Machine Learning at Extremes

by

Spencer Greenberg

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Mathematics

New York University

May 2016

Mehryar Mohri

Esteban Tabak

Dedication

To my parents, who taught me to love exploring ideas.

Acknowledgements

I would like to thank my advisors, professors Mohri and Tabak, for their indispensable guidance and mentorship, and for the many laughs we have shared together.

Abstract

Much can go wrong with machine learning algorithms and statistical techniques. This thesis investigates a number of cases where standard prediction methods perform poorly, and where standard approaches to analyzing such methods fail. For instance, extreme outliers in data can cause regression algorithms to make very poor predictions, while also calling into question boundedness assumptions used in the theoretical analysis of such algorithms. If the loss function of a regression algorithm is unbounded, as in many realistic scenarios, it raises the fundamental question: can we derive generalization guarantees when using unbounded loss functions? Another example is that the data points an algorithm is trained on may be of widely varying quality, rendering typical statistical estimators suboptimal. Even standard assumptions, for instance that it is desirable to completely minimize a machine learning algorithm's objective function, or that an algorithm's hypothesis set is data independent, or that smaller coefficients are more likely a priori than larger ones, as assumed in Ridge Regression, can be substantially violated in practice.

In this thesis we investigate cases that are extreme, in that they can dramatically thwart standard methods, but that actually occur when working with common algorithms. We provide solutions to address the problems listed above, and other problems that occur when making predictions from data, by applying methods based on outlier clipping, point reweighting, early stopping, and new choices for regularization penalties. Additionally, we give theoretical guarantees for learning with unbounded loss functions using an analysis based on relative deviations bounds. The solutions proposed in this thesis may significantly improve predictions in real-world applications, as well as our theoretical understanding of prediction algorithms.

Contents

List of Figures

List of Tables

Part I

Introduction

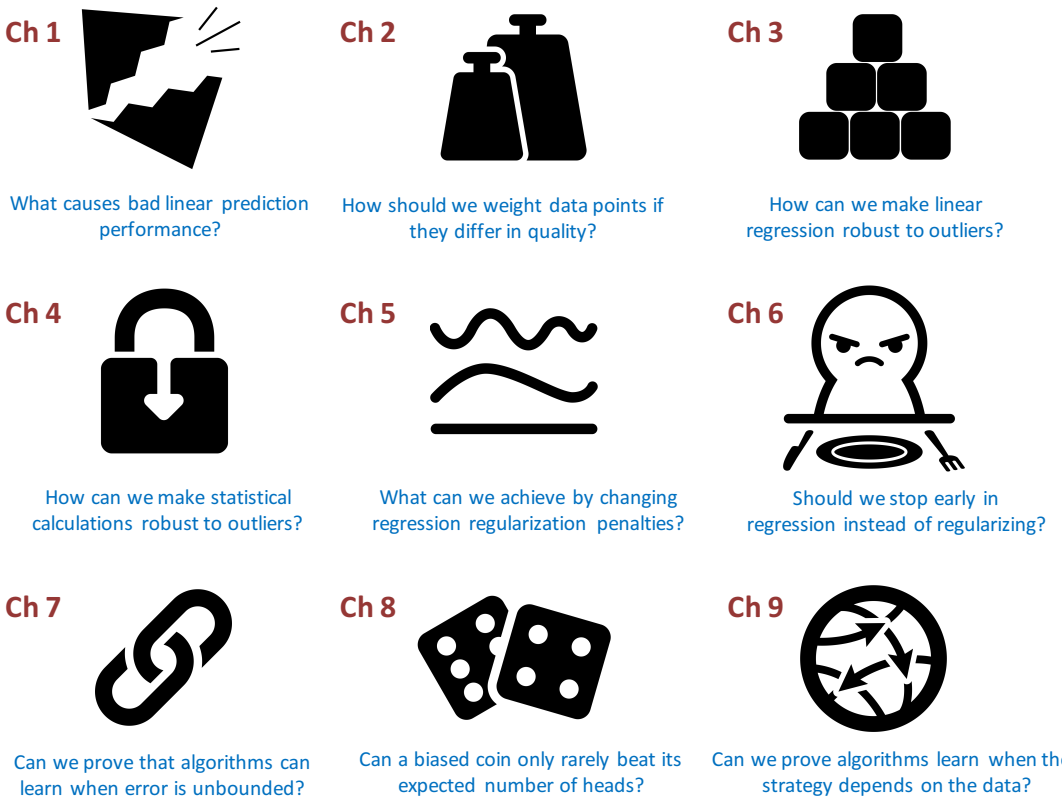

Figure 1: The image above summarizes the questions that we investigate in each chapter of this thesis.

Much can go wrong with machine learning algorithms and statistical techniques. This thesis investigates a number of cases where standard prediction methods perform poorly, and where standard approaches to analyzing such methods fail.

In Part II of this thesis we examine a number of specific examples where common algorithms can break down or have suboptimal performance. We do so by applying Ordinary Least Squares and Ridge Regression to synthetic test cases, each case specifically designed to illustrate one type of prediction failure. In Part III, we propose methods for handling a number of these failures, through techniques such as point reweighting, clipping, outlier

removal, and non-standard regularization. Finally, in Part IV, we introduce mathematical theory that can be useful for analyzing the performance of machine learning algorithms that do not conform to certain standard assumptions of machine learning theory. Our emphasis in this work is on regression (i.e. the prediction of real valued variables), rather than classification (i.e. the prediction of categories), but some of our results are very general, applying to many machine learning algorithms, as well as statistical estimation methods more generally.

More specifically, in this thesis, we discuss and propose solutions for the following issues, each of which can cause suboptimal prediction performance, throw off the results of statistical calculations, or prevent an algorithm from being analyzed successfully:

- **Data points of varying quality.** When data points differ in quality it generally causes standard prediction methods to have suboptimal performance, as they fail to take these differences into account. Higher quality points should be given more weight than lower quality ones. We explore an example of this phenomena in Chapter 1, and provide a framework for handling this type of problem in Chapter 2.

- **Outliers in data.** When extreme outliers occur in data, they hurt the prediction performance of many standard methods. Outliers can be removed or given lower weight. We study examples of outliers distorting predictions in Chapter 1, propose a method for handling outliers in the context of Ridge Regression in Chapter 3, and provide a method for handling outliers in univariate data in Chapter 4.

- **Prior information available.** When prior information is known about a prediction problem, it is important to build this information into the solution in order to improve performance. We examine some synthetic examples of this phenomena in Chapter 1, and discuss an approach to incorporating prior information in the context of kernelized

Ridge Regression in Chapter 5, through the introduction of regularization penalties that encode this information.

- **Overfitting of the training data.** The "complexity" of a machine learning algorithm must be appropriate for the learning problem at hand, otherwise noise may have too great an influence and overfitting may occur. Overfitting can be counteracted in a variety of ways, such as by introducing a regularization penalty, or by applying early stopping. An approach to regularization that generalizes the standard Ridge Regression regularization is analyzed in Chapter 5, and an efficient method for avoiding overfitting via gradient descent with cross-validated early stopping is discussed in Chapter 6.

- **Wasting time converging.** Machine learning algorithms can waste precious time during training by continuing to learn beyond the point of benefit. Early stopping is sometimes an appealing approach to solving this problem, as discussed in Chapter 6.

- **Unbounded loss functions.** When the error function used by a machine learning algorithm is unbounded, it invalidates many of the standard analyses regarding the ability of these algorithms to generalize what they have learned to future, unseen data. We derive generalization bounds for unbounded loss functions in Chapter 7, and prove a theorem about binomial distributions that is required in this analysis in Chapter 8.

- **Data dependent hypotheses.** When the hypothesis set explored by a machine learning algorithm depends on the data, it prevents the application of most approaches to analyzing an algorithm's ability to generalize to future data, such as using Rademacher Complexity or Uniform Stability. The case of data dependent hypothesis sets is surprisingly common. It happens, for instance, whenever an algorithm uses k-fold cross-validation to set parameters. We introduce a generalization of Rademacher Complexity and Uniform Stability in Chapter 9, that unifies the two theories and allows them to

apply to data dependent hypothesis sets.

Please note that the Python code files implementing the algorithms discussed in this thesis are placed in the public domain, and can be downloaded at the URL in the footnote below[1]. We now begin our discussion of machine learning at extremes: how standard methods can fail, and what can be done to improve them.

[1] http://spencergreenberg.com/code/doctoral_thesis_machine_learning_code.zip

Part II

What Can Go Wrong

Chapter 1

Breaking Ordinary Least Squares and Ridge Regression

Our investigation will begin with examining two very popular prediction algorithms, Ordinary Least Squares regression, and Ridge Regression. We will use hand constructed data sets to investigate many of the ways that these algorithms can make inaccurate predictions. Non-linear problems are the obvious cases where linear models like these fail, but as we will see there are many other ways to break these algorithms. On some linear problems, these algorithms are incapable of even learning at all.

The empirical test cases we examine will help us gain intuition for many of the topics discussed in later chapters. For instance, we will explore cases where the quality of different data points varies, which we will develop an approach to handle in Chapter 2. What is more, some of these cases will have outliers, and we will discuss algorithms to handle outliers in Chapters 3 and 4. We will also investigate cases where information is known that is external to the training set, which will tie into our discussion in Chapter 5 of how regularization can be used to encode prior information. Finally, we will discuss cases where

the potential for overfitting is high, which will be a theme that we touch on again in Chapter 6.

We now proceed to introduce Ordinary Least Squares regression and Ridge Regression, and investigate how these algorithms break.

1.1 Introduction

Ordinary Least Squares regression is probably the most commonly used prediction algorithm in the world. This algorithm makes predictions from data by using the linear solution coefficients w^{ols} that minimize the squared prediction error, given by:

$$w^{ols} \equiv \underset{w}{\operatorname{argmin}} \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2$$

where X is a matrix of training data with m points (one per column) and n features (one per row), and y is a column vector of real valued training labels. The solution to this optimization problem can be written:

$$w^{ols} = (XX^{\intercal})^{-1} Xy, \tag{1.1}$$

which we can then use to predict the label of any point x (represented as a column vector) by taking $x^{\intercal} w^{ols}$. Despite its success in a huge range of prediction problems, Ordinary Least Squares makes poor predictions for some types of problems, even some linear ones, and this has motivated alternative algorithms. Ridge Regression is probably the most widespread linear alternative. It improves performance in some contexts by solving a generalization of the Ordinary Least Squares optimization problem, and has linear solution coefficients defined as:

$$w^{ridge} \equiv \underset{w}{\operatorname{argmin}} \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2 + \lambda \sum_{i=1}^{n} {w_i}^2$$

where λ is a free parameter which determines how strongly to penalize large coefficients, typically set using k-fold cross-validation. The linear solution coefficients for Ridge Regression can also be written:

$$w^{ridge} = (XX^\intercal + \lambda I)^{-1} Xy. \tag{1.2}$$

But, as we will demonstrate, even Ridge Regression performs poorly on certain types of problems, even some linear ones with simple additive gaussian label noise.

In order to better understand the limitations of Ordinary Least Squares and Ridge Regression, when alternatives are preferable, and what can cause prediction algorithms to break down in general, this chapter will explore data sets where:

1. Ridge Regression outperforms Ordinary Least Squares
2. Ordinary Least Squares outperforms Ridge Regression
3. Modifications to the standard algorithms will outperform both Ridge Regression and Ordinary Least Squares
4. Neither Ridge Regression nor Ordinary Least Squares are able to learn

We draw conclusions about why Ridge Regression is usually preferable to Ordinary Least Squares, and about why, in some cases, alternative linear algorithms are preferable to both.

We will investigate cases that cause suboptimal performance for Ordinary Least Squares and Ridge Regression, and suggest ways that superior performance can be achieved. Unless otherwise noted, the examples we use will all be modifications to a simple "default case", which we will discuss first. All data sets used in this chapter can be downloaded at the URL found in

the footnote below[1] [2]. The Python code files for reproducing the performance numbers found in this chapter are in the public domain, and can be downloaded at the URL in the following footnote[3]. Throughout this chapter we will be referencing the R^2 results from Table 1.1, which show the percentage of label variance captured by various algorithms. The way we define R^2 allows the possibility of negative values, which occur when an algorithm underperforms what you would get just predicting the test set mean for all test set labels. However, since $R^2 < 0$ and $R^2 = 0$ both imply that a prediction algorithm is terrible, we do not differentiate between the two cases in this chapter, and record both as 0% accuracy for simplicity and ease of reading. More details about our R^2 calculations are provided in the caption beneath Table 1.1.

In Table 1.1, in addition to showing accuracy numbers for Ordinary Least Squares (OLS) and Ridge Regression (Ridge), which are the primary algorithms of interest in this chapter, we also show the accuracy of some other variations on these algorithms that provide points of comparison.

We now define these other algorithms. To simplify these descriptions, let us define R_j^2 to be the label R^2 achieved when using just the jth feature of the data to predict the training set labels, in a one dimensional regression of the form $a + bx_j$, where x_j is the jth feature of point x. With that definition in place, we are ready to describe the algorithms shown in Table 1.1:

- Lasso, which applies Lasso Regression, is much like Ridge Regression but with a penalty on the sum of the absolute value of the solution coefficients instead of the sum of the square of the solution coefficients. The penalty is still multiplied by a constant λ, which

[1] http://www.spencergreenberg.com/code/data_sets_breaking_ridge_regression.zip

[2] Due to size considerations, in the downloadable copy of our data sets, we have truncated the test set for each case to 2,000 points, rather than the 40,000 used in computing out of sample accuracies in this chapter.

[3] http://spencergreenberg.com/code/doctoral_thesis_machine_learning_code.zip

is set using 10-fold cross-validation.

- PCA, which performs principal component analysis on the positive definite matrix $XX^\intercal$ to reduce the dimensionality of the data to just d dimensions, and then runs Ordinary Least Squares on those d dimensions, with 10-fold cross-validation used to choose d. All values of d from 1 up to the numerically calculated rank of X are tried.

- Top-Fts, which applies Ordinary Least Squares to only the d features with highest R_j^2 values, with 10-fold cross-validation used to choose d.

- 1-Ft, which is like Top-Fts, but it applies Ordinary Least Squares to only the single feature with the highest R_j^2, ignoring all other features.

- Wgt-1d, which is like Ridge Regression, but the penalty applied to each feature is $\lambda(\frac{1}{R_j^2} - 1)$ instead of just λ, where as usual λ is set using 10-fold cross-validation. This means that features that perform better on their own (when used in one dimensional regressions) are penalized less than features that perform worse on their own. It is interesting to note that on the test cases in this chapter, this algorithm's performance was almost always between that of Ridge and Lasso, or falling only very slightly outside of this range. The only exception is one case where Wgt-1d performs much better than Ridge and Lasso. In Chapter 5 of this thesis we explore the construction of algorithms like this one.

1.1.1 Our Default Case

Our default case, which we use as a starting point for all the other cases, has been selected to be an easy problem for Ordinary Least Squares and Ridge Regression, as well as one of the simplest examples of multivariate linear regression. As such, the labels are generated by a linear model, the training set has plenty of data points, the noise-to-signal level in the labels

R^2 Accuracy of Different Algorithms on Different Data Sets

	Data Set	m	n	N/S	OLS	Ridge	Lasso	PCA	Top-Fts	1-Ft	Wgt-1d
1	Default	1000	100	0.3	91	91	91	91	90	7	91
2	Few Pts Per Ft	110	100	0.3	19	63	66	65	52	11	65
3	Noisy Labels	1000	100	3.0	0	5	3	0	0	0	3
4	Uncorrelated Fts	110	100	0.3	26	62	60	62	34	11	64
	Correlated Fts	110	100	0.3	0	89	88	88	79	30	88
5	Ft Outlier	1000	100	0.3	88	2	23	21	2	2	19
6	Irrelevant Fts	200	200	0.3	0	64	91	58	92	92	91
7	Fts Decreasing	100	100	0.3	0	64	80	56	72	55	75
8	Unnormalized Fts	105	100	0.3	0	0	0	0	55	4	46
	Normalized Fts	105	100	0.3	0	67	66	49	55	4	63
9	Prior Solutions	105	100	0.3	0	72	47	64	31	0	52
10	Singular Fts	100	400	0.3	0	16	4	20	0	0	15
11	Varying Pt Noise	1000	100	1.5	21	23	22	19	20	0	21
12	Label Outlier	1000	100	0.3	0	0	0	0	0	0	0
13	Heavy Tailed	1000	100	0.3	0	0	0	0	0	1	0
14	Non-Linear	1000	100	0.3	0	0	0	0	0	0	0

Table 1.1: Each row in the table above corresponds to a different data set. Performance on the default data set is shown in the first row, which has 1000 training points (m) and 100 features (n), with a noise-to-signal ratio (N/S) of 0.3, where all the values in the training data are independent and generated from an isotropic unit gaussian. Each other data set is a variation on the default data set as described in detail elsewhere in this chapter, having been modified to illustrate a particular idea. The highest accuracy scores in each row are colored to stand out.

All columns to the right of N/S represent the percentage of variance in the labels that was captured by different machine learning algorithms on 40,000 out of sample points. That is, we measure accuracy using the test set R^2 times 100, where R^2 is defined as one minus the ratio of the sum of squared prediction errors to the sum of squared differences between the test set labels and the test set label mean. These R^2 values can be negative (indicating a prediction accuracy worse than is achieved by the out of sample mean). In some of our test cases these negative values have enormous magnitudes. Since a negative value and zero value both indicate terrible predictions, all negative values are displayed as 0 for ease of reading. For cases where an algorithm could not be run (e.g. OLS when the number of features exceeds the number of points) the variance captured was also recorded to be 0. The first algorithms shown are Ordinary Least Squares regression (OLS), Ridge Regression (Ridge), and Lasso Regression (Lasso). Ridge and Lasso both apply 10-fold cross-validation, trying an exponentially increasing set of thirty λ (i.e. regularization parameter) values ranging from 10^{-6} to 10^{6} (as do all the other algorithms in this chapter that use k-fold cross-validation to set a free parameter). The remaining algorithms are described elsewhere in this chapter.

is modest, and both the features and label noise are drawn independently from a gaussian distribution. By modifying this basic example strategically, we will be able to degrade the performance of Ordinary Least Squares and Ridge Regression while illustrating many reasons why the two algorithms can fail, and how modifications to these algorithms can perform better. As seen in the "Default" row of Table 1.1, Ordinary Least Squares (OLS) and Ridge Regression (Ridge) each capture 91% of the variance in the test set labels for the default case.

Specifically, our default case consists of 1000 training points and 40,000 testing points (to ensure low out of sample variability). Each point has 100 features. A random vector of 100 linear model coefficients is generated from an isotropic unit gaussian (i.e. a multivariate gaussian distribution with zero mean and the identity matrix used as the covariance matrix) plus one constant term generated from a unit gaussian, and the labels for the training and testing points are then set by multiplying the training and testing points against these linear coefficients (plus the constant term), and then adding independent gaussian noise. The standard deviation of the noise is set so as to achieve a noise-to-signal ratio of 0.3. This is done by first computing the standard deviation of the noiseless labels (i.e. the labels before any noise has yet been added), then generating a vector of isotropic gaussian noise, then multiplying this noise vector by the desired noise-to-signal ratio times the standard deviation of the noiseless labels divided by the standard deviation of the noise vector itself. This gives the noise a standard deviation equal to the noise-to-signal ratio times the standard deviation of the noiseless labels, precisely as desired. The resulting noise is then added to the noiseless labels to form the final label vector, y.

The features for each point are generated independently from an isotropic unit gaussian distribution. After the data is generated, the mean and standard deviation of each feature of the training points is calculated. These means are subtracted from the corresponding

features of both the training data and testing data, and each feature in the training data and testing data is divided by the corresponding feature standard deviations (calculated from the training set). Note that for simplicity we do not add a feature of all 1's to the matrix X, so each of the linear prediction algorithms is forced to have an intercept of 0. We tested adding such a constant feature, but it did not cause our conclusions to change.

Except in the ways that are explicitly noted for each test case, all test cases use precisely the same data generation process that is used by the default case. Each of the test cases that follows is generated from scratch using a fixed random seed. That means that when parameters such as the number of features change, the data that is used will generally change in its entirety, since with a different number of features the sequence of random numbers being generated changes. This means that the performance numbers will vary with a change of random seed. We have, however, confirmed on a handful of different random seeds that the takeaways here do not change, even though the precise performance numbers do, except in places where that fact has been noted below.

You will find that many of the cases that follow are extreme examples. This is by design, as it is often the extreme examples that teach us the most. With that in mind, we will now explore a variety of modifications to the default case, which are carefully designed to affect the performance of Ordinary Least Squares and Ridge Regression.

1.2 Ridge Regression Outperforming Ordinary Least Squares

In this section, will discuss cases where Ridge Regression outperforms Ordinary Least Squares. Ridge Regression is a popular modification to Ordinary Least Squares for good reason. Ordinary Least Squares is often very effective at making predictions, but there are quite a few important cases where Ridge Regression achieves much better performance, with little drawback. In fact, the main drawback of Ridge Regression is increased running time, which is the result of needing to test many values of λ. In our default case Ordinary Least Squares and Ridge Regression achieve essentially the same accuracy. We now consider modifications to the default case where Ridge Regression performs significantly better.

1.2.1 Few Points Per Feature

A simple way to make Ridge Regression outperform Ordinary Least Squares is to make the ratio of training points to features small. In the "Few Pts Per Ft" row of Table 1.1, we modify the setup of the default case so that it has only 110 training points, rather than 1000. We still use 100 features though, giving us a very low point to feature ratio. As might be anticipated, this makes learning much harder for any algorithm. But Ordinary Least Squares is impacted much more greatly than Ridge Regression, capturing only 19% of the label variance compared to Ridge Regression's 63%.

The issue here is, of course, overfitting. With so many features per point, there are many vectors of solution coefficients that achieve high accuracy in the training set, but most of these will perform badly out of sample. Essentially, the model has too much flexibility due to having so many variables, so fits even the noise. By attempting to minimize the training error

only, Ordinary Least Squares inevitably selects one of these models that does not actually generalize well, being overly reactive to the label noise. Ridge Regression, on the other hand, selects a λ parameter through k-fold cross-validation. The parameter λ reduces the size of the solution search space, and eliminates from consideration solutions with very large coefficients. The result is a model that does not overfit the training data, and so generalizes well to the testing data.

1.2.2 Noisy Labels

Another advantage that Ridge Regression has over Ordinary Least Squares is greater resilience to label noise. To illustrate this, we modify the setup for the default case by adjusting the noise-to-signal ratio to 3 instead of 0.3. As can be seen in the "Noisy Labels" row of Table 1.1, this prevents Ordinary Least Squares from learning anything, capturing 0% of the variance, whereas Ridge Regression still manages to learn a little, capturing 5% of the variance. Ridge Regression's superior performance under high noise comes from its use of λ to adjust the bias-variance tradeoff. With $\lambda \approx 0$, it cares primarily about minimizing the prediction error, and subsequently the solution depends a lot on the specific realization of the noise. Since the noise is so large, having the solution be so sensitive to the exact labels is problematic. However, with λ larger, this dependence on the noise is reduced, as the algorithm trades off a greater bias (i.e. a solution closer to the zero vector) in exchange for reduced variance. By using k-fold cross-validation to select λ, an approximately optimal tradeoff between bias and variance is struck. When the noise level is very high, a larger λ tends to be chosen, as less reactivity to noise is desirable.

In Chapter 5 of this thesis we discuss in detail how the generalization performance of Ridge Regression is impacted by noise, and why regularization helps.

1.2.3 Correlated Features

Yet another situation that can throw off the performance of Ordinary Least Squares is when there are highly correlated, noisy features. Consider the "Uncorrelated Fts" row of Table 1.1. This case begins with the default case, but uses only 110 training points, and corrupts the training and testing features matrices by adding gaussian noise. Note that in the default case while the features are generated from an isotropic gaussian distribution, there is no noise added to them. Ordinary Least Squares performs poorly on this case, capturing only 26% of the variance. This is easily explained by the fact that there are 100 features, which is almost equal to the number of points. But comparing this result to the "Correlated Fts" row of Table 1.1, we see a new phenomena occur. This case is generated in precisely the same way as the "Uncorrelated Fts" case from Table 1.1, except that it starts with only 10 distinct features instead of 100, with each of those 10 features then being duplicated 10 times to produce a full set of 100 features. Finally, additive gaussian noise is added to the resulting training and testing features, exactly as in "Uncorrelated Fts". In this case though, the additive noise serves to prevent each feature from being identical to its duplicates.

The result is that for this new case "Correlated Fts", all algorithms shown in the table greatly improve in accuracy compared to the "Uncorrelated Fts" case, except Ordinary Least Squares, which now captures 0% of the variance! Ridge Regression handles this new problem easily, capturing 89%, compared to 62% in the "Uncorrelated Fts" case.

Why would all other algorithms in the table improve substantially while Ordinary Least Squares becomes incapable of learning? The trouble here seems to be that using nearly identical noisy features leads to multiple solution coefficient vectors that are very different from each other, but that have almost the same training set accuracy, leading to an arbitrary

choice being selected. The noise then becomes largely responsible for which of these solutions gets picked. For instance, if c_1 and c_2 are the coefficients for nearly identical features, Ordinary Least Squares is almost indifferent between making predictions using the model $2c_1 - c_2$ or the model $4c_1 - 3c_2$, since they will lead to almost the same predictions on the training set. This is related to the problem we saw in the "Few Pts Per Ft" case, which also involves having too many solutions to choose from, but here we compound the effect through a different mechanism. In "Few Pts Per Ft" the issue was that we had too many model parameters due to having too many features. Here, by making the features noisy duplicates of each other, we layer on top the problem that solutions with nearly identical performance can be constructed by combining the redundant variables in different amounts. This makes it even harder for Ordinary Least Squares to tell the difference between a useful and useless model, because so many models perform well on the training set.

Ridge Regression resolves the problem that Ordinary Least Squares has by penalizing large coefficients, and therefore preferring the model $2c_1 - c_2$ to the model $4c_1 - 3c_2$. If we compare the solution coefficients found by both algorithms for the "Uncorrelated Fts" case versus the "Correlated Fts" case, we find that in the latter case Ordinary Least Squares ends up with coefficients whose median magnitude is more than 3x that of the former. Ridge Regression has the opposite behavior, using coefficients of much smaller median magnitude in the latter case than the former [4].

[4]We note that, for one random seed, we found a case where Ordinary Least Squares actually improved its performance on "Correlated Fts" compared to "Uncorrelated Fts" , but on all the other random seeds that we tested, we found an effect in the opposite direction.

1.3 Ordinary Least Squares Outperforming Ridge Regression

While Ridge Regression is generally a superior prediction algorithm to Least Squares, it is not a strictly better algorithm. We will now discuss a case where Ridge Regression substantially underperforms Ordinary Least Squares.

1.3.1 Feature Outliers

A "feature outlier" is a training point that lies unreasonably far from the other training points. Of course, since points involve multiple dimensions, the question is raised as to which distance metric is appropriate for measuring how far points are from each other. Fortunately, we do not need to resolve this thorny issue to study the impact that feature outliers have on Ordinary Least Squares and Ridge Regression.

In the "Ft Outlier" row of Table 1.1, we apply our algorithms to a data set generated in the same manner as the default case, but we multiply all features of the first point by 500, producing a point that would be considered an extreme feature outlier using any reasonable distance function. We furthermore assign this point a label value of 1, so that its label is not unreasonably large, but is inappropriate given the features for that point[5].

Noting the performance shown in Table 1.1, we see that the impact of a feature outlier is very different from that of a label outlier (i.e. a point with extreme labels rather than extreme features, which we will consider later on, and which is also shown in the table). Ordinary

[5]Note that in all cases described in this chapter where we replace specific points in the data, such as when adding a feature outlier, the noise-to-signal ratio is set based on only the other data points (not including the point replacements), otherwise the amount of noise would be heavily distorted by the introduction of an outlier.

Least Squares handles the feature outlier well, capturing 88% of the variance, whereas Ridge Regression's performance is destroyed, capturing only 2%. This may be surprising to those who consider Ridge Regression a more robust algorithm than Ordinary Least Squares (e.g. due to the introduction of regularization), or who think that because it is a generalization of Ordinary Least Squares it must be a strict improvement.

Let us take a moment to consider what feature outliers do to these algorithms. In 1d space a feature outlier is a real problem because an extremely bad prediction will occur on that outlier unless the (single) solution coefficient is close to zero. This renders all predictions close to zero. In higher dimensions though, a vector of linear solution coefficients has the flexibility to predict a value close to zero for the feature outlier, while still making reasonable predictions on the other points. It simply requires the solution coefficient vector to be close to orthogonal with the point that is the feature outlier, a single linear constraint on the solution coefficients. Some distortion does occur in general, but it is not devastating. Hence, the reason why Ordinary Least Squares is not substantially impacted in the case at hand. It has plenty of dimensions to use to work around that outlier.

So then why does Ridge Regression perform so poorly? It is actually due to the k-fold cross-validation procedure, which itself is not robust to outliers, causing a poor selection of the λ parameter when a feature outlier is present. If we replace the standard k-fold cross-validation procedure with a more robust procedure that clips (i.e. winsorizes without removing) the top 5% largest errors and the bottom 5% smallest errors during cross-validation before taking the mean to get the average error, then performance of Ridge Regression on this data set jumps all the way back up to 88% (not shown in the table). This is the same accuracy as Ordinary Least Squares.

The reason that the standard k-fold cross-validation is problematic in the presence of a feature outlier is because such a point is terribly predicted when it happens to be in the out-of-sample set, causing a very large error that can swamp the error of the others points. So the feature outlier can cause a terrible selection of λ [6]. Note that while using a more robust variant on k-fold cross-validation has substantial benefits in the presence of feature outliers, it does not (as we will see) help with label outliers, since it does not resolve the problem that Ridge Regression's solution coefficients depend linearly on the label vector y.

In Chapter 3 of this thesis we explore a technique for handling feature outliers, analyze their impact mathematically, and discuss robust k-fold cross-validation at greater length.

1.4 Modifications Outperforming Ridge Regression and Ordinary Least Squares

Even in cases where Ridge Regression or Ordinary Least Squares perform reasonably well, we may be able to significantly outperform both of them by modifying the algorithms or using alternative (but related) algorithms. Here we discuss cases of this type.

1.4.1 Irrelevant Features

Let us consider a situation where only one feature is actually useful for predicting the labels. We construct this case by starting with the setup of the default case, then adjusting the number of points and features to both be equal to 200, and finally, generating the labels by only using the first feature, which we assign a coefficient of 1. The results are shown in the "Irrelevant Fts" row of Table 1.1. Ordinary Least Squares of course performs terribly since

[6] We note that we observed feature outliers to have somewhat less impact on Ridge Regression when using leave-one-out cross-validation to select λ, compared to 10-fold cross-validation.

there is only one point per feature, capturing 0% of the variance. On the other hand, Ridge Regression captures 64% of the variance, which it turns out is far from optimal. We can tell because Lasso Regression, shown as "Lasso" in the table, captures 91% of the variance. Lasso Regression involves minimizing an objective function much like that of Ridge Regression, but with a penalty on the absolute value of each coefficient, instead of the square of each coefficient. Hence, the Lasso Regression solution coefficients are defined by:

$$w^{lasso} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2 + \lambda \sum_{i=1}^{n} |w_i|$$

For this case, where we have many irrelevant features, Ridge Regression has a problem because the squared penalty it uses prefers many coefficients of small magnitude to one coefficient of large magnitude, but one large coefficient is precisely what is needed. To be more precise, if all of our coefficients have magnitude ϵ, then Ridge Regression assigns a penalty proportional to $\sum_{i=1}^{n} {w_i}^2 = n\epsilon^2$, which for reasonably large n is much smaller than what we get in a case where one coefficient has magnitude $n\epsilon$, and all the others are 0, which would yield a penalty $n^2\epsilon^2$. Hence, Ridge Regression has a built in preference to use many features a little bit, rather than one feature a lot, which is directly harmful in this case. Lasso Regression, on the other hand, assigns precisely the same penalty to n coefficients each with a magnitude ϵ as it does to one coefficient with a magnitude $n\epsilon$, and therefore is perfectly willing to use just a small number of non-zero coefficients. Relatedly, Lasso Regression tends to produce sparse solutions, where many coefficients are zero, which is well suited to this particular case.

As we might expect, the algorithm "1-Ft" performs well in this case, because it makes predictions using Ordinary Least Squares applied to only the single feature that best linearly predicts the labels in a one dimensional regression. In other words, it is perfectly suited to this case, capturing 92% of the variance. It is remarkable that Lasso Regression does almost

as well, despite making much weaker assumptions about the features, achieving 91% accuracy.

We see therefore that while Ridge Regression and Lasso Regression both have a preference for small solution coefficients, they make different implicit assumptions about the distribution of these coefficients, and therefore will be suited to somewhat different problems. For instance, Ridge Regression will tend to have an advantage when most features are about equally useful, whereas Lasso Regression will tend to have an advantage when most features are useless.

1.4.2 Features of Decreasing Usefulness

Previously we saw that Lasso Regression can have an advantage over Ridge Regression if most of the true coefficients are zero in the linear model generating the labels. It may therefore be tempting to conclude that this outperformance of Lasso Regression hinges on sparsity of the features. In the "Fts Decreasing" row of Table 1.1 we consider a related but non-sparse case where Lasso Regression again substantially outperforms Ridge Regression. This case follows the setup of the default case except that it uses only 100 training points, and the true coefficients are chosen to be $(1/2, 1/3, 1/4, \ldots, 1/(m+1))$. Because the label noise is independent of the features, this means that the first feature is the most useful for making predictions, the second feature the second most useful, and so on.

Why would Ridge Regression perform relatively poorly in this case? We believe the intuition is as follows. Since the number of points and features is equal, there is a very high potential for overfitting the data. To stop this from happening, Ridge Regression cannot choose a small λ. But this means that large coefficients are penalized significantly, so it prefers to use a greater number of features with smaller coefficients (even if they are not that

useful) to having larger coefficients in the earlier (i.e. most useful) features. For this case, doing so is a particularly bad idea. Lasso Regression, on the other hand, mostly ignores the less useful coefficients, assigning many of them a zero value. At first glance this may not seem like a great idea (since all the features are useful to a degree), but given a fixed noise level it is wiser to focus attention on the earlier features, which have a lower noise-to-signal ratio. In doing so, Lasso Regression captures 80% of the variance, compared to Ridge Regression's 64%.

We find that the "Wgt-1d" algorithm also does quite well in this case. It exploits the fact that, in this situation, a feature working well independently in a one dimensional regression is a good indicator of it working when combined with the other features. This algorithm captures 75% of the variance, which is slightly inferior to Lasso Regression's 80%.

The algorithm "1-Ft" shown in a columns of Table 1.1 gives us some further insight into this case. This algorithm simply tries to predict the training labels one-by-one using each feature in its own one-dimensional regression, then selects the single feature that performs the best on the training set, and uses that best feature alone to predict labels for the test set. On the "Fts Decreasing" case this simple algorithm captures 55% of the variance just using a single feature, not terribly far from Ridge Regression's 64%.

1.4.3 Poor Normalization

Ordinary Least Squares has the useful property that its prediction accuracy is invariant to full rank linear transformations of the points matrix X. But for Ridge Regression this is not the case. In fact, poor normalization of the features can lead to poor performance. Even multiplying one feature by a constant (e.g. changing the units of the feature from inches to feet) can have an impact on prediction accuracy, since large coefficients are penalized

by Ridge Regression, and if the values of a feature are made smaller, the corresponding coefficient must become larger to compensate (hence being penalized more). To illustrate this in an extreme form, we construct a case where the variance of the ith feature is set to 1.1^{i-1}, whereas the corresponding coefficient for the ith feature is set to the reciprocal $1.1^{-(i-1)}$. This setup also differs from the default case because we use just 105 training points.

Now, we examine the impact of leaving our features unnormalized (unlike the default case, which is normalized). The result is shown in the row "Unnormalized Fts" of Table 1.1. We compare this result with a case generated in exactly the same way, except using the standard normalization, where we subtract from each feature the training means and divide each feature by the training standard deviations. The result is shown in the row "Normalized Fts". Ridge Regression is unable to learn at all in the unnormalized case, capturing 0% of the variance, whereas after normalization it captures 67%. On poorly normalized data, Ridge Regression regularization sometimes cannot operate properly. We note that the only algorithm from Table 1.1 that performs reasonably well when the features are unnormalized is Top-Fts, which captures 55% of the variance.

1.4.4 Using Previous Solutions

One simple way to outperform Ordinary Least Squares and Ridge Regression, even on simple linear problems, is to use information that is external to our training data. For instance, we may have solved similar prediction problems in the past, or we we may be able to view our current prediction problem as coming from a set of other similar prediction problems, with the average of solution coefficients on those previous problems producing an estimate of the solution coefficients for this problem. For instance, suppose we are trying to build a model to predict the number of strikeouts that one particular baseball pitcher will

have in a game based on various factors related to him and the opposing team. In this case it could be very beneficial to incorporate models we have learned for other pitches, and use them to inform our model for this particular pitcher.

We demonstrate an example of this phenomenon in the "Prior Solutions" row in Table 1.1, where we treat the underlying linear coefficients that generate our labels as being drawn from a probability distribution over coefficient vectors. We generate a random mean vector with size equal to the number of features, and a random positive definite matrix to use as a covariance matrix, and then we draw 250 possible solution coefficient vectors from a multivariate normal distribution using this mean vector and covariance matrix as its parameters. Only one of these 250 vectors (selected at random) is then used to generate the labels for our training data. In this case we assume that, as the machine learning practitioner, have access to the other 249 vectors of coefficients which reflect our past solutions to similar problems. This information can then be exploited to better estimate the 250th coefficient vector, which is the one of interest.

But how can we incorporate the other information into our current prediction problem? One approach is to compute the average of the other 249 coefficients, which we will call b, and modify our Ridge Regression problem so that rather than using regularization to push our coefficients towards zero, we instead use it to push our coefficients to this vector b. Then, the parameter λ, which is selected via k-fold cross-validation, determines how much we push our solution towards b, which ideally should be a function of how close b is to the solution we are looking for. The modified Ridge Regression optimization problem can be written:

$$w^b = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2 + \lambda \sum_{i=1}^{n} (w_i - b_i)^2$$

Intuitively speaking, if the true coefficients are closer to b than to 0, we may find that this approach is superior to standard Ridge Regression.

In the "Prior Solutions" row in Table 1.1 we see that the number of points we use for this problem compared to the default case has been reduced to just 110, which serves to heighten the effect, but otherwise we imitate the default case. On this problem, Ridge Regression captures 72% of the variance, which is pretty good. But by using the modified Ridge Regression problem above, and setting b to the mean of the solution coefficients that were drawn from the same distribution as the one of interest, we capture 76% of the variance (not shown in the table). The size of the improvement, while modest here, will depend on how far the true coefficients are from the zero vector (that Ridge Regression pushes towards) compared to b (which this modified Ridge Regression pushes towards).

In Chapter 5 of this thesis we explore modifications to the Ridge Regression objective function (like the one mentioned above) in greater detail.

1.4.5 Using Priors

In the previous section we discussed how prediction accuracy can be improved by using previous solutions to similar problems. But even better than that is to have a probabilistic model for how our solution coefficients were generated. This model then can act as a prior over the coefficients we are learning, and therefore be used to push our results towards a priori more likely solutions.

For instance, if this prior is a multivariate normal, or can be locally approximated as one around the most likely solution coefficients, we can incorporate this information into the

Ridge Regression problem by further modification to the regularization term. In particular, we can write:

$$w^{b,\Lambda} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2 + \lambda (w - b)^\intercal \Lambda (w - b)$$

where Λ is a matrix that is proportional to the precision matrix (i.e. the inverse of the covariance matrix) of our normal prior on the solution coefficients. Applying this method with the same setup as the previous section, we use the inverse of the covariance matrix of the 249 coefficient vectors from other related prediction problems to set Λ, and we set b as in the last section, which models the mean of our prior. As usual we use 10-fold cross-validation to set λ.

This prior based approach incorporates more external information than we did previously, and subsequently leads to an improvement in the variance captured. While Ridge Regression achieves 72%, and Ridge Regression modified as in the last section yields 76%, we now capture 82% of the variance (not shown in the table).

In Chapter 5 of this thesis we explore in greater detail this idea of incorporating priors into Ridge Regression and generalizing the regularization constant to a full matrix Λ.

1.4.6 Highly Singular Feature Matrix

From a mathematical standpoint, Ordinary Least Squares can be defined as the linear model whose solution coefficients are given by Equation 1.1. Unfortunatly, when the number of features exceeds the number of points, the inverse of $XX^\intercal$ is not defined. In such cases, we simply recorded a zero in Table 1.1 for Ordinary Least Squares. However, a slight generalization to Ordinary Least Squares makes it applicable when the number of features

exceeds the number of points. We write:

$$w^{pseudo} = (XX^\intercal)^\dagger Xy = X^{\intercal\dagger} y \tag{1.3}$$

where † means the Moore-Penrose pseudo inverse. This formula agrees with Equation 1.1 when $XX^\intercal$ is invertible, but also is defined when $XX^\intercal$ is not invertible.

The pseudo inverse has an interesting connection to Ridge Regression. Specifically it is what we get for λ infinitesimally small, that is:

$$\lim_{\lambda \to 0^+} w^{ridge} = \lim_{\lambda \to 0^+} (XX^\intercal + \lambda I)^{-1} Xy = X^{\intercal\dagger} y = w^{pseudo}.$$

Using the pseudo inverse based solution can be viewed as an intermediate between Ordinary Least Squares and Ridge Regression, as it does not introduce any free parameters, but it generalizes Ordinary Least Squares to handle the case of singular feature matrices (one of the benefits of Ridge Regression). Perhaps surprisingly, there are rather simple cases where the pseudo inverse based algorithm actually outperforms Ridge Regression, when Ordinary Least Squares is not defined. We show one such case in the "Singular Fts" row of Table 1.1. Whereas Ridge Regression captures 16% of the variance, the pseudo inverse based method captures 21% (not shown in the table), despite Ridge Regression having the choice of $\lambda \approx 0$ available to it. Hence, this underperformance of Ridge Regression can be viewed as a failure of k-fold cross-validation to do its job of selecting the best λ.

It is tricky to generalize about precisely when this kind of case will occur, other than

to say that if the truly optimal λ is approximately zero, then the pseudo inverse method is safer than Ridge Regression, since we do not run the risk of getting an erroneously high λ. In our experiments, outperformance of the pseudo inverse based algorithm over Ridge Regression occurred when the number of points was 100 and the number of features 400, but otherwise all settings were identical to the default case[7].

We note from Table 1.1 that principal component analysis based regression (PCA) also performs well in this case, capturing 20% of the variance. This algorithm reduces the dimensionality from 400 down to 84 on this case, before running Ordinary Least Squares. It seems that the unusually good accuracy comes from the algorithm being able to discard most features without losing much information, reducing this to an easier problem than attempting to directly learn coefficients for all 400 dimensions.

A note about implementation is warranted here. In this chapter, we implement our algorithms using numpy, which is the standard scientific computing package for the Python programming language. There are at least four ways to implement Ordinary Least Squares using numpy, and they have quite different running times. In fact, we have seen the difference in speeds between the fastest and slowest differ by 5x-20x in different runs (when the number of points and features were about equal). From fastest to slowest, these approaches to implementing Ordinary Least Squares with numpy are:

- $w = solve(XX^\intercal, Xy)$, using a linear equation solver, which uses an LU decomposition with partial pivoting and row interchanges.

- $w = inv(XX^\intercal)Xy$, using a matrix inversion algorithm, which applies $solve(A, I)$ to

[7]We note, however, that with two choices of random seed (one with no intercept, one with an intercept achieved by adding a feature of all ones) we saw Ridge Regression perform better on this test case than the pseudo inverse based algorithm, though with all of the other random seeds that we tried the pseudo inverse method was superior.

perform the inverse, where I is the identity matrix.

- $w = lstsq(X^\intercal, y)$, using a least squares solver, which computes the singular value decomposition in order to solve the least squares problem.
- $w = pinv(X^\intercal)y$, using a pseudo inverse algorithm, calculated by again making use of the singular value decomposition.

Both the lstsq and pinv algorithms take a parameter rcond. As explained in the numpy documentation, singular values are set to zero if they are less than rcond multiplied by the largest calculated singular value. This raises a few important issues.

First, it is true that in theory Equation 1.3 defines a strict generalization of Ordinary Least Squares that allows for cases where there are more features than points. At first glance then, it seems desirable to simply discard with Ordinary Least Squares as it is traditionally defined, and use this pseudo inverse based algorithm instead. But doing so can come at a 5x-20x increase in running time!

Second, saying that we have implemented Ordinary Least Squares is ambiguous, since different implementations can have somewhat different behavior. But this is even more so the case with the pseudo inverse variation of Ordinary Least Squares defined by Equation 1.3, because the pseudo inverse in actual practice has a free parameter, which is the threshold rcond for deciding when a singular value should be treated as zero. Due to rounding error, we never expect to get precisely zero singular values, yet it is essential that the pseudo inverse be able to take the reciprocal only of non-zero singular values, hence the need for such a threshold. Ideally, the "zero" singular values will be so close to zero, and the non-zero singular values sufficiently large that any reasonable choice of rcond will give the same results, but

this is not guaranteed. We conclude that this pseudo inverse generalization of Ordinary Least Squares is theoretically preferable to the regular version, but it is not necessarily preferable in practice.

1.4.7 Varying Point Noise

In the "Varying Pt Noise" row of Table 1.1 we consider a case that is just like the default case, except that we use a noise-to-signal ratio of 1.5 instead of 0.3, and the label noise added to each point is generated from independent normal distributions with different standard deviations (rather than all sharing the same standard deviation). In particular, the standard deviation of the normal distribution from which the ith point's additive label noise is drawn is given by $\left(\frac{||X^i||_2}{10}\right)^{20}$. The effect of this is that some points are very reliable (when they have small 2-norm), and others very unreliable (when they have large 2-norm). Due to the high overall level of label noise, Ordinary Least Squares only captures 21% of the variance, and Ridge Regression 23%. But if the standard deviations of the noise levels for each point were known we could do better. This might happen, for instance, if the labels for each point came from a measurement made by a device, and the noise characteristics of the device were known to depend in a certain way on the sample being measured. To improve our accuracy, we simply reweight our points based on how reliable each is expected to be, using a generalization of Ordinary Least Squares that takes into account a point weight matrix, V, whose ith diagonal element is the weight assigned to the ith point. The solution coefficients are then given by:

$$w^{weighted} = (XVX^\intercal)^{-1}XVy$$

To demonstrate how point weighting can improve accuracy when knowledge of each point's potential noise level can be estimated, we use a very simple weighting scheme, assigning V_i^i to be the reciprocal of the standard deviation of the normal distribution that generated the

label noise for the ith point. This weighted variant of Ordinary Least Squares captures 30% of the variance, a substantial improvement over the other algorithms.

In Chapter 2 of this thesis we discuss strategies for weighting low quality data points less than high quality data points.

1.5 Ridge Regression and Ordinary Least Squares Unable to Learn

In some cases it is not simply that Ridge Regression and Ordinary Least Squares have suboptimal performance, but that they are essentially incapable of learning at all. Here we discuss these most extreme cases, and make suggestions as to what can be done about them.

1.5.1 Label Outliers

Having one or more points with training labels far from the other training labels can destroy the performance of both Ordinary Least Squares and Ridge Regression. We call such a case a "label outlier", to distinguish it from a feature outlier. Outliers such as these may come from various sources, such as human error, measurement error, data corruption, fat tailed probability distributions, or chance events. Ideally, we would remove extreme label outliers from our training sample.

We examine the impact of a large label outlier by generating a data set in the same manner that we create the default case, but then modifying the first point so that it has an associated label of 100,000. This is a couple of orders of magnitudes larger than the typical label of largest magnitude in the default case.

We can see the results of doing so in the "Label Outlier" row of Table 1.1. Ordinary Least Squares and Ridge Regression, which we would otherwise expect to capture 90% of the variance in the labels, instead both capture 0%.

The explanation for the poor performance of these algorithms in the presence of a label outlier is simple. Examining Equations 1.1 and 1.2 we see that both depend linearly on the label column vector y, and therefore on the label outlier. The only advantage that Ridge Regression has here comes from its ability to choose a large λ through k-fold cross-validation. A large λ is chosen to try to prevent the outlier from having a large influence on the solution, but as a consequence, prevents all other points from having an impact on the solution as well, leading to solution coefficients that are all approximately zero. Label outliers can be handled by either detecting them explicitly and clipping (i.e. winsorizing) them or removing them from the training set, or by systematically assigning smaller (carefully selected) weights to points that appear to be more extreme than to points that are less extreme.

In Chapter 3 and Chapter 4 of this thesis we explore techniques for handling label outliers.

1.5.2 Heavy Tailed Features

Another way to cause Ordinary Least Squares and Ridge Regression to fail is to use features that span an extremely wide range, for instance by choosing them from a very heavy tailed distribution. In this case we choose the matrix X in just the same manner as the default case, but then instead of using X as our features, we use 32^X, where the power is taken element-wise. This leads to features that follow a log-normal distribution. All other

aspects of this case follow the default case, so the underlying model is still linear. But a linear model applied to log-normal features produces an extreme distribution for the labels, as shown in Figure 1.1. And as can be seen in the "Heavy Tailed" row of Table 1.1, all of the algorithms listed have great difficulty in this case, capturing approximately 0% of the variance[8].

It could be argued that the squared loss function, which is what Ordinary Least Squares and Ridge Regression both attempt to reduce, is probably inappropriate in this case. If we temporarily remove from the testing labels just the lowest and highest one thousandth of the values, it causes the variance of the testing labels to fall by about 59%. While certainly these tail values are "outliers" in the sense of being extreme, they cannot be considered outliers in the sense of being erroneous data. Since they are in the test set, we cannot simply remove them. Quite the opposite, the most extreme values account for most of what matters in these predictions when the squared loss function is used. So either the loss function will need to be reconsidered, or one must focus mainly on predicting the most extreme values. This latter option is a challenging prediction problem, since these extreme values represent a small percentage of the data, and even the extreme values can be at quite different scales.

1.5.3 Non-Linearities

Perhaps the most obvious way of breaking Ordinary Least Squares and (linear) Ridge Regression is to use labels generated from a non-linear model. One such example is shown in the row "Non-Linear" in Table 1.1. This test case is just like the setup of the default case, except that the labels are generated by a linear model applied to the element-wise absolute value of the features, rather than a linear model applied to the features directly. This renders linear models worthless since, for the absolute value function applied to each

[8]With a few random seeds, we found cases where an algorithm was able to capture some of the variance, but there was little consistency to this pattern.

individual feature, on average there is no trend either upwards or downwards in the relationship between each feature and the labels. All algorithms shown in the table capture 0% of the variance. The simplest way to allow Ordinary Least Squares and Ridge Regression to learn in this context is to switch our features (e.g. generating new features from the existing ones, so that the problem becomes approximately linear in the new features, even though it isn't in the original features). Relatedly, for Ridge Regression we have the option of switching to a non-linear kernel function as our method of measuring the similarity between points. With an appropriately selected kernel, we will then be able to model non-linearities, including the element-wise absolute value function that is used as the non-linearity in this case.

In Chapter 5 of this thesis we discuss the process of kernelizing Ridge Regression.

1.6 Conclusion

In this chapter we have investigated the performance of Ordinary Least Squares regression and Ridge Regression on one default case where learning is easy, and on thirteen other cases (derived from the default case) where it is significantly more challenging for one or both of these algorithms to learn. Each case is designed to illustrate a principle about how Ordinary Least Squares or Ridge Regression can break down, or how they can be improved. Our major findings, somewhat oversimplified, are as follows:

1. Ridge Regression can have an advantage over Ordinary Least Squares when there are few features per point, when there is a high level of label noise, and when their are many correlated, noisy features.

2. Ordinary Least Squares can outperform Ridge Regression though when there are feature outliers in the data.

3. Ridge Regression can be outperformed by Lasso Regression in situations where there are either very few useful features (out of a large set of features), or where there are many useful features of highly varying usefulness levels in a setting where overfitting is likely.

4. Ridge Regression can be thrown off in certain circumstances when the features of the data are not normalized.

5. When prior information is available either about solutions to similar problems, or about the prior distribution from which the current problem was drawn, this information can be used to improve prediction performance by modifying Ridge Regression to encapsulate this prior information.

6. In certain cases where the feature matrix is highly singular, a modified Ordinary Least Squares algorithm that uses the pseudo inverse instead of the regular inverse can outperform even Ridge Regression.

7. When information is known about how noisy each point label is expected to be, it can be incorporated into the prediction algorithm to improve accuracy.

8. If there are extreme label outliers, or a very heavy tailed distribution from which our features were drawn, or a highly non-linear model from which our labels were generated, this can prevent Ridge Regression and Ordinary Least Squares from learning at all.

An overarching takeaway from these examples is that Ridge Regression tends to be more reliable than Ordinary Least Squares. But as reliable as Ridge Regression is, we have shown that one can still outperform it in a wide variety of circumstances, even when the underlying

model that generated the data is linear. Additionally, we have given examples where Ridge Regression and Ordinary Least Squares essentially cannot learn at all. In such cases, a machine learning practitioner's detailed understanding of the problem at hand is critical for achieving accurate predictions.

Label Rank vs. Label Magnitude for "Heavy Tailed" Case

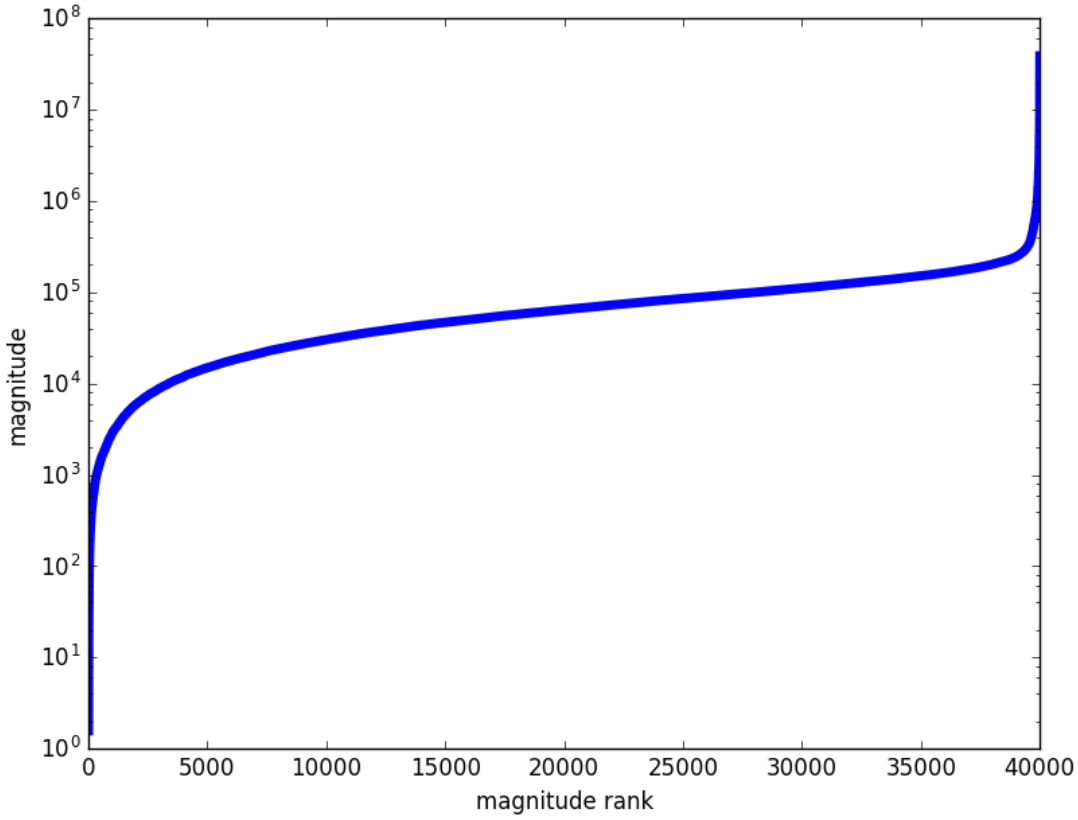

Figure 1.1: Here we have a log-plot of the test set labels for the "Heavy Tailed" case, with the x-axis showing us the rank of that label's magnitude compared to the other 40,000 test set labels, and the y-axis showing us the actual magnitude of the label. As we can see, this setup causes a small fraction of points (those at the right in the graph) to have very extreme values.

Part III

Fixing What Can Go Wrong

Chapter 2

Parameterized Weight Functions: Making Bad Data Count Less

In Chapter 1 we investigated many ways that machine learning algorithms can make suboptimal predictions. Now, we analyze in greater depth one such way, specifically, when the training data points vary substantially in quality. In such a scenario, we may have a direct measure of the quality of each data point, or we may only have access to a variable that correlates with data point quality. The question of how such information should be handled is pervasive, as it affects not just machine learning, but nearly any statistical estimation technique.

Intuitively, if some data points are better than others, we want to weight the higher quality data points more than the lower quality ones. Essentially, we have to devise a mapping from our measure of data point quality into the weight we assign each point. A "one size fits all" solution will not work, since quality can be measured in many different ways, and different such functions will be needed depending on the measure of quality. We have flexibility in choosing such a mapping, but we are not free to choose any mapping that we like. In this

chapter we explore the rules that govern these mappings, and how to construct mappings that satisfy all of the relevant rules. We produce a flexible framework for constructing functions that map point quality into point weight. These weights can then be plugged into the statistical estimation technique of interest, to adjust how much each data point counts. With this theory in hand, we will then explore a particular application of it in the context of regression in Chapter 3.

We now discuss different meanings of data point "quality", provide examples where data quality variability might arise, and introduce the notion of a "Weight Function", our approach to designing mappings from point quality to point weight.

2.1 Introduction

It is common when performing analysis on data that not all data points are equally good. For instance, data points may differ in how characteristic they are of the group one hopes to draw conclusions about, in how reliably measured they are, or in how "outlier-like" they are. In other words, some data points are worse and some points are better, and ideally we would assign a weight to each data point so that bad data counts less in our analysis than good data.

In practice, researchers usually perform analysis of data in one of two ways. Either they treat all data points equally (i.e. do not take into account how good or bad each data point is) or they draw an arbitrary cutoff point, saying that any data worse than the cutoff will be discarded completely (i.e. assigned 0 weight) and any data that is less than that bad will count fully in the analysis. But most of the time the "badness" of data lies on a continuous scale. It therefore is sometimes much preferable to weight points in a continuous fashion,

giving worse points less weight rather than counting each data point as either fully included or fully excluded. This is particularly true when performing data analysis on relatively small samples, where analysis can be compromised by a few very bad data points, yet throwing away data is costly. One bad outlier out of fifty points can ruin a standard deviation calculation, yet if we remove data too aggressively we may both bias our result and increase the variance of our estimate.

We provide a mathematical framework for thinking about how to convert a measure of poor data point quality, which we will call the "badness" z, of a data point, into a weight, $v(z)$. These weights can be used in data analysis to count bad data points less than good ones. Most statistical calculations and machine learning algorithms can be easily extended to work with weighted data. For instance an average can be replaced with a weighted average, and least squares regression can be replaced with weighted least squares regression. The main question then is how these weights $v(z)$ should be produced once we already have a badness z for each data point.

Consider four example scenarios where questions about how to map data quality into data point weights may arise:

1. A psychologist wants to know whether depressed patients get less sleep, on average, than non-depressed patients. But the survey the psychologist uses to measure depression does not have a well-defined cutoff to say who exactly is, or isn't depressed. The survey measures depression on a continuous scale. If the researcher picks a stark cutoff for what it means to be "depressed" (say, a score of at least 50 out of 100 on the depression scale) then that has strange implications, as it treats those with a score of 50 (who are barely representative of depression) identically to those with a score of 75 (who are

much more representative), and people with a score of 49 get ignored entirely despite being extremely similar to those who get scores of 50. So then we might ask: what are the psychologists options for computing the average level of sleep for depressed people, taking into account the fact that depression is not binary?

2. A social scientist has collected data from a hundred people about their views on gun control via interviews. The social scientist also has constructed a "unreliability" score for each person surveyed, based on whether their answers were self-contradictory or seemed like they were merely intended to please the interviewer. What are the social scientists options for counting the less reliable information less in their analysis of the data?

3. A physicist has made a series of repeated measurements. The physicist does not directly know the measurement error in each case, but has collected a variable for each measurement that is known to correlate with the measurement error. When analyzing the data, what are the physicists options for taking into account this correlate of the measurement error so as to count the more accurately measured data more than the less accurately measured data?

4. A machine learning researcher has developed a new method for scoring whether a data point is an outlier. What are this researcher's options for reducing the weight of points based on how outlier-like they seem, so that the more outlier-like points do not corrupt the result of running a regression algorithm?

In all of the above examples, assigning a weight to each data point could resolve the question at hand. For instance the psychologist in the first example, instead of taking the mean $\sum_{i=1}^{m} s_i$ of the sleep values s_i, could assign the weight v_i to the ith data point based on his measure of badness z_i, and then take the weighted mean $\sum_{i=1}^{m} v_i s_i$ where the v_i satisfy

$v_i \geq 0$ with $\sum_{i=1}^{m} v_i = 1$. However, it is unclear how to produce these weights from v_i from badness scores z_i associated with each point. Clearly we want points with higher badness to have lower weight, but beyond that we can wonder whether all options are equally good, such as:

$$v_i = \frac{\frac{1}{z_i}}{\sum_{i=1}^{m} \frac{1}{z_i}} \quad or \quad v_i = \frac{e^{-z_i}}{\sum_{i=1}^{m} e^{-z_i}}.$$

Certainly some choices must be better than others. We now consider what properties any function $v(z)$ that maps a badness z_i into a weight v_i should have.

2.2 Weight Functions

Suppose that we have a matrix X with one data point X^i per column, and for each point we have a score z_i that reflects some notion of badness for the point. Recall, a data point could have a larger badness due to being less characteristic of the group of interest, or because it is less reliably measured, or because it is more outlier-like than other data, or for some other reason. But regardless of the cause, data points with more badness should be counted less when performing analysis compared to data points with less badness, and our goal is to assign weights to the data points to reduce the total badness to make our analysis more accurate or more representative of what we are attempting to calculate. We assume $z_i \geq 0$, so that $z_i = 0$ means that a point is as good as possible. This assumption that badness has a minimum value makes sense in many contexts (e.g. when badness reflects variance, measurement error, outlierness, or non-representativeness of a group). Hence in this context a point can't be arbitrarily good, it can only be not at all bad (i.e. $z = 0$) reflecting the fact that it should not have its weight reduced at all during our calculations.

We would like to construct a function $v(z_i)$ that maps the badness z_i of each point X^i

to a weight that we will assign to that ith point during data analysis. We will call such a function a "Weight Function" if it has certain desirable properties that make it suitable for this purpose. Here, we assume that a weight of 1 is the maximum weight for a point (i.e. the point has the normal, full weight). Hence, we assume our function $v(z)$ takes values in the range 0 (no point weight) to 1 (maximum point weight). This choice of 1 for the maximum is not essential, the key is that the function must have a maximum, as we do not want to assign arbitrarily large amounts of weight to a single point, which would lead to highly biased analyses based on the data. Using 1 is simply a convenient choice.

It is important to note that the functions $v(z)$ will generally assign points unnormalized weights, meaning that the weights will not generally sum to 1. Hence the weights output by such a function will generally need to be normalized before they are used in data analysis. This is done by dividing each weight by the sum of all weights, causing the weights to then sum to 1.

Later on, we will introduce a real-valued free parameter α into these functions, producing what we call "Parameterized Weight Functions". This parameter α will determine how harshly the function reduces the weight of points of higher badness relative to ones of lower badness. For now, however, we assume our weight functions have no free parameters.

We would like our Weight Functions $v(z)$ to have a number of properties that make them suitable as a way to map badness into point weights. Intuitively, they should assign a value of 1 (i.e. maximum weight) to points that are not bad at all, they should not give more weight to points that are more bad than points that are less so, they should assign arbitrarily close to 0 weight to points that have arbitrarily large badness, they should limit the maximum amount of badness a point can have after weighting that point, and they should look like a uniform weighting of points for those points that are arbitrarily close to being not-at-all-bad.

The following formally defined function properties are designed to capture these intuitions. Any real-valued function that has all of the properties listed below, we will call a "Weight Function", and will be valid for consideration as a way of mapping badness values z into weights $v(z)$.

Properties that define a Weight Function $v(z)$

1. **Domain**: $v(z)$ is defined on $z \in [0, \infty)$, since we use 0 to mean that a point is not bad at all, but a point can have arbitrarily high badness. For instance, in regression a point can be perfectly on the regression line of the other points (0 badness) or it can be arbitrarily far away from the other points (arbitrarily high badness). As another example, a point can be associated with no measurement error (0 badness) or can have arbitrarily high measurement error (arbitrarily high badness).

2. **Range**: $v(z) \in [0, 1]$, since we use 0 to mean that a point is given no weight (and is hence totally ignored from all analysis), and a 1 to mean that a point is given the maximum possible weight (when that point is not bad at all).

3. **Monotonicity**: $v(z)$ must be a non-increasing function of z, since a point with greater badness should not have more weight than a point with less badness. This is clear since the purpose of introducing the notion of Weight Functions is to provide less weight to points with greater badness.

4. **Maximum**: $v(0) = 1$, since when the badness z is 0 the corresponding point is not bad at all, so should have maximum weight.

5. **Boundedness**: $\sup_{z \geq 0} v(z)z$ should be finite. Our weights are designed to reduce the badness of our points overall (once those weights are taken into account).

The badness of our ith point after it has been weighted by our weight function is $\frac{v(z_i)z_i}{\sum_{j=1}^{m} v(z_j)}$, so if we want our weight function to control the badness of our points no matter how large the badness gets, then we need $\sup_{z\geq 0} v(z)z$ to be finite. This will also prevent the average badness, after the points are weighted, from remaining arbitrarily large. Note that this bound also implies that $\lim_{z\to\infty} v(z) = 0$, which is crucial since arbitrarily bad points should have arbitrarily small weight. Otherwise a single extremely bad point may throw off the results of our whole analysis (e.g. one very extreme outlier may distort the mean of the whole sample, or one point with arbitrarily large measurement error throw off our entire analysis).

6. **Uniformity**: $\lim_{h\to 0^+} \frac{v(h)-v(0)}{h} = 0$, since points that are nearly not at all bad should all be treated nearly equally, since that will produce an unbiased estimate for our calculation of interest. Making the derivative of our weight function at $z = 0$ vanish (we use the right derivative since v is only defined for $z \geq 0$), implies that locally the function weights points that are nearly completely non-bad nearly uniformly. This makes sense because the unbiased way to perform data analysis with non-bad points is to weight them equally to each other, not to weight some more than others. It is only by virtue of a point being bad that we want to give it less weight, so $v(z)$ should locally approximate the uniform distribution near $z = 0$ by having a 0 right derivative there.

We now introduce a theorem which allows us to easily construct Weight Functions, $v(z)$, by starting with a non-decreasing function, $f(z)$, and passing it through a special non-linear transformation designed for this purpose.

Theorem 1 (Characterization of Weight Functions)**.** *If $f(z)$ is a non-decreasing, non-constant*

real-valued function defined on $z \in [0, \infty)$ which is continuous in some interval containing $z = 0$ (and which optionally may take on the value $+\infty$), then the function $v(z)$ defined as

$$v(z) = \frac{1}{1 + z(f(z) - f(0))}$$

satisfies all of the properties above and therefore is a Weight Function. Additionally, any Weight Function $v(z)$ can be written in the form above for some choice of $f(z)$, though $f(z)$ will only be non-decreasing when $\frac{1}{v(z)z} - \frac{1}{z}$ is non-decreasing.

We note that $f(z)$ does not need to be differentiable, and does not even need to be continuous (except near $z = 0$). It also does not need to be finite valued, but since it is non-decreasing, if it is ever equal to $+\infty$ it must remain at this value for all larger z as well.

Proof. We show, one by one, that each of the properties of a Weight Function is satisfied for any function $v(z)$ of the form specified in Theorem 1.

1. **Domain**: By the definition of $f(z)$ the domain of $v(z)$ is $z \in [0, \infty)$.

2. **Range**: $v(z)$ is of the form $\frac{1}{1+g(z)}$ for $g(z)$ a non-negative function, and therefore is in the range $[0, 1]$.

3. **Monotonicity**: $v(z)$ will be non-increasing whenever $z(f(z) - f(0))$ is non-decreasing. But z and $(f(z) - f(0))$ are both non-negative and non-decreasing since $f(z)$ is assumed to be non-decreasing and $z \geq 0$. Therefore the product of these factors, $z(f(z) - f(0))$, is non-decreasing.

4. **Maximum**: $v(0) = \frac{1}{1+0(f(0)-f(0))} = 1$.

5. **Boundedness**: $\sup_{z \geq 0} v(z)z = \sup_{z \geq 0} \frac{z}{1+z(f(z)-f(0))} = \frac{1}{\inf_{z \geq 0} \left(\frac{1}{z} + (f(z)-f(0))\right)}$. The only way this supremum can be infinite is if both of the positive terms $\frac{1}{z}$ and $(f(z) - f(0))$

approach zero simultaneously. But for small z when the latter term approaches zero the former term $\frac{1}{z}$ gets large. And as z grows arbitrarily large causing the former term to approach zero, the latter term $(f(z) - f(0))$ will always remain above a positive constant due to $f(z)$ being non-decreasing and non-constant. Hence, the supremum is bounded.

6. **Uniformity**: $\lim_{z\to 0^+} \frac{v(z)-v(0)}{z} = \lim_{z\to 0^+} \frac{\frac{1}{1+z(f(z)-f(0))}-1}{z} = \lim_{z\to 0^+} \frac{1}{z}\frac{-z(f(z)-f(0))}{1+z(f(z)-f(0))} = 0$ since $f(z)$ is assumed to be continuous near 0.

So far, we have shown that any function of the above form will have the properties of a Weight Function. But when can a given Weight Function $v(z)$ be written in that form? To find out, we consider the form of interest:

$$v(z) = \frac{1}{1 + z(f(z) - f(0))}.$$

We assume that $z > 0$ and that we are excluding any z where $v(z) = 0$. Then we rearrange to solve for $f(z)$. This gives us:

$$f(z) = \frac{1}{v(z)z} - \frac{1}{z} + f(0).$$

Since by definition $v(z)$ is defined on the domain $(0, \infty)$, and $\frac{1}{z}$ is defined and finite in this domain, this function $f(z)$ is also defined and finite in this domain. For all z where $v(z) = 0$, we simply define $f(z) = +\infty$, which is permitted in the definition of $f(z)$. For $z = 0$, which is the only remaining problematic point, we define $f(0)$ using the right hand limit $f(0) \equiv \lim_{z\to 0^+} f(z)$. This gives us:

$$\lim_{z\to 0^+} \frac{1}{v(z)z} - \frac{1}{z} + f(0) = \lim_{z\to 0^+} \frac{1}{z}\Big(\frac{1 - v(z)}{v(z)}\Big) + f(0) = -\lim_{z\to 0^+} \frac{1}{v(z)}\Big(\frac{v(z) - v(0)}{z}\Big) + f(0)$$

$$= - \lim_{z \to 0^+} \frac{1}{1} 0 + f(0) = f(0)$$

as expected, where we applied the properties that define $v(z)$. Hence, to express $v(z)$ in the desired form as function of $f(z)$, we can choose $f(z) = \frac{1}{v(z)z} - \frac{1}{z}$ for $z > 0$, and $f(z)$ will be non-decreasing precisely when $\frac{1}{v(z)z} - \frac{1}{z}$ is. □

We now extend our definition of a Weight Function by introducing a real-valued free parameter α. We will call this new type of function a "Parameterized Weight Function" and will write it $v_\alpha(z)$ instead of $v(z)$. This α parameter controls how strongly we reduce the weight of points based on on their badness. At one extreme ($\alpha = 0$) badness is not down-weighted at all (every point is assigned an equal weight of 1), and at the other extreme ($\alpha = \infty$) bad points are down-weighted as harshly as possible. Hence, we can think of $v_\alpha(z)$ as a family of Weight Functions with a harshness (i.e. how much increased badness reduces a point's weight) determined by α. Since the appropriate level of harshness is determined by whatever problem our Weight Functions are being applied to, it is convenient to introduce this as an explicit parameter.

Properties that define a Parameterized Weight Function $v_\alpha(z)$

1. **Weight Function Family**: $v_\alpha(z)$ must satisfy all of the properties of a Weight Function for each fixed $\alpha > 0$.

2. **Monotonicity**: $v_\alpha(z)$ must be a non-increasing function of α, since α quantifies the harshness of the weight function.

3. **Equal Treatment**: $v_0(z) = 1$, since when we choose $\alpha = 0$ we want to reduce the weight of bad points as little as possible, so in other words all points should receive the same weight no matter how bad each is.

4. **Extreme Treatment**: $\lim_{\alpha \to \infty} v_\alpha(z) = 0$ when $z > 0$, since when we choose α approaching infinity we want to reduce the weight of bad points the maximum amount possible, so in other words, all points that are even the slightest bit bad (i.e. that have $z > 0$) receive no weight.

Theorem 2 (Characterization of Parameterized Weight Functions)**.** *If $v(z)$ is a Weight Function, then $v(\alpha z)$ is a Parameterized Weight Function. Hence, if $f(z)$ is a non-decreasing, non-constant real-valued function defined on $z \in [0, \infty)$ which is continuous in some interval containing $z = 0$ (and which optionally may take on the value $+\infty$), then the function $v_\alpha(z)$ defined as*

$$v_\alpha(z) = \frac{1}{1 + \alpha z(f(\alpha z) - f(0))}$$

satisfies all of the properties above and therefore is a Parameterized Weight Function.

Proof. The symmetry between α and z cause the properties for a Weight Function (with respect to z) to immediately imply the properties of a Parameterized Weight Function (with respect to α). □

2.3 Generating Parameterized Weight Functions

Applying Theorem 2, it is straightforward to generate Parameterized Weight Functions, $v_\alpha(z)$, which for each $\alpha > 0$ represent valid functions for mapping each point's badness z into a corresponding weight that will be applied to that point. All we have to do is take a non-decreasing function $f(z)$ that is continuous at 0 and the theorem will produce a Parameterized Weight Function for us. We show a number of examples in Table 2.1, each generated using a different choice of $f(z)$. For all examples where a constant p appears, it is assumed to be a strictly positive real number.

List of Parameterized Weight Functions

$\mathbf{f(z)}$	$\mathbf{v_\alpha(z)}$	$\sup_{\mathbf{z\geq 0}} \mathbf{z}\, \mathbf{v_\alpha(z)}$
z^p	$\frac{1}{1+(\alpha z)^{p+1}}$	$\frac{1}{\alpha}\frac{p^{\frac{p}{p+1}}}{p+1}$
$\frac{e^z-1}{z}$	$\frac{1}{e^{\alpha z}-\alpha z}$	$\frac{1}{\alpha}\frac{1}{e-1}$
$\frac{e^{z^{p+1}}-1}{z}$	$e^{-(\alpha z)^{p+1}}$	$\frac{1}{\alpha}e^{-\frac{1+\log(p+1)}{p+1}}$
$\frac{1}{1-z^p}$ if $z<1$ otherwise ∞	$\frac{1-(\alpha z)^p}{1-(\alpha z)^p+(\alpha z)^{p+1}}$ if $z<\frac{1}{\alpha}$ otherwise 0	$<\frac{1}{\alpha}$
0 if $z<1$ otherwise ∞	1 if $z<\frac{1}{\alpha}$ otherwise 0	$\frac{1}{\alpha}$

Table 2.1: Some Parameterized Weight Functions, along with the functions $f(z)$ used to construct them by applying Theorem 2, and their bounding constants.

The last row in Table 2.1 corresponds to what is, in a sense, the simplest Parameterized Weight Function. It uses a stark cutoff of $\frac{1}{\alpha}$, and any point with a badness z less than this cutoff gets a full weight of 1, while all other points get 0 weight. The second to last option in Table 2.1 is a smoother variation of this. Any point with $z \geq \frac{1}{\alpha}$ still gets a weight of 0, but for $0 < z < \frac{1}{\alpha}$ the weights are strictly between 0 and 1. The first three functions in the table are smoother still, as they assign weights strictly between 0 and 1 to all points with $z > 0$.

Note that adding a constant to any such function f(z) does not affect the corresponding $v_\alpha(z)$. Also note that there are some interesting functions that nearly but not quite satisfy our necessary conditions for a Parameterized Weight Function: choosing $f(z) = 1$ yields $v_\alpha(z) = 1$, which is not a Parameterized Weight Function because it does not have the required limiting behavior with respect to α and also because $sup_{z\geq 0} v_\alpha(z)z$ is not bounded. On the other hand, the function $v_\alpha(z) = \frac{1}{1+\alpha z}$ nearly satisfies all the requisite conditions except that it does not have a 0 derivative at $z = 0$.

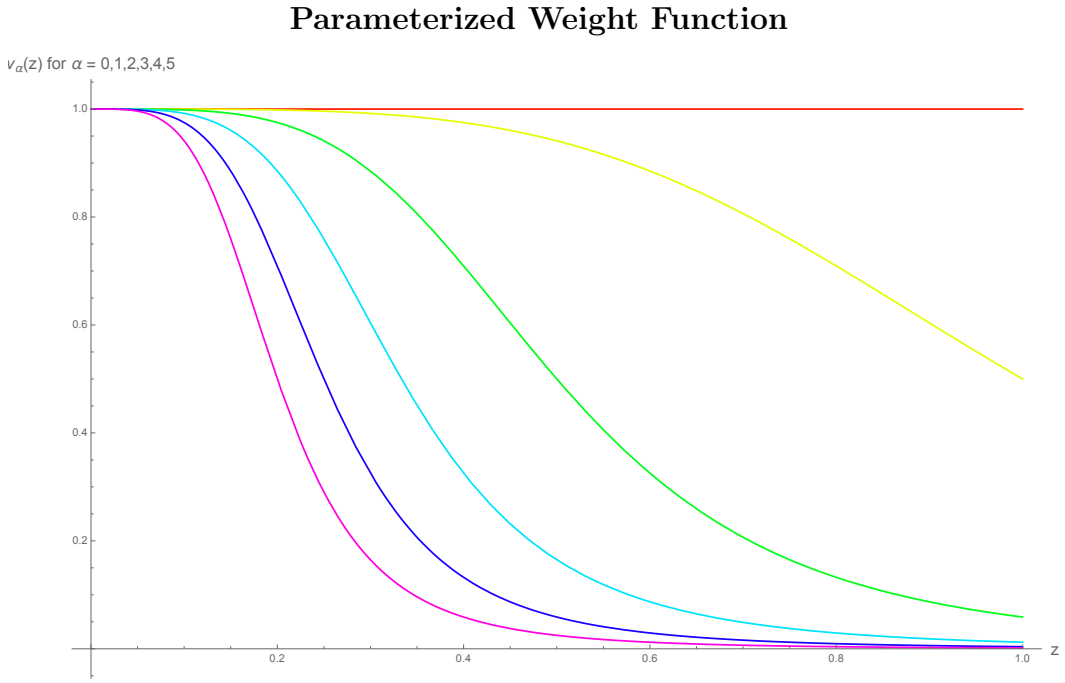

Figure 2.1: A sample Parameterized Weight Function $v_\alpha(z) = \frac{1}{1+z^4\alpha^4}$ plotted as a function of z for $\alpha = 0, 1, 2, 3, 4, 5$.

2.3.1 Polynomials

Can a function that is a polynomial in z be a parameterized weight function? No, because polynomials cannot both be equal to 1 when $z = 0$ and approach 0 as $z \to \infty$. But we can construct polynomial-like parameterized weight functions. For instance, we can construct a useful parameterized weight function that is (a) piecewise polynomial and (b) twice differentiable using the following procedure: starting with the odd looking non-decreasing function

$$f(z) \equiv \begin{cases} 0 & z < p \\ \frac{1}{z}\left(\frac{16}{(p-z+2)^3(3p^2-3p(2z+1)+3z(z+1)+2)} - 1\right) & p \le z \le p+2 \\ \infty & z > p+2, \end{cases}$$

which is defined for any $p \geq 0$, we generate a parameterized weight function $v_\alpha(z)$ by applying Theorem 2:

$$v_\alpha(z) \equiv \begin{cases} 1 & z < \frac{p}{\alpha} \\ \frac{1}{16}(p - \alpha z + 2)^3 \left(3p^2 - 3p(2\alpha z + 1) + 3\alpha z(\alpha z + 1) + 2\right) & \frac{p}{\alpha} \leq z \leq \frac{p+2}{\alpha} \\ 0 & z > \frac{p+2}{\alpha} \end{cases} .$$

We can gain a deeper understanding of this function by performing the change of variables $a \equiv \frac{p}{\alpha}$ and $b \equiv \frac{p+2}{\alpha}$, yielding:

$$v_{a,b}(z) \equiv \begin{cases} 1 & z < a \\ \frac{(z-b)^3\left(10a^2 + z(3b-15a) - 5ab + b^2 + 6z^2\right)}{(a-b)^5} & a \leq z \leq b \\ 0 & z > b \end{cases} .$$

This "polynomial" Weight Function is twice differentiable in z for all $z > 0$ (its first and second derivatives at both $a \equiv \frac{p}{\alpha}$ and $b \equiv \frac{p+2}{\alpha}$ are well defined and equal to zero, and it is of course continuous at these points as well). As z increases, this function starts strictly at 1 until it hits $z = a$ at which point it slopes down monotonically as a fifth degree polynomial in z until it hits $z = b$ where it then remains at zero forever after (see Figure 2.2). It is a piecewise polynomial not just in z, but (when using the appropriate parameterization) also in p and α, though it is not a polynomial in a and b. Piecewise polynomials that are k-times differentiable are as close as we can get to polynomials in this world of Weight Functions. The particular function $v_{a,b}$ introduced above is quite useful because it allows us to specify the badness level at which full weight ends (denoted "a") and the badness level at which zero weight begins (denoted "b"), corresponding to the cutoffs for a point being considered as good as possible and as bad as possible, respectively.

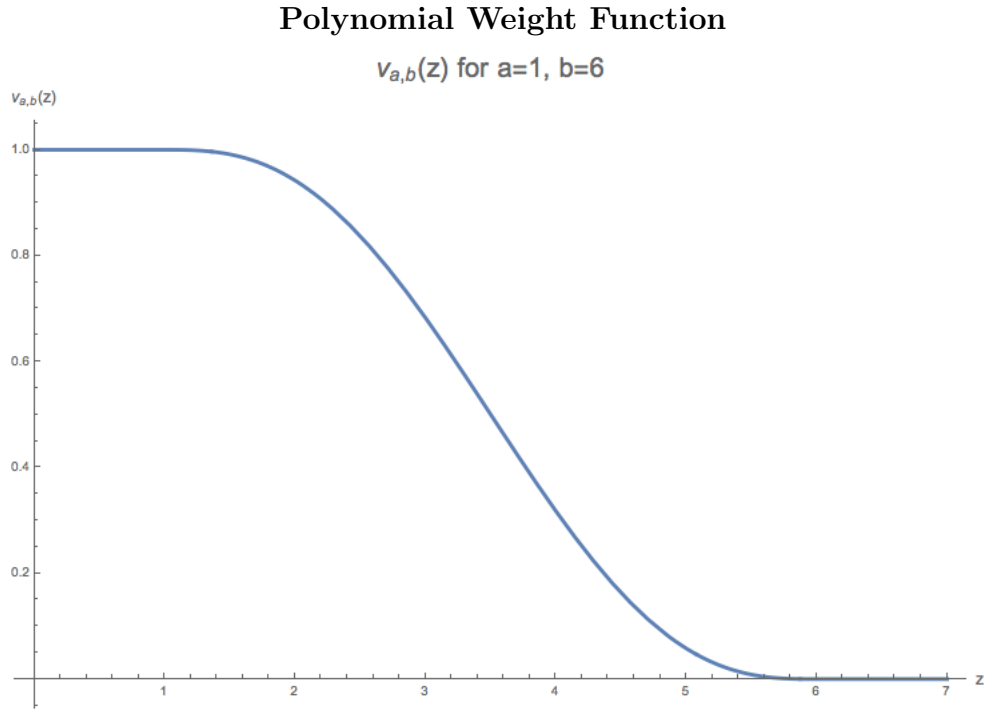

Figure 2.2: A twice differentiable piecewise polynomial Weight Function.

2.4 Speed of Decrease

Let us consider how quickly Parameterized Weight Functions of the form $v(\alpha z)$ decay as functions of z. We have:

$$v_\alpha(z)z = v(\alpha z)z \leq \sup_{z\geq 0} v(\alpha z)z = \frac{1}{\alpha}\sup_{u\geq 0} v(u)u = \frac{c}{\alpha}$$

for some finite $c > 0$, since by definition $\sup_{z\geq 0} v(z)z$ is finite. Therefore we have:

$$v_\alpha(z) \leq \frac{1}{\alpha}\frac{c}{z}$$

where c is a constant that depends on the shape of the function $v_\alpha(z)$, but not on α or z. Hence, one interpretation for α (when considering Parameterized Weight Functions of the form generated by Theorem 2) is as a parameter for controlling this upper bound on how fast the function decreases. This bound also implies that while Weight Functions need not be probability density functions nor even integrable functions, they do have to decay at least as fast as $\frac{1}{z}$. This rate of decay, however, is not typical: all the examples considered above decay faster than $\frac{1}{z}$.

2.5 Badness Range

The preceding discussion assumes that we already have a notion of badness defined for our data points of interest, whether it be some measure of outlierness, measurement error, representativeness of the group of interest, or something else. The switch from Weight Functions to Parameterized Weight Functions removes our need to worry about the unit of our badness scores, because α will play the role of setting this unit. But is is worth noting that if our badness scores are not naturally able to cover the range $[0, \infty)$ or $[0, \infty]$ we may want to modify these scores before plugging them into a Weight Function. For instance, if our scores z are in the range $[q, \infty)$ we can simply subtract q from each score. Or if they are in the range $[q, r]$ then we can apply a monotonic map such as

$$z \to \frac{1}{1 - \frac{z-q}{r-q}} - 1$$

so that the new range is $[0, \infty]$ before a Weight Function is used.

2.6 Weighted Least Squares

A simple example of a Weight Function arises naturally in the context of Ordinary Least Squares regression. From a statistics (as opposed to machine learning) perspective, if all the underlying variables being used in our linear model are normally distributed, plus our prediction errors are uncorrelated both with each other and with the other independent variables and the standard deviation of the prediction errors are all the same with zero expected values, then it is desirable to leave our data points unweighted. The Gauss-Markov theorem tells us that the standard (unweighted) Ordinary Least Squares solution is the "best linear unbiased estimator". However, if we drop the assumption that the prediction errors have identical standard deviations, then the points have to be weighted to produce the best linear unbiased estimator. In particular, theory says that the ith point should be weighted by $\frac{1}{\sigma_i^2}$, where σ_i is the standard deviation of the ith error (Draper and Smith, 1981). In practice though, this formula should not be used since it blows up to infinity. The standard deviations of each error are not actually known to infinite precision, and in almost all cases have to be estimated from the data. An erroneously low standard deviation estimate will cause the corresponding weight to explode, causing poor prediction performance. Hence, this formula is often adjusted to prevent exploding weights by writing $\frac{1}{\epsilon+\sigma_i^2}$, for some small $\epsilon > 0$ which smooths out the standard deviation estimates. If we use $z = \sigma_i$ as our measure of the badness of a data point, this corresponds precisely to one of our standard Weight Functions, $\frac{1}{1+(\alpha z)^2}$. Since this function will be normalized to have its weight sum to 1, α plays a role analogous to $\frac{1}{\epsilon}$. As $\alpha \to 0$ we reconstruct the uniform weighting which is optimal when all standard deviations are the same. On the other hand, as $\alpha \to \infty$ we reconstruct the theoretically optimal (but impractical) weight $\frac{1}{\sigma_i^2}$.

2.7 Selecting a Parameterized Weight Function

There is an infinite number of Parameterized Weight Functions to choose from. It would require extra information that the practitioner happens to know about his or her problem domain to narrow this infinite space down to just one function. Fortunately, there are many commonalities between all such functions, giving them a similar shape. They all start at 1 (for $z = 0$), then remain near 1 for a time as z increases, then monotonically decay down to zero (for large z), falling at a rate no slower than $\frac{c}{z}$, for some constant c (which might depend on α). When the Parameterized Weight Function is of the form $v(\alpha z)$, the bound becomes $\frac{c}{\alpha z}$, where c now does not depend on α. Given the striking similarities between all such functions, for a variety of practical purposes it will be sufficient to consider just one family of Parameterized Weight Functions, indexed by a parameter $p > 0$. We can think of α and p together determining where the function begins to descend substantially and how quickly it does so, or in other words, how wide of a flat top the function has near $z = 0$ and by what point it is assigning nearly zero weight. One useful family of parameterized weight functions is given by:

$$v_\alpha(z) = \frac{1}{1 + (\alpha z)^{p+1}}.$$

For this family of functions we can conveniently set α and p if we have an intuition about two points that the weight function should pass through.

2.7.0.1 Using Two Fixed Points

Consider again our example where a psychologist wishes to estimate the mean amount of sleep for depressed individuals in a sample. Scores on a depression inventory serve as a proxy for how closely each person fits the term "depressed". So if the depression test is out of 100

points, the psychologist may define the badness scores z using $z = 100 - d$, where d is the depression score of each individual. This choice simply causes the badness scores to be in the desired range $z \geq 0$, with a 0 badness for people who have the maximum possible depression score $d = 100$ (they are truly depressed by any definition), and a larger badness for anyone with a depression score lower than this. The exact function used to convert test scores into badnesses is not critical (e.g. another choice could be $z = \frac{100}{d} - 1$, which could be desirable because it would cause the badness scores to cover the domain $[0, \infty]$, but we consider the former option here just for simplicity).

To choose $v_\alpha(z)$ the psychologist can make use of his or her intuition about two points in our sample. If the psychologist can select an appropriate weight $v_0 > 0$ to assign any individuals who have a score z_0, and an appropriate weight v_1 (where $0 < v_1 < v_0$) to assign to individuals with a score of z_1 (where $0 > z_1 > z_0$). Then we can solve for α and p in the pair of equations: $v_\alpha(z_0) = v_0$ and $v_\alpha(z_1) = v_1$, which yields:

$$v(z) = \left(1 + \frac{1 - v_0}{v_0}\left(\frac{z}{z_0}\right)^{\frac{\log\left[\frac{v_0}{v_1}\frac{1-v_1}{1-v_0}\right]}{\log\left[\frac{z_1}{z_0}\right]}}\right)^{-1}$$

So for instance, if the psychologist decides that a person with badness score $z = 20$ should be considered 90% depressed (i.e. weighted 0.90 of what someone maximally depressed would be weighted), and someone with a depression score of $z = 60$ should be considered 10% depressed, then we can use ($z_0 = 20$, $v_0 = 0.90$), and ($z_1 = 60$, $v_1 = 0.10$) which yields: $\mathbf{v}(z) = (1 + \frac{z^4}{1440000})^{-1}$, as seen in Figure 2.3. If the ith depressed patient has sleep score s_i and badness z_i then to compute the average sleep for depressed individuals such that the continuum between "depressed" and "not depressed" is taken account, we would then just compute the weighted average sleep s for depressed individuals as: $s = \sum_{i=1}^{m} \frac{v(z_i)}{\sum_{j=1}^{m} v(z_j)} s_i$.

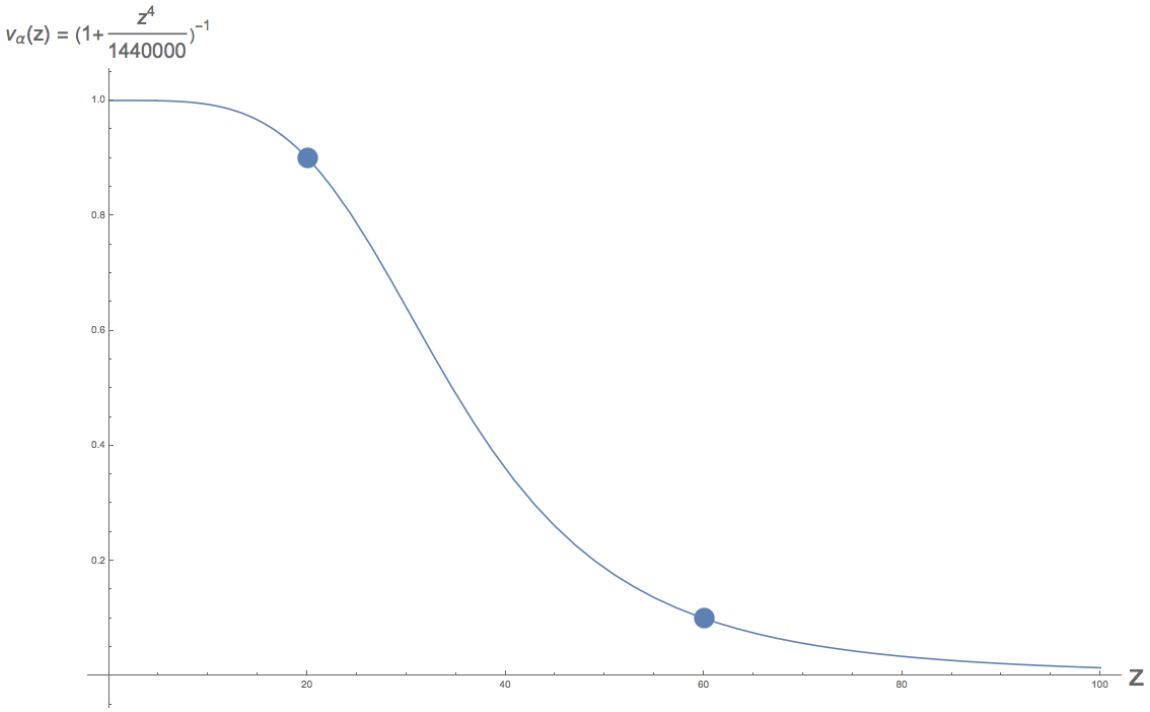

Figure 2.3: Sample Simple Weight Function of the form $v_\alpha(z) = \frac{1}{1+(\alpha z)^{p+1}}$, with parameters α and p set from two known points ($z_0 = 20$, $v_0 = 0.90$), and ($z_1 = 60$, $v_1 = 0.10$).

This procedure of fitting p and α using two points will always give us a decreasing function as long as $v_1 < v_0$ and $z_1 > z_0$. However it will only technically be a Weight Function when:

$$\frac{z1}{z0} < \frac{v_0}{v_1}\frac{1-v_1}{1-v_0}$$

because otherwise one of our properties that define a Weight Function that will not be satisfied (the right hand derivative at 0 will not be 0, meaning it will not locally weight points uniformly near 0).

2.7.0.2 Using Maximal Weight and Minimal Weight Cutoffs

Consider again our example where a social scientist has constructed an "unreliability" score for each person he or she has surveyed. It is desirable to reduce the weight of data

points that have greater unreliability to improve the quality of data analysis. If the sociologist has an intuitive sense of what cutoff score is low enough that we should consider a person getting that score "minimally" unreliable, and what cutoff score is high enough that we can consider a person getting that score "maximally unreliable", then we can use these two cutoffs to construct our Weight Function. Anyone data point below the former cutoff will receive a weight of 1, and any data point above the latter cutoff will receive a weight of 0. The "polynomial" Weight Function $v_{a,b}(z)$ that we constructed earlier is an appropriate choice in this case, with the parameters a and b representing the two cutoffs.

To be specific, let us assume that the sociologist's unreliability scores are real numbers in the range of 0 (not at all unreliable) to 5 (totally unreliable), and the sociologist decides that anyone with a score less than or equal to 1 should be considered totally reliable (and so gets a weight of 1) and anyone with a score of 4 or higher should be considered totally unreliable (and so gets a weight of 0). Then the appropriate weight function is simply $v_{1,4}(z)$. Of course, this function must be normalized before use so that the sum of weights assigned to all points is always 1.

2.7.0.3 Using External Validity

Consider again the example of the physicist who has collected a series of repeated measurements along with a quantity that correlates with measurement error (which serves as a measure of badness, z, for each data point). We assume that the correlate of measurement error is statistically independent of the repeated measurements that were made, otherwise reducing the weighting of data points based on it may bias the analysis. Now suppose that the physicist has also measured some third quantity that is known to correlate strongly with the repeated measurements, which we will call our "external validity measure". This third quantity can help us choose a Weight Function, by setting the free parameters of a Weight

Function family such that we maximize this correlation. The worse a data point is for use in our data analysis (due in this case to measurement error), the more its inclusion will tend to negatively impact the correlation between the repeated measurements and the external validity measure. Hence maximizing the correlation between the repeated measurements and the external validity measure by weighting data points provides a mechanism for deciding how low a weight to give to data points of different levels of badness, which means we can use it to determine α and p in our Parameterized Weight Functions.

We begin with a Parameterized Weight Function family, such as:

$$v(z) \equiv e^{-(\alpha z)^{p+1}}$$

Then to set the free parameters $\alpha \geq 0$ and $p > 0$ we simply try different combinations of these two parameters (e.g. $p \in \{0.5, 1, 1.5, \ldots, 5.5, 6\}$ and $\alpha \in \{2^{-10}, 2^{-9}, 2^{-8}, \ldots, 2^{9}, 2^{10}\}$). For each such combination we compute the weighted correlation between the repeated measurements and our external validity measure. The weighted correlation is just like a regular correlation, except each data point is weighted by $v(z)$ after this function has been normalized (so that the weight assigned to all the data points sum to one). Whichever combination of the parameters α and p maximizes this correlation between our repeated measurements and the external validity measure is our preferred choice of Weight Function parameters. In more general terms, we learn the parameters of our Weight Function family using an external quantity for assessing the quality of our (weighted) data points.

2.7.0.4 Using k-Fold Cross-Validation

Consider again the case of the machine learning researcher who has found a way of scoring how outlier-like (i.e. how "bad") each data point is for any given training set. For instance, in

an Ordinary Least Squares context, scoring methods could be those found in Stevens (1984). Furthermore, assume that the machine learning algorithm that he or she will be using for making predictions can make use of points weights, which determine how much each point should count during training. This is true for most popular algorithms. Then, by using a Weight Function, the machine learning researcher can turn the scores for how outlier-like each point is into point weights, hence causing the machine learning algorithm to rely less on outlier-like data and to ignore very extreme outliers altogether. We explore this approach to automatically handling outliers in the context of Ridge Regression in Greenberg et al. (2016b).

But which Weight Function should the machine learning researcher use to convert outlier scores into weights? Empirical test cases suggest that the Parameterized Weight Function:

$$v(z) = \frac{1}{1 + (\alpha z)^6}$$

is a reasonable choice. This function is very flat around $z = 0$ and only gradually descends to 0. We can then select $\alpha \geq 0$ using a robust form of k-fold cross-validation. Rather than selecting the α that gives the lowest error averaged across all k folds (as in regular k-fold cross-validation, which is not robust to outliers), we instead use the winsorized mean of the out of sample errors made on all out of sample points during cross-validation. The winsorized mean is like the regular mean, but it is much more robust to outliers. It is computed by first replacing any value above the ρ percentile value with the ρ percentile value, and any value below the $1 - \rho$ percentile value with the $1 - \rho$ percentile value, and only then computing the mean of the resulting values. Using this approach, the parameter α is automatically learned from the data based on how much outlier reduction is ideal in the particular training set being used. This lack of free parameters and automatic reduction of outliers is desirable as it prevents debates over precisely which data points should be included in regression analysis,

as was seen for example in a psychology context in Brandt (2012).

2.8 Conclusion

We have introduced the concept of a Weight Function, which represents the set of reasonable functions one can use to map a measure of how "bad" a data point is into a weight to be assigned to that point during (nearly any kind of) data analysis. We developed a framework for easily generating new Weight Functions simply by choosing an increasing function, and extended this definition tot Parameterized Weight Functions, which are Weight Functions with one free parameter that determines how harshly bad points should have their weights reduced. Finally, we discussed how researchers can use their intuition and measures of external performance to select which Weight Function to use in practice.

Chapter 3

Stabilized Ridge Regression

In this chapter, we investigate another source of problems for machine learning algorithms discussed in Chapter 1, namely, outliers in the training data. We investigate types and causes of outliers, and generalize Ridge Regression to a new algorithm which includes a parameter for penalizing outliers, while investigating why outliers have a damaging effect on Ridge Regression in the first place. We also explore some of the drawbacks of common approaches to removing outliers, such as modifying the loss function. The outlier robust algorithm that we introduce here leverages the work in Chapter 2 on Weight Functions. While Weight Functions are not specifically for outlier reduction in particular, they are applicable because "outlierness" is a measure of data point quality. Here, we limit our discussion of outliers to a regression context, but in Chapter 4 we discuss approaches for outlier detection in an arbitrary vector of real numbers.

3.1 Introduction

Despite their extreme popularity as prediction methods, Ordinary Least Squares regression and Ridge Regression are remarkably non-robust to outliers: the presence of a single "bad"

data point in the training set of either algorithm can severely damage their accuracy. In this work, we introduce a new variant of Ridge Regression, Stabilized Ridge Regression, which automatically accounts for possible outliers in the training data. Rather than handling outlier removal as a pre-processing step, or changing the loss function that we seek to minimize, e.g. switching from the L_2 loss function to the L_1 loss function, the algorithm automatically adjusts the weights of all data points based on how outlier-like they are. Our approach, which involves stabilizing the solution coefficients with respect to a point being dropped, operates by iteratively reweighting the data points and solving the Ridge Regression optimization problem under each reweighting. In doing so, it prevents extreme points from significantly harming prediction performance.

Our Python code for implementing Stabilized Ridge Regression is in the public domain, and can be downloaded at the URL found in the following footnote[1].

This chapter is organized as follows: we first introduce two different types of outliers and discuss the common methods used to handle outliers in practice. We then discuss the desirable properties for an algorithm robust to outliers. Next, we show that Ridge Regression (and by extension, Ordinary Least Squares) behaves undesirably in the presence of outliers, and that, even n-fold cross-validation, which is commonly used to select the tuning parameters of an algorithm, is problematic in the presence of outliers. Finally, we introduce our Stabilized Ridge Regression algorithm and demonstrate its performance in several cases of interest.

3.2 Outliers

Outliers, while hard to define formally, intuitively come in two flavors:

[1] http://spencergreenberg.com/code/doctoral_thesis_machine_learning_code.zip

1. A "label outlier" is a point with a label that is very large in magnitude relative to what would be expected from the rest of the data. As straightforward example, if all but one of the data point labels were generated by a normal distribution with mean 0 and standard deviation 1, but one data point had a label of 10, we can conclude that this label almost certainly was not generated from the same distribution as the other points. We call this point a "label outlier", and can predict that including it in the training set will likely have a detrimental effect on the performance of a prediction algorithm.

2. A "feature outlier" is a data point whose feature vector lies very far from other points in the data set. For instance, if all but one of the data points feature vectors were generated by a multi-variate normal distribution with mean $(0, 0, \ldots, 0)$ and with covariance matrix equal to the identity matrix, but one of the data points has feature vector $(10, 10, \ldots, 10)$ then it is almost certainly not generated from the same distribution as the other points, and will likely be harmful to the algorithm's prediction performance. We would call such a point a "feature outlier".

Of course, many real outliers are more subtle than these simple to identify cases, for instance a point whose label is not excessively large compared to the other labels, but is excessively large given what its feature vector is. And a real outlier can of course be both a feature outlier and a label outlier simultaneously, as this classification is not mutually exclusive.

The presence of outliers may have different causes, including input error, machine error, contamination of one data source with another, extreme chance events, or multimodal or heavy tailed probability distributions. What all outliers have in common is that they are risky to include when training a machine learning algorithm. Treating them like other points tends to make prediction less accurate in comparison with downweighting them.

3.2.1 Modifying Loss Functions to Mitigate Outliers

We now consider two common approaches to handling outliers, and discuss their limitations. Since most machine learning algorithms can be formulated in terms of minimizing a loss function, one approach to making such algorithms more robust to outliers is to modify this loss function. A properly chosen loss function can greatly reduce the impact of extreme points. The standard L_2 loss function of ordinary least squares and Ridge Regression is especially worrisome from the perspective of outliers because the most poorly predicted points (including outliers) have the most impact on the solution due to the squaring of errors. By switching to another loss function, such as L_p for $1 \leq p < 2$, this effect can be reduced. In particular, the L_1 loss function is highly robust against label outliers. A generalization of this idea, which allows for a wide range of loss functions, is used in the theory of M-estimation (Huber et al., 1964).

However, changing the loss function has significant disadvantages. First, while this approach can make the solution more robust to label outliers, it will typically not make the solution significantly more robust to feature outliers. In fact, when modifying Ordinary Least Squares regression to use the L_1 loss function, one finds in some cases that changing the feature of one point slightly can cause the solution to jump discontinuously (Ellis, 1998), which seems like a step in the wrong direction from a robustness perspective. Second, by changing the loss function to minimize, one is solving a fundamentally different problem from the original one, so one can get a very different solution from what ordinarily least squares or Ridge Regression would yield even in the absence of outliers.

For an instance of this, notice that $\text{argmin}_c \sum_i |y_i - c|^2$ and $\text{argmin}_c \sum_i |y_i - c|$ can have very different solutions: the mean for the first and the median for the second. For non-symmetric

distributions (including those without potential for large outliers) the mean and median can be far apart. Ideally, the choice of error function should be chosen based on one's goals and the prediction problem one's been handed, not on a desire for robustness. If the nature of the problem is such that miss-predicting a point's label by 2 units is 4 times worse than miss-predicting by 1 unit, then that's a property the algorithm should respect, making the L_2 error function more appropriate than the L_1 error function. Similarly, if one's goal is to accurately predict the mean of the label associated with a point, rather than say the median, the L_2 error function will again be more appropriate than other choices. Changing the loss function is often an inappropriate way to achieve outlier robustness because it does not preserve the original prediction goal.

Another approach to replacing the least squares problem with a more outlier robust problem is the Theil-Sen estimator (Blunck and Vahrenhold, 2006), which uses as the slope of the prediction line (for one dimensional prediction tasks) the median of the slopes of all lines passing through all pairs of data points. This method is certainly useful for some applications. But as with the approach of changing the loss function, this approach alters the goal that our algorithm is attempting to achieve, and therefore could lead to substantially different predictions even when no extreme outliers are presents.

Yet another strategy is to minimize the median of the squared errors (or another robust measure of scale applied to the prediction errors), rather than the sum of squared errors, known as S-estimation (Rousseeuw and Yohai, 1984). While this approach is robust to both label and feature outliers, it can have low statistical efficiency in outlier free data (Onur and Cetin, 2011).

An ideal approach to handling outliers would work for feature outliers in addition to label outliers, and would not require one to change one's ultimate goal (such as predicting the

mean of the label distribution at a point). It would also ideally produce predictions that are identical to or very close to Ridge Regression in the case when no outliers are present.

3.2.2 Pre-processing to Remove Outliers.

There are many approaches to pre-processing a training set to handle outliers, for instance by removing any point whose distance is more than D (using some notion of distance) from at least a fraction F of the other data points (Knorr et al., 2000a). Other approaches include reducing the dimensionality of the data set to just one or two dimensions where outlier detection is easier, or performing clustering on the data (Ben-Gal, 2005).

Pre-processing the data to remove outliers has the advantage of being simple and applicable to a wide range of problems, but it has disadvantages as well. How outlier-like a point is falls on a continuum, so some threshold will be required to decide whether a point should be removed. It is unclear how to set this threshold properly, and if it is not set properly, either too much data will be removed, which may both bias the solution and increase its variance, or too little will be removed, which will not handle the outlier problem.

A second issue with this approach is that it provides an all-or-nothing solution. Intuitively it seems that points that are somewhat outlier-like should count somewhat less during training than points that are not at all outlier-like, but this pre-processing approach to handling outliers either removes a data point completely or treats it equally to all other non-removed points.

A third potential issue for pre-processing based outlier removal is that it is not obvious that outliers can reliably be detected without taking into account the machine learning model that will be applied afterwards. In other words, without a model for the data, it is harder to

say what is or isn't an outlier, but pre-processing occurs precisely before one has a model for the data. Removal of one data point can influence which other data points appear to be outlier-like, which makes building the outlier removal process directly into the iterations of the training process especially desirable, especially compared to performing outlier removal in one pass of the data.

Unlike pre-processing approaches, it is desirable to have an algorithm that automatically determines the degree of "outlier-ness" of points, that reduces the weight of outliers rather than removing them altogether, and that is built into the machine learning algorithm itself so that it can leverage the model of the data in determining what is an outlier.

3.2.3 Desirable Properties of an Outlier-Robust Algorithm

Based on the discussion above about the inadequacies of the two previously mentioned approaches to removing outliers, we seek an approach to making Ridge Regression and Ordinary Least Squares regression robust to outliers that:

1. Handles both feature outliers and label outliers.

2. Does not redefine our prediction goal (e.g. minimizing the squared error or predicting the mean label for each data point), so the solutions produced by our approach should closely approximate that of Ridge Regression (or least squares) in the case where our data does not contain outliers.

3. Automatically determines how strongly to reduce the influence of points that are outlier-like, so that an arbitrary cutoff does not need to be set by hand.

4. Reduces the weight of outliers rather than removing them entirely, so that points that are only somewhat outlier-like can be handled in an appropriate way.

5. Is built into the machine learning algorithm itself, and uses the learned model (while it is being iteratively learned) in making its determination of what an outlier is.

3.3 Ridge Regression

We now consider Ridge Regression, and examine its behavior in the presence of outliers, which as we will see is far from desirable.

Ridge regression (which encompasses Ordinary Least Squares as a special case when $\lambda = 0$) makes predictions of labels from points by solving the following Ridge Regression optimization problem' for the solution coefficients w_λ:

$$w_\lambda \equiv \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} (w^\intercal X^i - y_i)^2 + \lambda \|w\|_2^2. \tag{3.1}$$

Here $\lambda > 0$ is a complexity parameter that determines the tradeoff between minimizing the empirical error (seeking low bias) and considering only small coefficients for w_λ (seeking low variance). The solution to the optimization problem is given by:

$$w_\lambda = (XX^\intercal + \lambda I)^{-1} Xy. \tag{3.2}$$

3.3.1 Ridge Regression and Outliers

It is widely believed that Ridge Regression is "robust", due to the flexibility it allows for selecting the complexity parameter λ. Intuitively, it may seem that being able to make a choice of λ can prevent the algorithm from performing badly when a data set has outliers. One reason for this false belief may come from the fact that Ridge Regression is "uniformly stable". This means that one can bound the maximum change in prediction error that can

occur due to adding (or removing) a single data point from the training set. If the predictions cannot change much when one data point changes, it intuitively seems that the algorithm must be robust against an extreme outlier in the data. Consider the following theorem adapted from Bousquet and Elisseeff (2002), which is based on the uniform stability of Ridge Regression.

Theorem 3. *Let $Z = (X, y)$ be our training set, and let $\widetilde{Z}$ be the same as Z except with any one point (column) dropped from X and the corresponding label (row) dropped from y. Let l_Z be the linear function solving the Ridge Regression optimization problem (Equation 3.1) trained on Z, and $l_{\widetilde{Z}}$ be the corresponding solution when instead trained on $\widetilde{Z}$. The prediction error achieved by l_Z at any point $x \in \mathbb{R}^n$ cannot differ from the prediction error achieved by $l_{\widetilde{Z}}$ at x by more than β, where*

$$\beta \leq 2\frac{\kappa^2 \mathcal{M}^2}{\lambda m}$$

with the assumption that the underlying distribution from which our points and labels are generated only produces points x such that $\|x\|_2 \leq \kappa$, and only produces labels y_0 such that $y_0 \in [0, \mathcal{M}]$.

The above theorem tells us that when one point is added to (or removed from) our training data, the predictions made by Ridge Regression on all other points cannot change "very much". There are at least three reasons why the stability based bound of Theorem 3 does not imply that Ridge Regression has satisfactory resistance to outliers, even when the conditions of the theorem are met.

1. The choice of λ typically depends on the data (for example, when we use k-fold cross-validation to set this parameter). But in Theorem 3 it is assumed that λ is fixed. If the procedure for selecting λ selects a small value, then β could be very large, so that the algorithm could actually be very influenced by the addition of one point.

2. β depends on the maximum possible label value, $\mathcal{M}$. In a world where outliers are possible we may have labels in our data set that are far larger than all the rest. If the label bound $\mathcal{M}$ is large, then β, the amount our predictions can change, may be large as well.

3. β depends on the maximum possible point norm, κ. But an outlier point may lie very far from the origin. If the point norm bound κ is large, then β may be large as well.

3.3.2 k-Fold Cross-Validation with Outliers

There is an additional reason why Ridge Regression, as it is usually implemented, is not robust to outliers. This is because k-fold cross-validation, despite its huge popularity, is itself not robust to outliers. So the standard procedure for selecting λ in Ridge Regression adds another source of non-robustness.

The process of performing k-fold cross-validation involves splitting the training data $Z = (X, y)$ into k disjoint subsets $S_1, \ldots, S_k$ of approximately equal size. Then, for various choices of λ (for instance from some exponentially growing set) we train once on each of the $Z - S_i$ for $i \in \{1, \ldots, k\}$, predict the labels in the corresponding S_i, and compute the sum of squared error for the predictions of all labels in S_i. We thus get k error values for the current choice of λ, one such error for each S_i. We sum these k errors together to get a total error estimate for the current choice of λ. We carry out this procedure for each λ in our set, and finally select the λ that yields the lowest error.

Consider what happens when the k-fold cross-validation algorithm is applied to a machine learning algorithm where the training data contains one very extreme outlier (which we assume is very hard to predict, and therefore will have an enormous prediction error when we

attempt to predict it). This outlier will fall into exactly one of the $S_1, \ldots, S_k$. Without loss of generality we can assume it is in S_1. Even if we suppose that the algorithm is not affected by outliers when it trains on $Z - S_i$ for $i \neq 1$ (even though it will be training on data containing an outlier), a strange outcome will occur when training on $Z - S_1$ and computing prediction errors on S_1. The outlier in S_1 will be very poorly predicted, causing a very large prediction error. If that outlier is sufficiently extreme, that error can overwhelm the prediction error of all other points combined, both from within S_1 and from the other S_i. Since all of the prediction errors are summed together, that means that the total error could be largely determined by the error made on that one outlier point. Hence, the parameter that we are applying k-fold cross-validation in order to learn may simply be determined by whichever choice of that parameter causes the best prediction of the single terrible outlier. This is of course an awful way to select the parameter because it depends mainly on one point in the data. It is because of this lack of robustness that our algorithm applies a robust variation on k-fold cross-validation, which we call Stabilized k-Fold Cross-Validation.

3.3.3 Label Outliers with Ridge Regression

As we now demonstrate, a single point with a sufficiently large label in the training set can destroy the performance of Ridge Regression, even if λ is chosen optimally. Suppose that point X^m is a label outlier in the sense that its corresponding label y_m is very large compared to all the other labels. Now, we will write the formula for the Ridge Regression solution coefficients, which solves:

$$\min_w \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2 + w^\intercal \Lambda w.$$

Note that for now we generalize the regularization matrix λI to instead be an arbitrary positive definite matrix Λ, to gain greater clarity on the impact of regularization. We have:

$$
\begin{aligned}
w_\Lambda &= (XX^\intercal + \Lambda)^{-1} Xy \\
&= (XX^\intercal + \Lambda)^{-1} \sum_{i=1}^{m} X^i y_i \\
&= (XX^\intercal + \Lambda)^{-1} X^m y_m + (XX^\intercal + \Lambda)^{-1} \sum_{i=1}^{m-1} X^i y_i
\end{aligned}
$$

If an entry of the column vector $(XX^\intercal + \Lambda)^{-1} X^m$ is sufficiently small in magnitude relative to y_m, then y_m will not be able to distort the corresponding coefficient of w_Λ. However, as y_m gets arbitrarily large relative to the elements of $(XX^\intercal + \Lambda)^{-1} X^m$, the solution coefficients can become arbitrarily distorted, The only hope we have to ensure this does not occur is to make a judicious choice of Λ, which must depend on y_m itself. However, since one almost always uses $\Lambda = \lambda \mathrm{I}$, the only choice is over the scalar parameter $\lambda \geq 0$ in

$$
w_\Lambda = (XX^\intercal + \lambda \mathrm{I})^{-1} X^m y_m + (XX^\intercal + \lambda \mathrm{I})^{-1} \sum_{i=1}^{m-1} X^i y_i.
$$

The problem is that λ does not provide enough flexibility to simultaneously eliminate the effect of the outlier label y_m while still preserving the effects of the rest of the points. If we get lucky and it turns out that $\|(XX^\intercal)^{-1} X^m\|$ is very small then nearly any choice of lambda may be fine. But in most cases the only way to make $\|(XX^\intercal + \lambda \mathrm{I})^{-1} X^m y_m\|$ small is to choose λ very large, which will have the effect of making $\|w_\Lambda\|$ small, preventing the algorithm from learning much of anything from the non-outlier points. We have that

$$
\lim_{y_m \to \infty} \frac{w_\Lambda}{y_m} = (XX^\intercal + \Lambda)^{-1} X^m
$$

meaning that as y_m gets large, the only label that ends up having a substantial effect on the solution is y_m, and that the solution coefficients are approximately proportional to y_m for

large y_m.

3.3.4 Feature Outliers with Ridge Regression

Intuitively, one might think that allowing the features of one point to grow unboundedly might produce a similar effect as letting that point's label do so. But the story for feature outliers is very different than that of label outliers. Suppose that the label y_m for our outlier point is modest in size, but that the entries of the point vector X^m are very large (we choose X^m without loss of generality, as the effect is symmetric for all points). For simplicity, we write $X^m = c\,z$ for a fixed column vector z and scalar c and consider what happens to the solution coefficients $w_\Lambda = (XX^\intercal + \Lambda)^{-1}Xy$ as $c \to \infty$. Substituting $c\,z$ for X^m in $XX^\intercal$, we can write:

$$XX^\intercal = \widetilde{X}\widetilde{X}^\intercal + c^2\,zz^\intercal \quad \text{and} \quad (XX^\intercal + \Lambda)^{-1} = (\widetilde{X}\widetilde{X}^\intercal + \Lambda + c^2\,zz^\intercal)^{-1}$$

where $\widetilde{X}$ is the matrix X with the m'th column removed. Since $c\,zz^\intercal$ is a rank one matrix added to $\widetilde{X}\widetilde{X}^\intercal + \Lambda$, we can apply the Sherman-Morrison formula to compute the inverse, so long as $1 + c^2\,z^\intercal(\widetilde{X}\widetilde{X}^\intercal + \Lambda)^{-1}z \neq 0$. Defining

$$\widetilde{R} \equiv (\widetilde{X}\widetilde{X}^\intercal + \Lambda)^{-1}$$

this gives us:

$$(XX^\intercal + \Lambda)^{-1} = (\widetilde{X}\widetilde{X}^\intercal + \Lambda + c^2\,zz^\intercal)^{-1} = \widetilde{R} - c^2\,\frac{\widetilde{R}\,zz^\intercal\widetilde{R}}{1 + c^2\,z^\intercal\widetilde{R}z}.$$

We return now to the equation for the Ridge Regression solution coefficients, and take the limit as $c \to \infty$. So long as $z^\intercal \widetilde{R} z \neq 0$, this yields

$$
\begin{aligned}
w_\infty \equiv \lim_{c\to\infty} w_\Lambda &= \lim_{c\to\infty} (XX^\intercal + \Lambda)^{-1} X y \\
&= \lim_{c\to\infty} (XX^\intercal + \Lambda)^{-1} X^m y_m + \lim_{c\to\infty} (XX^\intercal + \Lambda)^{-1} \sum_{i=1}^{m-1} X^i y_i \\
&= \lim_{c\to\infty} (\widetilde{R} - \frac{c^2\, \widetilde{R} z z^\intercal \widetilde{R}}{1 + c^2\, z^\intercal \widetilde{R} z}) c\, z y_m + \lim_{c\to\infty} (\widetilde{R} - c^2\, \frac{\widetilde{R} z z^\intercal \widetilde{R}}{1 + c^2\, z^\intercal \widetilde{R} z}) \sum_{i=1}^{m-1} X^i y_i \\
&= \lim_{c\to\infty} (\widetilde{R} z - \frac{c^2\, \widetilde{R} z z^\intercal \widetilde{R} z}{1 + c^2\, z^\intercal \widetilde{R} z}) c\, y_m + (\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) \sum_{i=1}^{m-1} X^i y_i \\
&= \widetilde{R} z \lim_{c\to\infty} (1 - \frac{c^2\, z^\intercal \widetilde{R} z}{1 + c^2\, z^\intercal \widetilde{R} z}) c\, y_m + (\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) \sum_{i=1}^{m-1} X^i y_i \\
&= \widetilde{R} z \lim_{c\to\infty} (\frac{1}{1 + c^2\, z^\intercal \widetilde{R} z}) c\, y_m + (\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) \sum_{i=1}^{m-1} X^i y_i \\
&= (\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) \sum_{i=1}^{m-1} X^i y_i \\
&= (\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) \widetilde{X} \widetilde{y} \\
&= \widetilde{w} - \frac{\widetilde{R} z z^\intercal \widetilde{w}}{z^\intercal \widetilde{R} z}.
\end{aligned}
$$

Now, in the special case where there is only one feature, the matrix $\widetilde{R}$ and the column vector z (assumed not to be 0) both become scalars, so $(\widetilde{R} - \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}) = (\widetilde{R} - \frac{\widetilde{R}^2 z^2}{\widetilde{R} z^2}) = 0$. Hence, our final expression for w_Λ in the one dimensional case is simply $w_\Lambda = 0$. Therefore, in this case the effect of a point with very large features is extremely bad, forcing the solution coefficient to go to zero.

In the case when the number of features is greater than 1, we see a different story. Our

solution coefficients are not forced to vanish, but they are influenced by the outlier point, changing from $\widetilde{w}$, had that point not been included, to $(I - \frac{\widetilde{R}zz^\intercal}{z^\intercal \widetilde{R} z})\widetilde{w}$ instead. To see what sort of effect this is, consider the prediction that the solution coefficients w_∞ yield for the outlier itself. One has:

$$\begin{aligned} X^{m\intercal} w_\infty = c\, z^\intercal w_\infty &= cz^\intercal(\widetilde{w} - \frac{\widetilde{R}\, zz^\intercal \widetilde{w}}{z^\intercal \widetilde{R} z}) \\ &= c(z^\intercal \widetilde{w} - \frac{z^\intercal \widetilde{R}\, zz^\intercal \widetilde{w}}{z^\intercal \widetilde{R} z}) \\ &= 0. \end{aligned}$$

Hence, as $c \to \infty$ the solution coefficients yield a 0 prediction for the outlier $X^m = c\, z$, but otherwise the solution coefficients still depend mainly on the features and labels of all the other points. As c gets increasingly large, Ridge Regression is forced to make an ever smaller prediction for the point X^m. Intuitively, the reason this happens is because Ridge Regression is trying to predict y_m as accurately as possible, and as X^m gets larger, the only way this can occur is if it makes smaller and smaller predictions for the unit vector in the direction of X^m. In one dimension this has devastating effects on our solution coefficients, but in higher dimensions this can be achieved while still maintaining somewhat reasonable predictions on other directions. But the predictions are still distorted, sometimes substantially so, which is not ideal.

We will now examine just how bad an arbitrarily large feature outlier is in terms of its distorting effect on our predictions. Let $Err^c[x_i, y_i]$ be the prediction error that Ridge Regression makes on the point (x_i, y_i) when trained on the feature matrix X and column vector of labels y, with $X^m = c\, z$, and let $Err^\infty[x_i, y_i] \equiv \lim_{c\to\infty} Err^c[x_i, y_i]$, be the error made in the limit as $c \to \infty$. Furthermore, let $\widetilde{Err}[x_i, y_i]$ be the prediction that Ridge Regression

makes on the point (x_i, y_i) when trained on the feature matrix $\widetilde{X}$ (which is just X with the mth column, X^m removed) and the label column vector $\widetilde{y}$ (y with the mth row value y_m removed). We can then consider the difference in generalization and empirical error between keeping the point X^m in our training set, and omitting it before training. We define the difference in generalization error as

$$\bar{\Delta} = \mathop{\mathbb{E}}_{x_i,y_i} Err^{\infty}[x_i, y_i] - \mathop{\mathbb{E}}_{x_i,y_i} \widetilde{Err}[x_i, y_i]$$

and for the difference in empirical error (on all points except for X^m) we write

$$\widehat{\Delta} = \frac{1}{m-1}\sum_{i=1}^{m-1} Err^{\infty}[x_i, y_i] - \frac{1}{m-1}\sum_{i=1}^{m-1} \widetilde{Err}[x_i, y_i]$$

with $x_i \equiv X^i$.

These quantities $\bar{\Delta}$ and $\widehat{\Delta}$ measure how badly Ridge Regression's prediction accuracy is affected by a single training point which has arbitrarily large feature values. The next theorem provides formulas for their values.

Theorem 4. *Let $\widetilde{w}$ be the solution coefficients for Ridge Regression when trained on $\widetilde{X}$, $\widetilde{y}$, and let $\widetilde{R} \equiv (\widetilde{X}\widetilde{X}^{\intercal} + \Lambda)^{-1}$. Furthermore, let $\widetilde{m} \equiv m-1$, the sample size once the point X^m is removed. Let $\bar{C} \equiv \widetilde{m}\ \mathbb{E}_{x_i,y_i}[x_i x_i]^{\intercal}$ and $\widehat{C} \equiv \sum_{i=1}^{m-1} X^i X^{i\intercal} = \widetilde{X}\widetilde{X}^{\intercal}$, which for mean centered data are unnormalized versions of the true and empirical covariance matrices, and let $\bar{r} = \widetilde{m}\ \mathbb{E}_{x_i,y_i}[x_i y_i]$. Then:*

$$\bar{\Delta} = \frac{1}{\widetilde{m}}\left(\frac{z^{\intercal}\widetilde{w}}{\sqrt{z^{\intercal}\widetilde{R}z}}\right)^2 \frac{z^{\intercal}\widetilde{R}\bar{C}\widetilde{R}z}{z^{\intercal}\widetilde{R}z} + \frac{2}{\widetilde{m}}\left(\frac{z^{\intercal}\widetilde{w}}{\sqrt{z^{\intercal}\widetilde{R}z}}\right)\frac{z^{\intercal}\widetilde{R}(\bar{r} - \bar{C}\widetilde{w})}{\sqrt{z^{\intercal}\widetilde{R}z}}$$

and

$$\widehat{\Delta} = \frac{1}{\widetilde{m}}\left(\frac{z^\intercal \widetilde{w}}{\sqrt{z^\intercal \widetilde{R} z}}\right)^2 \frac{z^\intercal \widetilde{R}\widehat{C}\widetilde{R} z}{z^\intercal \widetilde{R} z} + \frac{2}{\widetilde{m}}\left(\frac{z^\intercal \widetilde{w}}{\sqrt{z^\intercal \widetilde{R} z}}\right) \frac{z^\intercal (\mathrm{I} - \widetilde{R}\widehat{C})\widetilde{w}}{\sqrt{z^\intercal \widetilde{R} z}}.$$

Note that if $\Lambda \to 0$ and $\widehat{C}$ is invertible, then $\widetilde{R} \to \widehat{C}^{-1}$, yielding:

$$\lim_{\Lambda \to 0} \widehat{\Delta} = \frac{1}{\widetilde{m}} \frac{(z^\intercal \widetilde{w})^2}{z^\intercal \widehat{C}^{-1} z}.$$

Proof. The proof is nearly identical for $\bar{\Delta}$ and $\widehat{\Delta}$. In the former case, which corresponds to generalization error, we define

$$\mathbb{E}[.] \equiv \mathop{\mathbb{E}}_{x_i, y_i}[.]$$

which is the expected value over some new independently sampled point (x_i, y_i), whereas for the case of $\widehat{\Delta}$, which corresponds to empirical error, we define

$$\mathbb{E}[.] \equiv \frac{1}{\widetilde{m}} \sum_{i=1}^{m-1}.$$

which is the empirical mean over all training points. We furthermore define:

$$C \equiv \widetilde{m}\, \mathbb{E}[x_i x_i^\intercal] \qquad r \equiv \widetilde{m}\, \mathbb{E}[x_i y_i^\intercal]$$

$$\widetilde{R} \equiv (\widetilde{X}\widetilde{X}^\intercal + \Lambda)^{-1} \qquad Z \equiv \frac{\widetilde{R} z z^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z}.$$

Then we have, for $\Delta \equiv \bar{\Delta}$ in the generalization error case, and $\Delta \equiv \widehat{\Delta}$ in the empirical error

case,

$$
\begin{aligned}
\Delta &\equiv \mathbb{E}[\lim_{c\to\infty} Err^c[x_i, y_i]] - \mathbb{E}[\widetilde{Err}[x_i, y_i]] \\
&= \mathbb{E}[Err^\infty[x_i, y_i]] - \mathbb{E}[\widetilde{Err}[x_i, y_i]] \\
&= \mathbb{E}[(x_i^\intercal w_\infty - y_i)^2 - (x_i^\intercal \widetilde{w} - y_i)^2] \\
&= \mathbb{E}[(x_i^\intercal (\widetilde{w} - Z\widetilde{X}\widetilde{y}) - y_i)^2 - (x_i^\intercal \widetilde{w} - y_i)^2] \\
&= \mathbb{E}[((x_i^\intercal \widetilde{w} - y_i) - x_i^\intercal Z\widetilde{X}\widetilde{y})^2 - (x_i^\intercal \widetilde{w} - y_i)^2] \\
&= \mathbb{E}[(x_i^\intercal Z\widetilde{X}\widetilde{y})^2 - 2x_i^\intercal Z\widetilde{X}\widetilde{y}(x_i^\intercal \widetilde{w} - y_i)] \\
&= \widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal \mathbb{E}[x_i x_i^\intercal] Z\widetilde{X}\widetilde{y} - 2\widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal \mathbb{E}[x_i x_i^\intercal]\widetilde{w} + 2\widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal \mathbb{E}[x_i y_i] \\
&= \frac{1}{\widetilde{m}}\left(\widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal C Z\widetilde{X}\widetilde{y} - 2\widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal C\widetilde{w} + 2\widetilde{y}^\intercal \widetilde{X}^\intercal Z^\intercal r\right) \\
&= \frac{1}{\widetilde{m}}\left(\widetilde{w}^\intercal \frac{zz^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z} C \frac{\widetilde{R} zz^\intercal}{z^\intercal \widetilde{R} z}\widetilde{w} - 2\widetilde{w}^\intercal \frac{zz^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z} C\widetilde{w} + 2\widetilde{w}^\intercal \frac{zz^\intercal \widetilde{R}}{z^\intercal \widetilde{R} z} r\right) \\
&= \frac{1}{\widetilde{m}} \frac{(z^\intercal \widetilde{w})^2}{z^\intercal \widetilde{R} z} \frac{z^\intercal \widetilde{R} C \widetilde{R} z}{z^\intercal \widetilde{R} z} + \frac{2}{\widetilde{m}} \frac{z^\intercal \widetilde{w}}{z^\intercal \widetilde{R} z} z^\intercal \widetilde{R}(r - C\widetilde{w}) \\
&= \frac{1}{\widetilde{m}}\left(\frac{z^\intercal \widetilde{w}}{\sqrt{z^\intercal \widetilde{R} z}}\right)^2 \frac{z^\intercal \widetilde{R} C \widetilde{R} z}{z^\intercal \widetilde{R} z} + \frac{2}{\widetilde{m}}\left(\frac{z^\intercal \widetilde{w}}{\sqrt{z^\intercal \widetilde{R} z}}\right) \frac{z^\intercal \widetilde{R}(r - C\widetilde{w})}{\sqrt{z^\intercal \widetilde{R} z}}.
\end{aligned}
$$

In the case of empirical error, we use the additional fact that $\widetilde{R}r = \widetilde{R}\widetilde{X}\widetilde{y} = \widetilde{R}\widetilde{w}$ to simplify further. Observing that $C = \bar{C}$ and $r = \bar{r}$ in the case of generalization error, and that $C = \widehat{C}$ in the case of empirical error, completes the proof. □

We now introduce iteratively reweighted Ridge Regression, which provides the algorithmic framework for our new algorithm, Stabilized Ridge Regression. This algorithm will behave similarly to Ridge Regression when no outliers are present, but unlike Ridge Regression it is inherently robust to outliers.

3.4 Iteratively Reweighed Ridge Regression

Let m be the number of data points in our training set, and let X be the $n \times m$ matrix of points, with one point X^i per column, and one feature X_i per row. Likewise, let the $m \times 1$ column vector y represent the labels of our training set, with one label y_i for each point X^i. We assume here, and throughout this chapter that X has had a row of all 1's added, to handle the intercept (i.e. constant term) of linear regression. With this assumption in place, we can simply treat the constant term as one more feature like any other throughout this work.

Consider Algorithm 1, the iteratively reweighted Ridge Regression algorithm. This algorithm is very versatile. For instance for $1 < p \leq 2$, if we choose to assign weights to points using the reweighting function $v_i \rightarrow \frac{1}{|X^{i\intercal}w - y_i|^{2-p}}$, this algorithm converges to $w = \text{argmin}_{\widetilde{w}} \sum_{i=1}^{m} (X^i \widetilde{w} - y_i)^p$. Hence this choice of reweighting function replaces the L_2 loss function with the L_p loss function. Examining this family of reweighting functions one sees immediately why they do not appropriately handle feature outliers: they do not account for the extremeness of the feature vector X^i except through its influence in predicting the label. As we shall see, the extremeness of the feature vector should be handled more explicitly. Furthermore, from examining this family of reweighting function one can also see why they can produce quite different solutions than Ordinary Least Squares even when there are no outliers: as p gets close to 1, where strong outlier robustness occurs, such functions significantly reduce the weight of points that are not very outlier-like. In other words, it is not merely the outliers that are substantially affected by these re-weighting schemes, making them questionable as methods of outlier reduction.

Note that in the case where the reweighting function always assigns the constant 1 for all v_i, the Iteratively Reweighted Ridge Regression algorithm above simply gives the Ridge Regression solution $w = (XX^\intercal + \lambda I)^{-1} Xy$. In other words, this algorithm generalizes Ridge

Algorithm 1 Iteratively Reweighted Ridge Regression

1: Initialize the non-negative weight vector v to be a vector of all 1's (indicating that initially all points count equally).

2: Set the solution coefficients w by solving the weighted Ridge Regression problem for the solution coefficients $w \rightarrow \text{argmin}_{w'} \sum_{i=1}^{m} v_i (X^{i\intercal} w' - y_i)^2 + \lambda \|w'\|_2^2$, with solution $w = (XVX^\intercal + \lambda I)^{-1} XVy$. Here, λ is the complexity parameter of Ridge Regression determining how strongly to push the solution towards the zero vector $\vec{0}$, I is the identity matrix, and V is a diagonal matrix with the non-negative weights v_i as its diagonal elements.

3: Adjust the weights $v_i \rightarrow \Omega_i(w, X, y)$ using some "reweighting" function Ω which may depend on the solution w, the training points X and the training labels y.

4: Normalize the weights v_i so that they sum to the number of points, m, by taking $v_i \rightarrow \frac{v_i}{\frac{1}{m}\sum_{j=1}^{m} v_j}$.

5: If the solution coefficients w have not changed in 2-norm by more than ϵ compared to the solution coefficients from the previous iteration, stop, and use these last solution coefficients to predict future data. Otherwise, return to Step 2.

Regression.

3.5 Stabilized Ridge Regression

We propose a new algorithm that we call Stabilized Ridge Regression. It is implemented using iteratively reweighed Ridge Regression (Algorithm 1) but with a unique reweighting function that contains a free parameter α, measuring the degree that outliers need to be handled. Let $R \equiv (XVX^\intercal + \lambda I)^{-1}$. Our algorithm uses the following specially chosen reweighting function:

$$v_i \rightarrow \mathbf{v}_\alpha(z_i)$$

where

$$\mathbf{v}_\alpha(z_i) = \frac{1}{1 + (\alpha z_i)^6}$$

and

$$z_i = \frac{X^{i\intercal} R X^i}{1 - v_i X^{i\intercal} R X^i} |X^{i\intercal} w - y_i|$$

To set α we introduce a robust variation on k-fold cross-validation, which we call Stabilized k-Fold Cross-Validation. As seen below in Algorithm 2, the cross-validation process proceeds much like ordinary k-fold cross-validation, but computes error in a slightly different and much more robust fashion. As usual, we use some wide ranging set of α values, for instance, $\alpha \in \mathbb{A} \equiv \{10^{-6}, 2 * 10^{-6}, 4 * 10^{-6}, \ldots, 2^{40} * 10^{-6}\}$. For the parameter $0 < \rho < 100$ in the algorithm, we generally use $\rho = 5$, representing clipping at the 5th and 95th percentile for error values. This choice performs well so long as less than 5% of the data consists of substantial outliers. In situations where outliers could realistically be even more common than 5%, the value of ρ should be increased. For reasons that will be discussed later, we add a further rule to the Stabilized Ridge Regression Algorithm, that whenever more than half the points are assigned a weight less than $\frac{1}{2}$ (before the weights have been normalized to sum

to the number of points) then we skip that α and do not consider it during cross-validation. This allows us to skip large values of α, since once this condition is triggered for a given α, we need not consider larger ones. When training on many choices for α, one can further speed up the algorithm by starting with the lowest α and using the final weights from the previous α as the starting point for the subsequent larger α.

When we require $\lambda > 0$ to avoid overfitting or to make it possible to invert the feature similarity matrix, we can do so by applying k-fold stabilized cross-validation to that parameter as well, trying combinations of λ and α to find one that performs well. When $\lambda = 0$ we call the algorithm Stabilized Least Squares because it corresponds to an outlier robust modification of the Ordinary Least Squares algorithm.

Finally, we note that it is usually a good idea to normalize the features of the training data. We recommend this is done by computing the median and median absolute deviation (i.e. the median of the absolute value of the differences of each value from the median) of each feature in the training data. Then, subtract the corresponding median from each feature in both the training and testing data, and then divide each result by the corresponding median absolute deviation. This is not strictly necessary, but it serves to center the data, and make the importance of each feature be of comparable importance from the perspective of the regularization term. The normalization that is typically done (subtracting the feature means, and dividing by the feature standard deviations) is a bad idea in this context. Outliers can cause the standard deviation and even the mean to be very unreasonable values, which can dramatically distort the results. Hence, we base normalization on the median and median absolute deviation rather than on the mean and standard deviation.

A concise python programming language implementation of the core of the Stabilized Re-

gression algorithm is provided in section 3.9. Stabilized k-Fold Cross-Validation would be applied to the provided function to learn the parameter α (for Stabilized Least Squares) or to learn both the parameters α and λ (for Stabilized Ridge Regression).

3.5.0.1 More Flexible Reweighting Functions

For more flexibility in the assignment of the weights, another option would be to introduce a second reweighting function parameter, such as $p > 1$ in:

$$\mathbf{v}_\alpha(z_i) = \frac{1}{1 + (\alpha z_i)^p}$$

This would require cross-validation over both p and α (when $\lambda = 0$ is an acceptable choice) or over p, α and λ (when $\lambda > 0$ is required). In practice though, we find this extra flexibility in the weight function is generally not needed.

3.5.0.2 Using the Weights of Stabilized Ridge Regression

We note that when the Stabilized Ridge Regression algorithm has been run, it provides not only a method for making label predictions, but it also provides the weights, v_i, to assign to each point in the training data. These weights can then be used to estimate, for example, the covariance matrix of the features matrix or the mean of the features in a robust fashion (e.g. by replacing averages with weighted averages). Thus, the algorithm tells us how to weight our training data for any statistical calculation where robustness is needed. Let us therefore switch our perspective momentarily, and rather than viewing Stabilized Ridge Regression as a machine learning algorithm, view it as a way to produce robust statistics (i.e. using it to calculate weights for each training point, and then to compute a robust version of the statistics of interest by applying those weights). In this case, we can think of the labels, y, as providing the measure of external validity that is used to estimate the free parameter α

and set our weights for each training point. In other words, the Stabilized Ridge Regression algorithm will tell us how to weight our training data for statistical calculations to make those calculations robust to outliers, but it has a free parameter α that needs to be set, which measures how harshly we reduce the weight of outliers. How do we set this free parameter? We do so by considering which choice of α will allow us to best (robustly) predict the variable y from the features. So if we think that our features should allow us to predict y, we can robustly estimate the covariance of our features or other statistics of our features by running Stabilized Ridge Regression and then using the α that best allows us to use our features to predict y. What's more, y is used in the weight calculations themselves. Training points that do not yield accurate predictions of their own label are given less weight.

Consider the following example where such a procedure is useful. Suppose a psychologist needs to estimate the covariance matrix of a set of variables that reflect past behaviors related to impulsivity (e.g. number of car accidents, number of fights, number of jobs quit) that was gathered on 300 people (e.g. so that a factor analysis of the variables can be performed). But the data seems to have many outliers, due to a combination of human entry error and wide behavioral variation. A regular covariance matrix calculation will produce a very inaccurate covariance matrix due to these outliers, leading to a distorted factor analysis. But suppose that the psychologist has a measure of external validity, which in this case might be scores for each participant on an impulsivity personality test. By choosing these test scores for the labels, y, the psychologist can then run Stabilized Ridge Regression on the data to produce a weight for each data point, which can then be used to robustly estimate the covariance matrix.

Switching perspectives once more, yet another way to think about Stabilized Ridge Regression is as a means to solve for α. In this case, we can think of α as a way of measuring how outlier like our feature matrix X is, with respect to prediction variable y. In other words,

α can be thought of a way of capturing the outlier-ness of a data set (X, y) as a single number. If α is large, then standard statistical methods like Ordinary Least Squares regression, Ridge Regression, covariance matrix estimates, and standard deviations calculations will tend to be very inaccurate estimators. If on the other hand α is small, then we are in the realm where our usual statistical methods work well.

Algorithm 2 Stabilized Cross-Validation

1: Divide our training data $Z = (X, y)$ into k mutually exclusive subsets, $F_1, \ldots, F_k$ each with nearly the same number of points.

2: Create k new overlapping training sets $Z_1, \ldots, Z_k$ defined as $Z_i = Z - F_i$.

3: For each $\alpha_j \in \mathbb{A}$, and for each $Z_i \in \{Z_1, \ldots, Z_k\}$, train our Stabilized Ridge Regression algorithm on Z_i with $\alpha = \alpha_j$, and compute the squared prediction error on every point in F_i.

4: For each $\alpha_j \in \mathbb{A}$, estimate the mean of all the squared out-of-sample prediction errors using a robust mean estimator. In principle any robust estimate of the mean would do, but the standard estimator (sum the values, divide by their number) will not work. The robust mean estimator we have used in the examples below takes all such squared errors associated with α_j and computes the ρth percentile and 100 - ρth percentile. Any value below the ρth percentile gets set to the ρth percentile value, and any value above the 100 - ρth percentile gets set to the 100 - ρth percentile value. Then, once this clipping has been performed, one takes the mean computed in the usual way.

5: Choose the α_j with the lowest average truncated squared prediction error.

6: Train a new Stabilized Ridge Regression algorithm with the winning α_j on the full training set Z, and use its solution to make predictions on future data.

3.6 Stability of Solutions

To explain where the reweighting function used in the Stabilized Ridge Regression algorithm comes from, we will consider how much our solution coefficients w change when we drop one point from our data set. Considering the impact of dropping a point from the training set is of course equivalent to considering the impact of adding a point to the training set, but for convenience we will refer to such cases as a point "being dropped". Our reweighting function is selected so as to bound the impact that an extreme label or feature outlier can have on our predictions, as measured by considering how predictions change when that point is dropped. As discussed, this does not make the algorithm uniformly stable (a commonly used notion of stability, see Bousquet and Elisseeff (2002)). But uniform stability is actually not the right notion of robustness when outliers are being considered. Rather, our approach makes the change in predictions of Stabilized Ridge Regression not too large when dropping an extreme point, assuming the point we are predicting is not too extreme. Any linear prediction method will predict an arbitrarily large value when applied to a point whose feature vector has sufficiently large norm. This is simply part of what it means to have a linear model. So strictly speaking, for linear methods, uniform stability is not possible without assumptions about the magnitude of the feature vectors that the algorithm will be applied to (an assumption we cannot make since we are precisely considering the case of outliers). Hence, our interest is not in uniform stability per se, but in limiting how much outliers change our solution coefficients, which can be investigating by assessing how much our solution coefficients can change when a point is dropped from our training data.

Recall that the reweighting function used in Stabilized Ridge Regression is parameterized by a non-negative number α, which is used to set the tradeoff between not eliminating the effect of true outliers, and removing the effect of non-outliers. At the extreme of $\alpha = 0$

we simply reproduce Ridge Regression, with the usual lack of robustness to outliers, and at the opposite extreme as $\alpha \to \infty$ outliers are penalized to an extreme degree, which will be discarding too much information. Since there is not a clear cut distinction between outliers and non-outliers, this tradeoff will always exist, but our algorithm addresses this problem by learning the best choice of α for the given problem using our robust version of k-fold cross-validation.

3.6.1 Stability of Weighted Ridge Regression

If an outlier is to destroy the prediction accuracy of an algorithm, it must do so by changing the solution coefficients w, since those coefficients fully determine the predictions made. The ultimate measure of how much the solution coefficients are effected by the removal of a single point is by examining the difference in solution coefficients including and removing the point. In other words, robustness against the insertion of an outlier hinges on the magnitude of the quantity:

$$\widetilde{w} - w$$

where w is the vector of solution coefficients, and $\widetilde{w}$ is the same solution coefficients but as they would have been had we not been given one of the training points. Without loss of generality, we assume that the point being dropped is $z_m = (X^m, y_m)$, the last point in the data set. We will demonstrate that through a careful selection of the weight v_i that we assign to each point, one can prevent $\widetilde{w} - w$ from becoming huge in magnitude, even in the case where the point being dropped is arbitrarily extreme, with arbitrarily large label or arbitrarily large feature vector norm.

To study the change in solution coefficients w when the last point in the training set is dropped, we will use $\widetilde{X}$, $\widetilde{y}$ and $\widetilde{V}$ to mean the points X, labels y, and diagonal weight

matrix V, respectively, after the mth point is removed from each (i.e. exactly as they would have been had there never been a point (X^m, y_m)). We note that since we are performing iterative reweighting in our Stabilized Ridge Regression algorithm, the weights themselves depend on the data, and therefore in general the entries of $\widetilde{V}$ are different than the entries of V. Additionally, the former is an $m-1 \times m-1$ positive diagonal matrix whereas the latter is an $m \times m$ positive diagonal matrix.

We define $R \equiv (XVX^\intercal + \lambda I)^{-1}$ and $\widetilde{R} \equiv (\widetilde{X}\widetilde{V}\widetilde{X}^\intercal + \lambda I)^{-1}$, where we assume $\lambda > 0$ to make sure the matrix is invertible and positive-definite. Finally, we let $\bar{V}$ be identical to the diagonal weight matrix V, except with the mth row and mth column of V removed (so that $\bar{V}$ and $\widetilde{V}$ are both $m-1 \times m-1$ and so can be subtracted from each other), and we define:

$$V^\Delta \equiv \widetilde{V} - \bar{V}.$$

The formula that we wish to derive for $\widetilde{w} - w$ will be designed to take a special form, such that it separates into two terms: (1) the direct effect that label y_m, feature vector X^m and corresponding weight v_m have on the change in solution coefficients when we drop the mth point, and (2) a term independent of the point being dropped except due to the indirect effect of the weights on all other points changing when we drop this point. Applying the relation $w = RXVy$, we get:

$$\begin{aligned}
\widetilde{w} - w &= \widetilde{R}\widetilde{X}\widetilde{V}\widetilde{y} - RXVy \\
&= (\widetilde{R}\widetilde{X}\bar{V}\widetilde{y} - RXVy) + (\widetilde{R}\widetilde{X}\widetilde{V}\widetilde{y} - \widetilde{R}\widetilde{X}\bar{V}\widetilde{y}) \\
&= (\widetilde{R}\widetilde{X}\bar{V}\widetilde{y} - RXVy) + \widetilde{R}\widetilde{X}V^\Delta\widetilde{y}.
\end{aligned}$$

Now

$$\widetilde{X}\bar{V}\widetilde{y} = \sum_{i=1}^{m-1} v_i X^i y_i = XVy - v_m X^m y_m.$$

Furthermore

$$\begin{aligned}\widetilde{R} &= (\widetilde{X}\widetilde{V}\widetilde{X}^\intercal + \lambda I)^{-1} = \Big(\sum_{i=1}^{m-1} v_i X^i X^{j\intercal} + \lambda I\Big)^{-1} = (XVX^\intercal + \lambda I - v_m X^m X^{m\intercal})^{-1} \\ &= (R^{-1} - v_m X^m X^{m\intercal})^{-1} = R + v_m \frac{RX^m X^{m\intercal} R}{1 - v_m X^{m\intercal} R X^m}\end{aligned}$$

by the Sherman-Morrison formula, as long as $v_m X^{m\intercal} R X^m \neq 1$, a condition automatically satisfied, since $v_m X^{m\intercal} R X^m = v_m X^{m\intercal}\Big(P + v_m X^m X^{m\intercal}\Big)^{-1} X^m$ for a positive definite matrix P, which means Corollary 1 implies $v_m X^{m\intercal} R X^m < 1$.

Putting the above expression for $\widetilde{R}$ together with the expression for $\widetilde{X}\bar{V}\widetilde{y}$, we form an expression for $\widetilde{R}\widetilde{X}\bar{V}\widetilde{y}$, and so can write:

$$\begin{aligned}\widetilde{R}\widetilde{X}\bar{V}\widetilde{y} - RXVy &= \Big(R + v_m \frac{RX^m X^{m\intercal} R}{1 - v_m X^{m\intercal} R X^m}\Big)(XVy - v_m X^m y_m) - RXVy \\ &= RXVy - v_m RX^m y_m + v_m \frac{RX^m X^{m\intercal} R}{1 - v_m X^{m\intercal} R X^m}(XVy - v_m X^m y_m) - RXVy \\ &= v_m RX^m \Big(\frac{X^{m\intercal} R(XVy - v_m X^m y_m)}{1 - v_m X^{m\intercal} R X^m} - y_m\Big) \\ &= v_m RX^m \Big(\frac{X^{m\intercal} R(XVy - v_m X^m y_m) - y_m(1 - v_m X^{m\intercal} R X^m)}{1 - v_m X^{m\intercal} R X^m}\Big) \\ &= v_m RX^m \Big(\frac{X^{m\intercal} RXVy - y_m}{1 - v_m X^{m\intercal} R X^m}\Big) \\ &= \frac{v_m RX^m}{1 - v_m X^{m\intercal} R X^m}\big(X^{m\intercal} w - y_m\big).\end{aligned}$$

So far we have shown that our solution coefficients w change by the following amount due to dropping point $z_m = (X^m, y_m)$:

$$\widetilde{w} - w = v_m A_m + B_m.$$

where

$$A_m = \frac{RX^m}{1 - v_m X^{m\intercal} R X^m}\left(X^{m\intercal} w - y_m\right) \text{ and } B_m = \widetilde{R}\widetilde{X}V^{\Delta}\widetilde{y}.$$

We next examine the terms of the above equation to see when it will produce a large change in solution coefficients due to the dropped point being an outlier.

3.6.2 Examining B_m

The term B_m does not contain the dropped point (X^m, y_m), and it does not depend on the dropped weight, v_m, except for indirectly within $V^{\Delta} = \widetilde{V} - \bar{V}$, since this expression accounts for the other weights v_i changing as a consequence of the mth point disappearing. In other words, B_m is independent of the point being dropped except through this points influence on the change in weights. We can think of B_m as capturing the stability of the reweighting function that assigns a weight to each point. As we see in Lemma 1, B_m is bounded, no matter how extreme an outlier the dropped point is.

Lemma 1 (Bounding B_m)**.** *The maximum value of $\|B_m\|_2$ is bounded by the following expression, which does not depend on the label y_m or the feature vector X^m of the point being dropped:*

$$\sup_{z_m = (X^m, y_m)} \|B_m\|_2 < \sqrt{2} m \|\widetilde{R}\widetilde{X}\|_2 \|\widetilde{y}\|_2.$$

Proof. Observe that:

$$\begin{aligned}
\sup_{z_m} \|B_m\|_2 &= \sup_{z_m} \|\widetilde{R}\widetilde{X}(\widetilde{V} - \bar{V})\widetilde{y}\|_2 \\
&\leq \|\widetilde{R}\widetilde{X}\|_2 \max_{\widetilde{V},\bar{V}} \|\widetilde{V} - \bar{V}\|_2 \|\widetilde{y}\|_2 \\
&= \sqrt{(m-1)^2 + m^2} \|\widetilde{R}\widetilde{X}\|_2 \|\widetilde{y}\|_2 \\
&< \sqrt{2}m \|\widetilde{R}\widetilde{X}\|_2 \|\widetilde{y}\|_2
\end{aligned}$$

where we made use of the fact that $\widetilde{V}$ and $\bar{V}$ are both positive diagonal matrices, with the former summing to $m-1$ and the latter summing to at most m. □

In practice though, a good choice for our reweighting function will keep B_m much smaller than this bound suggests. At one extreme, one can consider reweighting functions that weight all points uniformly regardless of the training data, assigning the constant weights $v_i = 1$ and $\widetilde{v}_i = 1$. The weights v_i and $\widetilde{v}_i$ each must sum to the number of points, m and $m-1$ respectively, so the diagonal elements of V are all 1, and therefore so are the diagonal elements of $\bar{V}$. Similarly, the diagonal elements of $\widetilde{V}$ are also all one. Hence, for this constant reweighting function, $B_m = 0$, since $V^\Delta = 0$.

At the other extreme, one can consider what happens when the reweighting function assigns all of the weight to one point (which is not the outlier being dropped), and no weight to any other points. In this example, when the most extreme outlier is dropped, we assume the weight switches completely to another point, causing $\|V^\Delta\|_2$ to achieve its maximum value of $\sqrt{(m-1)^2 + m^2}$. This upper bound reflects a strange and poor choice of reweighting function, because the weight is all focussed on one point, and which point it is focussed on is highly influenced by another point (i.e. the one being dropped).

In a truly ideal case, with an ideal reweighting function, an extreme outlier would be assigned (essentially) zero weight, and assuming the other points are not outlier-like at all, they would be equally weighted with all the remaining weight. In this ideal case we have $V^{\Delta} = \widetilde{V} - \bar{V} = \text{diag}(1 - \frac{m}{m-1}) = \frac{-1}{m-1} I$. Therefore, we would have

$$B_m = \frac{-1}{m-1} \widetilde{R} \widetilde{X} \widetilde{y} = \widetilde{R} \left(\frac{-1}{m-1} \sum_{i=1}^{m-1} X^i y_i \right).$$

Except in extreme cases where the label or feature distributions have infinite means, $\frac{1}{m-1} \sum_{i=1}^{m-1} X^i y_i$ will approach a constant as m gets large. Note that the quantity that B_m converges to in this case does not depend on X^m or y_m at all, which means that B_m is impervious to the outlier being dropped.

Let us generalize this ideal case to one where there are k extreme outliers instead of just one, and analyze how the solution coefficients change when any one of these k outliers is dropped. We again assume an ideal situation where each outlier gets assigned 0 weight, with the remaining weights uniformly distributed between the other points, all of which are not outlier-like at all. This implies then that $V^{\Delta} = \text{diag}(\frac{m-1}{m-k} - \frac{m}{m-k}) = \frac{-1}{m-k} I$. If g is the fraction of good points, that is, the fraction that are not outliers, then $k = (1-g)m$, and so in this case $V^{\Delta} = \frac{1}{g} \frac{-1}{m} I$. Therefore, the magnitude of B_m still converges to a constant as m gets large, but the value of that constant is larger when the fraction of non-outliers is smaller.

The ideal cases above hint at what to look for in our choice of reweighting function: it should be such that extreme outliers are assigned very small weights, and points that are not outlier-like are all given approximately all the same weight near 1.

3.6.3 Examining A_m

The term A_m is a major problem from the point of view of stability when dropping an outlier. Unlike B_m it can get arbitrarily large when an arbitrarily bad point is dropped (specifically, when $|y_m|$ gets arbitrarily large). Intuitively speaking, A_m captures the direct influence of the point $z_m = (X^m, y_m)$ on our solution, rather than how our weights v_i change when the point is dropped, which is captured by B_m.

An important aspect of A_m is that it appears with v_m multiplying it, which is the weight assigned to the point being dropped. Since our goal is to limit how much influence an outlier can have on the solution coefficients when the outlier is dropped, we need to set this weight v_m so as to limit the effect of A_m.

3.6.3.1 The Outlier Measure and Reweighting Function

With this goal in mind, we now examine a special choice for the weight v_m. First, we define a measure of how outlier-like the mth point is, given by:

$$d_m \equiv |\frac{X^{m\intercal} A_m}{\sqrt{X^{m\intercal} R X^m}}| = \frac{\sqrt{X^{m\intercal} R X^m}}{1 - v_m X^{m\intercal} R X^m} |X^{m\intercal} w - y_m|.$$

For our weights, we use

$$v_m \equiv g_\alpha(d_m)$$

where $g_\alpha(d)$ is chosen to be a non-increasing, continuous function defined on $d \geq 0$, with range $[0, 1]$ and satisfying $g_\alpha(0) = 1$. Here, α is a parameter determining how strongly to lower the weight of outliers, which we will discuss in detail later. This function $g_\alpha(d)$ must be non-increasing as a function of d because we cannot assign more weight to points that are more outlier-like than points that are less outlier-like. It must be non-negative

because it represents the weight given to each point, which cannot be negative. We give it an upper bound, which is also its value for $d_m = 0$, because we want it to treat non outlier-like points equally, not to give them arbitrarily large weight which could cause other useful data points to be ignored. The choice of 1 for the upper bound is arbitrary, since our weights will ultimately be normalized to sum to the number of points. For now we work with unnormalized weights v_m, but we will discuss the impact of normalizing them later on. We choose $g_\alpha(d)$ to be a continuous function of d; as a consequence, we will be able to apply a fixed-point theorem to show the existence of a solution for the weights. We also require that $g_\alpha(d)$ assigns approximately uniform weights near 1 to all points with $d \approx 0$. We enforce this by requiring that when $\alpha > 0$ the right hand derivative satisfies $\lim_{d\to 0^+} \frac{g_\alpha(d)-g_\alpha(0)}{d} = 0$, which makes g locally a uniform distribution near 0. The reason we want this restriction is due to the fact that $d \approx 0$ corresponds to points that are not at all outlier-like. In order for our Stabilized Ridge Regression algorithm to produce predictions that are very similar to Ridge Regression in cases without outliers, we need all points with $d \approx 0$ to be assigned the same (maximal) weight. It is not desirable to reduce the weight of non-outliers, and doing so will cause poorer predictions. Finally, we also require that g_α has the property:

$$\sup_{d\geq 0} g_\alpha(d)d \leq \frac{C}{\alpha}$$

for some bounding constant C. Note that all of these properties we need for $g_\alpha(d)$ make it conform precisely to the definition of a Parameterized Weight Function (Greenberg et al., 2016a). There are infinitely many such Parameterized Weight Functions to choose from, but we use this particular one, $\mathbf{v}_\alpha(z_i) = \frac{1}{1+(\alpha z_i)^6}$, as it is convenient due to (a) its wide region of uniformity near $z_i = 0$ (so that most points can be nearly equal weighted), and (b) its relatively simple form. In figure 3.1 we show what this function looks like for various α.

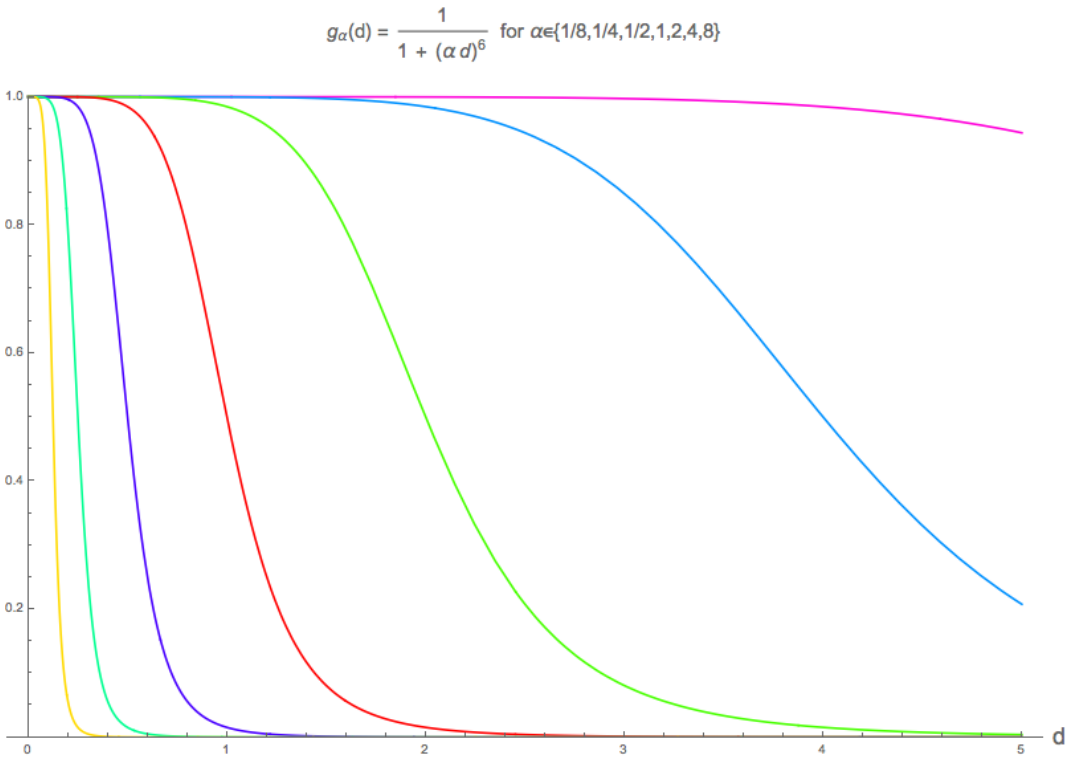

Figure 3.1: Our reweighting function for various values of α.

3.6.3.2 Bounding A_m

With the above properties of $g_\alpha(d)$ in mind, we can write:

$$\begin{aligned} v_m A_m &= v_m d_m \frac{A_m}{d_m} = g_\alpha(d_m) d_m \frac{A_m}{d_m} = g_\alpha(d_m) d_m \frac{\frac{RX^m}{1 - v_m X^{m\intercal} R X^m}\left(X^{m\intercal} w - y_m\right)}{\frac{\sqrt{X^{m\intercal} R X^m}}{1 - v_m X^{m\intercal} R X^m}\left|X^{m\intercal} w - y_m\right|} \\ &= g_\alpha(d_m) d_m \frac{RX^m}{\sqrt{X^{m\intercal} R X^m}} \operatorname{sign}(X^{m\intercal} w - y_m) \end{aligned}$$

Therefore:

$$\begin{aligned}\|v_m A_m\|_2 \leq \frac{C}{\alpha}\frac{\|RX^m\|_2}{\sqrt{X^{m\intercal}RX^m}} &= \frac{C}{\alpha}\sqrt{\frac{X^{m\intercal}R^2X^m}{X^{m\intercal}RX^m}} \\ &\leq \sqrt{\lambda_{\max}[R]}\frac{C}{\alpha} \\ &= \frac{1}{\sqrt{\lambda_{\min}[XVX^\intercal + \lambda I]}}\frac{C}{\alpha} = \frac{1}{\sqrt{\lambda_{\min}[XVX^\intercal] + \lambda}}\frac{C}{\alpha} \leq \frac{1}{\sqrt{\lambda}}\frac{C}{\alpha}.\end{aligned}$$

Hence, by varying α we can change the bound on the magnitude of $v_m A_m$, which is precisely what we need to prevent large outliers from greatly affecting our solution coefficients. Recall that B_m is not only bounded, but even more importantly, does not depend on the dropped point (X^m, y_m) at all except through its influence on how dropping this point changes the weights of other points. It is A_m that will blow up due to an extreme outlier, and it is precisely our choice of weight v_m that allows us to control this.

3.6.3.3 Impact of Normalizing the Weights

Let us again consider the quantity $\|v_m A_m\|_2$, which we took care to bound previously. This quantity does not yet take into account the normalization that will occur on the v_i to cause them to sum to the number of points, m. Because of our property that the right hand derivative of $g_\alpha(d)$ evaluated at $d = 0$ must be 0, our bound on $\|v_m A_m\|_2$ will not explode once normalization has been taken into account in realistic situations. Most points will not be very outlier-like, which means they will be assigned weights close to 1. The normalization constant that we divide every weight by is $\frac{1}{m}\sum_{i=1}^m v_\alpha(i)$. If at least a fraction q of our points have a weight of at least $1 - \epsilon$ for some positive ϵ, then the normalized weight satisfies

$$\frac{v_m}{\frac{1}{m}\sum_{i=1}^m v_i} \leq \frac{v_m}{\frac{1}{m}(qm(1-\epsilon))} = \frac{v_m}{q(1-\epsilon)}.$$

Even in a really extreme case where only half the points are not outlier-like (i.e. $q = \frac{1}{2}$) and where not being outlier-like is defined very broadly to include any point with weight at least $\frac{1}{2}$ (i.e. $\epsilon = \frac{1}{2}$) we get a very moderate bound after normalization, with $\frac{v_m}{\frac{1}{m}\sum_{i=1}^{m} v_i} \leq 4v_m$. Hence we see that the property of assigning non-outlier like points a weight close to 1 is essential for keeping our bound small once normalization is taken into account. When α gets very large, this property is lost, since essentially all data will be considered an outlier. But we need not worry about the high α case since that case corresponds to ignoring virtually all of our data, which means we will not predict accurately and so our stabilized k-fold cross-validation will not select such an α due to bad out of sample predictions. This also tells us that we need not try arbitrary high α values during cross-validation. When performing iterations for a specific α, if we ever find that too many points have too little weight (before normalization) we can simply stop and not try any α larger than the current one. In practice, we find that if more than half of the points have an unnormalized weight of less than $\frac{1}{2}$ that is good enough grounds not to try higher α. Using this criteria also implies that the bound on $v_i A_i$ can't grow by more than a factor of 4 once we normalize the weights, which is also desirable. Allowing arbitrary large α prevents us from maintaining such a bound.

3.6.3.4 Relation to Other Work

Consider the quantity we used in the derivation above for measuring outlier-ness, $d_m \equiv \frac{\sqrt{X^{m\intercal} R X^m}}{1 - v_m X^{m\intercal} R X^m} |X^{m\intercal} w - y_m|$. This is close to the square root of Cook's distance (Chatterjee and Hadi, 1986), though normalized differently. Cook's distance is sometimes used in the statistics literature as a means to detect outliers so that they can be removed as part of a pre-processing step. The difference is that Cook's distance uses no regularization parameter λ, has no weights v_i, and is not used as part of an iterative algorithm. Furthermore, we do not use d_m directly, but pass it first through our function $g_\alpha(d_m)$ to produce a weight.

3.6.3.5 Changing the Dropped Point

In the above derivation for d_m we assumed it was the mth point that was dropped. But of course, we care about the influence of all points. So generalizing the work above for all points, and normalizing the weights so that they sum to the number of points, we will assign to the ith point a final normalized weight of the following form:

$$v_i = \frac{g_\alpha(d_i)}{\frac{1}{m}\sum_{j=1}^{m} g_\alpha(d_j)}$$

where

$$d_i = \frac{\sqrt{X^{i\intercal} R X^i}}{1 - v_i X^{i\intercal} R X^i} |X^{i\intercal} w - y_i|.$$

We note that this is a recursive equation for v_i, since the weights occur on both sides, including within the definitions for R and w. This is why Stabilized Ridge Regression is an iterative algorithm. We begin with uniform weights $v_i = 1$, which is analogous to Ridge Regression, then use that to compute R and w which are used to estimate what the weights should be, then the whole process is repeated until we converge closely enough to a final set of weights.

3.6.3.6 The Existence of a Solution

Since the weights must each be non-negative and they must sum to m, the set of possible weight vectors v is an m-1 dimensional closed simplex. Our weight vector v is ultimately defined using an equation of the form $v = F(v)$, where F is a function that maps this compact convex simplex into itself. Since $g_\alpha(d)$ is defined to be continuous, F is continuous as well, so we can apply the Brouwer fixed-point theorem, which tells us that there exists a point v_0 satisfying $v_0 = F(v_0)$. That means there is at least one solution for our weight vector. In our empirical tests of Stabilized Ridge Regression, we have found that the iterations of the algorithm converge quite rapidly to a solution for the weights, usually within 40 iterations,

but typically much faster. Intuitively, we can see why iteratively applying F to a starting weight vector will converge to a solution. The presence of an outlier tends to make that outlier seem less outlier-like than it really is. Once we detect it as an outlier, the next iteration its weight will be reduced, so it will seem more outlier-like than it did before. This process continues with the weight of the outlier progressively falling until either it converges, or the process stops when the outlier's weight becomes indistinguishable (from a machine precision perspective) from 0. The story can be more complicated due to the fact that outliers can mask each other. One large positive label outlier can make a moderate positive label outlier seem like it isn't an outlier at first, until the larger outlier has sufficiently reduced weight (after a few iterations). Another case is where a large feature outlier in one direction makes a moderate feature outlier in the opposite direction seem like more of an outlier than it really is until the weight in the first outlier has been reduced. A number of iterations can be necessary therefore for the weights to resolve themselves.

3.6.3.7 Interpreting d_m

The quantity d_m is a measure of how outlier-like a point is. The left factor $\frac{\sqrt{X^{m\intercal}RX^m}}{1-v_m X^{m\intercal}RX^m}$ measures how much of a feature outlier that point is. It utilizes $X^{m\intercal}RX^m$, which (when the features are mean centered) serves the purpose of a multi-dimensional z-score, measuring how far the point X^m is from the other points. The denominator $(1 - v_m X^{m\intercal}RX^m)$ causes our measurement of how outlier-like a feature outlier is to spike up to infinity as it overwhelms the effects of all other points. This is desirable because it means that as a feature outlier approaches being maximally problematic, its score for how outlier-like it is approaches infinity.

Looking now at the right factor of the equation for d_m, we see that it measures how much of a label outlier the point in question is by looking at how much we mispredict that point. Putting both factors together, d_m measures how much of an outlier the point is capturing

both the notion of a feature outlier and a label outlier. Note that if a point is not at all a label outlier (i.e. prediction error is 0) then d_m is 0, which makes sense because such points do not change our solution coefficients at all when dropped. Similarly, if a point is not at all a feature outlier (i.e. $X^{m\intercal}RX^m = 0$, where we have no constant term, so our hyperplane must pass through the origin) then again $d_m = 0$ as expected, because here too dropping the point can't effect our solution coefficients.

3.6.3.8 Interpreting α

There are at least three valid interpretations for the parameter α in g_α.

1. The parameter α reflects how aggressively we reduce the weight of outliers. A priori, there is no exact dividing line between outliers and non-outliers. By setting α via stabilized k-fold cross-validation we are learning from the data the magnitude at which outliers start having a detrimental impact on prediction performance. When $\alpha = 0$ we do not penalize outliers at all, but as α grows we assign progressively less weight to even fairly innocuous points. Generally, we will not select a reweighting function that separates points starkly into outliers and non-outliers by assigning 0 weight and maximal weight respectively. But if we were to make such a choice, α would be the determinant of how many points are deemed outliers.

2. Additionally, α reflects a tradeoff between bias and variance. The lager the value of α, the more data we are ignoring or at least partially ignoring. Inevitably, some of this data may have been useful, and so we are potentially biasing our solution. But the data that we are ignoring is the most extreme data, which had a large impact on our solution coefficients. So by throwing this data away, we reduce the variance of our algorithm's learning process. When $\alpha = 0$ we minimize bias but accept the possibility of huge variance (stemming from outliers), and as α grows we reduce our variance by

reducing the effect of those points that are most extreme.

3. Finally, α controls the maximum size of A_i and B_i for each data point, (X^i, y_i). A_i reflects the change in solution coefficients caused by dropping (or equivalently, by adding) the ith point that is directly due to the label and feature vector that point. On the other hand, B_i accounts for the change in solution coefficients caused by changes in weights that occur when this point is dropped. For $\alpha = 0$ we weight our points uniformly, which forces B_i to be 0, because dropping the point causes no change in our weights beyond normalization. But this choice $\alpha = 0$ means that A_m can grow unboundedly, making the algorithm highly sensitive to outliers. On the other hand, as α gets larger we constrain A_i (in fact, A_i falls like $\frac{1}{\alpha}$), but now B_i can become larger since our weights become more sensitive to the data.

We note that α is quite separate in its effects from the standard free parameter λ in Ridge Regression, and hence supplements it rather than replacing it. The parameter λ acts on the features, helping us handle cases where there are too many or they are highly correlated, whereas α acts on the points, helping us handle cases where there are extreme points that could impact our prediction performance. While both λ and α make the algorithm more "robust" in an intuitive sense, they provide very different types of robustness that protect us from different problems.

3.7 Empirical Results

We now consider applying the Stabilized Ridge Regression algorithm to empirical data, and compare its out of sample accuracy to that of Ridge Regression and Ordinary Least Squares on synthetic examples where outliers are present. For Stabilized Ridge Regression we use $\lambda = 0$ since in the synthetic cases being used the features of the training data are uncor-

related (so no λ regularization is needed), and we use use 10-fold stabilized cross-validation to select α. For Ridge Regression we do what is typically done, using 10-fold cross-validation to select λ. For Stabilized Ridge Regression our k-fold stabilized cross-validation tries forty values of α growing exponentially (by a factor of about 2) from 1E-6 to 1E6, and for Ridge Regression we have the k-fold cross-validation try these same forty values for λ .

Here, we consider two types of outliers: label outliers, where an unreasonably large label is assigned to one of the pre-existing features, and feature outliers, where a feature of unreasonably large magnitude is assigned to one of the pre-existing labels. In each of the figures we use 250 training points and 50 features. The features are generated from an isotropic multi-variate normal distribution, and the labels are a random linear combination of the features plus noise. We calculate out-of-sample performance on a test set of 10,000 points that are withheld from training. We use $R^2 \equiv 1 - \frac{err}{var}$ as our measure of performance, where err is the average squared prediction error, and var is the variance of the set set labels. With this choice 1 is the best possible performance and 0 means that our accuracy is only as good as predicting the average test set label for all points (i.e. it is the performance of the best possible constant predictor). By this definition, a negative R^2 implies performing worse than the best constant predictor. We can interpret this performance measuring as the fraction of the label variance that our prediction method has accounted for.

In the first plot in Figure 3.2, we consider what happens when one point in the training set has its label replaced with an outlier of increasingly large magnitude. As this magnitude increases, Stabilized Ridge Regression is essentially unaffected, whereas both Ridge Regression and Ordinary Least Squares have their performance completely destroyed. Ridge regression outperforms Ordinary Least Squares slightly in this case, but only in

regions where both perform terribly, so this outperformance has little benefit. However, Ridge Regression at least prevents the R^2 from going negative, unlike Ordinary Least Squares.

In the second plot in Figure 3.2, we consider what happens when one label in the training set has its feature replaced with a feature outlier of increasing magnitude. For simplicity we choose the outlier feature to be a vector where every element has the same value. As the magnitude of this vector increases, Stabilized Ridge Regression is again essentially unaffected, whereas Ridge Regression has its performance destroyed rapidly. Ordinary least squares loses a significant amount of accuracy, but performs much better than Ridge Regression in this case. As we have discussed, this is due to Ridge Regression's use of k-fold cross-validation, which is itself not robust to outliers.

In the third plot of Figure 3.2, we consider what happens when many label outliers of moderate size are including in the training set by replacing the labels of a number of training points with numbers generated uniformly at random with moderately large variance compared to the natural variance of the labels. As the number of such outliers increases, we have the expected effect: the performance of all algorithms in negatively impacted. But Stabilized Ridge regression has much slower decline in performance than Ridge Regression and Ordinary Least Squares.

These three plots in Figure 3.2 tell the same fundamental story. Not only is Stabilized Ridge Regression impervious to the impact of outliers in all three cases, but Ridge Regression and Ordinary Least Squares have their prediction performance severely impacted by outliers. It will surprise some to see that Ridge Regression does not perform significantly better in the presence of outliers than Ordinary Least Squares in these cases, and in one of these cases it actually performs worse. One might think that the flexibility to choose λ would help Ridge

Regression be significantly more robust in the presence of outliers. But the fact that k-fold cross-validation is generally used to select the λ parameter presents a unique problem when outliers are present. As we have mentioned, the usual k-fold cross-validation is itself not robust to outliers, causing Ridge Regression to do a poor job of selecting λ. For instance, if there is one extreme outlier, the λ parameters will mainly be selected to minimize the error on that single outliers in the one k-fold cross-validation fold where that point is withheld. Hence, λ will depend almost entirely on the outlier, not on the rest of the training data.

3.7.1 Case Study

As a case study, we demonstrate the result of applying Stabilized Ridge Regression to the Computer Hardware data set (Ein-Dor and Feldmesser, 1987) from the UCI Machine Learning Repository (Lichman, 2013). This data set reflects characteristics of different computer CPU's, with the goal of predicting the relative performance of each processor. Outliers are most problematic in smaller data sets, since then each point has a greater influence on the final predictions. And the Computer Hardware data set is indeed small, with 209 points and 6 features to use for prediction (called MYCT, MMIN, MMAX, CACH, CHMIN and CHMAX). The goal is to predict the label PRP. As before, we measure prediction performance using R^2. We use Stabilized Ridge Regression to train on 80% of the data (selected one time at random), and test on the remaining 20%. We find that the algorithm assigns all points essentially equal weights (of about 1.01), except for three points. The point with the most extreme label is modestly underweighted, with a weight of 0.60. The algorithm considered this point to be only somewhat outlier-like, despite its large label. The point with the second most extreme label is assigned a much lower weight, 0.002, because it appears to have elements of both a label and a feature outlier, and is considered sufficiently outlier-like overall to reduce its weight dramatically. Finally, the point with the third most extreme label is assigned a weight

of 0.02. On the out-of-sample data Stabilized Ridge Regression achieves an R^2 of 0.87 which compares positively to Ridge Regression and Ordinary Least Squares, which both achieve an R^2 of only 0.73.

3.8 Conclusion

We showed that Ridge Regression and n-fold cross-validation perform poorly in the presence of outliers. We introduced a new algorithm, Stabilized Ridge Regression, which modifies Ridge Regression in order to make it inherently robust to outliers. This new algorithm performs similarly to Ridge Regression in the absence of outliers, but it automatically reduces the weight of outliers when they are present. This approach to stabilizing an algorithm (iterative reweighting with weights based on the change in solution coefficients when a point is dropped, which are passed through a parameterized weight function) can be generalized to other cases beyond ridge regression, which we intend to explore in subsequent work.

3.9 Python Code

```
import numpy as np
from numpy import multiply as mult #element-wise matrix multiplication operator


def stabilized(X, y, alpha, lambd=0, maxIters=50, weightDeg=6, prec=1E-8, singMin=1E-12):
        """Returns the column matrix of solution coefficients for stabilized regression.
        X : the n x m matrix of training points with 1 point per column, 1 ft. per row
        y : the m x 1 matrix of training labels with one point label per row
        alpha >= 0 : how strongly to penalize outliers (alpha=0 gives ridge reg.)
        lambd >= 0 : how strongly to penalize large coeffs (ridge reg. regularization)
```

```
maxIters > 1 : the maximum number of iterations before terminating
weightDeg > 1 : the point weights will vi = 1 / (1 + (alpha * zi)^weightDeg)
prec > 0 : will halt if root mean square change in coeffs is less than this
singMin > 0 : when solving lst. squares problems singular values are set to 0
if their ratio to the largest singular value is less than singMin
"""
X = np.matrix(X) #make the feature matrix into a matrix object
y = np.matrix(y) #make the label vector into a column matrix object
if len(y.T) != 1: raise RuntimeError("Label vector y needs exactly one column!")
if len(X.T) != len(y): raise RuntimeError("y row count != X col. count!")

numFeatures = len(X) #assume feature mat. has 1 point per col. and ft. per row
numPoints = len(y) #assume label mat. is a col. vector with 1 pt. label per row
v = np.matrix(np.ones((numPoints, 1))) #the weights we assign to each point
lastc = None #solution coefficients from the last iteration
regularizer = lambd * np.eye(numFeatures) #regularizing mat. lambda*I of ridge reg.
ones = np.matrix(np.ones((numFeatures, 1))) #column vector of all 1's

for iterationNum in xrange(maxIters):
        RX = np.linalg.lstsq(X * mult(v, X.T) + regularizer, X, rcond=singMin)[0]
        c = RX * mult(v, y) #solution coefficients using the current weights

        if (lastc is not None) and np.sqrt(np.mean(np.square(c-lastc))) < prec:
                return c

        absE = np.abs(X.T * c - y) #abs. value of the label prediction errors
        d = (ones.T * mult(X, RX)).T  #di = [X^i' R X^i]_i
        weightedDRatio = np.sqrt(d)/(1.0 - mult(v, d)) #sqrt(di) / (1 - vi di)
        v = 1.0 / (1.0 + np.power(alpha*mult(weightedDRatio, absE), weightDeg))
        v *= numPoints/float(np.sum(v)) #normalize weights to sum to num. of pts.
        #stop if > half of pts. have weight < 0.5
```

```
            if np.sum(v>0.50) < numPoints*0.50:
                    if lastc != None: return lastc #if have a valid solution
                    else: return c #otherwise, return what we have

            lastc = c #save the last solution coefficients

    return c
```

3.10 Acknowledgements

We thank the UCI Machine Learning Repository for the Computer Hardware data set used in our experiments.

3.11 Chapter Notes

Lemma 2. *For any column vector b, any positive definite matrix P, and any real number σ, we have:*

$$\sigma b^\intercal (P + \sigma bb^\intercal)^{-1} b = \frac{\sigma b^\intercal P^{-1} b}{1 + \sigma b^\intercal P^{-1} b}.$$

Proof. By the Sherman-Morrison formula (for the inverse of a rank-1 update to a matrix), we have

$$(M + uv^\intercal)^{-1} = M^{-1} - \frac{M^{-1} uv^\intercal M^{-1}}{1 + v^\intercal M^{-1} u}$$

so long as $1 + v^\intercal M^{-1} u \neq 0$. This implies that for any column vector q:

$$q^\intercal (I + qq^\intercal)^{-1} q = q^\intercal \Big(I - \frac{qq^\intercal}{1 + q^\intercal q} \Big) q = q^\intercal q - \frac{q^\intercal q q^\intercal q}{1 + q^\intercal q} = \frac{q^\intercal q (1 + q^\intercal q) - q^\intercal q q^\intercal q}{1 + q^\intercal q} = \frac{q^\intercal q}{1 + q^\intercal q}.$$

Now, since our matrix P is positive definite, it has a unique positive definite square root $P^{1/2}$ with $P^{1/2}P^{1/2} = P$. If we let $q \equiv P^{-1/2}b\sqrt{\sigma}$, then applying the formula above, we can write:

$$\begin{aligned}
\sigma b^\intercal (P + \sigma bb^\intercal)^{-1} b &= \sigma b^\intercal (P^{1/2}(I + \sigma P^{-1/2} bb^\intercal P^{-1/2})P^{1/2})^{-1} b \\
&= \sqrt{\sigma} b^\intercal P^{-1/2} (I + P^{-1/2} b\sqrt{\sigma}\sqrt{\sigma} b^\intercal P^{-1/2})^{-1} P^{-1/2} b\sqrt{\sigma} \\
&= \frac{\sqrt{\sigma} b^\intercal P^{-1/2} P^{-1/2} b\sqrt{\sigma}}{1 + \sqrt{\sigma} b^\intercal P^{-1/2} P^{-1/2} b\sqrt{\sigma}} \\
&= \frac{\sigma b^\intercal P^{-1} b}{1 + \sigma b^\intercal P^{-1} b},
\end{aligned}$$

which completes the proof. □

Since the right-hand side of the identity of Lemma 2 is always less than 1, the following holds.

Corollary 1. *For any vector b, any positive definite matrix P, and any positive number $\sigma > 0$, we have:*

$$\sigma b^\intercal (P + \sigma bb^\intercal)^{-1} b < 1.$$

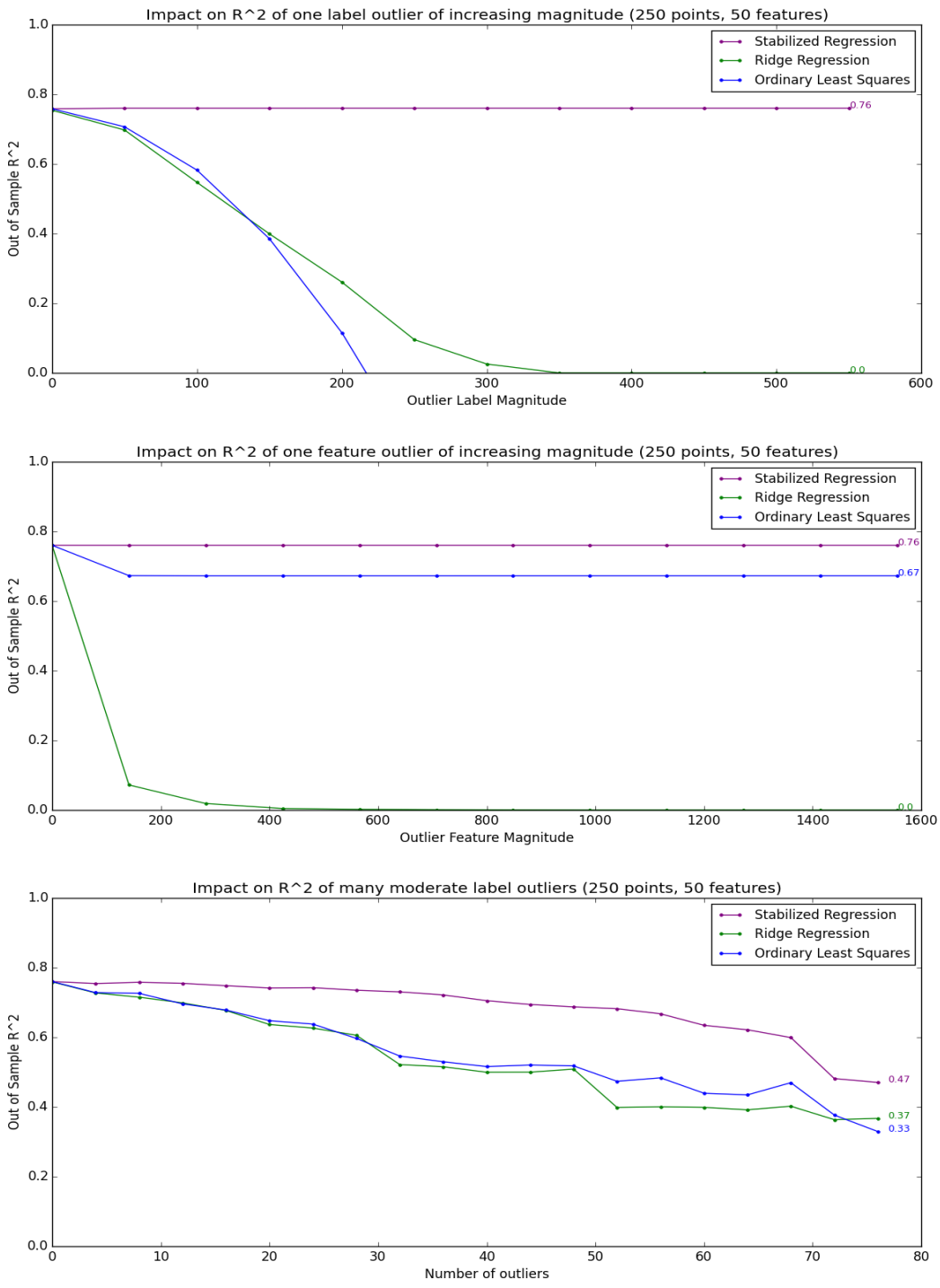

Figure 3.2: How does Stabilized Ridge Regression perform when there are outliers? We examine how the R^2 is affected by a single very large label outlier (top figure), a single very large feature outlier (middle figure) and many moderate sized label outliers (bottom figure). These results are compared to the those of Ordinary Least Squares, and of Ridge Regression.

Chapter 4

Safe Statistics

In Chapter 3 we developed an approach to eliminating the negative effects of outliers in the context of Ridge Regression. In this chapter, we consider the more generic problem of detecting outliers in an arbitrary vector of real numbers, and how to calculate common statistics without having to worry about them being heavily distorted by outliers. Solutions to this problem can also be applied in a machine learning context, for instance to remove or clip outliers in the training labels, or to pre-process each feature in advance of learning. Hence, the solutions proposed here could be helpful in the outlier examples that we used to break Ridge Regression in Chapter 1.

In some ways, the univariate outlier detection that we address here is harder than that of Chapter 3. Since that was in the context of a supervised learning problem, we had an objective function that we hoped to minimize, and we were able to use that objective function to set the strength of outlier penalization. In the setting we consider in this chapter though there is no clear objective function that we can leverage. On the other hand, in some ways the setting analyzed in this chapter is significantly easier, since we consider only univariate data, which removes the possibility of feature outliers and avoids complicated questions about

the right way to measure the distance between points. In univariate data, our outliers are expected to be greater than or less than all the other values. The real trick is in figuring out how large or small a value needs to be such that we should declare it an outlier, and then deciding how to handle the values that we claim are outliers.

4.1 Introduction

Suppose that we are given a vector v of m real numbers, with each number drawn independently from some fixed distribution. We would like to compute statistics on v, for instance to estimate the mean or standard deviation of the distribution from which its values were drawn, or to use v as the training labels for a machine learning algorithm. But how can we do so safely if extreme outliers may be present? It could be that human error, measurement error, or data corruption has altered some of these values or inserted extra irrelevant ones, so that what we have access to does not truly come from the distribution of interest. Or it could be that our data really does come from the distribution of interest, but bad luck has simply caused an unusually extreme value that could heavily distort our calculations. In this chapter, we introduce two simple algorithms for detecting and handling outliers in univariate data for situations like these. These algorithms are designed so that, once applied, statistics can be calculated safely, without the fear of extreme outliers corrupting the results.

The first algorithm we propose, MAD Removal, is appropriate when we have strong reasons to think that our data should have been drawn from a specific probability distribution, but data corruption or other errors may have caused some extreme values that do not fit this distribution, and furthermore, the location and scale parameters of the theoretical probability distribution are not necessarily known. For instance, we may have strong reason to believe our data should come from a normal distribution, but human error may have corrupted

some of our values, and we do not know the mean and variance of the normal distribution in advance. What makes such scenarios tricky is that we must robustly estimate the location and scale parameters from values that may contain extreme outliers, while making sure that the interpretation of the resulting algorithm does not depend on the sample size, even though the reliability of our location and scale parameter estimates will inevitably be sample size dependent. In this context we can think of an "outlier" as a value that is far out in the tail of the known distribution, where there is a very low probability of finding a value more extreme than that.

The second algorithm we propose, Impact Clipping, does not require any known distribution, and is for cases where we would like to protect ourselves from any one value in our data impacting a statistical calculation too much. For instance, we may have a set of values for which we need to compute the standard deviation, but we may be concerned that extreme outliers could lead to just one or two values dominating the standard deviation calculation, making it unreliable. What makes scenarios like this tricky is that we cannot simply look at the sensitivity of the statistic of interest to each value, since with such an analysis one outlier can mask the existence of another, or cause a non-outlier to seem like an outlier. In this context we can think of an "outlier" as a value which, when left unaltered, will have an excessive influence on the calculation of our statistic of interest, regardless of the presence or absence of other extreme outliers.

For both algorithms, it is extreme values that are of concern, because values near the center of the distribution, even if erroneous, are unlikely to cause significant issues for nearly any application, unless appearing in very large numbers.

We will begin the chapter by discussing some common causes of outliers, followed by some

existing outlier handling methods from the literature, and then briefly introducing each of our two proposed algorithms. We will then discuss in Section 4.4 how they excel in two substantially different settings for handling outliers, followed by a discussion in Section 4.5 of how big an outlier needs to be to distort the sample mean and standard deviation. After that, in Section 4.6 we will introduce a list of traits that we would ideally like any outlier handling algorithm to satisfy, which these two algorithms were designed to do. And finally, in Sections 4.7 and 4.8 we will finish the chapter by discussing MAD Removal and Impact Clipping in greater detail. All of the Python code from this chapter, including implementations of MAD Removal and Impact Clipping can be downloaded at the URL found in the footnote below[1].

4.2 Causes of Outliers

Outliers, which we can informally define as "data points that are unlike the rest of our data", can be caused by many different phenomena. Common causes include the following.

1. **Human input error**, for instance someone reversing two digits or making a typo during data entry.

2. **Machine error**, such as a malfunctioning measurement device or a corrupted data file.

3. **Accidental inclusion of wrong data**, such as when data from one source is mistakenly mixed with data from another source.

4. **Bad luck**, when a very unlikely extreme event really occurs.

5. **Heavy tailed distributions**, such as the distribution of company sizes in the United States.

[1] http://www.spencergreenberg.com/code/safe_statistics_code.zip

For the purposes of this chapter, we do not need to detect which sort of outlier has occurred in our data. In the case of MAD Removal, we simply wish to remove data that is unlikely to have come from the true distribution that we expect, and in the case of Impact Clipping, we simply want to protect statistics that we calculate from being excessively distorted by any single extreme value.

4.3 Some Techniques in the Literature

Many outlier handling techniques exist, ranging from the simple to the complex. Some techniques involve visualizing the data, such as the popular box plot method (Laurikkala et al., 2000), which visually displays the median, lower quartile, upper quartile, largest value less than the upper quartile + 1.5 IQR, lowest value greater than the lower quartile - 1.5 IQR, and all individual values that fall outside of these boundaries, where IQR stands for the interquartile range. When a researcher has the time to visually inspect the data, and enough domain knowledge to make the call about what is and is not an outlier, this can be a useful method for outlier removal. But when automation is required, or sufficient domain knowledge is not available to make such a determination in a non-subjective fashion, visualization techniques may be undesirable. Other classic methods, including Dixon's Q test (Dean and Dixon, 1951) and Grubbs' test (Grubbs, 1950), have problems being applied or interpreted in the presence of multiple outliers, as we will discuss later on. Some other common methods do not apply in the unsupervised context that is the focus of this chapter, since they required a training set for our algorithm to learn from (in which previous outliers or pervious non-outliers have been labeled as such), the so called "type 2" and "type 3" methods in the categorization of Hodge and Austin (2004). Still other techniques, which are designed for multi-dimensional data, make little sense to apply to univariate data, such as those based on the convex hull, factor analysis, or k-means clustering. The convex hull of

one dimensional data is useless, factor analysis has no effect on one dimensional data, and k-means clustering is more suitable for finding inner regions in univariate data that have low probability, rather than determining which extreme values are too extreme. But only extreme values are important here, since they are the only values that are likely to mess up analysis performed on one dimensional data. Another approach, "Distance-based Outliers" as defined by Knorr et al. (2000b), asks whether at least a specified fraction of points in our data lie at least a specified distance away from each point under consideration. In the case of univariate data however, this approach may be a step in the wrong direction, since all we really need to do is discover cutoffs to determine what values are too extreme, yet this approach introduces two new parameters (the fraction of data to be considered, and the distance threshold) that seem as difficult to determine as cutoffs are if not more so. A similar issue occurs when we define outliers as the n points that have the greatest average (or maximal) distance to their k nearest neighbors (Niu et al., 2011), as the hard work is then in setting n and k, which may be as hard or harder to set than a one dimensional outlier cutoff value, which we focus on in this chapter.

4.3.1 Introduction to MAD Removal

The first algorithm we propose, "MAD Removal", is appropriate when we have strong reason to believe that our data should have been drawn (at least approximately) from a specific probability distribution, but that some fraction of our values may be erroneous or may have been corrupted. The location (e.g. mean) and scale (e.g. standard deviation) of this a priori distribution do not have to be known to apply MAD Removal.

The purpose of the MAD Removal algorithm is to ensure that any erroneous or corrupted values do not substantially distort the conclusions we draw from the data. It works simply

by removing any values that lie too far from the center, but using a very robust approach to determine what "center" and "too far" mean. Specifically, it removes values that are too many median absolute deviations (i.e. MADs, as defined in Equation 4.1) from the median, and the cutoff threshold is chosen so that, for our known a priori distribution, only about a fraction p of values will be removed on average. By setting p to a small value, for instance $p = 0.001$, we ensure that applying MAD Removal is safe, in the sense that if our data has no corruption and simply comes from the expected distribution, there is virtually no cost to applying the algorithm. On the other hand, if there really are erroneous values or corruptions in the data, the MAD Removal can save us from drawing wrong conclusions based on extreme outliers. Note that we are only concerned here about extreme values, because although non-extreme values may also be the result of data corruption or errors, unless they are extremely common such values will have little impact on relevant statistics that we calculate from the data. Note also that the median and MAD here are particularly appealing statistics when outliers may be present. They each have a breakdown point of 50% (Rousseeuw and Croux, 1993), which is the highest breakdown possible, and which is twice the breakdown of the interquartile range.

In this chapter, we focus on applying MAD Removal to the case where the data is known to have approximately a normal distribution (except for possible data corruptions), since that is am extremely common case of interest, and because the central limit theorem implies that variables that are linear combinations of nearly independent other variables will tend to produce gaussian distributed data. But in principle the same techniques apply to a very wide range of possible distributions. The only aspect of the algorithm that hinges on the distribution is the function that maps sample size into an appropriate cutoff value (i.e. how many MADs away from the median we consider an outlier) such that only a fraction p of values will be removed, on average, when the values really are all drawn independently from that theoretical distribution. In Algorithm 3 we explain our empirical approach for

fitting this function. A significant benefit of fitting this function empirically is that the method can then be applied to non-normal distributions easily, plus it avoids potentially very complicated theoretical analysis of probability distributions. For random variables $X_1, \ldots, X_m$ whose values are drawn independently from some distribution, the quantity $(X_i - \text{median}_k(X_k))/\text{mad}_k(X_k)$ can be tricky to study analytically.

4.3.2 Introduction to Impact Clipping

The second algorithm that we introduce in this chapter, "Impact Clipping", is appropriate for situations where we have no a priori knowledge of what the distribution that generated our data is, but we are concerned that extreme outliers could corrupt our statistical calculations. This is an especially worrisome issue when sample sizes are small, or when values come from a poorly understood source.

The purpose of the Impact Clipping algorithm is to make sure that our statistic of interest are not impacted too greatly by any one extreme value in the data. It is premised on the idea that if the decision of whether or not to clip (i.e. limit the magnitude of) a single value in our data makes a large difference to our estimate of a statistic, then that value is risky to include unaltered as it is likely to greatly increases the variance in our estimation process. Therefore, its magnitude should be clipped.

The Impact Clipping algorithm starts by calculating an initial safe and conservative estimate of the statistic of interest, as well as a "safe" mean estimate. It does so by sorting the values and then clipping (i.e. winsorizing) any value that is not within the middle $\mathcal{F}$ fraction

of values. So values smaller than all those in the middle $\mathcal{F}$ fraction are set to the smallest of the middle values, and values greater than all those in the middle $\mathcal{F}$ fraction are set to the largest of the middle values. The algorithm then examines the non-middle values one by one, starting with the closest to the (safely estimated) mean. For each, it considers how much the current estimate to the statistic of interest would be altered if, instead of clipping that value, we include that value in the middle set of unclipped values. If the statistic of interest changes by more than a fraction $\mathcal{I}$, then we leave that value clipped and stop trying to include values from that side (i.e. either the left of the sorted vector or the right), since the other values in that direction are even more extreme. If the statistic of interest does not rise by more than fraction $\mathcal{I}$ then we unclip the current value being considered, add it to the set of middle (i.e. unclipped values), and adjust our estimate of the mean and standard deviation accordingly.

In this chapter, we use the standard deviation as our statistic of interest for Impact Clipping. This is a good choice for many purposes, since it is the least robust statistic that is commonly calculated (and therefore will be conservative when the statistic of interest is a somewhat more robust statistic like the mean). Additionally, the standard deviation can be updated in constant time from a previously known value when one value in our data goes from being clipped to being unclipped, which is desirable from an efficiency standpoint.

4.4 Two Settings for Handling Outliers

The appropriate way to handle outliers depends on context. In this chapter we consider two different contexts for outlier handling, both of which arise frequently.

4.4.1 Case 1: Known Distribution With Possible Corruption.

We may, for theoretical reasons or based on passed experience, have good reason to believe that our values were drawn from a distribution that approximately matches some known distribution, yet still have to consider the possibility that some of the values are erroneous or have been corrupted. For instance, a psychologist may have good reason to believe that the amount of a certain behavior is normally distributed in the population, but have to contend with the possibility that some participants in their study mistakenly entered unrealistically high values. Or to give another example, a physicist may have theoretical reasons to believe that a variable that is measured electronically should follow an exponential distribution, but be wary of occasional machine failures that yield very large measurement error.

It is in cases like these cases that our MAD Removal algorithm applies. As mentioned, in this chapter we focus on the case where the theoretical distribution is a normal distribution, but the same process will apply to a wide range of distributions, so long as they are parameterized only by a parameter of location (like the mean) and a parameter of scale (like the variance).

4.4.2 Case 2: Unknown Distribution With Possible Extreme Values.

In some cases we do not have a priori knowledge of what distribution our data will be generated by. However, if our use for the vector v is such that an extreme outlier could cause high variance in our estimates or inaccurate results, we may want to reduce outliers. For instance, a financial analyst may want to estimate the standard deviation of transaction prices, but worry that there could be a few extreme prices that dominate the calculation. Or an economist may be performing regression analysis, and be concerned that the prediction

variables (i.e. training labels) may have extreme values that would distort the solution coefficients. For cases like these where no known theoretical distribution can be assumed, and where the main concern is preventing extreme values from having too great an influence on our calculations, we recommend outlier clipping (i.e. winsorizing) rather than outlier removal. One advantage of outlier clipping is that it create less bias than outlier removal. But additionally, in this case we cannot be confident that extreme values do not truly come from the distribution of interest, since we do not know that distribution. All that we can say is that extreme values increase the variance of our statistical calculations, and may cause them to be unreliable. Hence removing values completely may be discarding genuine value, whereas clipping is a more moderate approach that simply limits the influence of such values.

It is this sort of situation that Impact Clipping algorithm was designed for. The algorithm is based on the fact that in these cases our main worry is that our statistics will be overly influenced by any one value. Hence, the algorithm is designed to clip extreme values in such a way that the impact any such extreme value can have on a statistic of interest (in terms of percentage change caused by including that value unclipped) is limited. As mentioned, we focus in this chapter on the standard deviation because it is generally the least robust statistic of common interest.

In the next section, we consider how large a value an outlier must be in order to create problems for some common statistics.

4.5 Outlier Impact on the Mean and Standard Deviation

The sample mean and standard deviation can be arbitrarily corrupted by a single sufficiently bad outlier. Let us consider how big an outlier value has to be to increase (i.e. distort) the mean and standard deviation by a given amount $d > 0$. Without loss of generality, we assume that the last value of our vector v, i.e. the mth value, is our outlier, and we denote the value of this outlier $z \equiv v_m$. Let μ_m be the sample mean calculated using all m values (i.e. including the outlier value z), and let μ_{m-1} be the mean when calculated using just the first m-1 values (i.e. with the outlier omitted). Similarly, let σ_m be the sample standard deviation with all m values, and σ_{m-1} be the standard deviation when the outlier is omitted. Hence:

$$\mu_m = \frac{1}{m}\sum_{i=1}^{m} v_i \qquad \mu_{m-1} = \frac{1}{m-1}\sum_{i=1}^{m-1} v_i$$

$$\sigma_m = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(v_i - \mu_m)^2} \qquad \sigma_{m-1} = \sqrt{\frac{1}{m-2}\sum_{i=1}^{m-1}(v_i - \mu_{m-1})^2}$$

We now would like to figure out how big z will have to be to cause different distortion fractions, d, to each statistic. For the sample mean, we therefore would like to solve for the outlier value z such that:

$$\frac{\mu_m}{\mu_{m-1}} = 1 + d$$

Solving for z in terms of μ_{m-1} yields

$$z = \mu_{m-1} + m\, d\, \mu_{m-1}$$

so the size of the outlier depends linearly on the mean of the non-outlier values, the vector size m, and the distortion factor d. In particular, as m gets larger, the size the outlier needs

to be to distort the mean a fixed amount grows proportionally. So for example, using data which (without the outlier value) has mean $\mu_{m-1} = 1$, the mean will be distorted by 20% when $z = 1 + 0.2m$.

For the standard deviation, we similarly want to solve for the outlier value $z > \mu_{m-1}$ such that

$$\frac{\sigma_m}{\sigma_{m-1}} = 1 + d$$

which gives us

$$z = \mu_{m-1} + \sqrt{\frac{m}{m-1}}\sqrt{1 + (m-1)\left((d+1)^2 - 1\right)}\sigma_{m-1},$$

which grows linearly in μ_{m-1} and σ_{m-1}. For m large, we can approximate this formula as:

$$z \approx \mu_{m-1} + \sqrt{m}\sqrt{d(2+d)}\ \sigma_{m-1}.$$

So for instance, if the data without the outlier has a mean $\mu_{m-1} = 0$ and a standard deviation $\sigma_{m-1} = 1$ then the outlier distorts the standard deviation by 20% when $z \approx 0.66\sqrt{m}$. Comparing this $\sqrt{m}$ dependence to the m dependence that we witnessed for the mean makes it clear why the standard deviation is much less robust to outliers. For m reasonably big, the outlier value will need to be much larger to corrupt the mean than to corrupt the standard deviation.

4.6 Desired Traits for Outlier Detection Algorithms

We now propose a list of traits that it is usually desirable for any method of handling outliers (in univariate data) to satisfy. MAD Removal and Impact Clipping are designed to

have these traits.

1. **Has the ability to handle multiple outliers**. Since in real data we cannot be confident that there will be at most one outlier, the ability to detect just a single outlier is usually of limited usefulness. The classical Dixon's Q test (Dean and Dixon, 1951), based on an assumption of normally distributed data, which involves subtracting the value of a possible outlier to the sample value closest to it and dividing by the range of the data, is an example of a simple method for detecting outliers that is only designed to detect one outlier per data set.

2. **Causes little bias / loss of efficiency on data from "nicely behaved" distributions**. If a distribution is nicely behaved, we expect it to very rarely have outliers worth worrying about. In such cases, we ideally want any outlier handling algorithm applied to such data to cause very little bias or loss of efficiency on average. Thanks to the central limit theorem, sums of independent or nearly independent random variables tend to produce data that is normally distributed, so normally distributed data occurs commonly in practice. Furthermore, normally distributed data has "thin tails", meaning that it is very unlikely to produce extreme outliers by chance alone. It is therefore the quintessential case of a random process that is essentially free of extreme outliers, and so for most purposes serves as a good model of a "nicely behaved" distribution that we expect a good outlier handling method to affect very little. When this is the case, we are safe applying the method to data that we have strong reasons to think should be normally distributed, as there is little cost to doing so if it really is normally distributed as we expect, but if there are extreme outliers due to errors or data corruption, the outlier handling algorithm should help. As an example of an algorithm that violates this mandate, consider winsorization. As usually applied, it means that for some percentile q, the values below the qth percentile are set to the qth percentile,

and the values above the (100-q)th percentile are set to the (100-q)th percentile. If we apply this process to independent, normally distributed values when the sample size is large then we underestimate the standard deviation by about 9% for $q = 5\%$, 18% for $q = 10\%$, and 36% for $q = 20\%$. Even more distortion occurs if, instead of winsorizing, we simply remove values below the qth percentile and above the $(100 - q)th$ percentile. In that case, we underestimate the standard deviation by about 21% for $q = 5\%$, 34% for $q = 10\%$, and 54% for $q = 20\%$. Any algorithm that removes or clips extreme values will have a bias towards reducing the standard deviation, but this level of bias is unacceptably large for many applications. The main exception, where we would not care whether our outlier handling algorithm causes significant bias or a substantive loss of statistical efficiency on normally distributed data, is when we have strong reasons to believe that our data was generated by some specific other distribution. In such cases, it is the performance on that specific distribution that is of interest.

3. **Uses an outlier criteria which does not depend on the extremity of more extreme outliers.** When we are trying to tell whether one particular point is an outlier, our answer should not hinge on the level of extremeness of another, even more extreme outlier. Otherwise, our decision procedure for determining what is an outlier is not itself robust to outliers! When the criteria for detecting an outlier is impacted by the extremity of other outliers, it leads to the two very unappealing phenomena of "masking", where one large outlier hides another, and "swamping", where the presence of an outlier will make a non-outlier seem like an outlier. For an example of masking, suppose that our outlier handling criteria is to simply remove every value that is more than 4 standard deviations away from the mean of the values. This presents a problem because the standard deviation calculation is highly sensitive to the extremeness of any outliers that our present. Therefore the presence of one very extreme outlier may

prevent us from removing another, somewhat less extreme outlier, due to the first outlier making the standard deviation artificially high. To be specific, suppose we have 100 values that are generated from a standard normal distribution with mean 0 and standard deviation 1, and then we add two outliers, one with the value 100, and the other with the value 1000. Due to the presence of the outliers the mean and standard deviation end up being about 11 and 99, meaning that the smaller outlier value of 100 is only about 0.9 standard deviations from the mean, even though it would have close to a 0% chance of having been drawn from a standard normal distribution! In other words, the larger outlier makes the smaller outlier look like a non-outlier due to the masking effect.

4. **Has a meaningful interpretation for its outlier cutoffs.** Ideally, we should be able to meaningfully interpret the cutoffs used by any univariate outlier handling algorithm in its determination of what is and is not deemed an outlier. Having such an interpretation can help us conclude whether the outlier handling procedure is appropriate to use in our particular case. As an example of an outlier detection method that lacks a meaningful interpretation, consider Grubbs' test (Grubbs, 1950), which assumes that the values of interest were drawn from a normal distribution. It evaluates each potential outlier by performing a significance test at significance level α. It does so by using a random variable which is constructed to be the variance of the sample without that potential outlier included, divided by the variance of the sample when that potential outlier is included. This test can be applied for each extreme value in decreasing order of extremeness. Whenever a value fails the test, it can be removed from the sample and then the process can be repeated. Now, if the data were in fact normally distributed with exactly one potential outlier added, then the cutoff for the test would have precisely the intended interpretation, calculating the probability of

getting a change in variance at least this extreme by including the potential outlier compared to excluding it, under the null hypothesis that the remaining values were drawn from a normal distribution with no outliers. The problem is that when there are multiple outliers this intuitive interpretation fails, potentially in a large way, since the remaining data will not in that case be normally distributed. Hence, the cutoff for the test statistic does not have a meaningful interpretation in the presence multiple outliers, since its interpretation is based on an assumption that we know to be violated.

4.7 Details of the MAD Removal Algorithm

We now discuss the MAD Removal Algorithm in greater detail.

In some cases we have a strong theoretical reasons to believe that our data should be drawn from a particular distribution, but with the possibility of data errors or corruptions. In such a case, values lying much further from the center than is typical with that distribution are suspicious, and are likely to be the result of human error, measurement error, or data corruption, rather than truly having been drawn from the distribution of interest. It is precisely such a situation that the MAD Removal algorithm is designed for. For each value, this algorithm simply looks at how many median absolute deviations (i.e. MADs) that value lies from the median, where MADs are defined as:

$$\text{mad}(v) = \text{median}(|v - \text{median}(v)|). \tag{4.1}$$

with the absolute value and subtraction applied element-wise. Note that for the normal distribution, one MAD is equal to about 0.67 standard deviations. In the MAD Removal algorithm, any value that lies too many MADs from the median is removed. But how many

MADs is too many? We set the cutoff so that when applied repeatedly to vectors of m values drawn from our known theoretical distribution, only a fraction p of those values is removed, on average. Note that this is not quite the same as saying that each individual vector element has a probability p of being removed when drawn from the theoretical distribution, since we are looking at the average number of elements removed across many random draws of such vectors, not within each such vector.

The reason that we remove data in MAD Removal rather than clip (i.e. winsorize) it is because, due to our strong expectation about the true distribution that generated the data, and a small value for p (such as p=0.001), outliers have a substantial chance of being bad or corrupted data, rather than being regular values that are merely far from the center due to unlikely chance events.

If we have strong reasons to expect our chosen theoretical distribution, the MAD Removal algorithm can be applied safely, with virtually no cost. If the data actually was all drawn from this distribution, then the algorithm will scarcely alter the values at all, so statistics of interest will be essentially unaltered. On the other hand, if there are extreme outliers that are due to corruptions or mistakes, these will generally be removed by the algorithm. And, in fact, the MAD Removal algorithm will generally have little to no effect on data drawn from distributions that are even "nicer" than our theoretical distribution. For instance, if we use the normal distribution as our theoretical distribution, a very small fraction of values will be removed on average from data that accidentally comes from a uniform distribution or symmetric triangular distribution instead of a normal distribution. Hence, the algorithm will behave nicely even if our assumptions about the theoretical distribution are wrong, as long as they are wrong in the sense of us overestimating the distribution's likelihood of extreme values rather than underestimating it.

The MAD Removal algorithm is designed to satisfy all of our desired traits from Section 4.6 for an outlier detection algorithm. It easily handles multiple outliers. It has almost no bias and very high efficiency when applied to normally distributed data (when p is chosen to be a small number such as $p = 0.001$, and when the theoretical distribution is as nicely behaved or less nicely behaved than a normal distribution, as will usually be the case). Additionally, our determination of what is deemed an outlier is highly robust, because both the MAD and median are themselves highly robust, insensitive to the extremeness of all values except the least extreme 50% of the data. What is more, the MAD Removal algorithm has a meaningful interpretation for its cutoff value, which is that it is set to remove only a fraction p of values on average when the data comes from our theoretical distribution.

The idea of using MADs from the median to determine what is an outlier is of course not a new concept. However, typically static cutoffs have been recommended for what is deemed to be too far from the center, for instance 2.5 or 3.0 MADs from the median (Rousseeuw and Croux, 1993). Other recommendations include forming so-called "modified z-scores" for each value v_i, given by,

$$z_i \equiv 0.6745 \frac{|v_i - median|}{MAD}$$

where the constant in front is the MAD of a standard normal distribution, and then we discard values when these z-scores are bigger than 3.5 (Iglewicz and Hoaglin, 1993). But any static cutoff like this lacks a meaningful fixed interpretation, as the interpretation changes with both the sample size and the underlying distribution that we expect. The challenge we are faced with is setting the cutoff in a meaningful, sample size independent way.

Once we have assumed a particular theoretical distribution for our data (but with the possibility of data corruption or errors), it is non-trivial to construct a function that maps sample size into a cutoff such that a fraction p of values will be removed, on average, for values sampled independently from our theoretical distribution. We introduce a procedure for learning such a mapping, which we find through a process of numerical estimation and curve fitting. Details of this process are shown in Algorithm 3, under the assumption that the theoretical distribution is a normal distribution. However, the approach can easily be modified for a wide range of theoretical distributions, so long as the distribution has no unknowns other than a location parameter (such as the mean) and a scale parameter (such as the standard deviation) which the algorithm accounts for automatically. The only changes in the procedure that are needed are (a) randomly sampling from the theoretical distribution instead of from the normal distribution, and (b) setting the asymptotic constant α_p so that it corresponds to the number of MADs from the median one needs to include such that a fraction p of values from the theoretical distribution are included in the limit as the sample size goes to infinity. Once the appropriate map from sample size to a corresponding cutoff is fit, it can be used as in Algorithm 4 to apply the MAD Removal procedure.

Note that in the special case where our sample size is large, and the underlying probability distribution is a symmetric one, the MAD removal algorithm becomes very simple. It tells us to remove any value that is outside of the range $[median - \alpha_p * MAD, median + \alpha_p * MAD]$, where $\alpha_p = \frac{F^{-1}(1-\frac{p}{2})-F^{-1}(1/2)}{\frac{1}{2}(F^{-1}(3/4)-F^{-1}(1/4))}$, and $F^{-1}(p)$ is the inverse cumulative distribution function of our assumed distribution. When $p = 0.001$, and the underlying data is normally distributed, we get $\alpha_p \approx 4.87854$. Using this asymptotic approximation rather than the more reliable complete algorithm leads to underestimating the outlier cutoff by about 20% when $m = 42$, as well as 3% when $m = 220$, and 1% when $m = 600$. For $p = 0.01$ on the other hand, we have $\alpha_p \approx 3.81893$, and for $p = .0001$ we have $\alpha_p \approx 5.7682$.

Those accustomed to null hypothesis testing may be tempted to use $p = 0.05$, but this is not small enough, and defeats the purpose of this algorithm. The choice of p, which determines on average how many values will be removed when your data really is generated by your distribution of interest, needs to be set so that very little data is removed when no outliers are present. This is so that the outlier removal method is safe to use with very little cost, meaning that applying it when there are no outliers will have almost no impact on subsequent analyses performed. The choice $p = 0.05$ will remove 5% of data when there are no outliers, which is often too much (e.g. when subsequently calculating the standard deviation). We recommend $p = 0.001$. Smaller values of p can of course be chosen to be even more cautious, but there are diminishing returns for using smaller p's, and we would only recommend them when subsequent calculations that will be performed on the data are extremely sensitive to missing values, or if the theoretical distribution of interest has very heavy tails.

Once the MAD Removal algorithm has been applied, we can compute the statistics that we like on our vector of interest. For instance, we can compute the "MAD Mean", μ_{MAD}, which is just the sample mean of a vector after MAD Removal has been applied, or the "MAD Standard Deviation", σ_{MAD}, which is just the sample standard deviation after MAD Removal has been applied. These statistics are of course far more robust to outliers than their ordinary counterparts.

It is important to emphasize that the MAD Removal algorithm is only intended for cases where there are strong reasons to believe that the data was generated by our chosen theoretical distribution (plus possible data errors or corruptions), or another distribution that is even more nicely behaved than our chosen one. Only in such cases can values that are extreme

reasonably be viewed as errors or corruptions rather than simply legitimate values that lie far from the center. If such an assumption cannot be made, and the data in fact may be truly drawn from a thicker tailed distribution, we do not recommend applying MAD Removal. For instance, if we choose the normal distribution as our distribution of interest, then when MAD Removal is applied to a log-normal distribution, $5\% - 10\%$ of the largest values may be removed by the algorithm, which will substantially distort its subsequently estimated mean and standard deviation.

What makes the MAD Removal algorithm particularly unsuitable for situations where the expected distribution is unknown is that, in the limit as the same size goes to infinity, the algorithm will continue to produce a bias estimate of statistics like the standard deviation, even though in that limit an outlier (no matter how big) has no impact on the result. The reason is because MAD Removal is designed to remove data that is unlikely to have come from the distribution of interest, rather than to produce an unbiased estimate if the data comes from other, thicker tailed distributions. In cases where there is no strong reason to expect a particular distribution, the Impact Clipping algorithm is preferred.

In Figures 4.1 and 4.2 we plot the fraction of values that MAD Removal (using a normal distribution as the underlying theoretical distribution) ends up removing, on average, when applied to data drawn from various distributions, for three different choices of p. The straight horizontal lines achieved in the normal case at the top of Figure 4.1 shows precisely what we expect, that when applied to values that actually were drawn from a normal distribution, a fraction p of values are removed on average.

4.7.1 Implementing MAD Removal

To implement MAD Removal, we begin with a minimum and maximum vector size, $t_1 \geq 3$ and $t_2 >> t_1$ respectively, to be used in constructing the model (we recommend $t_1 = 4$ and $t_2 = 120$). We also need to select a number, $p > 0$, which reflects the fraction of the data that we desire the algorithm to remove when values are independent and drawn from a normal distribution (we recommend $p = 0.001$). Finally, we need to select and a number of samples, s, to use in our simulations for each size of vector (since we will be fitting a part of our the model empirically). We recommend $s = 90,000$ but more can be used for increased accuracy in the constructed model.

The details of how we approximate the map from the sample size into the appropriate outlier cutoff is shown in Algorithm 3, which focusses on the normal distribution case. Note that when curve fitting during this algorithm, we fit the cases where m is even separately from the cases where m is odd, since the median behaves quite differently in each (i.e. it averages the middle two values in the former, and selects just the lone middle value in the latter). That means that there will be two sets of numbers for a,b,c,d (see Algorithm 3) and the cutoff function $r(m) = f(m, a, b, c, d)$ will use one such set of a,b,c,d when m is even, and the other when m is odd. The resulting cutoffs from this curve fitting procedure, in comparison to the empirical cutoffs these curves are fit against, are show in Figure 4.3. The magnitude of the error of the fit curves versus the empirical values is shown in Figure 9.1

The cutoff function that we fit during this process is then applied in Algorithm 4. And the corresponding Python code for applying MAD Removal for a previously fit cutoff function is shown in Section 4.11. The implementation given has running time on the order of $m \log(m)$ since it relies on sorting the values. However, if a fast selection algorithm like Hoare's selection

algorithm is used to compute the median and MAD, it will have running on the order of just m.

4.7.2 Extensions of Mad Removal

In principle, the MAD Removal concept may be generalizable to higher dimensions, though doing so does present challenges. Rather than a vector of values v, in the multidimensional context we have a matrix V with one point V^k per column, and one feature V_j per row. We assume that each feature of each point is drawn independently from some known distribution, except with possible corruption or errors. Then, one potential approach for applying MAD Removal in this context is to define a robust alternative to the covariance matrix, for instance the matrix D where:

$$D_i^j = \sqrt{\text{median}\Big((V_i - \text{median}(V_i)) * (V_j - \text{median}(V_j))\Big)}$$

where * is the element-wise multiplication operator, and the differences are also taken element-wise. Note that this is a multi-dimensional generalization of the MAD, since

$$D_i^i = \sqrt{\text{median}\Big((V_i - \text{median}(V_i))^2\Big)} = \text{mad}(V_i).$$

Then, we can measure each point's natural distance to the center of the distribution using:

$$\text{dist}_i = \sqrt{(V^i - medians)^\intercal D^{-1}(V^i - medians)}.$$

where medians is a vector where the jth element is the median of the jth feature of V. This quantity dist_i generalizes to multiple dimensions the notion of "number of MADs from the median".

One would then empirically, for each dimension n and each number of points m, find the cutoff such that when each feature of each point is drawn independently from the chosen theoretical distribution, on average only a fraction p of them would have a *dist* value larger than that cutoff. Points with *dist* values greater than that would be discarded. This approach seems to naturally generalize the univariate one, but it has challenges, including the difficulty of accounting for both m and n (rather than just m as in the univariate case). Furthermore, the matrix D would need to be investigated to make sure it is always positive definite. And finally, there is the question of how well this method would work in the case where the features of each point really do come from the chosen theoretical distribution, but where they are not drawn independently. We leave these questions open as a potential future line of inquiry.

Note that MAD Removal, as defined here, is most appropriate when the underlying theoretical distribution that we have reason to believe our data comes from (though with possible corruptions) is at least reasonably symmetric. While the MAD statistic does not apply just to symmetric distributions, it operates by calculating a symmetric range around the median within which 50% of the data lies (Rousseeuw and Croux, 1993), making it most suitable to symmetric situations. For highly non-symmetric distributions, the MAD Removal algorithm could be modified so that the MAD is replaces with the interquartile range, and rather than finding a single cutoff such that only a fraction p of our data is removed when it comes from our theoretical distribution, we could instead find two cutoffs, one to the left of the median and one to the right, such that when our data is drawn from our theoretical distribution a fraction $p/2$ will be removed on average from each side of the median.

Algorithm 3 Preprocessing For the MAD Removal Algorithm (Normal Distribution Example)

1: For each vector size, m, from t_1 to t_2

2: Generate s vectors v^i of length m, with each entry independent and normally distributed (i.e. drawn from our theoretical distribution of interest) with mean 0 and standard deviation 1.

3: For each such vector v^i calculate the vector $z^i = \frac{1}{\alpha_p}|\frac{v^i - \text{median}(v^i)}{\text{mad}(v^i)}|$. We will use the notation erf^{-1} to mean the inverse of the erf function, which itself is defined as $\text{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$, where the constant $\alpha_p \equiv \frac{\text{erf}^{-1}(1-p)}{\text{erf}^{-1}(\frac{1}{2})} \geq 0$ is the asymptotic adjustment factor (for the normal distribution) handling the case of large m (i.e. the asymptotic number of MADs from the median to remove a fraction p of values when they are drawn from our theoretical distribution as $m \to \infty$). For all symmetric distributions the asymptotic adjustment factor is $\alpha_p = \frac{F^{-1}(1-\frac{p}{2}) - F^{-1}(1/2)}{\frac{1}{2}(F^{-1}(3/4) - F^{-1}(1/4))}$, where $F^{-1}(p)$ is the inverse cumulative distribution function.

4: Assemble all the z^i into a size $s\,m$ one dimensional vector $g = (z^1, \ldots, z^m)$.

5: Define u_m to be the $(1-p)th$ percentile value of g, so that approximately a fraction p of values in g are larger than u_m. Note that there is one u_m per vector size m.

6: Form the vectors $q^{\text{odd}} = (u_{t_1}, u_{t_1+2}, u_{t_1+4}, \ldots)$ and $q^{\text{even}} = (u_{t_1+1}, u_{t_1+3}, u_{t_1+5}, \ldots)$.

7: Solve for the optimal parameters a,b,c, and d in the function

$$f(m, a, b, c, d) = 1 + \frac{1}{b\frac{m^d}{\log(m)^c} - a}$$

to minimize the sum of squared errors $\sum_{m=t_1}^{t_2}(f(m, a, b, c, d) - q_{m-t_1+1})^2$ using the Levenberg-Marquardt algorithm with 5000 iterations and starting point $a = b = c = d = 1$, first for $q = q^{\text{even}}$ then for $q = q^{\text{odd}}$, giving us two sets of a,b,c,d.

8: The MAD Removal algorithm will remove values outside of $\text{median}(v) \pm \text{cutoff}(m)\,\text{mad}(v)$ where $\text{cutoff}(m) = \alpha_p r(m) = \alpha_p f(m, a, b, c, d)$ and a,b,c,d are the optimal constant values found in our prior step (taking care to use the a,b,c,d fit to q^{even} if $m + t_1$ is even, and the values fit to q^{odd} if $m + t_1$ is odd). On average about a fraction p of those values will be removed when the data is normally distributed. Note that $\lim_{m\to\infty} r(m) = 1$.

Algorithm 4 The MAD Removal Algorithm (applying the previously created model)

If v.size $< t_1$: return v unaltered.

if mad(v) = 0: return v unaltered.

cutoff $= \alpha_p \, r(m)$

boundary = cutoff $*$ mad(v)

If mode = "remove outliers":

    For each $v_i >$ median(v) + boundary: remove v_i from v.

    For each $v_i <$ median(v) − boundary: remove v_i from v.

Otherwise, if mode="clip outliers":

    For each $v_i >$ median(v) + boundary: set $v_i =$ median(v) + boundary.

    For each $v_i <$ median(v) − boundary: set $v_i =$ median(v) − boundary.

4.8 Details of the Impact Clipping Algorithm

In situations where we do not have a strong theoretical reason to expect a particular underly distribution, the MAD Removal algorithm is not appropriate. Hence, we introduce the Impact Clipping algorithm. The idea in this case is that we want to make sure the calculations we perform on our vector v are not overly influenced by any single value in the vector. For instance, if our sample standard deviation was altered by 50% based on the decision whether or not to clip the value of one particularly extreme value in our vector, then we should view the result as suspect. In such a case, the variance of our standard deviation estimation procedure itself may be unacceptably high. The Impact Clipping algorithm therefore is designed to limit the amount that any one value can impact our statistics, while additionally satisfying the criteria for an outlier removal algorithm from Section 4.6. In particular, the algorithm was carefully designed so that the criteria for whether a value is determined to be an outlier (and therefore, whether that value is clipped) does not depend on the level of extremeness of outliers more extreme than the value under consideration. This

prevents the swamping and masking problems that plague many outlier handling techniques.

In principle, for Impact Clipping any statistic of interest could be used when setting the maximum amount that a single value is allowed to change the result. For our implementation in this chapter though, we choose the standard deviation because it is the least robust statistic that is commonly of interest. Hence, Impact Clipping, as described here, is designed to limit the impact of any one value in the vector v on the standard deviation that we estimate using v. If it turns out that what we are most interested in is another statistic, like the mean, which is less sensitive to outliers than the standard deviation, then this procedure will still work fine as implemented here. It will simply clip values more aggressively than is strictly required to limit the impact of any one value on the mean.

Impact Clipping has two parameters. To understand these parameters, it is easiest to think about u, which is a copy of v that has been sorted in ascending order, and which is 0 indexed, so that $u = (u_0, \ldots, u_{m-1})$. The first parameter is $\mathcal{F}$, which is the approximate fraction of the values at the middle of u that we we want to be certain not to alter with the algorithm. We recommend setting $\mathcal{F} = 0.90$, since it is unlikely that more than 5% of values on either side of u will be extreme outliers. However, in cases where a larger rate of extreme values may occur, a smaller $\mathcal{F}$ can be used, such as $\mathcal{F} = 0.80$.

The second parameter of Impact Clipping is $\mathcal{I}$, which is the fractional amount that a value is allowed to increase our current estimate of the standard deviation (when we go from clipping it, to no longer clipping it) such that we are willing to allow that value to be unclipped. Our algorithm will start off treating as clipped any value outside of the middle $\mathcal{F}$ fraction of values of u. This means that, to start, values less than that of the left edge of the middle $\mathcal{F}$ values of

u are set to the value at the left edge of the middle, and values greater than that of the right edge of the middle $\mathcal{F}$ are set to the value at the right edge of the middle. We then proceed to try adjusting our clipping so that it starts instead one value to the left, or one value to the right. We try the values closest to our original (clipped) estimate of the mean first. We only accept a clipping change (i.e. shift which value we start clipping on either side at) when doing so does not cause our current (clipped) standard deviation estimate to increase by more than $\mathcal{I}$.

We recommend setting $\mathcal{I} = 0.20$, meaning that a value will only be unclipped if doing so increases the standard deviation by less than 20%. If this sounds high, keep in mind that when a value is unclipped, it raises the clipping of the other potential outlier values (on its same side from the middle that are more extreme than it) to precisely its own value. So the amount the standard deviation changes depends not just on the value itself, but on the amount that unclipping it raises the clipping of other values. In other words, the change in standard deviation is based on raising the clipping of all values this extreme to the value under consideration. Using $\mathcal{I} = 0.05$ is too small for most purposes, and $\mathcal{I} = 0.50$ is too large. Note that for $\mathcal{F} = 0.90$ and $\mathcal{I} = 0.20$ when $m = 30$ only about 1 in 1000 values will be clipped on average when our data comes from a normal distribution, as shown in Figure 4.5. Having little impact on normally distributed values is desirable, as explained in Section 4.6. It is also important to note that for small $\mathcal{F} = 0.90$ a somewhat larger $\mathcal{I}$ should be used, since when there are fewer values in the middle unclipped region, changing the clipping impacts more values at once and so is expected to cause a somewhat larger standard deviation increase.

Note that in cases where we might apply Impact Clipping to estimate quantities like the mean and standard deviation, one may be tempted to instead simply replace them with alternative, more robust statistics, such as swapping the mean with the median and the standard deviation with the interquartile range. In some cases, the median and interquartile

range are no less appropriate for achieving our particular goals than the mean and median, and so this substitution is perfectly reasonable. But in other cases, what we actually want is an outlier robust estimate of the mean and standard deviation, not just any measure of location and scale. In such cases replacing (for instance) the sample mean with the sample median can be misleading, as the two can differ substantially even when outliers are not present, and even in the limit as the sample size goes to infinity. An example of this phenomena is shown in Figure 4.7. Hence, if what we truly care about is the mean, then we should estimate it in an outlier robust way, not switch to an alternative statistic like the median.

In Figures 4.5 and 4.6 we plot the fraction of values that Impact Clipping (using the standard deviation as our statistic of interest) ends up removing, on average, when applied to data drawn from various distributions, for three different choices of $\mathcal{I}$. Notice that while the fraction of values clipped generally decreases with m (since the bigger m is, the less influence each value has on the standard deviation), there are occasional discontinuities where the fraction clipped jumps up. This is caused by the discrete nature of the operation of taking the middle fraction $\mathcal{F}$ of values. We are constrained in our selection of middle values by three requirements: (1) the number of middle values must be at least one, (2) we must keep an equal number of values on either side of the middle so as to avoid directional bias, and (3) the number of middle values must be an integer even though $\mathcal{F}m$ usually is not an integer. To satisfy all these constraints, our initial guess for how many values the middle (i.e. initial unclipped set) should have is either $\mathcal{F}m$ or 2 (whichever is bigger), and then we round down either by one or by two, whichever leaves us with an equal number of values on both sides of the middle. With $\mathcal{F} = 0.90$ this leaves us with a jump at $m = 21$, where we go from having one value on each side of the middle to two values on each side of the middle, with the next jumps happening at $m = 41$, $m = 61$, etc. As the sample size gets larger the jumps become

less impactful, since going from 1 value on each side of the middle to 2 is a much greater change than, for example, going from 9 on each side to 10.

4.8.1 Implementing Impact Clipping

For integers $s \geq 0$ and $e < m$, we define the clipped mean $\mu_{s,e}$ of a zero-indexed vector $u = (u_0, u_1, \ldots, u_{m-1})$ of real numbers, whose values are sorted in increasing order, to be:

$$\mu_{s,e} \equiv \frac{1}{m}\left(s\, u_s \;+\; \sum_{i=s}^{e} u_i \;+\; (m-e-1)\, u_e\right). \tag{4.2}$$

This is just the mean that we would get if the values less than u_s in u were all set to u_s, and the values greater than u_e were all set to u_e. In other words, we have winsorized u and then taken its mean. Analogously, we define the clipped mean of squares:

$$\mu^{sq}_{s,e} \equiv \frac{1}{m}\left(s\, u_s^2 \;+\; \sum_{i=s}^{e} u_i^2 \;+\; (m-e-1)\, u_e^2\right) \tag{4.3}$$

which is the same as the clipped mean, except with each element of u being squared. Finally, we can define the clipped sample variance as:

$$\sigma^2_{s,e} \equiv \frac{m}{m-1}(\mu^{sq}_{s,e} - \mu^2_{s,e}). \tag{4.4}$$

where the factor $\frac{m}{m-1}$ is needed because an unbiased sample mean normalizes by $\frac{1}{m-1}$, unlike the mean which normalizes by $\frac{1}{m}$. This formula is simply the usual one for writing the variance in terms of the mean and mean of squares, but applied to the clipped mean and clipped mean of squares.

The quantities $\mu_{s,e}$ and $\mu^{sq}_{s,e}$ have the convenient property that they can each be updated in

constant time when s decreases by one or e increases by one. Therefore, $\sigma^2_{s,e}$ shares the same property, since it is merely a function of $\mu_{s,e}$ and $\mu^{sq}_{s,e}$, as shown in its definition above. In particular, we can efficiently update these quantities as s decreases by one or e increases by one by writing:

$$\mu_{s-1,e} = \mu_{s,e} + \frac{s}{m}(u_{s-1} - u_s) \tag{4.5}$$

$$\mu^{sq}_{s-1,e} = \mu^{sq}_{s,e} + \frac{s}{m}(u^2_{s-1} - u^2_s) \tag{4.6}$$

and

$$\mu_{s,e+1} = \mu_{s,e} + \frac{m-e-1}{m}(u_{e+1} - u_e) \tag{4.7}$$

$$\mu^{sq}_{s,e+1} = \mu^{sq}_{s,e} + \frac{m-e-1}{m}(u^2_{e+1} - u^2_e). \tag{4.8}$$

With this notation in place, we can explain the Impact Clipping algorithm in detail. We start with a vector v that we want to handle outliers in, make a copy u, which we sort in increasing order. We then begin our process with a fraction p of values (in the middle of u) that we want to leave unmodified, giving us τ unmodified values in the center (which is at most p m), ensuring an equal number of values to the left and right of those middle values. We then set s to the index of the smallest of the middle τ values, and e to the index of the largest of the middle τ values. We next get an initial estimate of the mean $\mu_0 \equiv \mu_{s,e}$ and compute the standard deviation $\sigma_{s,e}$ using Equations 4.2, 4.3 and 4.4. At this stage, we are treating all values to the left of s and to the right of e as being clipped in our calculation of the standard deviation. We will progressively try unclipping them, in order of how close they are to μ_0, so that we handle the least extreme values first. If the corresponding new

standard deviation (which will either be $\sigma_{s,e+1}$ if we're unclipping a value from the right, or $\sigma_{s-1,e}$ if we're unclipping a value from the left) is not more than a fraction $\mathcal{I}$ bigger than the previous standard deviation $\sigma_{s,e}$, then we permanently unclip that value and decrease s by one (if the value was to the left) or increase e by one (if the value was to the right). However, if this allowed fraction $\mathcal{I}$ to increase the standard deviation is exceeded, we will no longer attempt to unclip values any more from the corresponding side. Eventually the algorithm will end, either because both sides have had a value which, when unclipping, exceeds the allowed standard deviation increase, or because the ends $s = 0$ and $e = m - 1$ are reached. We now return our original vector v, except that values less than u_s are set to u_s, and values greater than u_e are set to u_e, where here s and e are the final values of these variables (which indicate where the last unclipping occurred). The details of this algorithm are shown as Algorithm 5, and Python code for the algorithm is given in Section 4.11. This implementation has running time on the order of $m \log(m)$ since it relies on sorting all of the values.

4.9 Differences Between MAD Removal and Impact Clipping

The Impact Clipping algorithm is designed to behave very differently than MAD Removal in the limit as the number of values goes to infinity. Since MAD Removal is premised on the idea that we should have a certain underlying distribution for the data, but that the values may be corrupted for some reason, it makes sense to remove bad seeming data even when the number of values is very large. But Impact Clipping makes no such assumption. In its case, our concern is with not letting any single value impact the standard deviation too much. But as the number of values grows, each individual one has less effect on the standard deviation, so we become increasingly willing to leave values unclipped. Therefore, unlike MAD Removal,

Algorithm 5 Impact Clipping

1: Create a zero-indexed copy of the vector v, called u, and sort the values of u in ascending order.

2: Select the middle τ values of u (which will be left unclipped), where τ is set to $\mathcal{F}\,m$ after rounding it down to the nearest integer, or to the integer beneath that one, whichever leaves τ with at least one element and puts an equal number of values to the left of τ as to the right of τ.

3: Define two integers, the start value $s = (m - \tau)/2$ and end value $e = (m + \tau)/2 - 1$, which are the left most and right most of the middle τ values.

4: Save our original mean estimate $\mu_0 = \mu_{s,e}$ using Equation 4.2.

5: Set $continueLeft = True$ and $continueRight = True$, which signify that we have not yet stopped trying to add back into our standard deviation calculation the values to the left of s and to the right of e respectively.

6: If $s = 0$ then set $continueLeft = False$, and if $e = m-1$ then set $continueRight = False$, indicating that there are no more values left to try in the corresponding direction (i.e. to the left of s or to the right of e).

7: If $continueLeft = False$ and $continueRight = False$ then finish and return a copy of v with every value less than u_s set to u_s, and every value greater than u_e set to u_e.

8: If $continueLeft = True$ and (continueRight = False or $|u_{s-1} - \mu_0| \leq |u_{e+1} - \mu_0|$), then:

9: Efficiently calculate $\mu_{s-1,e}$ and $\mu^{sq}_{s-1,e}$ and $\sigma^2_{s-1,e}$ using Eq. 4.5, 4.6 and 4.4.

10: If $\sigma_{s-1,e}/\sigma_{s,e} - 1 > \mathcal{I}$, then set $continueLeft = False$

11: Otherwise: set $s \to s - 1$

12: Otherwise:

13: Efficiently calculate $\mu_{s,e+1}$ and $\mu^{sq}_{s,e+1}$ and $\sigma^2_{s,e+1}$ using Eq. 4.7, 4.8 and 4.4.

14: If $\sigma_{s,e+1}/\sigma_{s,e} - 1 > \mathcal{I}$, then set $continueRight = False$

15: Otherwise: set $e \to e + 1$

16: Return to Step 6.

Impact Clipping is asymptotically unbiased.

Another substantial difference between Impact Clipping and MAD Removal is that, in the former, we clip values without removing them, whereas in the latter we remove values completely. The reason for the difference is that, in the former case, we have no particular reason to believe that data is corrupted or invalid, since we know nothing about our data's underlying distribution. We simply want to protect ourselves from extreme values that might excessively distort our statistics. Clipping is a way to protect ourselves while introducing less bias than would occur with outlier removal. On the other hand, for MAD Removal, a value that is way outside what is expected from our assumed theoretical distribution can reasonably be said to be a corrupt or erroneous value, and so removing it makes sense, and is not expected to introduce significant bias.

4.10 Conclusion

We have introduced two methods for handling outliers in data. The first, MAD Removal is appropriate when the data should have been generated by a known distribution, but may have been corrupted or contain errors. The second, Impact Clipping, is appropriate when the underlying distribution is unknown, and we are concerned with limiting the impact that any one value in our data can have on statistics of interest, such as the standard deviation.

4.11 Python Code

```
import math
import numpy as np

def MAD_Removal(values, mode="remove outliers", normalFractionAffected=".001"):
        """Clips (i.e. Winsorizes) or removes potential outliers from values (a list
        or 1d numpy array of real numbers). Identifies a lower and upper cutoff such
        that if the elements of values are independent and normally distributed
        then on average only about a fraction normalFractionAffected of elements
        will not lie between the cutoffs. Cutoffs are based on the probability of
        normal values being that many median absolute deviations (i.e. MADs) away
        from the median, given the length of values.

        Elements not within cutoffs are clipped or removed based on mode setting.

        If mode='clip outliers' then winsorize by setting values below lower cutoff
        to lower cutoff, and values above the upper cutoff to the upper cutoff.

        If mode='remove outliers' then values not falling between lower and upper
        cutoff will be removed completely from the returned copy of values.

        normalFractionAffected must be one of these text strings:
        '.0001', '.001', '.01'

        Never modifies values, always returns a copy.
        """
        values = np.array(np.copy(values)).flatten() #copy as 1d flat numpy array
        if len(values) <= 3: return values #don't modify if less than 4 values
        median = np.median(values)
```

```
        mad = np.median(np.abs(values - median)) #the median absolute deviation
        if mad <= 1E-16: return values #if middle 50% all the same, just return
        #allowed distance from median is sample size dependent multiple of MAD
        cutoff = madAdjustment(len(values), normalFractionAffected)
        boundary = mad * cutoff
        if mode == "clip outliers": #set large and small values to the cutoffs
                values[values>median + boundary] = median + boundary #clip big values
                values[values<median - boundary] = median - boundary #clip small values
                return values
        elif mode == "remove outliers": #completely remove large and small values
                #keep only values within boundary of median
                return values[np.abs((values-median)/float(boundary)) <= 1]
        else:
                raise ValueError("mode must be 'clip outliers' or 'remove outliers'")


def madAdjustment(sampleSize, normalFractionAffected):
        """gives the number of MADs from the median to achieve only a fraction
        normalFractionAffected outside of median +- cutoff*mad when the data
        really does come from a normal distribution."""
        m = sampleSize
        even = (m % 2 == 0) #true if and only if sampleSize is even
        #standardize the format of normalFractionAffected as a string
        normalFractionAffected = str(normalFractionAffected).lstrip("0").rstrip("0")
        if normalFractionAffected == ".0001":
                asymptoticMultipier = 5.7682 #=InverseErf[1-.0001]/InverseErf[0.5]
                #handle even and odd cases separately, using numerically fit parameters
                if even: r = correction(m, a=0.670852, b=0.160374, c=1.87124, d=1.52869)
                else: r = correction(m, a=1.06162, b=0.275104, c=2.17383, d=1.5079)
        elif normalFractionAffected == ".001":
                asymptoticMultipier = 4.87854 #=InverseErf[1-.001]/InverseErf[0.5]
```

```
                if even: r = correction(m, a=0.858321, b=0.242921, c=1.45477, d=1.36824)
                else: r = correction(m, a=1.59485, b=0.468591, c=1.85156, d=1.35865)
        elif normalFractionAffected == ".01":
                asymptoticMultipier = 3.81893 #=InverseErf[1-.01]/InverseErf[0.5]
                if even: r = correction(m, a=1.50225, b=0.538382, c=1.27693, d=1.24351)
                else: r = correction(m, a=1.16243, b=0.338736, c=0.894065, d=1.21593)
        else:
                raise ValueError("normalFractionAffected must be one of these text\
                strings: '.0001', '.001', '.01'")
        return asymptoticMultipier * r

def correction(m, a, b, c, d):
        """the correction to adjust the asymptotic cutoff for sample size m"""
        return 1.0 + 1.0/float(b * ((m ** d) / (math.log(m) ** c)) - a)




def impactClipping(values, initialFractionToInclude=0.90, maxStdDevIncrease=0.15):
        """An outlier clipping method that returns a copy of values (in the original
        order), where we first compute the variance and mean of the middle
        initialFractionToInclude of the values, clipping (i.e. winsorizing) the others
        and try unclipping each of the unclipped values one at a time in order of how
        close they are to the original clipped mean. If adding one back ever raises
        the standard deviation (that we have so far) by more than a maxStdDevIncrease
        fraction then we stop unclipping values from that side.
        """
```

```
sortedValues = np.array(np.copy(values)).flatten() #copy as 1d flat array
sortedValues.sort() # sort ascending
m = len(sortedValues)
numToInclude = m*initialFractionToInclude
maxInTheMiddle = int(math.floor(numToInclude)) #rounded down
if maxInTheMiddle <= 2: maxInTheMiddle = 2
#make sure that the number of values to the left and right are the same
if (m % 2 == 0) == (maxInTheMiddle % 2 == 0): numToInclude = maxInTheMiddle
else: numToInclude = max(maxInTheMiddle - 1, 0)
#set so that approximately initialFractionToInclude are in the middle
numberOnEachSide = int(round((m-numToInclude)/2.0))
start = numberOnEachSide #left edge of the middle (i.e. unclipped) values
end = m - numberOnEachSide - 1 #right edge of the middle values
#calculate initial stats clipping values left of start and right of end
mean = (start*sortedValues[start] + np.sum(sortedValues[start:end+1])\
        + (m - end - 1)*sortedValues[end])/float(m)
avgSq = (start*(sortedValues[start]**2)\
        + np.sum(np.power(sortedValues[start:end+1],2))\
        + (m - end - 1)*(sortedValues[end]**2))/float(m)
var = (avgSq - mean**2) * (m/float(m-1))
origMean = mean #used to decide order of adding back clipped values
continueLeft = True #we stop trying values left of start when False
continueRight = True #we stop trying values right of end when False
while True:
        if start == 0: continueLeft = False
        if end == m - 1: continueRight = False
        if (not continueLeft) and (not continueRight): break
        newMean = mean
        newAvgSq = avgSq
        newStart = start
        newEnd = end
```

```
        #smallest dist. to original clipped mean decides if next is left or right
        if continueLeft and ((not continueRight)\
                or abs(sortedValues[start-1]-origMean)\
                < abs(sortedValues[end+1]-origMean)):
                mult = start/float(m)
                newMean += mult * (sortedValues[start - 1] - sortedValues[start])
                newAvgSq += mult \
                * (sortedValues[start - 1]**2 - sortedValues[start]**2)
                newStart -= 1
        else:
                mult = (m - end - 1)/float(m)
                newMean += mult * (sortedValues[end+1] - sortedValues[end])
                newAvgSq += mult * (sortedValues[end+1]**2 - sortedValues[end]**2)
                newEnd += 1

        newVar = (newAvgSq - newMean**2) * (m/float(m-1))
        #check if the standard deviation increased too much
        #change to another statistic if the std dev is not the stat of interest
        condition = math.sqrt(newVar/float(var)) - 1.0 > maxStdDevIncrease
        if condition:
                if newStart == start - 1: continueLeft = False
                else: continueRight = False
        else: #otherwise, keep the new values for each variable
                mean, avgSq, var = newMean, newAvgSq, newVar
                start, end = newStart, newEnd
valuesCopy = np.array(np.copy(values)).flatten()
valuesCopy[valuesCopy<sortedValues[start]] = sortedValues[start]
valuesCopy[valuesCopy>sortedValues[end]] = sortedValues[end]
return valuesCopy
```

MAD Removal - Fraction of Values Removed (Normal, Uniform, Bimodal)

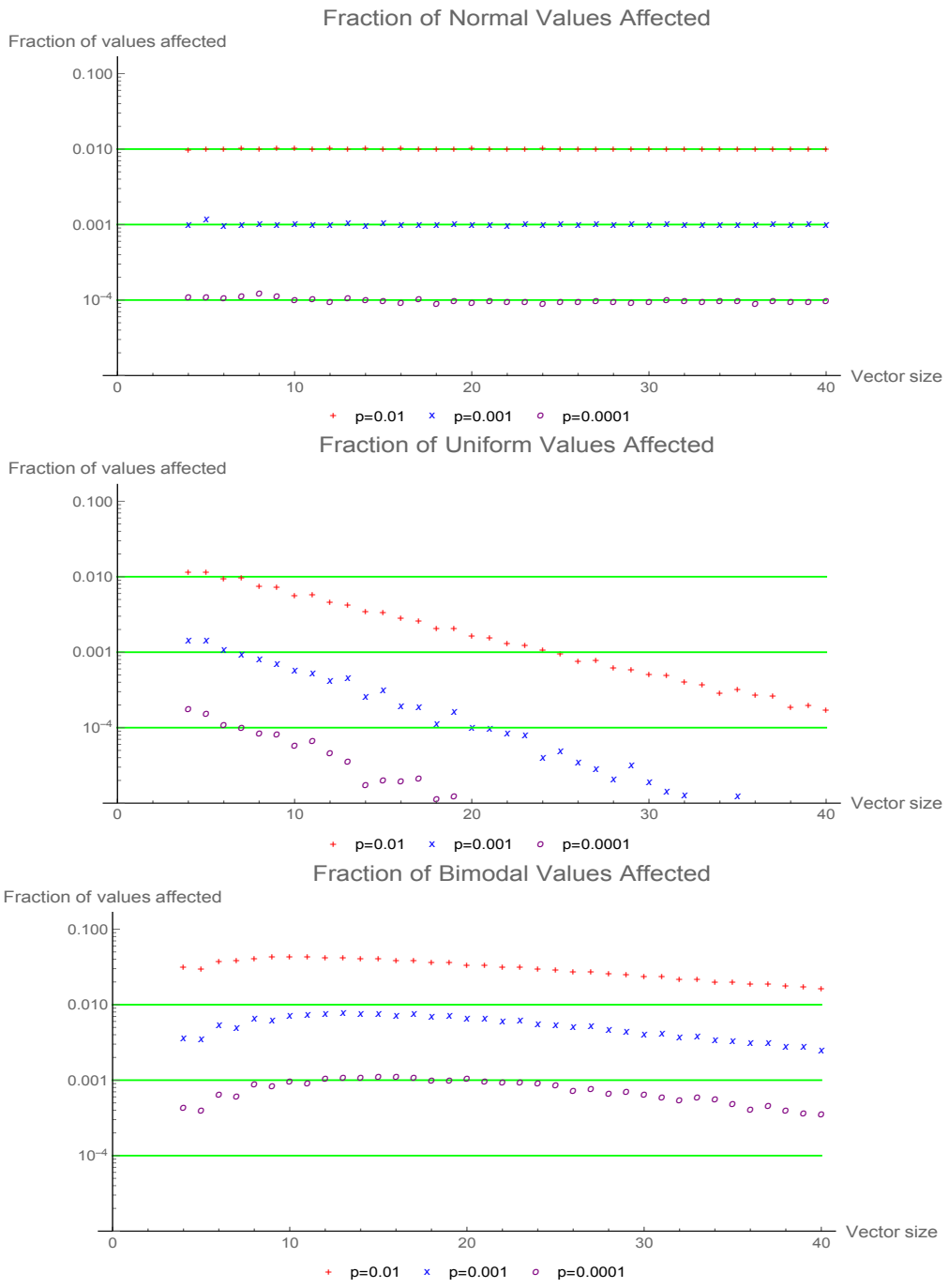

Figure 4.1: The average fraction of values that are removed by the MAD Removal algorithm (using a normal distribution as its underlying theoretical distribution) when applied to a large number of vectors of varying sizes, where the vector elements are independent and identically distributed and drawn from (1) a normal distribution, (2) a uniform distribution, and (3) a bimodal distribution (consisting of two normally distributed bumps of equal standard deviation, with their means 6 standard deviations apart). Each dotted line corresponds to a different value of p, the fraction of values expected to be removed when the values are independent and drawn from a normal distribution.

MAD Removal - Fraction of Values Removed (Log Normal, Cauchy, Exp)

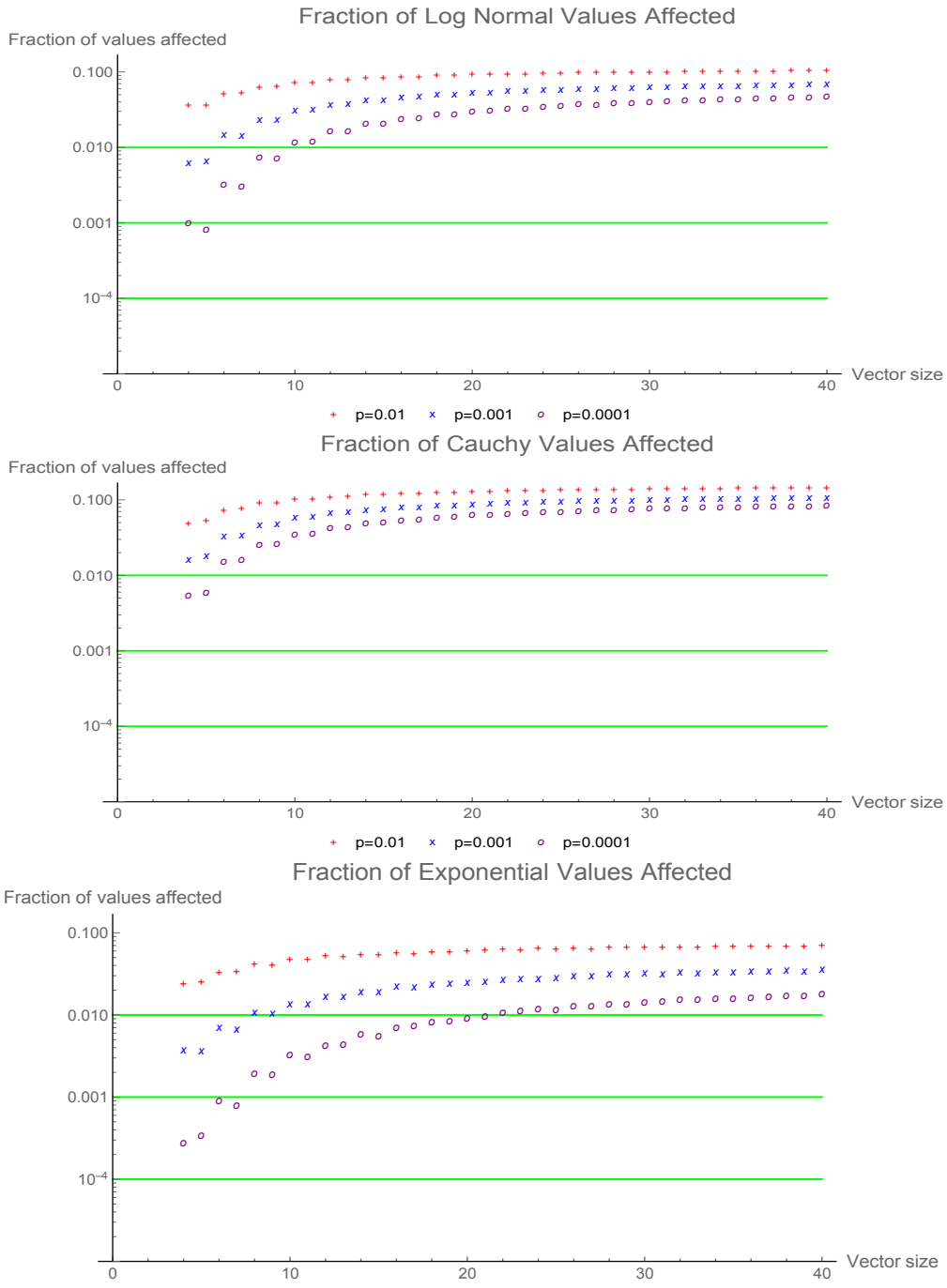

Figure 4.2: The average fraction of values that are removed by the MAD Removal algorithm (using a normal distribution as its underlying theoretical distribution) when applied to a large number of vectors of varying sizes, where the vector elements are independent and identically distributed and drawn from (1) a log normal distribution, (2) a Cauchy distribution and (3) the exponential distribution. Each dotted line corresponds to a different value of p, the fraction of values expected to be removed when the values are independent and drawn from a normal distribution.

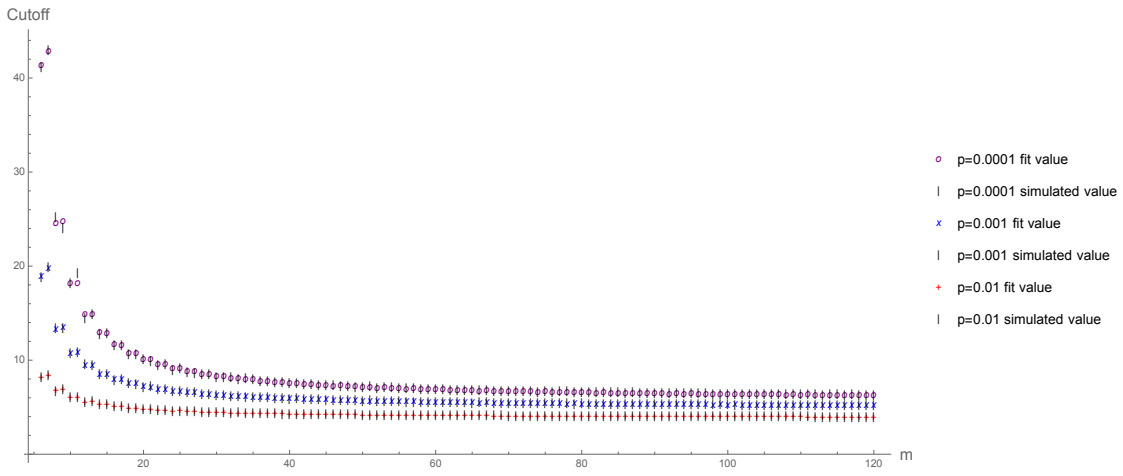

Figure 4.3: This figure shows the empirically determined cutoff values at different sample sizes m and for difference choices of p. We compare these empirically derived values to the values of the function cutoff$(m) = \alpha_p r(m) = \alpha_p f(m, a, b, c, d)$, where a,b,c,d have been fit via a least squares minimization (as described in Algorithm 3) to match the empirical values. Note that for a given p there are in fact two sets of parameters a,b,c,d, as one set is used when m is even, and another is used when m is odd, but for easy viewing in the figure we choose the appropriate such function for each m to make the plot seamless, as though it is just one function. As can be seen in the figure, our curve fitting procedure is able to closely fit the empirical cutoff values. Note that as m gets large these curves asymptote to α_p, which gives 3.82, 4.88, and 5.77 MADs from the median for p equal to 0.01, 0.001, and 0.0001 respectively. Furthermore, note that the cutoffs are decreasing as m increases, meaning that the larger the sample size, the less lenient our cutoffs are. At m=100 we have a cutoff that is 4.5%, 7.4% and 10.2% higher than the $m \to \infty$ asymptote for p equal to 0.01, 0.001, and 0.0001 respectively

Differences Between Cutoffs Via Simulation Versus Fit To Those Values

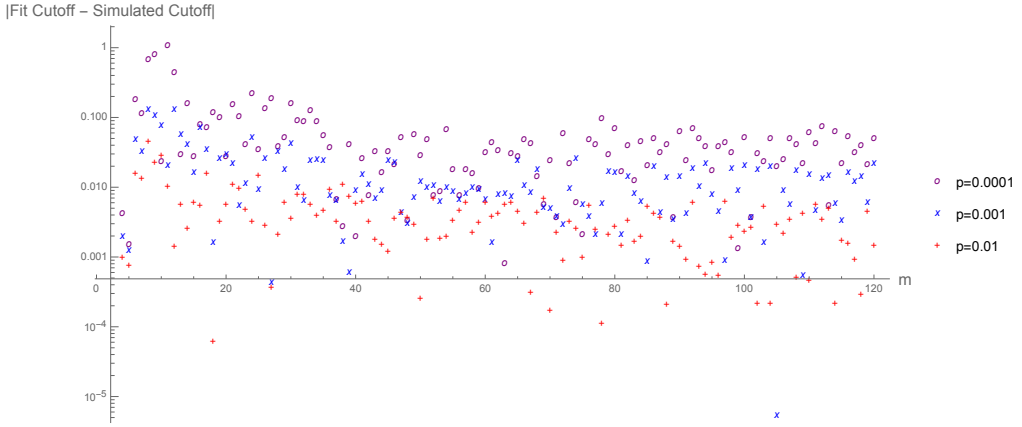

Figure 4.4: This logarithmic plot shows the same information as Figure 4.3, but now we have taken the the absolute value of the differences between the empirically determined cutoff values and the fit values cutoff$(m) = \alpha_p r(m)$, rather than looking at the raw (i.e. non-difference) values. As can be seen in the chart, the fit overshoots the empirical values slightly, but the difference is so small as to have no practical significance.

Impact Clipping - Fraction of Values Clipped (Normal, Uniform, Bimodal)

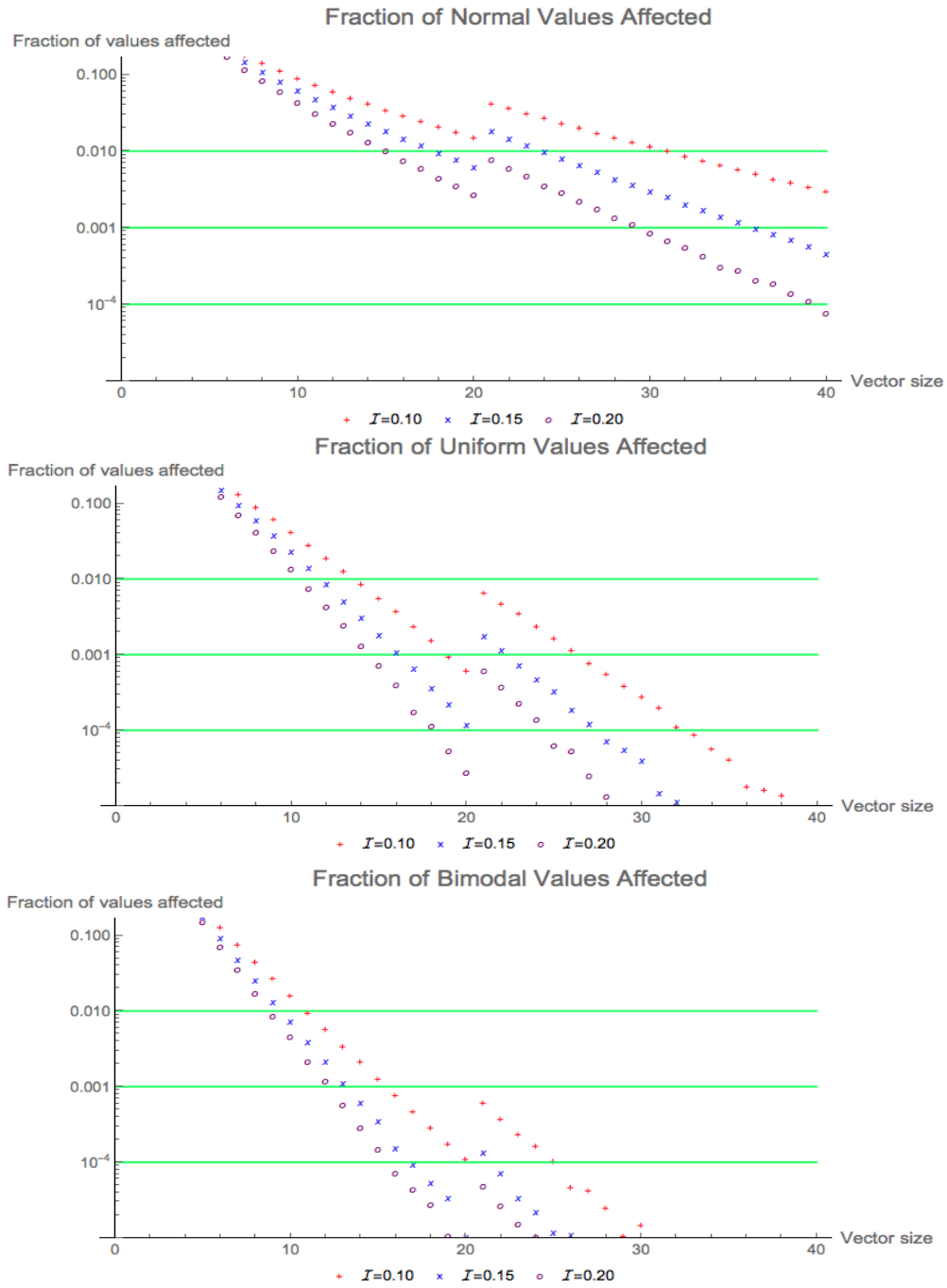

Figure 4.5: The average fraction of values that are clipped by the Impact Clipping algorithm when applied to a large number of vectors of varying sizes, where the vector elements are independent and identically distributed and drawn from (1) a normal distribution, (2) a uniform distribution, and (3) a bimodal distribution (consisting of two normally distributed bumps of equal standard deviation, with their means 6 standard deviations apart). Each dotted line corresponds to a different value of $\mathcal{I}$, the fraction by which the standard deviation is allowed to increase in order for us to be willing to unclip a previously clipped value.

Impact Clipping - Fraction of Values Clipped (Log Normal, Cauchy, Exponential)

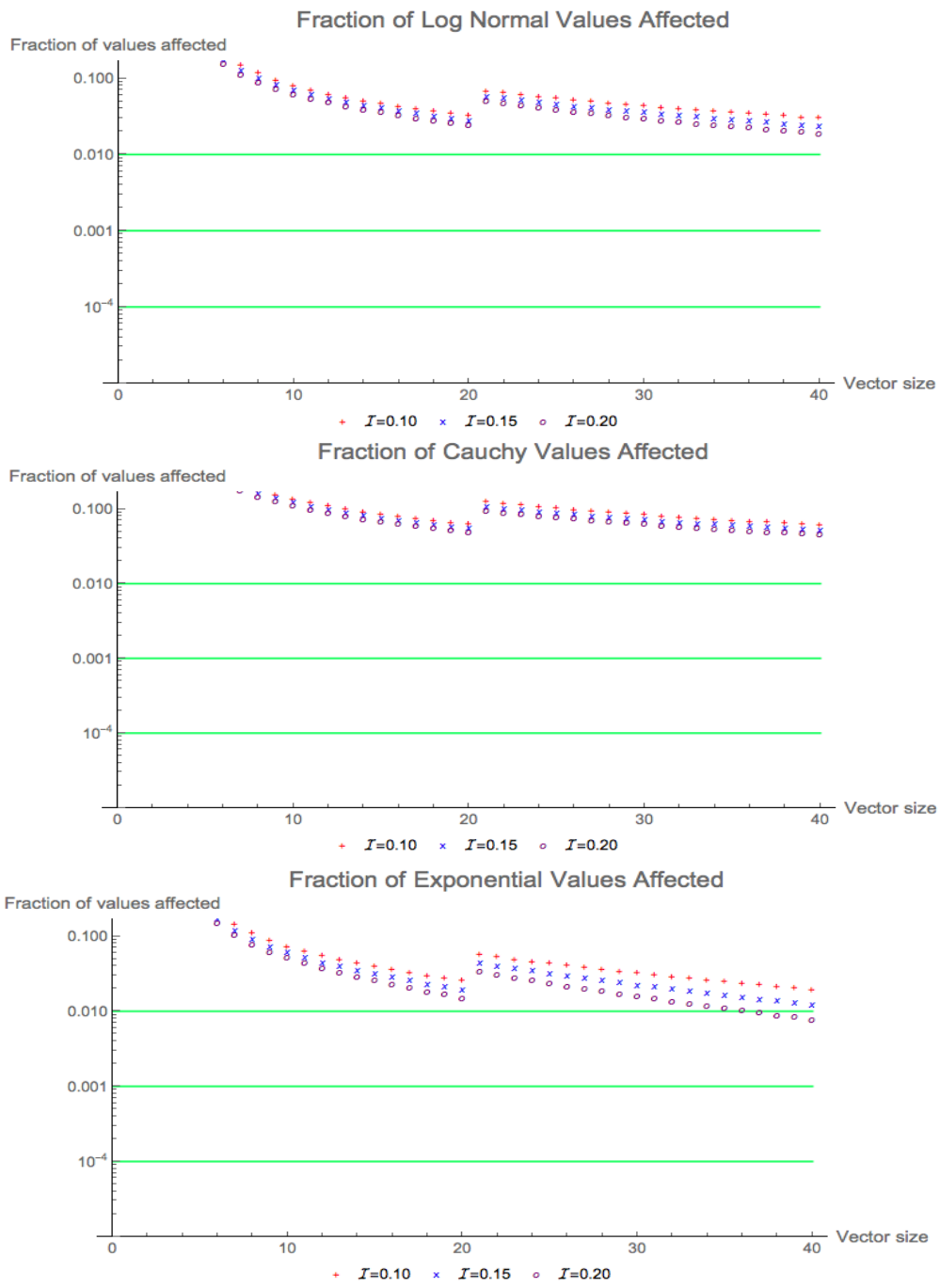

Figure 4.6: The average fraction of values that are clipped by the Impact Clipping algorithm when applied to a large number of vectors of varying sizes, where the vector elements are independent and identically distributed and drawn from (1) a log normal distribution, (2) a cauchy distribution, and (3) an exponential distribution. Each dotted line corresponds to a different value of $\mathcal{I}$, the fraction by which the standard deviation is allowed to increase in order for us to be willing to unclip a previously clipped value.

A Bimodal Distribution Where the Mean and Median Are Far Apart

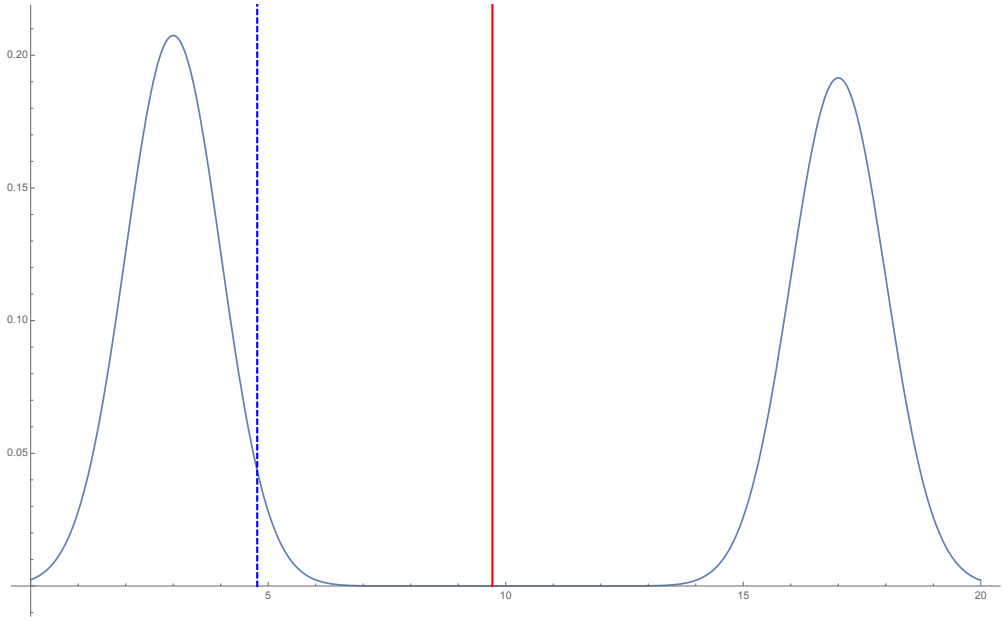

Figure 4.7: The dashed blue line shows the median of this bimodal distribution, and the solid red line shows the median. Since the bump on the left contains 52% of the probability, the median lies within the left bump, whereas the mean is nearly equidistant between the two bumps.

Chapter 5

Generalized Regularization for Least Squares Regression

We have already examined one generalization of Ridge Regression in Chapter 3, in the context of handling outliers. Here we generalize Ridge Regression in a different direction by modifying its regularization penalty. It is well known that by switching from the L_2 penalty of Ridge Regression, to the L_1 penalty of Lasso Regression, we can learn sparse solutions. And as we saw in Chapter 1, on some prediction problems this tendency towards sparsity allows Lasso Regression to significantly outperform Ridge Regression, such as when the labels were generated from a linear model that only used a few of the features. That raises an intriguing question: what do other sorts of regularization terms do to our solution? And when would we want a regularization penalty that is neither that of Ridge Regression nor that of Lasso Regression?

In this chapter, we replace the standard L_2 regularization of Ridge Regression with an arbitrary positive definite quadratic form. This leads to an optimization problem that is substantially more general that Ridge Regression, but that is still analytically solvable,

making possible the closed form analysis of the generalization ability of such an algorithm when trained on a linear problem. The conclusion from such an analysis is that the best regularization is one chosen for the prediction problem at hand. In other words, regularization is a way to encapsulate prior knowledge about our problem. In this way, generalizing the regularization penalty provides us with a tool for tackling some of the ways that we saw Ridge Regression perform sub-optimally on empirical cases in Chapter 1. The more prior information we build into our model, the better we will perform, and generalized regularization can be a convenient means to do so.

We also discuss in this chapter the case of regularization that depends on the data itself, and give examples of how that can be used to change the algorithm's behavior. Many common approaches to analyzing the generalization ability of machine learning algorithms fail if the regularization is data dependent. In Chapter 9 however, we will develop a learning bound that permits this sort of data dependency.

We will now discuss our approach for generalizing the regularization of Ridge Regression and (by extension) Ordinary Least Squares.

5.1 Introduction

To prevent overfitting in Ordinary Least Squares regression, and to improve performance in the presence of co-linearity of features, it is common to introduce an L_2 penalty on the solution coefficients. This results in the Ridge Regression algorithm, which has as its vector

of feature coefficients the solution to the following optimization problem:

$$w_\lambda \equiv \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} (w^\intercal X^i - y_i)^2 + \lambda ||w||_2^2. \tag{5.1}$$

Here X is an $n \times m$ matrix of training data with one data point X^i per column and one feature X_j per row. We use the $m \times 1$ column vector y to represent the labels of our training set, with one label y_i corresponding to each point X^i in X. The parameter $\lambda > 0$ is the complexity parameter for Ridge Regression which determines the strength of regularization, and therefore how much the solution coefficients should be pushed towards 0. Hence, we can think of Ridge Regression as using a one parameter family $\lambda ||w||_2^2$ of regularization norms, one norm for each $\lambda > 0$. In real world implementations, Ridge Regression generally uses data dependent regularization, since λ is generally chosen using the data (e.g. using k-fold cross-validation). This is a minimal data dependency however, as the training data is used only to select this one constant.

In this chapter we consider what happens when we generalize this regularization 2-norm to a much broader class of norms, replacing the penalty term $\lambda ||w||_2^2$ with $(w-b)^\intercal \Lambda (w-b)$ instead, and allowing this regularization to depend in significant ways on the training data. Here Λ is a real-valued positive semi-definite matrix which we will refer to as the "regularization matrix" or when useful for clarity, the "feature regularization matrix", since this matrix determines how features are treated. The real valued vector b (the "bias" vector) biases w by pushing our solution coefficients to be more b-like. The restriction that Λ is positive definite guarantees that $(w-b)^\intercal \Lambda (w-b)$ has the finite minimum $w = b$. This proposed change of norm leads to what we call in this chapter "Generalized Ridge Regression", which we contrast with "Ridge Regression" (referring to the standard Ridge Regression algorithm). This particular approach to generalizing Ridge Regression, where we let the regularization be

a quadratic form that may depend on the training data, is appealing because:

1. It allows for more flexible inclusion of prior information than Ridge Regression.
2. It encompasses a wide range of different algorithmic behaviors, including those that mimic feature selection, feature extraction, outlier removal, and principal component analysis.
3. It is usable with kernels for non-linear predictions.
4. It still has a simple analytic solution that is easy to compute.

This contrasts with most other ways to modify the Ridge Regression objective function, such as Lasso Regression, where no simple analytic solution exists.

We will investigate a number of questions that arise from our choice of generalized regularization. We first discuss the relationship between prior information and regularization, and proceed to introduce the optimization problem that our regularization generalization corresponds to. We also discuss how generalization error in ideal linear cases relates to the choice of our regularization parameters. Then, we investigate how kernels can be used in this context, and how changing the regularization term relates to modifying the kernel. We also introduce some one parameter families of regularization matrices which can serve as an alternative to the one parameter family used in Ridge Regression. We expect these forms to lead to superior prediction performance for some classes of problems, as they encode different prior information and make different assumptions than Ridge Regression. Additionally, we explore how a regularization matrix can perform behaviors that are similar to feature selection, feature extraction and principal component analysis, and discuss how Ridge Regression behaves in the context of repeated features.

Note that in a number of sections of this chapter we use weights v_i for each training point (X^i, y_i), with $\sum_{i=1}^{m} v_i = m$, and we sometimes write this collection of weights as the diagonal elements of a diagonal weight matrix V. These weights allow us to count different data points different amounts, not unlike how a diagonal Λ matrix lets us count different features different amounts.

5.2 Regularization and Prior Information

One way to view the penalty term in standard Ridge Regression is as a prior that is being imposed on the prediction problem, where we assume that each coefficient is more likely to be small than to be large. The free parameter λ is then learned (for instance using k-fold cross-validation) as a way of setting an appropriate strength for this prior. When the Ridge Regression penalty is generalized to include a regularization matrix Λ, this can viewed as encompassing a much wider range of possible priors.

A simple interpretation of the Λ matrix can be seen when we write:

$$(w-b)^\intercal \Lambda (w-b) = \sum_{i=1}^{n}\sum_{j=1}^{n} \Lambda_i^j (w_i - b_i)(w_j - b_j)$$

$$= \sum_{1\leq i\leq n} \Lambda_i^i (w_i - b_i)^2 + 2\sum_{1\leq i\leq n}\sum_{1\leq j<i} \Lambda_i^j (w_i - b_i)(w_j - b_j).$$

Here it is apparent that if Λ does not depend on the data, then it encodes prior information about how large we expect each $(w_i - b_i)^2$ and each $(w_i - b_i)(w_j - b_j)$ to be. While $\Lambda = \lambda I$ with $b = 0$ penalizes all features equally, and while diagonal Λ with $b = 0$ gives each feature its own penalty, a full positive definite matrix for Λ also assigns each cross term $(w_i - b_i)(w_j - b_j)$ its own penalty Λ_i^j. Hence, it can capture the possibility that some coefficients w_i are more

or less likely to be large (or far from b_i when $b_i \neq 0$) when other coefficients w_j are large (or far from b_j when $b_j \neq 0$). This could be used, for instance, in image prediction problems where each feature corresponds to a pixel, where we use $b = 0$ and we set the off diagonal element Λ_i^j to be negative when i and j are nearby pixels to encode the prior knowledge that coefficients for nearby pixels should be correlated (while taking care to ensure that Λ is still positive definite).

5.2.1 Bayesian Interpretation of Λ

From a Bayesian perspective, we can think of Λ as switching from an isotropic gaussian prior (where the center is at the 0 vector, and the covariance matrix is a multiple of the identity matrix) to a general gaussian with arbitrary center and covariance matrix. Hence, our generalized form of regularization allows us to encode more complex prior knowledge about what sort of coefficients we expect.

To see this in detail, consider the case where our data is truly drawn from a linear model with additive gaussian noise, where each training point is drawn independently from the others. Then the label y_i for point X^i is given by:

$$y_i =^\intercal X^{i\intercal} w^* + \epsilon$$

where ϵ is a gaussian random variable with mean equal to the zero vector, and standard deviation σ. Suppose further that the prior probability we assign to solution coefficient vector w is given by the multivariate normal probability density function

$$P(w) \equiv \frac{1}{\sqrt{(2\pi)^n |\Psi|}} e^{\frac{-1}{2}(w-b)^\intercal \Psi^{-1}(w-b)}.$$

Then, applying Bayes' rule, the posterior probability $P(w \mid Z)$ of a given solution coefficient w, once we have observed our training data $Z \equiv (X, y)$, is given by:

$$P(w \mid Z) = \frac{P(w)P(Z \mid w)}{P(Z)} \propto P(w) \prod_{i=1}^{n} P(Z_i \mid w).$$

$$\propto e^{\frac{-1}{2}(w-b)^\intercal \Psi^{-1}(w-b)} \prod_{i=1}^{n} e^{-\frac{(w^\intercal X^i - y_i)^2}{2\sigma^2}}.$$

$$= e^{\frac{-1}{2}(w-b)^\intercal \Psi^{-1}(w-b) - \sum_{i=1}^{m} \frac{(w^\intercal X^i - y_i)^2}{2\sigma^2}}.$$

$$= e^{\frac{-1}{2\sigma^2}\left((w-b)^\intercal \sigma^2 \Psi^{-1}(w-b) + \sum_{i=1}^{m}(w^\intercal X^i - y_i)^2\right)}.$$

Now, since $e^{\frac{-1}{2\sigma^2}x}$ is a monotonic decreasing function of x, we maximize $P(w \mid Z)$ by solving

$$\min_{w} \sum_{i=1}^{m} (w^\intercal X^i - y_i)^2 + (w-b)^\intercal \Lambda (w-b),$$

where $\Lambda \equiv \sigma^2 \Psi^{-1}$. Hence, we see that in this case b reflects the coefficient vector w that we think is a priori most likely. On the other hand, Λ reflects both the anticipated level of label noise (via the constant σ), and the ellipsoid describing how a priori likely are all possible values of w (via the precision matrix Ψ^{-1}).

As an example where prior information is readily available, suppose that we have solved many very similar regression problems to the one we are interested in now, and view the current problem as having been drawn as a random sample from the set of all of these problems. Then, we can use the average solution coefficients of the previous problems for b, and use the distribution of the solution coefficients in these previous problems to estimate Λ. In that

case, b has the simple interpretation of being the mean solution coefficient vector, and Λ encodes how much deviation from this best guess we can expect (and in what directions this deviation is more and less likely).

Of course in general there is no universally "right" prior. The appropriate prior to use depends on the pre-existing information that we have about the prediction problem, so different problems will best be solved using different priors. Therefore we should expect that the standard Ridge Regression regularization norm (corresponding to $\Lambda = \lambda I$, $b = 0$) will sometimes be an excellent choice, but in other instances we will be much better served by choosing a different Λ and b.

In some cases, the most useful choices for Λ will not be "priors" at all, but rather, will depend on the training data. Introducing a data dependent regularization matrix Λ and bias vector b can cause a wide range of interesting behaviors. As we will see, some such data driven choices of Λ can even behave similarly to other algorithms that are usually performed separately from the regression, such as feature selection, feature extraction, and principal component analysis. To explore data dependent regularization, and to gain a deeper understanding of this generalized form of regularization in general, we begin by investigating the Generalized Ridge Regression solution.

5.3 The Generalized Ridge Regression Optimization Problem

The following lemma introduces our Generalized Ridge Regression optimization problem, which includes the feature regularization matrix Λ, the bias vector b, and the diagonal point

weight matrix V. We will assume throughout this chapter that the feature matrix X has had a column of all 1's appended. This feature plays the role of a constant term in our linear prediction function, allowing us to treat the constant term just like any other feature.

Lemma 3. *The unique solution to the Generalized Ridge Regression problem*

$$\underset{w\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} v_i(w^\intercal X^i - y_i)^2 + (w-b)^\intercal\Lambda(w-b).$$

for a real-valued positive definite symmetric regularization matrix Λ *is*

$$w_\Lambda = (XVX^\intercal + \Lambda)^{-1}(XVy + \Lambda b).$$

Proof. It is easily seen than setting the derivative

$$\frac{d}{dw}\left(\sum_{i=1}^{m} v_i(w^\intercal X^i - y_i)^2 + (w-b)^\intercal\Lambda(w-b)\right)$$

equal to zero yields the result above. Since by construction $XV^\intercal X$ is positive semi-definite and Λ is positive definite, the matrix $(XVX^\intercal+\Lambda)$ is positive definite, so its inverse exists. □

We note that $\lim_{||\Lambda||\to\infty} w_\Lambda = b$. This reflects a prior that locates all of the probability for the solution coefficients precisely at the vector b. As $||\Lambda||$ grows, the probability is increasingly concentrated around b.

5.4 Generalization Error of Generalized Ridge Regression

Previously, we explored how the maximum likelihood solution relates to the bias vector b and regularization matrix Λ. But generally it is not the maximum likelihood solution that

we care about per se, but rather, achieving a low generalization error. We now investigate the generalization error (i.e. the average prediction error on new data) under the same assumptions that we used before, namely, that our data is generated by a linear model with additive gaussian noise, and that our features were drawn from a multivariate gaussian distribution.

This setup will allow us to study the way that Λ and b impact the generalization error in this context, but also will give us intuition about the role that Λ and b play in general. There is no need in this case to introduce a prior distribution over the solution coefficients as we did previously, since our model is already determined. Recall that our Generalized Ridge Regression solution coefficients are given by $w = R(XVy + \Lambda b)$, where $R \equiv (XVX^\intercal + \Lambda)^{-1}$. Our first theorem on this topic expresses the generalization error, G, in terms of Λ (the regularization matrix), b (the bias vector), $\vec{\epsilon}$ (the vector of additive noise that has corrupted the vector of labels), w^* (the true solution coefficients that generated our data), C (the true covariance matrix of the features) and μ (the true mean of the features).

Theorem 5. *The generalization error of Generalized Ridge Regression, when trained on X and y that are generated from a linear model $y = X^\intercal w^* + \vec{\epsilon}$ with $\vec{\epsilon} \sim Norm(0, \sigma^2 I)$ and $X \sim Norm(\mu, C)$, is given by:*

$$G = ||ARXV\vec{\epsilon} + AR\Lambda(b - w^*)||^2 + \sigma^2$$

where $A \equiv \sqrt{C + \mu\mu^\intercal}$ and $||.||$ represents the 2-norm of a vector.

The theorem illustrates what impacts the generalization error. First, we see that it helps to have σ, the additive label noise, be very small. When that happens $||\vec{\epsilon}||$ will tend to be relatively small as well, so the main contributor to the error will be $AR\Lambda(b - w^*)$ which essentially measures the bias introduced by Λ and b. As $||\Lambda|| \to 0$ (corresponding to having less and less regularization) or $b \to w^*$ (corresponding to our guess for w^* being increasingly

perfect) this bias term disappears as well. So zero generalization error occurs in the limit as the label noise variance goes to zero, and either $||\Lambda|| \to 0$ or $b \to w^*$. If the underlying model generating the data were non-linear, additional bias would still remain caused by the optimal linear model differing that the true model.

The other term inside the norm, which is $ARXV\vec{\epsilon}$, is precisely A times the Generalized Ridge Regression solution coefficients (with $b = 0$) that we would have gotten had we trained our model on the noise vector $\vec{\epsilon}$ (instead of on the labels y). So that term is small when the regression model would have been able to accurately learn (when trained on labels consisting of pure mean 0 noise) that the appropriate solution coefficients is something close to the zero vector. In a sense, this is a measure of the algorithms propensity to overfit the training data, with more overfitting tending to result in a larger vector $RXV\vec{\epsilon}$. Of course when the label noise level is high (i.e. $||\vec{\epsilon}||$ is large) it becomes harder for the algorithm to determine that the coefficients should be close to zero, and hence the generalization error due to this term will tend to be larger.

As $||\Lambda||$ increases, we see a tradeoff between the term $ARXV\vec{\epsilon}$, which tends to shrink in magnitude, and the term $AR\Lambda(b - w^*)$, which tends to grow in magnitude. Hence, $||\Lambda||$ controls the tradeoff between variance and bias, which is reflected in these two terms, respectively.

While the magnitude $||\vec{\epsilon}||$ is dependent on σ, the direction that the vector $\vec{\epsilon}$ points is purely due to chance in this model, and is equally likely to point in any direction. If we get lucky, then $ARXV\frac{\vec{\epsilon}}{||\vec{\epsilon}||}$ will be small, and if we get unlucky it will be large. On the other hand, unlike the vector $\vec{\epsilon}$, we may have some a priori sense of the direction that $b - w^*$ may point. When that is the case, we can choose Λ to encapsulate that information, which will make $\Lambda(b - w^*)$ small, and tend to improve the generalization error. This reinforces the previous bayesian interpretation of Λ. Even in this non-bayesian context, Λ and b are encapsulating prior information about what w^* might be, and if we select them well the generalization error will tend to improve.

The proof of the theorem follows.

Theorem 5. We will make use of Corollary 4 from the chapter notes, which says that for a vector $x_0 \sim Norm(\mu, C)$, we have $\mathbf{E}_{x_0}\,(x_0^\intercal v)^2 = ||\sqrt{C+\mu\mu^\intercal}\,v||^2$.
We proceed to analyze the generalization error, G of our Generalized Ridge Regression solution. For $x_0 \sim Norm(\mu, C)$ and $y_0 = x_0^\intercal w^* + \epsilon$ with $\epsilon \sim Norm(0, \sigma)$ we can write:

$$G \equiv \mathbf{E}_{\mathbf{x_0},\mathbf{y_0}}\,(x_0^\intercal w - y_0)^2 = \mathbf{E}_{\mathbf{x_0},\epsilon}\,(x_0^\intercal w - x_0^\intercal w^* - \epsilon)^2 = \mathbf{E}_{\mathbf{x_0},\epsilon}\,(x_0^\intercal (w - w^*) - \epsilon)^2$$

$$= \mathbf{E}_{\mathbf{x_0}}\,(x_0^\intercal (w - w^*))^2 + \mathbf{E}_\epsilon\,\epsilon^2 - 2\mathbf{E}_{\mathbf{x_0},\epsilon}[\epsilon\, x_0^\intercal (w - w^*)]$$

$$= \mathbf{E}_{\mathbf{x_0}}\,(x_0^\intercal (w - w^*))^2 + \sigma^2 = ||\sqrt{C+\mu\mu^\intercal}\,(w - w^*)||^2 + \sigma^2$$

$$= ||A\,(R(XVy + \Lambda b) - w^*)\,||^2 + \sigma^2$$

$$= ||A\,(R(XV(X^\intercal w^* + \vec{\epsilon}) + \Lambda b) - w^*)\,||^2 + \sigma^2$$

$$= ||AR\left((XVX^\intercal - R^{-1})w^* + XV\vec{\epsilon} + \Lambda b\right)||^2 + \sigma^2$$

$$= ||AR\,(-\Lambda w^* + XV\vec{\epsilon} + \Lambda b)\,||^2 + \sigma^2$$

$$= ||ARXV\vec{\epsilon} + AR\Lambda(b - w^*)||^2 + \sigma^2$$

□

Since the vector $\vec{\epsilon}$ is random with $\vec{\epsilon} \sim Norm(0, \sigma^2 I)$, we can also consider the expected value of the generalization error (with respect to $\vec{\epsilon}$), to see how the average generalization error varies with respect to b, Λ and σ. We do so in the following corollary.

Corollary 2. *The average generalization error of Generalized Ridge Regression, when trained on X and y that are generated from a linear model $y = X^\intercal w^* + \vec{\epsilon}$ with $\vec{\epsilon} \sim Norm(0, \sigma^2 I)$*

and $X \sim Norm(\mu, C)$*, is given by:*

$$\mathbf{E}_{\vec{\epsilon}}\, G = \sigma^2 ||ARXV||^2_{Frob} + ||AR\Lambda(b - w^*)||^2 + \sigma^2$$

where $A \equiv \sqrt{C + \mu\mu^\intercal}$*,* $R \equiv (XVX^\intercal + \Lambda)^{-1}$*,* $||.||_{Frob}$ *represents the Frobenius norm of a matrix, and* $||.||$ *the 2-norm of a vector.*

Proof. By applying the preceding theorem, and Corollary 3 from the chapter notes to produce the trace, we have:

$$\mathbf{E}_{\vec{\epsilon}}\, G = \mathbf{E}_{\vec{\epsilon}}\, ||ARXV\vec{\epsilon} + AR\Lambda(b - w^*)||^2 + \sigma^2$$

$$= \mathbf{E}_{\vec{\epsilon}}\, ||ARXV\vec{\epsilon}||^2 + ||AR\Lambda(b - w^*)||^2 + \mathbf{E}_{\vec{\epsilon}}\, 2\vec{\epsilon}^\intercal V^\intercal X^\intercal R^\intercal A^\intercal AR\Lambda(b - w^*) + \sigma^2$$

$$= \mathbf{E}_{\vec{\epsilon}}\, ||ARXV\vec{\epsilon}||^2 + ||AR\Lambda(b - w^*)||^2 + \sigma^2$$

$$= \mathbf{E}_{\vec{\epsilon}}\, \vec{\epsilon}^\intercal V^\intercal X^\intercal R^\intercal A^\intercal ARXV\vec{\epsilon} + ||AR\Lambda(b - w^*)||^2 + \sigma^2$$

$$= \sigma^2 Tr[V^\intercal X^\intercal R^\intercal A^\intercal ARXV] + ||AR\Lambda(b - w^*)||^2 + \sigma^2$$

$$= \sigma^2 Tr[ARXVV^\intercal X^\intercal R^\intercal A^\intercal] + ||AR\Lambda(b - w^*)||^2 + \sigma^2$$

$$= \sigma^2 ||ARXV||^2_{Frob} + ||AR\Lambda(b - w^*)||^2 + \sigma^2.$$

□

This formula for the average generalization error has a very clean interpretation. The first term, $\sigma^2 ||ARXV||^2_{Frob}$, is the generalization error that is attributable to the label noise in the training data. In this case, the label noise (whether in the training set or testing set) has standard deviation σ. As σ goes to 0 this first term disappears, as expected, because then the training data has no label noise. What's more, as $||\Lambda||$ gets large, R tends to shrink, causing this term to shrink. Hence, $||\Lambda||$ can be interpreted as the magnitude of regularization, and as

this magnitude grows, the generalization becomes less impacted by label noise in the training data. However, even for $||\Lambda||$ fixed, the exact choice of Λ is going to influence the size of this term. Hence, the generalization error can be made smaller by selecting a Λ that reduces $||\sqrt{C+\mu\mu^\intercal}(XVX^\intercal+\Lambda)^{-1}XV||_{Frob}$, which means that, in an ideal situation, we would want Λ to depend on X as well as the true feature mean μ and true feature covariance matrix C. This provides a glimpse of why we may want Λ to depend both on our training and our prior knowledge simultaneously, suggesting the possibility of using data driven choices for Λ and b.

The second term, $||AR\Lambda(b-w^*)||^2$, is the generalization error attributable to bias in our model. This is the only term that b, our "bias vector" impacts. If b is perfect and equals the true solution coefficients w^* then this term disappears, as it also does if $||\Lambda|| \to 0$ (but typically at the cost of increasing the first term). So Λ and b both contribute bias, and either one can (at least theoretically) be chosen so that there is no contribution from this term. The term can hence be made smaller by a judicious choice of b and Λ, which might incorporate prior knowledge about w^*, as well as information about X, C and μ which all appear in this term. As $||\Lambda||$ grows larger, this term tends to grow, while the first term simultaneously shrinks, reflecting the tradeoff between bias and variance.

The third and final term, σ^2, is the simplest, and reflects the out of sample label noise. There is nothing that can be done to remove this source of error, other than collecting data in a manner that adds as little extra noise as possible (e.g. using the most accurate measurement tools available).

Recall that typically in linear regression settings the choice $b = 0$ is used, such as is done for Ridge Regression. But when will an alternative choice for b outperform the choice of 0? Is there something special about $b = 0$ that makes it preferable? These questions can

be addressed by comparing the generalization error when 0 is chosen versus when b takes its general form. We will have a lower generalization error for a non-zero b occurs precisely when:

$$\sigma^2||ARXV||^2_{Frob} + ||AR\Lambda(b - w^*)||^2 + \sigma^2 < \sigma^2||ARXV||^2_{Frob} + ||AR\Lambda w^*||^2 + \sigma^2.$$

We see that this happens when

$$||AR\Lambda(b - w^*)||^2 < ||AR\Lambda w^*||^2$$

or equivalently, when

$$(b - w^*)^\intercal Q(b - w^*) < w^{*\intercal} Q w^*$$

where $Q \equiv \Lambda^\intercal R^\intercal A^\intercal AR\Lambda$. Hence, we see that $b = 0$ is not necessarily a good choice, and in particular we want to choose b to make $b - w^*$ small, which $b = 0$ will only do if w^* is already small. However, the notion of "smallness" here is not a simple 2-norm, but rather it is smallness as measured by the norm induced by the matrix Q. Sometimes $b = 0$ will be an excellent choice, for instance, when it is likely that most solution coefficients are small, but if we have a way of guessing w^*, or estimating it from extra data, we may be much better off using this guess for b.

5.5 Kernelizing the Optimization Problem

The Generalized Ridge Regression optimization problem from Lemma 3 can be generalized further if we allow for the possibility that the points X^i may each be transformed by some feature map ϕ before using them for learning (introducing a kernelized form of the algorithm).

This produces a kernel Generalized Ridge Regression optimization problem:

$$w_\phi \equiv \underset{w\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} v_i(w^\intercal\phi(X^i) - y_i)^2 + (w-b)^\intercal\Lambda(w-b).$$

We define the kernel function κ which measures the similarity between any pair of points x_1 and x_2:

$$\kappa(x_1, x_2) \equiv \phi(x_1)^\intercal\phi(x_2)$$

and the $m \times m$ kernel matrix K whose ith, jth entry K_i^j represents the similarity between points i and j of X:

$$K \equiv [\kappa(X^i, X^j)]_{i,j=1}^m.$$

Finally, we define $\Phi \equiv \phi(X)$ to be the $n \times m$ matrix $[\phi(X^i)]_{i=1}^m$ constructed from applying ϕ to each point (i.e. column) of X, and note that then $K = \Phi^\intercal\Phi$.

Lemma 4. *The unique solution to the kernel Generalized Ridge Regression problem*

$$\underset{w\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} v_i(w^\intercal\phi(X^i) - y_i)^2 + (w-b)^\intercal\Lambda(w-b).$$

is given by

$$w_\phi = \left(\Phi V\Phi^\intercal + \Lambda\right)^{-1}(\Phi V y + \Lambda b).$$

Proof. This follows immediately from Lemma 3 and the definition of Φ. □

5.5.1 Point Regularization Matrix

Now we introduce some lemmas which will allow us to rewrite this expression in a form that converts dot products into kernel point similarity measurements, enabling us to finish kernelizing the algorithm. Basically, we want to replace our "feature regularization matrix" Λ with a "point regularization matrix" Ω that has the same effect when applied to the point similarity matrix $\Phi^\intercal\Phi$ that Λ does when applied to the feature similarity matrix $\Phi\Phi^\intercal$. In essence, the following lemma tells us that we can pass Φ through the inverse that shows up in the analytic solution for Generalized Ridge Regression, but doing so replaces the feature regularization matrix Λ with the point regularization matrix Ω.

Theorem 6. *For any real matrices Φ, Λ and Ω, the following relationship holds*

$$\big(\Phi\Phi^\intercal + \Lambda\big)^+\Phi = \Phi\big(\Phi^\intercal\Phi + \Omega\big)^+$$

whenever the equation

$$\Lambda\Phi = \Phi\Omega$$

is satisfied. Furthermore, when $\Phi\Phi^\intercal$ is invertible, the only Λ that satisfies this equation is

$$\Lambda = \Phi\Omega\Phi^+$$

and when $\Phi^\intercal\Phi$ is invertible, the only Ω that satisfies this equation is

$$\Omega = \Phi^+\Lambda\Phi.$$

Proof. The relationship

$$\Lambda\Phi = \Phi\Omega$$

is equivalent to

$$(\Phi\Phi^\intercal + \Lambda)\Phi = \Phi(\Phi^\intercal\Phi + \Omega)$$

which implies that

$$\Phi = (\Phi\Phi^\intercal + \Lambda)^+\Phi(\Phi^\intercal\Phi + \Omega)$$

which finally implies

$$\Phi(\Phi^\intercal\Phi + \Omega)^+ = (\Phi\Phi^\intercal + \Lambda)^+\Phi$$

as desired.

Now, in the case where $\Phi\Phi^\intercal$ is invertible, we have that

$$\Lambda\Phi = \Phi\Omega$$

implies

$$\Lambda\Phi\Phi^\intercal = \Phi\Omega\Phi^\intercal$$

which is equivalent to

$$\Lambda = \Phi\Omega\Phi^\intercal(\Phi\Phi^\intercal)^{-1} = \Phi\Omega\Phi^+.$$

On the other hand, when $\Phi^\intercal\Phi$ is invertible, we have that

$$\Lambda\Phi = \Phi\Omega$$

implies

$$\Phi^\intercal\Lambda\Phi = \Phi^\intercal\Phi\Omega$$

which is equivalent to

$$\Omega = (\Phi^\intercal \Phi)^{-1} \Phi^\intercal \Lambda \Phi = \Phi^+ \Lambda \Phi.$$

This completes the proof.

□

It is important to note that if we are given Λ and simply need to pick an Ω, and are not picky about which Ω we get, we can use:

$$\Omega = \Phi^+ \Lambda \Phi.$$

In the case where $\Phi^\intercal \Phi$ is invertible this will be a unique choice, but more generally it will not as adding a matrix M to Ω that is in the null space of Φ will preserve the property that $\Lambda\Phi = \Phi\Omega$, since then $\Phi(\Omega + M) = \Phi\Omega$.

We note that Ω is not always positive definite, or even symmetric, despite the fact that Λ is positive definite by assumption. For instance, consider this case:

$$\Phi = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

$$\Phi^+ = \Phi^{-1} = \begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}$$

$$\Lambda = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Omega = \begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix} \neq \Omega^\intercal.$$

It is also interesting to note that in the special case of Ridge Regression (where $\Lambda = \lambda I$), if we assume that $\Phi\Phi^\intercal$ is invertible, we obtain the positive definite matrix:

$$\Omega = \Phi^+\Lambda\Phi = \lambda\Phi^+\Phi = \lambda\Phi^\intercal(\Phi\Phi^\intercal)^{-1}\Phi.$$

as one of our solutions. This result may be surprising as we might expect to get $\Omega = \lambda I_m$, where I_m is the $m \times m$ identity matrix, as it is well known from the process of kernelizing Ridge Regression that

$$(\Phi\Phi^\intercal + \lambda I_n)^{-1}\Phi = \Phi(\Phi^\intercal\Phi + \lambda I_m)^{-1}.$$

But we recall that Ω does not have a unique solution, and $\Omega = \Phi^+\Lambda\Phi$ is simply a useful and general purpose choice. As noted previously, if we start with a valid Ω we can add to it any matrix that is in the null space of Φ and still have a valid Ω. In this case, we observe that the matrix $\lambda(I_m - \Phi^\intercal(\Phi\Phi^\intercal)^{-1}\Phi)$ is zero when left multiplied by Φ as required, and adding it to $\Omega = \lambda\Phi^\intercal(\Phi\Phi^\intercal)^{-1}\Phi$ gives λI_m.

5.5.2 Representer theorem form

We now complete our kernelizing process by showing that we can write our predictions in a form that explicitly depends on the similarity (as measured by our kernel function) between each point X^i and the point x that we are predicting the label of. This is the form given by

the representer theorem (Schlkopf et al., 2001).

Theorem 7. *Whenever the matrix $\Phi\Phi^\intercal$ is invertible, the solution to the kernel Generalized Ridge Regression optimization problem*

$$w_\phi \equiv \underset{w\in\mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m}(w^\intercal\phi(X^i) - y_i)^2 + w^\intercal\Lambda w$$

defines a linear function $l(x) \equiv w_\phi^\intercal\phi(x)$ which can be written

$$l(x) = \sum_{i=1}^{m} \alpha_i\, \kappa(x, X^i)$$

for kernel similarity function $\kappa(x_1, x_2) \equiv \phi(x_1)^\intercal\phi(x_2)$, $\alpha \equiv \big(K + \Omega\big)^{-1}y$, and Ω as defined in Lemma 6.

Proof. Using Lemma 4 and the fact that $\Lambda = \Phi\Omega\Phi^+$, followed by Lemma 6, followed by the definition of the kernel matrix K, we have that the Kernel Ridge Regression optimization problem is solved by the coefficient vector:

$$\begin{aligned}
w_\phi =& \big(\Phi\Phi^\intercal + \Lambda\big)^{-1}(\Phi y + \Lambda b)\\
=& \big(\Phi\Phi^\intercal + \Lambda\big)^{-1}(\Phi y + \Phi\Omega\Phi^+ b)\\
=& \big(\Phi\Phi^\intercal + \Lambda\big)^{+}\Phi(y + \Omega\Phi^+ b)\\
=& \Phi\big(\Phi^\intercal\Phi + \Omega\big)^{+}(y + \Omega\Phi^+ b)\\
=& \Phi\big(K + \Omega\big)^{+}(y + \Omega\Phi^+ b).
\end{aligned}$$

Defining the column vector

$$\alpha \equiv \big(K + \Omega\big)^{+}(y + \Omega\Phi^+ b),$$

the prediction that our linear model gives at point x can be written as:

$$\begin{aligned} l(x) \equiv \phi(x)^\intercal w_\phi =& \phi(x)^\intercal \Phi \big(K + \Omega\big)^+ (y + \Omega \Phi^+ b) \\ =& ([\kappa(x, X^i)]_{i=1}^m)^\intercal \alpha \\ =& \sum_{i=1}^m \alpha_i \, \kappa(x, X^i). \end{aligned}$$

□

Hence we see that whereas Λ is added to $\Phi\Phi^\intercal$ to regularize the solution coefficients w_ϕ, in the kernelized formulation the corresponding $\Omega = \Phi^+ \Lambda \Phi$ is added to $K = \Phi^\intercal \Phi$ to regularize the linear point coefficients α. As we have noted, passing from Λ to Ω is possible when $\Phi\Phi^\intercal$ is invertible, which requires at least as many points as features. But in the case where there are more features than points, we can think of the defining equation of our prediction algorithm as being $l(x) = \phi(x)^\intercal \Phi (K + \Omega)^+ y$, without ever actually needing to reference the quantity

$$w_\phi = (\Phi\Phi^\intercal + \Lambda)^{-1} (\Phi y + \Lambda b).$$

That corresponds to the previously mentioned case where we define Λ in terms of Ω, instead of the reverse.

We originally named b the bias because the solution coefficients get biased towards b. But the equation

$$\begin{aligned} w_\phi &= \Phi \big(K + \Omega\big)^+ (y + \Omega \Phi^+ b) \\ &= \Phi \big(K + \Omega\big)^+ (y + \Phi^+ \Lambda \Phi \Phi^+ b) \\ &= \Phi \big(K + \Omega\big)^+ (y + \Phi^+ \Lambda \Phi \Phi^\intercal (\Phi\Phi^\intercal)^{-1} b) \end{aligned}$$

$$= \Phi\big(K + \Omega\big)^{+}(y + \Phi^{+}\Lambda b)$$

shows that b is a bias in another sense as well. The effect of b is equivalent to simply adding $\Omega\Phi^{+}b$ or equivalently of adding $\Phi^{+}\Lambda b$ to the vector of labels, y.

5.5.3 Regularization as a Change in Kernel

We now show that generalizing the regularization term $||w||_2^2$ to $w^{\intercal}\Lambda w$ and adding point weights v_i to our optimization are equivalent to solving the ordinary Ridge Regression problem except with a certain change to our kernel function and transformation to our labels y. In other words, changes of the regularization matrix Λ can be thought of as a type of kernel change.

Theorem 8. *The linear function $l(x) \equiv w_{\phi}^{\intercal}\phi(x)$ that solves the Kernel Ridge Regression problem with positive definite regularization matrix Λ, feature map ϕ, and point weights v_i, whose linear coefficients are given by:*

$$w_{\phi} \equiv \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} v_i (w^{\intercal}\phi(X^i) - y_i)^2 + w^{\intercal}\Lambda w$$

is identical to the linear function $l(x) \equiv w_{\psi}^{\intercal}\psi(x)$ that solves the Kernel Ridge Regression problem with standard L_2 norm based regularization, feature map ψ, and label map η , whose linear coefficients are given by:

$$w_{\psi} \equiv \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} (w^{\intercal}\psi(X^i) - \eta(y_i))^2 + ||w||_2^2$$

where $\psi(X^i) = \sqrt{v_i}\, L^{-1}\phi(X^i)$, $LL^{\intercal} = \Lambda$ and $\eta(y_i) = \sqrt{v_i}y_i$.

Proof. By assumption, the regularization matrix Λ is positive definite. Hence, there is a

unique, real, lower triangular matrix L with positive diagonal entries such that

$$LL^\intercal = \Lambda \tag{5.2}$$

which is found via the Cholesky decomposition of Λ. Note that if Λ is positive semi-definite but not positive definite, L will still exist, though it will not be invertible and so will not work for our purposes.

Now, we perform a change of variables $w = (L^\intercal)^{-1}u$, writing:

$$\begin{aligned}
w_\phi &\equiv \operatorname*{argmin}_w \sum_{i=1}^{m} v_i(w^\intercal\phi(X^i) - y_i)^2 + w^\intercal\Lambda w \\
&= \operatorname*{argmin}_w \sum_{i=1}^{m} v_i(w^\intercal\phi(X^i) - y_i)^2 + w^\intercal LL^\intercal w \\
&= \operatorname*{argmin}_w \sum_{i=1}^{m} v_i(w^\intercal\phi(X^i) - y_i)^2 + ||L^\intercal w||_2^2 \\
&= \operatorname*{argmin}_u \sum_{i=1}^{m} v_i\left(((L^\intercal)^{-1}u)^\intercal\phi(X^i) - y_i\right)^2 + ||L^\intercal(L^\intercal)^{-1}u||_2^2 \\
&= \operatorname*{argmin}_u \sum_{i=1}^{m} (u^\intercal\sqrt{v_i}\, L^{-1}\phi(X^i) - \sqrt{v_i}\, y_i)^2 + ||u||_2^2 \\
&= \operatorname*{argmin}_u \sum_{i=1}^{m} (u^\intercal\psi(X^i) - \eta(y_i))^2 + ||u||_2^2
\end{aligned}$$

So solving the Ridge Regression problem with a full regularization matrix Λ and weights v_i using feature map ϕ is equivalent to solving the Ridge Regression problem using standard L_2 regularization (with $\lambda = 1$) under the feature map ψ and label map η. □

5.6 Point Removal

How much flexibility does choosing a regularization matrix give us? Is it enough, for instance, to effectively remove one of the data points (say, an outlier) from our training set, without impacting the rest of the point predictions? As we will see, the answer is yes, and this can be achieved using just a rank-1 update to the regularization matrix. This is very different than the Ridge Regression case, where the only way to remove the influence of a point that has a very large feature values is to make λ very large, which will have a profound negative impact on the predictions for the other points.

As we will see in the following theorem, if we have a regularization matrix $\widetilde{\Lambda}$, we can change it to a new regularization matrix Λ via a rank-1 update such that our predictions are exactly as though they would have been had one of the points in our data set not been there, and we had used the regularization matrix $\widetilde{\Lambda}$ for training on that modified data set (which lacks that point). We will assume, without loss of generality, that it is the first point in the data set that is being removed.

Theorem 9. *Let w and $\widetilde{w}$ be the solution coefficients for two Generalized Ridge Regression problems, with regularization matrices Λ and $\widetilde{\Lambda}$ respectively, trained on the features and labels (X, y) and $(\widetilde{X}, \widetilde{y})$ respectively, where $\widetilde{X}$ is just X with the first point (i.e. column) removed, and $\widetilde{y}$ is just y with the first point (i.e. row) removed. Furthermore, let $C \equiv XX^\intercal$, $\widetilde{C} \equiv \widetilde{X}\widetilde{X}^\intercal$, $R \equiv (C + \Lambda)^{-1}$ and $\widetilde{R} \equiv (\widetilde{C} + \widetilde{\Lambda})^{-1}$, so that $w = RXy$ and $\widetilde{w} = \widetilde{R}\widetilde{X}\widetilde{y}$. Then*

$$w = \widetilde{w} \quad when \quad \Lambda\widetilde{w} = Xy - C\widetilde{w},$$

which for $n > 1$ leaves us an infinite number of choices for Λ. If we further require Λ to be such a choice of regularization matrix to ensure $w = \widetilde{w}$ while minimizing the Frobenius norm

$||\Lambda - \widetilde{\Lambda}||_F$, *which is the sum of the squared differences between elements of* Λ *and* $\widetilde{\Lambda}$, *then we have a unique solution:*

$$\Lambda = \widetilde{\Lambda} + \frac{u\,\widetilde{w}^{\intercal}}{\widetilde{w}^{\intercal}\widetilde{w}}$$

with

$$u = Xy - (C + \widetilde{\Lambda})\widetilde{w}$$

and so Λ *represents a rank-1 update to the matrix* $\widetilde{\Lambda}$, *which is sufficient to completely remove the effect of one point.*

This choice of Λ ensures that we exactly reproduce the predictions we would have gotten had we been using $\widetilde{\Lambda}$ for regularization and had the first point (X^1, y_1) not in our data set, while still trying to keep Λ as close to $\widetilde{\Lambda}$ as possible to distort the original problem as little as possible.

Proof. We want to set Λ so that

$$w = (C + \Lambda)^{-1}Xy = \widetilde{w}$$

That's equivalent to showing that:

$$Xy = (C + \Lambda)\widetilde{w}$$

or

$$\Lambda\widetilde{w} = Xy - C\widetilde{w}$$

as required. Now, we want to find the choice of Λ satisfying the above equation that minimizes

$||\Lambda - \widetilde{\Lambda}||_F^2$. Let us use $q \equiv Xy - C\widetilde{w}$ so that our constraints can be written $\Lambda\widetilde{w} - q = 0$. Then, introducing a vector ρ of Lagrange multipliers to reflect our constraints, we differentiate the Lagrangian with respect to the elements of Λ subject to the constraints and set the result equal to zero, writing:

$$\frac{d}{d\Lambda_{rs}} \; ||\Lambda - \widetilde{\Lambda}||_F^2 + \rho^\intercal(\Lambda\widetilde{w} - q)$$

$$= \frac{d}{d\Lambda_{rs}} \; \sum_{i=1}^{n}\sum_{j=1}^{n}(\Lambda_{ij} - \widetilde{\Lambda}_{ij})^2 + \sum_{k=1}^{n}\rho_k\Big(\sum_{t=1}^{n}\Lambda_{kt}\widetilde{w}_t - q_k\Big)$$

$$= 2(\Lambda_{rs} - \widetilde{\Lambda}_{rs}) + \rho_r\widetilde{w}_s = 0.$$

Hence, we find that:

$$\Lambda_{rs} = \widetilde{\Lambda}_{rs} - \frac{1}{2}\rho_r\,\widetilde{w}_s$$

and so

$$\Lambda = \widetilde{\Lambda} - \frac{1}{2}\rho\,\widetilde{w}^\intercal$$

which we see is a rank-1 update to $\widetilde{\Lambda}$. Now, we need to solve for the Lagrange multipliers ρ, so we apply our constraint that $\Lambda\widetilde{w} = q$ by multiplying both sides of the equation by $\widetilde{w}$ on the right. This gives us

$$q = \Lambda\widetilde{w} = \widetilde{\Lambda}\widetilde{w} - \frac{1}{2}\rho\,\widetilde{w}^\intercal\widetilde{w}.$$

which means that

$$\rho = \frac{2}{\widetilde{w}^\intercal\widetilde{w}}(\widetilde{\Lambda}\widetilde{w} - q) = \frac{2}{\widetilde{w}^\intercal\widetilde{w}}(\widetilde{\Lambda}\widetilde{w} + C\widetilde{w} - Xy) = \frac{2}{\widetilde{w}^\intercal\widetilde{w}}((\widetilde{\Lambda} + C)\widetilde{w} - Xy)$$

Therefore

$$\Lambda = \widetilde{\Lambda} + \frac{(Xy - (C + \widetilde{\Lambda})\widetilde{w})\,\widetilde{w}^\intercal}{\widetilde{w}^\intercal\widetilde{w}}$$

as required. Note, that the resulting matrix Λ may or may not be positive definite, depending

on the training data. However, even if it is not positive definite, the solution coefficients w based on Λ can still be defined using $w = (C + \Lambda)^{-1}Xy$ so long as the inverse exists. Alternatively, we could have chosen a positive definite Λ to remove the effect of the point in question, but in that case it may not be representable with a rank-1 update to $\widetilde{\Lambda}$. $\square$

5.7 Choices for the Regularization Matrix

As we have seen, generalizing the penalty in Ridge Regression to a full regularization matrix Λ provides a lot of flexibility. But what sort of choices for Λ other than $\Lambda = \lambda I$ might be useful in practice?

It is often desirable that our regularization matrix has a free "complexity parameter" $\alpha \geq 0$, like the λ of Ridge Regression, which allows us to match the strength of our regularization to the complexity of the problem at hand. Since the right choice of this parameter is very difficult to derive theoretically, and depends on aspects of the underlying problem like the noise level, number of features, and sample size, we generally learn this parameter from the data itself (e.g. using k-fold cross-validation). This parameter α is designed to tradeoff between bias and variance. When $\alpha = 0$ we want minimum bias, so generally we will want our algorithm to become equivalent to Ordinary Least Squares regression. At the other extreme as $\alpha \to \infty$, we want the solution will depend minimally on the data and therefore to have low variance, being driven entirely or almost entirely by some kind of baseline assumption or prior (e.g. that few features are important or that large coefficients are unlikely).

With this in mind, we will now introduce some families of regularization matrices Λ, each containing a free complexity parameter α.

5.7.1 Diagonal Λ Based on Feature Badness

When Λ is a diagonal matrix, its action is to penalize individual features different amounts. This can be useful when one has a way to determine which features are more likely to be reliable than others, or which coefficients are less likely to be close to zero than others.

Suppose that for each feature X_j we have a measure z_j of how "bad" we expect this feature to be, relative to the other features. This could be reflective of each feature's estimated measurement noise, our prior probability on it having a large value, whether that feature is correlated with the other features, or something else. We call z_j the "feature badness" of the ith feature. Then, we can use a diagonal matrix for Λ with $\Lambda_{ii} = \kappa_\alpha(z_i)$, where κ_α is a function that maps the feature badness z_i for the ith feature into the penalty that we will be applying for that feature. Here α is our complexity parameter.

We can think of the two extreme endpoints of α (i.e. $\alpha \to \infty$ and $\alpha = 0$) as determining whether we will only use the most reliable feature (ignoring all the others), or whether we will treat all features equally (ignoring how bad each feature is, which means we are using Ordinary Least Squares). When α is between these two extremes we will trust the bad features less than the better features by penalizing the bad ones more, but all features will get used. In essence, α prevents us from overfitting by reducing how much we trust potentially unreliable features. This encompasses the idea of reducing the dimension of our problem by removing features (i.e. feature selection), but it also includes a soft and more general form of feature selection. Rather than dropping features, it allows us to trust features different amounts. Those that seem very bad might end up being almost entirely removed, but all features may get used to some extent, or all features worse than some cutoff value could be removed completely. Any traditional feature selection algorithm, as long as it assigns a score

to each feature based on how bad or useless that feature seems, could be used as the measure of feature badness in order to apply this approach.

In the simplest case, we will have $\kappa_\alpha(z_i) = \alpha \ \forall i$, which reproduces Ridge Regression (using α in place of the usual λ). This corresponds to a prior that larger coefficients are less likely than small ones, while treating all coefficients equally. A somewhat more interesting choice would be $\kappa_\alpha(z_i) = \alpha z_i \ \forall i$, where in this simple case we assume the z_i are constants independent of the training data. This would be appropriate when we have a priori knowledge of how likely each coefficient is to be large, which we encode in the z_i. However, it is worth noting that the unit or scale of each feature matters here, so it is common to first normalize the features (e.g. by subtracting the training data feature means from each training and testing feature, and dividing by the training data feature standard deviations). Then the z_i do not have to take into account the scale or units of the features.

In fact, if we assume the features are centered so that each has mean 0, then we can think of Ridge Regression, after the features have been normalized to have unit variance, as being simply a case of using a diagonal regularization matrix (without feature normalization). To see why, note how regularizing the kth feature with the feature dependent penalty $\frac{\lambda}{\sigma_k^2}$ ends up being equivalent to normalizing each feature by multiplying X_k by a constant σ_k (through a change of variables $r_k = w_k/\sigma_k$):

$$\min_w \sum_{i=1}^{m} (X^{i\intercal} w - y)^2 + \sum_{k=1}^{n} \frac{\lambda}{\sigma_k^2} w_k^2$$

$$= \min_w \sum_{i=1}^{m} (\sum_{k=1}^{n} X_k^{i\intercal} w_k - y)^2 + \sum_{k=1}^{n} \frac{\lambda}{\sigma_k^2} w_k^2$$

$$= \min_{w} \sum_{i=1}^{m} (\sum_{k=1}^{n} \sigma_k X_k^{i\intercal} \frac{w_k}{\sigma_k} - y)^2 + \sum_{k=1}^{n} \frac{\lambda}{\sigma_k^2} w_k^2$$

$$= \min_{r} \sum_{i=1}^{m} (\sum_{k=1}^{n} \sigma_k X_k^{i\intercal} r_k - y)^2 + \lambda \sum_{k=1}^{n} r_k^2.$$

So, in a sense, regularization with a diagonal matrix is very popular, as it occurs implicitly whenever we divide our features by constants to normalize them. It is worth noting that in Ordinary Least Squares, normalization of features has no effect. When we move to Ridge Regression, normalizing features only has an impact precisely because it distorts the penalty of our regularization. This choice of normalization can end up mattering a lot.

Let us consider an extreme example, where the choice of feature normalization (or equivalently, of an appropriate diagonal regularization matrix) can be essential for achieving adequate prediction performance. Suppose that the true coefficients of the linear model that generated our data are equal to 1 for the first $n/2$ features, and equal to ρ for the remaining $n/2$ features, where ρ is some large number. Furthermore, suppose that those first $n/2$ features of our training data all have standard deviations of 1, and last $n/2$ features have standard deviations of $1/\rho$. Finally, suppose that the number of features is large compared to the number of points, or that the label noise is high, so that a significant amount of regularization will be needed to prevent overfitting. Now, here's the conundrum. In this setup, all features contribute to the final labeling of a point about the same amount (since a huge coefficient multiplied by values with a correspondingly tiny standard deviation lead to values approximately equal to 1). If we choose our Ridge Regression regularization parameter λ to be large in order to prevent overfitting adequately, then it wipes out the features with standard deviations equal to $1/\rho$, since the tiny standard deviation magnifies the impact of regularization. On the other

hand, if we do not make our regularization strong, then we overfit our sample. We get poor performance either way.

5.7.2 Feature Penalty Functions

We have previously discussed the idea of introducing a function $\kappa_\alpha(z)$ to map a measure of how bad a feature is (i.e. z) into a regularization penalty for that feature, and we have introduced very simple cases. But more generally, what properties would we want such a function κ_α to have? We propose that it is useful to have such a function satisfy the following properties, which we use as the definition of a "Feature Penalty Function."

Definition of a Feature Penalty Function

We would like our choice of function $\kappa_\alpha(z)$ to have the following properties. Any function with these properties we will call a Feature Penalty Function.

1. $\kappa_\alpha(z)$ is defined on the domain $z \geq 0$ (since negative badness does not make sense in this context) and $\alpha \geq 0$ (since a negative regularization strength also does not make sense).

2. $\kappa_\alpha(z)$ should be a non-decreasing function of z, since features of greater badness should have a penalty at least as large as features of less badness.

3. $\lim_{z\to\infty} \kappa_\alpha(z) = \infty$, since an infinitely bad feature should have infinite penalty so that it is ignored completely.

4. $\kappa_\alpha(0) = 0$, since a feature with no badness should be trusted fully, and therefore not penalized at all.

5. $\lim_{\alpha \to \infty} \kappa_\alpha(z) = \infty$ when $z > 0$, since as α goes to ∞ we maximally distrust all features, so we have no trust at all (i.e. infinity penalties) for all features that have any badness at all.

6. $\kappa_0(z) = 0$, since with $\alpha = 0$ we completely ignore how bad each feature is, so no penalty is given to any point (which means we are performing Ordinary Least Squares regression).

There are infinitely many possibilities for feature penalty functions, but here we give two convenient examples.

Lemma 5. *The functions*

$$\kappa_\alpha(z) = \Big(z + 1\Big)^\alpha - 1$$

and

$$\kappa_\alpha(z) = \alpha\, z$$

are both Feature Penalty Functions.

The proof is trivial, it simply involves evaluating each of these two functions against each of the relevant properties. We note that for z close to 0 these two Feature Penalty Functions closely agree, as the second function is the linear taylor series approximation of the first function.

Note that if our choice of how we measure badness (and therefore, how we define z) does not naturally cover the range $[, \infty)$, it may be beneficial to transform it so that it does cover the full range.

5.7.3 Measuring Feature Badness

5.7.3.1 One Dimensional Prediction Error

How can we measure the badness of a feature in practice? One approach is to predict our labels using each feature on its own (without any of the other features) and use the prediction error for the ith feature as our value for z_i. This makes use of the intuition that a feature which performs well on its own is more likely to be useful (when combined with other features) than one which does not. If we think in terms of priors, we have replaced the Ridge Regression prior (that coefficients are small) with a new prior which says that a feature is more likely to be useful in our final model if it is useful when it is used alone, and hence it should receive less penalty than other features. The complexity parameter $\alpha \geq 0$ regulates the strength of our prior, just like $\lambda \geq 0$ determines the strength of the prior in Ridge Regression.

In keeping with the above, if we are using linear (i.e. non-kernelized) regression, we could use the linear prediction error for each feature's badness. In that case, the badness would be:

$$z_i \equiv Var(y) - \frac{Cov(X_i, y)^2}{Var(X_i)}$$

where $Var(y)$ is the variance of a vector y, and $Cov(X_i, y)$ is the covariance between vectors X_i and y. We would then apply a Feature Penalty Function to convert these badness scores into penalties. However, since the range of the z_i is not $[0, \infty)$ we might want to transform first, for instance by dividing by $Var(y)$ to put it into the range $[0, 1]$, which gives simply $1 - Cor(X, y)^2$, and then applying a transformation such as $z \to \frac{1}{1-z} - 1 = \frac{1}{Cor(X,y)^2} - 1$ to

make the range $[0, \infty]$. Note that in the kernelized case, it is useful to consider non-linear predictions instead of just linear ones. For instance, in a polynomial case we could fit a one dimensional kth degree polynomial of the ith feature against our labels, and use the error of those predictions for our z_i.

5.7.3.2 Relatedness to Other Features

Another approach to defining the feature badness is to consider how related each feature is to every other feature. For example, we can define:

$$z_i = 1 - \frac{1}{n-1} \sum_{j \neq i}^{n} |Cor(X_i, X_j)|^p.$$

were $Cor(X_i, X_j)$ is the correlation between vector X_i and vector X_j and $p > 0$. This measures how correlated each feature is to all the other features, and assigns a high badness if the feature is highly correlated to the others. The larger p is, the more the largest correlations a feature has matters compared to its smaller correlations. Given an equation such as this one, we might choose to transform it to the range $[0, \infty)$. Regardless though, we would then apply a Feature Penalty Function to map the scores into penalties. This choice of how to measure badness will reduce the reliance on features that do not have a strong linear relationship to any other features in the data (whether positive or negative). This can be useful because in some circumstances a feature having little to do with the other features makes it less likely to be useful in making predictions. For instance, this can occur because features that are dominated by noise rather than signal will not be strongly related to the other features, or because features that appear only rarely (e.g. rare words in text based features) will have low correlations to other features and also be of little use in making predictions. In other cases information that is important will tend to crop up in multiple features, and so a feature having low similarity to the other features is a sign of low importance.

In other situation one might want to define a badness score that is exactly the reverse of the one above, taking $z_i \to 1 - z_i$ in the previous formula. This new notion of badness may be useful in a setting where many features are redundant, and it is important to hone in on the most unique information, so we want to give unique information the least penalty. Sometimes high correlation to other features makes a feature more likely to be useful, and in other cases it makes it more likely to be useless. This illustrates that, inevitably, the choice of how to define feature "badness" is dependent on the prior information we have. Choosing the right regularization for a problem is an exercise in asking yourself what you believe you already know about that problem.

5.7.4 Principal Component Based Λ

We look now at another special choice of Λ, one that induces behaviors that have the flavor of principal component analysis. Consider the singular value decomposition of our training features matrix X. We write:

$$X = U\Sigma Q$$

where U and Q are orthogonal matrices and Σ is a diagonal matrix with all non-negative diagonal elements. We call U the principal component eigenvectors of X. We are going to assume that $m > n$ (i.e. there are more points than features) and that Σ has all positive diagonal elements. This allows us to make an unconventional choice for the shapes of the matrices. Namely we choose U and Σ to both be of size $n \times n$, so U and Σ are both invertible. This means that Q has size $n \times m$ and that Q is therefore not invertible.

Theorem 10. *Let Λ_o be any positive definite matrix and let the columns of U be the principal component eigenvectors of X. Assume that the singular values of X are all positive. Then, in*

Generalized Ridge Regression, using the regularization matrix

$$\Lambda = U\Lambda_o U^\intercal$$

is equivalent to performing Generalized Ridge Regression with regularization matrix Λ_o on a new set of features, equivalent to the original features in our data set once they have been projected onto the principal component eigenvectors.

Proof. By definition, the principal component eigenvectors are given by the columns of U, and the projection coefficients that we get from projecting each point in X onto each of them is given by $U^\intercal X = U^\intercal U \Sigma Q = \Sigma Q$. So performing Generalized Ridge Regression with the projected coefficients as features, where we use the matrix of eigenvectors U to project the point x_0 that we are going to be predicting, would give us the following quantity for the solution coefficients w_{pca} applied to predict the point x_0:

$$x_0^\intercal w_{pca} = x_0^\intercal U(\Sigma Q(\Sigma Q)^\intercal + \Lambda_o)^{-1}(\Sigma Q)y$$

$$= x_0^\intercal U(U^\intercal X(U^\intercal X)^\intercal + \Lambda_o)^{-1}(U^\intercal X)y$$

$$= x_0^\intercal U(U^\intercal X X^\intercal U + U^\intercal U\Lambda_o U^\intercal U)^{-1}(U^\intercal X)y$$

$$= x_0^\intercal U U^\intercal (X X^\intercal + U\Lambda_o U^\intercal)^{-1} U(U^\intercal X)y$$

$$= x_0^\intercal (X X^\intercal + U\Lambda_o U^\intercal)^{-1} X y$$

$$= x_0^\intercal (X X^\intercal + \Lambda)^{-1} X y.$$

□

Hence, we see that we can choose our regularization to operate on the PCA feature space instead of the original feature space. And if we choose Λ_o to be a diagonal matrix,

we can directly control the penalty applied to the PCA eigenvector features. For instance, applying ideas previously discussed, we can assign a badness to each PCA feature based on the one dimensional prediction accuracy we can achieve with each PCA feature. Or we can penalize each of these features based on the corresponding singular values. Larger singular values in the principal component analysis are associated with greater variance captured by a feature, so we can, for instance, think of the reciprocal of each singular value as a measure of badness, and then apply a Feature Penalty Function to map these badness values into penalties.

How does a PCA based approach to regularization differ from Ridge Regression? This difference becomes strongly apparent when we consider what happens in the presence of features that are repeated, or nearly repeated. In the principal component analysis case, nearly identical features will get collapsed together into a single new feature. We now investigate however what happens with Ridge Regression in the presence of repeated features, which is quite different than the PCA case.

5.7.5 Repeated Features in Ridge Regression

We now consider what happens when our data set contains repeated features, and how the solution coefficients relate to what would have been achieved if those features were not repeated. As we will see in the lemma, repeated features X_j have their Ridge Regression penalty weakened by a factor $\frac{1}{r_i}$ compared to if the feature had only appeared once.

Lemma 6. *Let X be an n by m matrix of training points, with one point per column and one feature per row. Let the matrix $\bar{X}$ be identical to the matrix X, except with each feature X_j of X included exactly $r_j \geq 1$ times. Therefore $r = \sum_{j=1}^{n} r_j$ is the total number of features in $\bar{X}$. Finally, let x be any vector of length n (the point we will be making a label prediction for) and let $\bar{x}$ be a vector of length r thats consist purely of values from x but with x_j repeated $r_j \geq 1$*

times, with each value $\bar{x}_k$ being a repetition of x_j precisely whenever feature $\bar{X}_k$ is a repetition of feature X_j. Then the prediction function that is the solution to the Ridge Regression problem with training data $(\bar{X}, y)$ will assign the same predicted label to point $\bar{x}$ that the solution to the Generalized Ridge Regression problem with training data (X, y) and diagonal regularization matrix $\Lambda = \text{diag}(\frac{\lambda}{r_j})_{i=j}^n$ assigns to point x. In other words, $\bar{w}^{*\intercal}\bar{x} = w^{*\intercal}x$, where

$$\bar{w}^* \equiv \underset{\bar{w}\in\mathbb{R}^r}{\text{argmin}} \sum_{i=1}^{m}(\bar{w}^\intercal \bar{X}^i - y_i)^2 + \lambda \sum_{k=1}^{r} \bar{w}_k^2 \tag{5.3}$$

and

$$w^* \equiv \underset{w\in\mathbb{R}^n}{\text{argmin}} \sum_{i=1}^{m}(w^\intercal X^i - y_i)^2 + \lambda \sum_{j=1}^{n} \frac{{w_j}^2}{r_j}.$$

Furthermore, $\bar{w}_k^* = \frac{w_j^*}{r_j}$ whenever the feature $\bar{X}_k$ is identical to the feature X_j.

This means that the linear function solving the optimization problem (5.3) where there are repeated features is the same one that would have been achieved if each repeated feature had occurred only once, except that the regularization penalty on each repeated feature is reduced. The penalty strength is $\frac{\lambda}{r_j}$ instead of the usual λ due to the feature being repeated $r_j \geq 1$ times. So repeated features act just like non-repeated features, except that they are pushed less strongly towards 0. Hence, if the algorithm needs to push one of the repeated features to 0 to achieve high prediction accuracy, a larger λ must be selected than if the feature had not been repeated.

Proof. We observe that the Ridge Regression optimization problem (5.3) for training data $\bar{X}$ does not change at all if we swap variable $\bar{w}_j$ for variable $\bar{w}_k$ so long as features $\bar{X}_j$ and $\bar{X}_k$ are identical. What's more, the final values $\bar{w}_j^*$ and $\bar{w}_k^*$ for these variables must be the same, due to the complete symmetry of how such variables appear in the optimization problem. We use the vector w^+ to represent the subset of variables from $\bar{w}$ corresponding to only the

unique features of $\bar{X}$, so w_j^+ corresponds to feature X^j, which we recall appears r_j times in $\bar{X}$. Define the functions $\bar{F}$ and F:

$$\bar{F}(\bar{w}) \equiv \sum_{i=1}^{m} \left(\bar{w}^\intercal \bar{X}^i - y_i\right)^2 + \lambda \sum_{k=1}^{r} \bar{w}_k^2$$

$$F(w) \equiv \sum_{i=1}^{m} \left(w^\intercal X^i - y_i\right)^2 + \lambda \sum_{j=1}^{n} \frac{w_j^2}{r_j}$$

Exploiting the symmetry for variables corresponding to the same (repeated) feature, and then applying a change of variables $w_j = r_j {w_j}^+$, we write:

$$\min_{\bar{w}\in\mathbb{R}^r} \bar{F}(\bar{w}) = \min_{\bar{w}\in\mathbb{R}^r} \sum_{i=1}^{m} \left(\bar{w}^\intercal \bar{X}^i - y_i\right)^2 + \lambda \sum_{k=1}^{r} \bar{w}_k^2$$

$$= \min_{\bar{w}\in\mathbb{R}^r} \sum_{i=1}^{m} \Big(\sum_{k=1}^{r} \bar{w}_k \bar{X}_k^i - y_i\Big)^2 + \lambda \sum_{k=1}^{r} \bar{w}_k^2$$

$$= \min_{w^+\in\mathbb{R}^r} \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} r_j {w_j}^+ X_j^i - y_i\Big)^2 + \lambda \sum_{j=1}^{n} r_j {w_j^+}^2$$

$$= \min_{w\in\mathbb{R}^r} \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} w_j X_j^i - y_i\Big)^2 + \lambda \sum_{j=1}^{n} \frac{w_j^2}{r_j}$$

$$= \min_{w\in\mathbb{R}^r} \sum_{i=1}^{m} \left(w^\intercal X^i - y_i\right)^2 + \lambda \sum_{j=1}^{n} \frac{w_j^2}{r_j} = \min_{w\in\mathbb{R}^r} F(w).$$

This shows us that if $\bar{w}^*$ is the minimizer of $\bar{F}(\bar{w})$ then w^* is the minimizer of $F(w)$, where $w_j^* = r_j \bar{w}_k^*$ whenever $\bar{X}_k$ is a repetition of the feature X_j, or using our previous notation, that $w_j^* = r_j {w^+}_j^*$. This relationship implies that for x and $\bar{x}$ as defined in the statement of the lemma:

$$\bar{w}^{*\intercal} \bar{x} = \sum_{k=1}^{r} \bar{w}_k^* \bar{x}_k = \sum_{j=1}^{n} r_j w_j^+ x_j = \sum_{j=1}^{n} w_j^* x_j = w^{*\intercal} x$$

This completes our proof. □

5.8 Conclusion

We have seen that generalizing the 2-norm regularization term $||w||^2$ used in least squares regression to a full quadratic form $(w-b)^\intercal \Lambda (w-b)$ provides a flexible, and analytically tractable way of encoding prior knowledge into regression problems, and encompasses a wide range of algorithmic behaviors.

5.9 Chapter Notes

Lemma 7. *When z is a vector valued random variable that follows a multivariate normal distribution with mean equal to the zero vector, and covariance matrix equal to the identity matrix (so each z_i is independent of the other z_j), then for any matrix M:*

$$\mathbf{E}_z[z^\intercal M z] = Tr[M]$$

where $\mathbf{E}_z[.]$ is the expected value with respect to z, and $Tr[.]$ is the matrix trace.

Proof.

$$\mathbf{E}_z[z^\intercal M z] = \mathbf{E}_z[\sum_{i,j} M_i^j z_i z_j] = \sum_{i,j} M_i^j \mathbf{E}_z[z_i z_j] = \sum_i M_i^i \mathbf{E}_z[z_i^2]$$

$$= \sum_i M_i^i Var[z_i]. = \sum_i M_i^i = Tr[M_i^i].$$

□

Corollary 3. *When x is a random variable that follows a multivariate normal distribution*

with mean μ and positive definite covariance matrix C, then for any matrix M:

$$\mathbf{E}_x[x^\intercal Mx] = Tr[(C + \mu\mu^\intercal)M]$$

Proof.

$$\mathbf{E}_x[x^\intercal Mx] = \mathbf{E}_x[(x - \mu + \mu)^\intercal M(x - \mu + \mu)]$$

$$= \mathbf{E}_x[((x - \mu)^\intercal + \mu^\intercal)M(x - \mu) + ((x - \mu)^\intercal + \mu^\intercal)M\mu]$$

$$= \mathbf{E}_x[(x - \mu)^\intercal M(x - \mu) + \mu^\intercal M(x - \mu) + (x - \mu)^\intercal M\mu + \mu^\intercal M\mu]$$

$$= \mathbf{E}_x[(x - \mu)^\intercal M(x - \mu)] + \mu^\intercal M\mu$$

$$= \mathbf{E}_x[(C^{-1/2}(x - \mu))^\intercal C^{1/2}MC^{1/2}C^{-1/2}(x - \mu)] + \mu^\intercal M\mu$$

$$= \mathbf{E}_{z\sim Norm(0,I)}[z^\intercal C^{1/2}MC^{1/2}z] + \mu^\intercal M\mu = Tr[C^{1/2}MC^{1/2}] + \mu^\intercal M\mu$$

$$= Tr[CM] + Tr[\mu^\intercal M\mu] = Tr[CM] + Tr[\mu\mu^\intercal M] = Tr[(C + \mu\mu^\intercal)M]$$

where we have made use of Lemma 7 to introduce the trace $Tr[.]$. □

Corollary 4. *When x is a random variable that follows a multivariate normal distribution with mean μ and positive definite covariance matrix C, then for any vector v:*

$$\mathbf{E}_x\,(x^\intercal v)^2 = ||\sqrt{C + \mu\mu^\intercal}\,v||^2$$

where $\sqrt{C + \mu\mu^\intercal}$ is the positive definite square root of the positive definite matrix $(C + \mu\mu^\intercal)$, and $||.||$ means the 2-norm of a vector.

Proof.

$$\mathbf{E}_x(x^\intercal v)^2 = \mathbf{E}_x x^\intercal vv^\intercal x = Tr[(C + \mu\mu^\intercal)vv^\intercal]$$

$$= v^{\intercal}(C + \mu\mu^{\intercal})v = v^{\intercal}\sqrt{C + \mu\mu^{\intercal}}\sqrt{C + \mu\mu^{\intercal}}v = ||\sqrt{C + \mu\mu^{\intercal}}\, v||^2$$

where we have made use of Corollary 3 to introduce the trace $Tr[.]$.

□

Chapter 6

Impatient Learning for Least Squares Regression

We have spent a substantial amount of time so far investigating Ordinary Least Squares regression, which is probably the most popular prediction algorithm on earth, and Ridge Regression, which is the most common way to regularize Ordinary Least Squares. As we saw for empirical examples in Chapter 1, Ridge Regression corrects a number of problems that can trip up Ordinary Least Squares. Ridge Regression's biggest advantages are that it prevents overfitting, handles noisy labels and correlated features well, and can be kernelized to make non-linear predictions.

However, Ridge Regression also has some drawbacks. Just to form the covariance matrix $XX^\intercal$ takes time on the order of mn^2 (when non-kernelized) which can be too large for large datasets that have many features. Plus, common implementations that use k-fold cross-validation require setting the regularization parameter λ by training with many different λ values on each of 5 or 10 different folds of data, which increases running time by a factor of more than 100 compared to Ordinary Least Squares. Another thorny issue is that if our

grid of λ values to test is not fine enough, or not wide enough, then we can end up with an insufficiently good choice of λ, meaning that practitioner expertise sometimes is drawn upon to determine which λ values to try. What is more, the standard implementation of Ridge Regression based on solving a least squares problem requires loading all data into memory at once, making it inappropriate for very large data sets. Finally, the standard Ridge Regression algorithm is rigid in that it can't be be adapted to problems with different loss functions.

In this chapter, keeping these issues in mind, we explore an alternative to Ridge Regression that has many of the same benefits, has no difficult to deal with free parameters that need tweaking, is potentially much faster for very large data sets, and which can be adapted easily to changes in loss function.

6.1 Introduction

Ridge Regression is the most popular way to modify Ordinary Least Squares regression to prevent overfitting. However, the standard implementation of Ridge Regression can be unacceptably slow when the number of points and features are both large. In this chapter we consider an alternative approach to preventing overfitting in Least Squares Regression. We solve the Ordinary Least Squares problem iteratively using gradient descent with early stopping, which is a well known alternative to regularization that causes a machine learning problem to have fewer "effective" parameters, with the number of effective parameters increasing as the training time increases (Weigend and Rumelhart, 1992). More specifically, to decide when to stop we apply the gradient descent procedure in parallel to κ different folds of data (i.e. training κ sub-algorithms), each fold with its own mutually exclusive out of sample data set, an iterative form of k-fold cross-validation. Our predictions then are made by taking a weighted average of the predictions of these κ sub-algorithms, where sub-algorithms with

more out of sample error get less weight. Combining the sub-algorithms in this way allows us to use all of our data, rather than needing to sacrifice some of it to form a fixed validation set. Furthermore, this approach gives us a simple and effective stopping criteria: when the average out of sample fold error does not improve more than a certain percentage after an iteration of all κ sub-algorithms, we stop. We will refer to this procedure as "Impatient Learning".

We find that this algorithm is generally much faster than the standard implementation of Ridge Regression with k-fold cross-validation, especially when the number of points and features is large. But it has some other desirable properties relative to Ridge Regression, as well. Its only free parameter is easy to set and does not require tuning, it is easy to adapt to other differentiable, convex loss functions beyond the squared error loss function of Ridge Regression, and it can be modified to handle very large data sets. What is more, certain types of prior knowledge can easily be built into the algorithm, such as the knowledge that certain coefficients must be positive. As we will show, it is also very easy to kernelize Impatient Learning so that it can be applied to non-linear learning problems.

Our Python code for implementing Impatient Learning is in the public domain, and can be found at the URL in the following footnote [1].

In the next section we will more formally define the Impatient Learning algorithm. To increase applicability, we introduce the algorithm for arbitrary differentiable loss functions L, which measure how bad it is to predict the label a for a point when in fact b is the label. Later, when we want to empirically investigate the accuracy of Impatient Learning compared to standard Ridge Regression we will use the standard squared loss function $L(a, b) = (a - b)^2$.

[1] http://spencergreenberg.com/code/doctoral_thesis_machine_learning_code.zip

6.2 Impatient Learning

We define θ to be a vector of parameters that specify a hypothesis function $h_\theta(X)$. A hypothesis function maps a matrix of points X (with m points X^i, one per column, and n features X_j, one per row) into a vector of predicted labels for those points or, when it is convenient to consider functions acting on a single point at a time, maps a point x into a predicted label for that point. Let θ^* correspond to an initial guess, h_{θ^*}, of what a good hypothesis function might be. If no guess can be given, $\theta^* = 0$ can be used, which indicates that we expect many of our parameters to be 0 or close to 0.

We now define our objective function $G(\theta, S) = L(h_\theta(X), y)$ which we will be using our algorithm to reduce the value of by updating parameter θ. Note that when applied to vectors of values (rather than individual values) we will typically define L as $L(\vec{a}, \vec{b}) = \sum_{i=1}^{m} L(a_i, b_i)$, since most common objective functions take this form, but this additive form of loss function is not required. For the gradient of our objective function with respect to θ we will write $\nabla G \equiv \nabla_\theta G(\theta, S)$. Let $\eta(\theta, S, \nabla G)$ be a function which computes (or approximates) the optimal step-size to move along the negative of the given gradient ∇G, for a particular current training set $S = (X, y)$, to reduce the function $G(\theta, S)$ as much as possible. Hence $\eta(.)$ computes the solution to the one dimensional optimization problem:

$$\eta(\theta, S, \nabla G) = \underset{\eta'}{\text{argmin}}\ G(\theta - \eta' \nabla G).$$

Note that in some cases we may choose ∇G independently of S, so it is important to treat ∇G as an independent input to the function $\eta(.)$.

Our learning algorithm will make use of κ sub-algorithms, A_1, A_2, ..., A_κ, each of which is solving its own learning problem on a slightly different subset of the data. We will make κ copies of our training data $S = (X, y)$, with each copy $S_1, \ldots, S_\kappa$ omitting approximately a fraction $\frac{1}{\kappa}$ of the training data, chosen so that each data point in S is missing from exactly one of these copies S_i.

Now, we will train our κ sub-algorithms in parallel, each on a different S_i, measuring the error E_i at each iteration for each algorithm on the data that each algorithm is not trained on, and sum these errors $E \equiv \sum_{i=1}^{\kappa} E_i$ to produce an estimate of the true out of sample error. If after an iteration (of all κ sub-algorithms) this out of sample error estimate has not decreased a sufficiently large percentage, then we halt the entire algorithm. We generally discard this final step if it made the out of sample error worse, since it may have caused overfitting. However, as a variant of the algorithm that we can use if we choose, when the error actually gets worse with the last step rather than simply not improving as much as is desired, an approach to squeeze slightly more performance out of the algorithm is to take a final gradient step (after discarding the last regular step), but where we optimize the step size with respect to the out of sample error for that fold, rather than the in sample error for that fold. This is important if even one step is too many (i.e. overfitting occurs immediately).

Note that once the entire algorithm finishes, E_i serves as an estimate of the reliability of sub-algorithm A_i. Hence, when selecting our weights w_i we can use this to our advantage,

weighting each algorithm's prediction by a decreasing function of its error E_i, and then normalizing so that the weights sum to 1. Alternatively, we can simply assign each algorithm A_i the same weight, $w_i = \frac{1}{\kappa}$, which treats all sub-algorithms as equals, but this is throwing away potentially valuable information about the accuracy of each sub-algorithm. We suggest calculating the R^2 of each sub-algorithm, which we'll denote R_i^2, and then assigning weights $w_i = \frac{R_i^2}{\sum_{j=1}^{\kappa} R_j^2}$, where we define the R^2 of a prediction function $h(x)$ on an out of sample data set (X,y) in a slightly idiosyncratic way as:

$$R^2 \equiv 1 - \frac{L(h(X), y)}{\min_c L(c\,\vec{1}, y)}$$

when this quantity is positive, and 0 when this quantity is non-positive, where $\vec{1}$ is the vector of all ones, and c is the scalar constant that minimizes $L(c\,\vec{1}, y)$ for the given loss function L. In the case of the squared loss function, c is just the mean of the label vector y. Note that this definition means that R^2 is the fraction of loss captured by our algorithm, relative to the optimal constant (out of sample) predictor. We call this "R^2" since in the common case where L is the squared loss function, it simply gives us the fraction of variance captured by our predictor, and coincides with the typical notion of R^2. We set it to 0 in the rare case where the calculation is negative because this situation implies an algorithm's predictions are worse than a constant predictor, which likely means the algorithm learned nothing at all or overfit, and therefore that we do not want to rely on it.

Using this weighting scheme, a perfect predictor will be assigned a maximal weight of 1 (prior to normalizing the weights so that they sum to 1), and a predictor that is worse than the optimal constant predictor will be assigned a weight of 0. While theoretically an algorithm with perfect predictions should receive all the weight rather than an unnormalized weight of 1, we do not know the true error of any algorithm, only an estimate based on a

small portion of withheld data. Hence, using a weighting scheme that explodes to infinity as the error for an algorithm goes to 0 is risky, as 0 error out of sample, while impressive, does not imply a perfect predictor.

In the Impatient Learning algorithm, the total validation error, E, which is the sum of errors made by each algorithm on its own validation set, serves as a proxy for the out of sample generalization error for the whole combined system. In fact though, we expect the whole system to produce a lower out of sample error than E, since it has the benefit of combining together all of the different sub-algorithms, each trained on slightly different data, whereas the components that go into E each use only one such algorithm. A step-by-step description of the Impatient Learning algorithm is shown as Algorithm 6.

If the total validation error, E, as a function of the number of iterations, has many local maxima and minima then the stopping rule suggested above, $\frac{E_{old}-E_{new}}{E_{old}} < \mathcal{E}$, will stop at the first local minima, which will generally be too early. But our experiments suggest that, for the squared loss, this function of E is smooth and does not have problematic local minima. See page 233 for examples. It is a non-trivial observation that there is typically a smooth curve, with a unique minimum describing the out of sample error as a function of the number of iterations, since this makes the stopping criteria much more reliable. Choosing when to stop in cases where there can be multiple local minimums is a non-trivial problem, and one generally must resort to one of a variety of heuristics, which may or may not be reliable in any particular instance (Prechelt, 1998).

Algorithm 6 Impatient Learning

1: Impatient Learning can be sensitive to how the labels are centered, so we run an outlier clipping algorithm (e.g. Impact Clipping or Winsorization) on y to clip any extreme labels, then compute the mean, and subtract this mean from the clipped labels.

2: Split our training set $S = (X, y)$ into κ mutually exclusive subsets V_1, V_1, ..., V_κ with sizes that are as close as possible to each other. Set V_i will be the validation set for sub-algorithm A_i, which we will use to measure A_i's out of sample error.

3: Make κ overlapping subsets of S called S_1, S_2, ..., S_κ, each defined as $S_\kappa = S - V_\kappa$. Set S_i will be the training set for sub-algorithm A_i.

4: Make κ copies of our starting parameter vector θ^*, call these θ^1, θ^2, ..., θ^κ. These represent the parameters for each of our κ sub-algorithms.

5: For each sub-algorithm A_i, update the corresponding parameter vector θ^i via gradient descent using S^i by taking $\theta^i \rightarrow \theta^i - \eta \nabla G \equiv \theta^i - \eta(\theta^i, S_i, \nabla_{\theta^i} G(\theta^i, S_i)) \nabla_{\theta^i} G(\theta^i, S_i)$.

6: For each sub-algorithm A_i, record the prediction error $E_i = G(\theta^i, V_i)$ made on its validation set V_i, and compute aggregated out of sample error estimate $E_{new} = E_1 + E_2 + \ldots + E_\kappa$.

7: If it is not iteration 1, compare total out of sample error estimate E_{new} to its prior value E_{old}. If $\frac{E_{old} - E_{new}}{E_{old}} < \mathcal{E}$ stop looping and go to Step 8, since it is not worth continuing, and further iterations may overfit. In this case, if the error estimate has actually gotten worse, discard the last iteration since it may cause overfitting. If $\frac{E_{old} - E_{new}}{E_{old}} \geq \mathcal{E}$ update the error setting $E_{new} \rightarrow E_{old}$, and go to Step 5.

8: Predict points using a weighted average of the predictions of the individual sub-algorithms. Hence, our prediction for a point x will be $h(x) = w_1 h_{\theta^1}(X) + w_2 h_{\theta^2}(x) + \ldots + w_\kappa h_{\theta^\kappa}(x)$. The weights w_i can be chosen in various ways (discussed elsewhere in Section 6.2 of this chapter), but they should be non-negative, sum to 1, and give more weight to the sub-algorithms A_i that have lower out of sample fold error E_i.

6.3 Iterative Least Squares Regression

We now consider the interesting special case of Impatient Learning where our loss function is the squared loss function, which is used by Ordinary Least Squares and Ridge Regression.

In this case, we have the objective function:

$$G = \sum_{i=1}^{m} (X^{i\intercal} w - y_i)^2.$$

The gradient, which is the vector in the direction we will be stepping along, is then given by:

$$\frac{d}{dw} G = 2 \sum_{i=1}^{m} X^i (X^{i\intercal} w - y_i) = 2 \sum_{i=1}^{m} X^i X^{i\intercal} w - 2 \sum_{i=1}^{m} X^i y_i$$

$$= 2XX^\intercal w - 2Xy \equiv Dw - b$$

where

$$D \equiv 2XX^\intercal \ \ and \ \ b \equiv 2Xy.$$

Then each, gradient descent step, which converts our previous coefficient vector w_{k-1} to our new coefficient vector w_k by subtracting a multiple of the gradient, is given by:

$$w_k = w_{k-1} - \eta_{k-1}(Dw_{k-1} - b) = (I - \eta_{k-1} D) w_{k-1} + \eta_{k-1} b \tag{6.1}$$

where η_k is the step size, which may be chosen to be a different value at each step k. Now, recursively applying this equation to itself t times, we find that:

$$w_k = (I - \eta_{k-1} D) w_{k-1} + \eta_{k-1} b$$

$$= (I - \eta_{k-1} D)((I - \eta_{k-2} D) w_{k-2} + \eta_{k-2} b) + \eta_{k-1} b$$

$$= (I - \eta_{k-1}D)((I - \eta_{k-2}D)((I - \eta_{k-3}D)w_{k-3} + \eta_{k-3}b) + \eta_{k-2}b) + \eta_{k-1}b$$

$$= (I - \eta_{k-1}D)(I - \eta_{k-2}D)(I - \eta_{k-3}D)w_{k-3}$$

$$+ \left((I - \eta_{k-1}D)(I - \eta_{k-2}D)\eta_{k-3} + (I - \eta_{k-1}D)\eta_{k-2} + \eta_{k-1}\right)b$$

$$= \left(\prod_{j=1}^{t}(I - \eta_{k-j}D)\right) w_{k-t} + \sum_{s=1}^{t}\left(\prod_{j=1}^{s-1}(I - \eta_{k-j}D)\right)\eta_{k-s}\ b$$

Hence, setting $t = k$ we have:

$$w_k = \left(\prod_{j=1}^{k}(I - 2\eta_{k-j}XX^\intercal)\right) w_0 + \sum_{s=1}^{k}\left(\prod_{j=1}^{s-1}(I - 2\eta_{k-j}XX^\intercal)\right) 2\eta_{k-s}\ Xy \tag{6.2}$$

where w_0 is our starting guess for the solution coefficients. We notice that this equation for w_k consists of two terms. The first is simply some matrix (which depends on the number of iterations) applied to the starting coefficients, w_0, which reflect our best a priori guess for the solution. Often we will use $w_0 = 0$, in which case this term will disappear. The second term is just a matrix (which depends on the number of iterations) applied to Xy.

To better understand Equation 6.2, consider what happens in the case where the step-size η_k is a constant, $\eta > 0$. Then we have

$$w_k = (I - 2\eta XX^\intercal)^k w_0 + \sum_{s=0}^{k-1} 2\eta(I - 2\eta XX^\intercal)^s Xy$$

Under the assumption that $XX^\intercal$ is invertible, and that $0 < \eta < \frac{1}{\sigma_{\max}}$, where $\sigma_{\max}$ is the largest singular value of $XX^\intercal$, we can write an explicit formula for the right hand sum, which is:

$$w_k = (I - 2\eta XX^\intercal)^k w_0 + 2\eta(2\eta XX^\intercal)^{-1}(I - (I - 2\eta XX^\intercal)^k)Xy$$

$$= (I - 2\eta XX^\intercal)^k w_0 + (XX^\intercal)^{-1}(I - (I - 2\eta XX^\intercal)^k)Xy$$

$$= (I - 2\eta XX^\intercal)^k w_0 + (XX^\intercal)^{-1}Xy - (XX^\intercal)^{-1}(I - 2\eta XX^\intercal)^k Xy$$

$$= (I - 2\eta XX^\intercal)^k w_0 - (I - 2\eta XX^\intercal)^k (XX^\intercal)^{-1}Xy + (XX^\intercal)^{-1}Xy$$

$$= (I - 2\eta XX^\intercal)^k (w_0 - (XX^\intercal)^{-1}Xy) + (XX^\intercal)^{-1}Xy$$

$$= (I - 2\eta XX^\intercal)^k (w_0 - w_{LS}) + w_{LS}$$

where

$$w_{LS} \equiv (XX^\intercal)^{-1}Xy$$

is the solution coefficient vector for Ordinary Least Squares regression. As we will see, $(I - 2\eta XX^\intercal)^k$ approaches 0 as k increases, so as expected the gradient descent algorithm (with fixed step-size) follows a path from w_0 (when $k = 0$) to w_{LS} (as $k \to \infty$). In detail, if we diagonalize $XX^\intercal$ writing

$$XX^\intercal \equiv U\Sigma U^\intercal$$

where U is orthonormal and Σ is diagonal with diagonal entries $\sigma_i > 0$, which are the singular values of $XX^\intercal$, then we have:

$$(I - 2\eta XX^\intercal)^k = (I - 2\eta U\Sigma U^\intercal)^k = (UIU^\intercal - 2\eta U\Sigma U^\intercal)^k$$

$$= (U(I - 2\eta\Sigma)U^\intercal)^k = U(I - 2\eta\Sigma)^k U^\intercal$$

$$= U \begin{bmatrix} (1 - 2\eta\sigma_1)^k & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & (1 - 2\eta\sigma_n)^k \end{bmatrix} U^\intercal.$$

Now, since $XX^\intercal$ is assumed to be invertible, the σ_i are always positive by definition. And as long as $w_0 \neq w_{LS}$, there is no harm assuming η is also strictly positive. Therefore $(1 - 2\eta\sigma_i)^k \to 0$ as $k \to \infty$ so long as $\eta < \frac{1}{\sigma_{\max}}$.

In summary, we can think of gradient descent, using the squared loss function with a constant step-size, as following the path $(I - 2\eta XX^\intercal)^k(w_0 - w_{LS}) + w_{LS}$, indexed by the iteration number k, which takes it from starting point w_0 towards the least squared solution w_{LS}. We can view this as a form of shrinkage, where for iteration 0 we start at w_0 and blend increasingly into w_{LS} as the iterations increase, only reaching w_{LS} in the limit of an infinite number of iterations. If we think of w_0 as our best a priori guess for the solution coefficients w, this also explains why early stopping can prevent overfitting. The fewer iterations, the closer our solution coefficient will generally be to our starting guess w_0, much like with Ridge Regression where the greater the λ, the closer our solution coefficients are to the zero vector.

6.3.0.1 Locally Optimal Step Size

While using a constant step size $\eta_k = \eta$ makes the gradient descent algorithm easy to analyze, there is an advantage to choosing η_k more carefully so as to achieve faster convergence. Early stopping means that we generally will not have to convergence all the way to the solution of interest, but it is still desirable to get to convergence quickly. Hence, we consider the locally optimal step size to take when performing gradient descent with the squared error loss function. For the moment, assume that we will be stepping in the direction $-\Delta G$ by a step size η, where the gradient of our loss function at the current point is

$$\Delta G \equiv 2(XX^\intercal w - Xy) = 2X(X^\intercal w - y) = 2X\epsilon$$

and ϵ is the vector of prediction errors ϵ_i given by:

$$\epsilon_i \equiv X^{i\intercal} w - y_i.$$

We also define the scalar $a_i \equiv X^{i\intercal}\Delta G = 2X^{i\intercal}X\epsilon$, which will simplify our formulas. Then we take the derivative of the squared loss function with respect to η, evaluated at the updated solution coefficient $w - \eta\Delta G$, and assign it equal to 0 to solve for the optimal η, yielding:

$$\frac{d}{d\eta}\sum_{i=1}^{m}(X^{i\intercal}(w-\eta\Delta G)-y_i)^2 = \sum_{i=1}^{m}\frac{d}{d\eta}(X^{i\intercal}w-\eta X^{i\intercal}\Delta G-y_i)^2$$

$$=\sum_{i=1}^{m}\frac{d}{d\eta}(\epsilon_i-\eta a_i)^2 = -2\sum_{i=1}^{m}(\epsilon_i-\eta a_i)a_i = -2\sum_{i=1}^{m}\epsilon_i a_i + 2\eta\sum_{i=1}^{m}a_i^2 = 0$$

Therefore, solving for the locally optimal η to minimize the loss, we have:

$$\eta = \frac{\sum_{i=1}^{m}\epsilon_i a_i}{\sum_{i=1}^{m}a_i^2} = \frac{\sum_{i=1}^{m}\epsilon_i X^{i\intercal}\Delta G}{\sum_{i=1}^{m}\Delta G^{\intercal}X^i X^{i\intercal}\Delta G}$$

$$= \frac{\epsilon^{\intercal}X^{\intercal}\Delta G}{\Delta G^{\intercal}XX^{\intercal}\Delta G}$$

$$= \frac{1}{2}\frac{\epsilon^{\intercal}X^{\intercal}X\epsilon}{\epsilon^{\intercal}X^{\intercal}XX^{\intercal}X\epsilon} = \frac{1}{2}\frac{\epsilon^{\intercal}K\epsilon}{\epsilon^{\intercal}K^2\epsilon}$$

where in a linear context $K \equiv X^{\intercal}X$. This leads to the locally optimal solution coefficient update

$$w_k = w_{k-1} - \eta_{k-1}\Delta G = w_{k-1} - \frac{\epsilon^{\intercal}K\epsilon}{\epsilon^{\intercal}K^2\epsilon}X\epsilon.$$

We note that in the special case where $\epsilon^{\intercal}K\epsilon \equiv \sum_{i=1}^{m}\epsilon_i a_i = 0$, we get the locally optimal step size $\eta = 0$, even if the denominator $\epsilon^{\intercal}K^2\epsilon = 0$ as well. This should be added to the implementation of this algorithm as a special case.

We will now discuss the speed of Ridge Regression, and compare it to the speed of Impatient Learning.

6.4 Ridge Regression Speed

How fast is Ridge Regression? When the complexity parameter λ of Ridge Regression is fixed, the solution coefficients w_λ can be solved for analytically, so the algorithm is usually implemented using a linear solver to find the solution to:

$$(XX^\intercal + \lambda I)w_\lambda = Xy, \tag{6.3}$$

where X has one training point X^i per column and one feature X_j per row, y is a column vector of the training labels, and I is the identity matrix. Usually a column of 1's is adding to X to represent a constant term in our resulting linear equation.

The process of forming the matrix $XX^\intercal$ has running time on the order of $m\,n^2$, where m is the number of points and n is the number of features. Solving the linear equation of Equation 6.3 using commonly applied methods has running time on the order of n^3. This leads to a total running time on the order of $n^2 \max(n, m)$. When the number of features is larger than the number of points, this running time can be improved by introducing a linear version of the kernel matrix, defined in this case as $K \equiv X^\intercal X$. We write:

$$w_\lambda = (XX^\intercal + \lambda I)^{-1}Xy = X(X^\intercal X + \lambda I)^{-1}y = X(K + \lambda I)^{-1}y.$$

It takes time on the order of $m^2 n$ to form the kernel matrix, and time on the order of m^3 to invert $(K + \lambda I)$ using typically applied methods. Hence, the total running time is on the order of $m^2 \max(n, m)$. Hence, when Ridge Regression is implemented with speed in mind, forming $XX^\intercal$ when $m \geq n$ and forming $K \equiv X^\intercal X$ when $m < n$, it has running time on the order of $\min(n^2 \max(n, m), m^2 \max(n, m)) = \max(n, m) \min(n, m)^2$. Since $\max(n, m) \min(n, m)^2 > \min(n, m)^3$ we are in trouble when $\min(n, m)$ is large. The worst

case scenario is when $m \approx n$ which gives us a running time on the order of m^3, which for some applications will be unacceptably slow. For $m = 5000$, m^3 is more than one hundred billion. In the 3rd part of Figure 6.3 we see a comparison of the number of seconds to train Ridge Regression against this theoretical model on the randomly generated learning problems discussed in Section 6.5, where for simplicity we only use the $XX^\intercal$ based Ridge Regression implementation.

Another source of slowness for Ridge Regression, as it is typically implemented, is in selection of the λ parameter. Typically, one uses k-fold cross-validation to test a wide range of λ values, selecting the one that yields the lowest out of sample error. This can significantly slow our learning process, even though the slow down is only by a constant factor. Let us consider a specific case where $m > n$. If we are using 10-fold cross-validation, we will have to train 10 models, each on 90% of our data points, which will increase our training time by a factor of approximately 9. If we are trying, say, 30 values of λ that exponentially increase from 10^{-6} to 10^6, that will increase our training time by another factor of 30. In total, it will take about 270 times longer to train this model than it does to train Ordinary Least Squares, which may be unacceptably long. We could, of course, reduce the number of folds or the number of λ values tried, but if the number of folds is too small or the grid for our lambda values is not fine enough or does not span a wide enough range, we may make a poor choice for λ. An alternative which can be faster is to use a method for approximating the k-fold cross validation error or the leave one error error, such as the "Generalized Cross-Validation" method (Golub et al., 1979), but we do not consider such approaches in this chapter.

6.5 Impatient Learning Speed

It is known that gradient descent is often a slow algorithm for function optimization, as it is typically inefficient for computing a high number of decimal points of accuracy unless we

have strong convexity of our objective function. With convexity, but not strong convexity, to get an accuracy within $\mathcal{A}$ of the optimal value we need a number of iterations on the order of $1/\mathcal{A}$, whereas with strong convexity we only need iterations on the order of $\log(1/\mathcal{A})$ (Gordon and Tibshirani, 2012). Strong convexity is sometimes lacking in least squares problems (e.g. whenever the number of features is greater than the number of points, or when a feature is duplicated), and even when it is satisfied, convergence may still be slow in practice if the condition number is high, even though asymptotically the gap between the current solution and the optimal solution falls exponentially fast.

Optimization algorithms have the goal of finding, to high precision, the input that maximizes or minimizes the value of a function. Most machine learning algorithms can be recast as optimization algorithms. Machine learning involves searching a given space of parameters or functions to find one that minimizes an objective function. These observations naturally point us in the direction of using optimization algorithms to solve machine learning problems, which is widely done. However, the goals of mathematical optimization and machine learning differ in two important ways:

1. Optimization algorithms try to efficiently find the input minimizing a function to many digits of accuracy, whereas machine learning algorithms should try to efficiently find a single input (e.g. parameter vector representing a hypothesis) that has close to optimal out of sample prediction accuracy. Usually the latter does not require resolving parameters to many digits of accuracy, and doing so wastes time. The seventh or even fifth digit of our parameters is usually irrelevant in machine learning.

2. Optimization algorithms take for granted that the function they are minimizing is the actual function one is trying to minimized, whereas in machine learning the objective function is not actually the function one hopes to minimize. Empirical loss functions

are defined over a finite sample of training data, and minimizing this function (i.e. approaching producing zero loss) is generally undesirable, as it may imply substantial overfitting. When a regularization term is added to correct the overfitting problem, the objective function still does not directly measure the goal one hopes to achieve (i.e. out of sample accuracy). Regularization merely serves as a convenient way of incorporating a priori knowledge (such as a preference for small coefficients over large coefficients), or of making our objective function more closely (but still imperfectly) representative our actual goal. The fact that our training labels have random noise means that no function of our training data will ever perfectly represent our true goal.

For these reasons using a method like gradient descent, which is known to be slow for optimization, is not necessarily dumb in a machine learning context. The hope is that stopping early makes this slow optimization algorithm into a fast machine learning algorithm. As we have discussed, high precision is unnecessary in most machine learning applications, and in fact is wasteful. By stopping as soon as further iterations are useless, the Impatient Learning algorithm speeds up the gradient descent process, sometimes dramatically. Furthermore, stopping early takes into account the fact that it is not the objective function per se that we are truly interested in minimizing, but rather the generalization error. The objective function is merely a proxy for this goal.

So how fast is linear Impatient Learning with a squared loss function? Examining Section 6.3.0.1, we see that at each iteration we have to compute the point error vector $\epsilon = X^\intercal w - y$ and the update vector $\frac{\epsilon^\intercal X^\intercal X \epsilon}{\epsilon^\intercal (X^\intercal X)^2 \epsilon} X\epsilon$ for each fold. If we perform this calculation efficiently we will always be multiplying a matrix by a vector (never a matrix by a matrix) so it can be done in time on the order of $m\,n$, which is the number of points times the number of features. Hence, if $\mathcal{I}$ is the number of iterations, then the total running time for impatient

learning is given by $\kappa\, m\, n\, \mathcal{I}$, where recall that κ is the number of folds. But how many iterations does the algorithm take? Intuitively, the faster that overfitting begins to occur, the sooner the algorithm will stop. Since higher levels of noise mean more overfitting, we therefore would expect to find that Impatient Learning is faster when there is more noise, which as we shall see, is indeed the case.

To empirically investigate the running time of Impatient Learning (with the squared error loss function and locally optimal step size) as a function of the number of points, the number of features and the label noise, we conduct experiments on synthetic linear learning problems. The results are shown in Figures 6.2 and 6.3. For these experiments we generated 2241 linear learning problems. Of these, 5 were removed due to Ridge Regression taking an excessively long time to train for unknown reasons, leaving 2236 learning problems. Each used an underlying linear model with each coefficient generated from an independent unit gaussian, and a gaussian additive constant term. Additive gaussian noise was then added to each label. Points were drawn independently from an isotropic gaussian (i.e. using unit variance and zero correlation between features). The number of training points, number of training features, and signal to noise ratio (of the noise added to the labels) for each learning problem were drawn uniformly and independently at random from the ranges [50,3000], [20,500], and [0.0,1.0] respectively. Impatient Learning with the squared error loss function and Ridge Regression were both applied to each learning problem using 10-fold cross validation. Ridge Regression's 10-fold cross-validation tested 30 lambda values spanning exponentially from 10^{-6} to 10^{6}. For Impatient Learning, we used $\mathcal{E} = 0.000005$, indicating that we would stop once an iteration failed to improve the average out of sample error estimate by at least 0.0005%. There is little reason to select $\mathcal{E}$ smaller than this, but an $\mathcal{E}$ that is 10 or even 100 times bigger can be chosen for faster stopping in slow cases. Our usual recommendation is $\mathcal{E} = 0.0001$. We chose a smaller value for our empirical test cases simply so that we could explore how many iterations Im-

patient Learning would take when there is very little restriction on its ability to converge fully.

The Impatient Learning algorithm typically finishes in remarkably few iterations on these test cases. The median number of iterations was only 5, and the median absolute deviations of the number of iterations was just 2. The minimum was 1 iteration, but there were many large outliers, with the maximum being 900 iterations. The mean was 11.7, and the standard deviation 38 due to these outliers. As can be seen in Figure 6.2, the number of features and the number points seem to have little relationship to the average number of iterations, though there are more extreme outliers in problems with smaller numbers of points or greater number of features. There was a fairly strong relationship involving the noise to signal ratio $\mathcal{N}$, which is the standard deviation of the additive noise applied to the labels divided by the standard deviation of the labels before noise is applied. We found that the number of iterations was roughly related to $\frac{1}{\sqrt{\mathcal{N}}}$. In some cases, when the noise was very low a very large number of iterations was used. This makes sense intuitively because when there is little noise machine learning looks more like function optimization, since noise is a major cause of overfitting and of why our objective function does not match our true goal. Without overfitting and noise to worry about, grinding out many digits of accuracy in our optimization may actually be beneficial, and so Impatient Learning starts to show its warts as an optimization algorithm.

In Figure 6.3 we can see that the total training seconds for Impatient Learning on these synthetic cases is reasonably well modeled using a power of the quantity $\frac{nm}{\sqrt{\mathcal{N}}}$. While the power shown in the curve fit in the figure is less than 1, the curve appears to undershoot somewhat for large values of the x-axis, so for large numbers of features and points may actually be closer to 1. Note that a log-log plot is shown, meaning that monomials of the form ax^b appear as straight lines.

In our implementations of Impatient Learning and Ridge Regression the latter was generally much slower. The median of the ratio of the training times showed Ridge Regression to be 8.25x slower, and Ridge Regression was slower in 99% of cases. These numbers are not particularly telling though, since they will vary with the learning problem, as well as with the details of how each algorithm is implemented and even the programming language used. Overall though, our empirical investigations suggest that Impatient Learning is substantially faster than Ridge Regression, and that the size of this benefit will be greater when there is more noise, and when the number of points and features is large. It is worth also noting that the variability of training time for Impatient Learning can be substantial. Depending on the setting for $\mathcal{E}$, the algorithm can occasionally run for many iterations with little gain, even though it is usually fast and usually needs few iterations.

But what about the accuracy of Impatient Learning? While intuitively we expect it to avoid overfitting due to the trick of estimating test set error after each iteration, it is not obvious that the predictions it makes will be that similar to those of Ridge Regression, or even that the accuracy will be similar. They both push the solution coefficients towards zero (if we initialize $\theta^* = \vec{0}$) relative to Ordinary Least Squares, but they do so in different ways.

On our synthetic test cases, we considered the R^2 of Impatient Learning minus the R^2 of Ridge Regression. The average of this quantity was -0.0033 with a median of -0.00040, reflecting a slight average reduction in accuracy using Impatient Learning. The standard deviation of this quantity was 0.012. We found that 72% of the time Ridge Regression was more accurate, even though these accuracy differences were usually very small. When $\mathcal{N}$ was small, Impatient Learning almost never outperformed Ridge Regression by a meaningful amount, probably due to the "optimization like" nature of these cases, where high precision has value. What's more, as the number of points grew, the likelihood of large differences in

performance between the two algorithms shrank considerably, and Impatient Learning seemed to perform somewhat better relative to Ridge Regression in these cases as well compared to when there were fewer points.

Overall, our empirical investigation suggests that Impatient Learning tends to be substantially faster, but slightly less accurate than Ridge Regression, with Impatient Learning gaining the biggest speed advantage when the number of points and features is large, and doing the worst (along both the dimensions of speed and accuracy) when the label noise is low. Results will, of course, vary from data set to data set, and what is true on these synthetic test cases will not hold for all problems. But these results suggest that Impatient Learning can be a desirable alternative to Ridge Regression when a small accuracy loss would be worth trading for a large speed gain, assuming it is not a machine learning problem with low label noise. In other words, Impatient Learning can be better than Ridge Regression when you are, well, impatient.

6.5.1 Kernelizing Iterative Least Squares Regression

Let us now consider what happens when we kernelize Impatient Learning by applying a vector valued transformation function ϕ to each data point X^i, which gives us the matrix

$$\Phi \equiv [\phi(X^i)]_{i=1}^m$$

with one transformed point per column. As we will see, this will lead to an extremely simple formula that we can use for learning functions that are non-linear in the original variables. We define the kernel (i.e. point similarity) function

$$\mathcal{K}(X^i, X^j) = \phi(X^i)^\intercal \phi(X^j)$$

where we use the kernel matrix

$$K \equiv \Phi^\intercal \Phi = [\phi(X^i)^\intercal \phi(X^j)]_{i=1,j=1}^m.$$

This is a non-linear generalization of our previous definition for K. We consider a new point x^0 (represented as a column vector), which we would like to make a prediction for, and its transformation by ϕ into our new space, which is given by

$$\phi_0 \equiv \phi(x_0).$$

Finally, we define the column vector of similarities between x_0 and all the points X^i as

$$\vec{k}_0 \equiv \phi_0^\intercal \Phi = [\mathcal{K}(x_0, X^i)]_{i=1}^m.$$

Then, we will predict that the label for the point ϕ_0 is y_0 by applying Equation 6.2 to the new features Φ instead of the original features X. By passing Φ from the right through to the left, this yields:

$$\begin{aligned}
y_0 &\equiv \phi_0^\intercal \left(\prod_{j=1}^{k} (I - 2\eta_{k-j} \Phi\Phi^\intercal) \right) \Phi\Phi^\intercal (\Phi\Phi^\intercal)^{-1} w_0 + \phi_0^\intercal \sum_{s=1}^{k} \left(\prod_{j=1}^{s-1} (I - 2\eta_{k-j} \Phi\Phi^\intercal) \right) 2\eta_{k-s}\ \Phi y \\
&= \vec{k}_0^\intercal \left(\prod_{j=1}^{k} (I - 2\eta_{k-j} \Phi^\intercal\Phi) \right) \Phi^\intercal (\Phi\Phi^\intercal)^{-1} w_0 + \vec{k}_0^\intercal \sum_{s=1}^{k} \left(\prod_{j=1}^{s-1} (I - 2\eta_{k-j} \Phi^\intercal\Phi) \right) 2\eta_{k-s}\ y \\
&= \vec{k}_0^\intercal \left(\prod_{j=1}^{k} (I - 2\eta_{k-j} K) \right) u_0 + \vec{k}_0^\intercal \sum_{s=1}^{k} \left(\prod_{j=1}^{s-1} (I - 2\eta_{k-j} K) \right) 2\eta_{k-s}\ y \\
&= \vec{k}_0^\intercal \left(\left(\prod_{j=1}^{k} (I - 2\eta_{k-j} K) \right) u_0 + \sum_{s=1}^{k} \left(\prod_{j=1}^{s-1} (I - 2\eta_{k-j} K) \right) 2\eta_{k-s}\ y \right) \qquad (6.4)
\end{aligned}$$

$$\equiv \vec{k}_0^{\intercal} u_k$$

where we have defined a new starting parameter vector

$$u_0 \equiv \Phi^{\intercal}(\Phi\Phi^{\intercal})^{-1} w_0$$

which takes the place of our previous starting coefficients w_0, and a corresponding kth iteration of this vector given by u_k.

The object u_k reflects a new type of solution coefficient vector. Unlike w_k which acts on feature vectors (i.e. individual points) such as x_0, instead u_k acts on point similarity vectors, like ϕ_0, which measure how similar a point is to every other point. We now note that the intermediate Expression 6.4 above for y_0 is identical to Equation 6.2 once it has been left multiplied by the point x^0, except that Xy is now replaced with y, $XX^{\intercal}$ is now replaced with K and x^0 is now replaced with $\vec{k}_0$. Since w_k satisfies the recurrence relation of Equation 6.1, which is

$$w_k = (I - 2\eta_{k-1} XX^{\intercal})\, w_{k-1} + 2\eta_{k-1} Xy$$

that immediately implies that

$$u_k = (I - 2\eta_{k-1} K)\, u_{k-1} + 2\eta_{k-1} y.$$

Hence, we now know how to perform gradient descent on this problem in a kernelized fashion to produce Kernelized Impatient Learning. We simply start with our best guess for our solution coefficient vector, which is given by u_0 (comparable to our previous w_0), and then we iterate according to the above formula for u_k, which expresses it in terms of u_{k-1}. Predictions

are made by taking dot products with u_k, not against points such as x_0, but against vectors of how similar each point is to every other point, such as $\vec{k}_0 = [K(x_0, X^i)]_{i=1}^m$.

We note now that the recursive relationship for w_k can also be better understood by writing it as

$$w_k = w_{k-1} - \eta_{k-1}(Dw_{k-1} - b) = w_{k-1} - 2\eta_{k-1}X(X^\intercal w_{k-1} - y)$$

$$= w_{k-1} - 2\eta_{k-1}X\vec{\epsilon}_{k-1}$$

where we define $\vec{\epsilon}_k$ to be the vector of training set prediction errors (on each point X^i) made by the algorithm after iteration k. In other word, at each iteration, the algorithm is adjusting w_{k-1} by the matrix X applied to the vector of errors made at each point, to produce w_k. But the recursive relationship for u_k is even more intuitive. We have:

$$u_k = (I - 2\eta_{k-1}K)\, u_{k-1} + 2\eta_{k-1}y$$

$$= u_{k-1} - 2\eta_{k-1}(Ku_{k-1} - y)$$

But since Ku_{k-1} is the vector of predictions for each point after $k-1$ iterations, that means that the vector of point prediction errors after k-1 iterations is given by $\vec{\epsilon}_{k-1} \equiv Ku_{k-1} - y$. Hence:

$$u_k = u_{k-1} - 2\eta_{k-1}\vec{\epsilon}_{k-1}.$$

So at each iteration, the kernelized form of the algorithm simply adds to the coefficients a multiple of the prediction errors at the previous iteration! Hence, we have an extraordinary simple implementation of the kernelized form of gradient descent for the squared error loss function. We simply start at u_0 and subtract a multiple of the error each time.

We note that our final kernelized update formula only depends on the error, so despite this derivation holding only in the case of the squared error loss function, one could simply replace the definition of error by using a different loss function to compute $\vec{\epsilon}$, to produce a different algorithm (though it would not necessarily correspond to the standard regression objective function with that loss function, except in the squared error case). Finally, we note that this form of the algorithm gives us a way of reducing the impact of outliers on the final solution, should we choose to do so. The only way that a point can corrupt our coefficients is if the corresponding entry of $\vec{\epsilon}_{k-1}$ is large. Hence, before subtracting a multiple of $\vec{\epsilon}_{k-1}$ from the coefficients, we can simply apply any 1-dimensional outlier removal algorithm to the vector $\vec{\epsilon}_{k-1}$, and either assign 0 for any value deemed an outlier, or winsorize that value so that it can't be more than a fixed amount.

What is the locally optimal step size in the kernelized case? Well, we replace X with Φ, and $K \equiv \Phi^\intercal \Phi$, so the locally optimal step size is still just

$$\eta = \frac{1}{2} \frac{\epsilon^\intercal K \epsilon}{\epsilon^\intercal K^2 \epsilon}.$$

and our complete update becomes

$$u_k = u_{k-1} - 2\eta_{k-1}\epsilon = u_{k-1} - \frac{\epsilon^\intercal K \epsilon}{\epsilon^\intercal K^2 \epsilon}\epsilon.$$

Hence, the formula for performing gradient descent based kernel regression with a squared loss function is shockingly simple, even simpler than the standard linear case. The drawback is that we can no longer avoid matrix to matrix multiplications as we could in the linear case. We are forced to form the kernel matrix K all at once, which takes time on the order of nm^2.

6.6 Subsampling

The Impatient Learning approach has a natural extension to work with very large data sets. Rather than computing the gradient from all points with each set S_i, we could use some random subsample of points from each S_i (e.g. 1000 randomly selected points for each gradient calculation). If the number of points is large enough, we should still expect to produce a smooth and convex relationship between the validation error E and the number of iterations, which is important for being able to easily select the appropriate stopping time. But it is hard to know how large is large enough. When subsampling, an alternative variation is to (when taking the gradient step) compute the optimal step size on both the subsample of randomly selected points, as well as on the out of sample test set for that fold. We then only take a step if the two step sizes agree in sign, and if they do, then the step size we use is whichever has a smaller magnitude. This ensures that the fold test set error never decreases, even if the the gradient calculations are not very accurate, which may happen in high dimensions.

By choosing a subsample of points for gradient computation, we no longer have to load all points in memory, allowing the possibility of working with files that are larger than available RAM. What is more, we now gain some of the benefits of stochastic gradient descent (Gardner, 1984), in particular, if there are many redundant points they will no longer slow down our processing time, and the algorithm may become much faster for large sample sizes.

6.7 Prior Knowledge

The Impatient Learning algorithm encodes a best guess of the solution coefficients, given by θ^*, as the starting parameters. This can be thought of as a way of encoding prior

knowledge about what the solution might be, or it can simply be set to $\theta^* = 0$ in cases where no reasonable guess can be made. However, the Impatient Learning algorithm can also be easily extended to encode other prior knowledge of the practitioner. Any linear equality or inequality constraints on the parameter vector can be captured by projecting the gradient onto these constraints before calculating the step size. So for instance if one has reason to believe that certain coefficients are positive, or bounded, this knowledge can be encoded by projecting the gradient accordingly. Doing so, however, may slow down convergence of the algorithm.

6.8 Conclusion

We have defined Impatient Learning, a simple alternative to Ridge Regression based on a k-fold cross-validated form of gradient descent with early stopping, which does not require regularization. We have also derived a very simple formula for kernelizing Impatient Learning, so that it can be applied to non-linear problems. Testing the linear form of Impatient Learning empirically on linear synthetic cases, we found that it is generally substantially faster than k-fold cross-validation based Ridge Regression, especially when the number of points and features is large, though it is slightly less accurate, making it appropriate when the user is impatient. In our tests, Impatient Learning performed its worst, both in terms of speed and accuracy, when there was a low amount of label noise.

Impatient Learning has a handful of other advantages over the standard implementation of Ridge Regression, beyond just speed. Its one free parameter is a single number $\mathcal{E}$, which can usually just be set to 0.0001, requiring no tuning. Ridge Regression, on the other hand, requires a vector of λ values to test, which can be more difficult to choose appropriately. What is more, Impatient Learning is easy to adapt to other differentiable, convex loss functions

beyond just the squared loss, and to situations where there are linear constraints on the coefficients, unlike the standard implementation of Ridge Regression. Finally, Impatient Learning can easily be modified to use subsamples of the data for each gradient calculation, so that it can be applied to extremely large data sets, even ones that do not fit in memory.

Impatient Learning Convergence

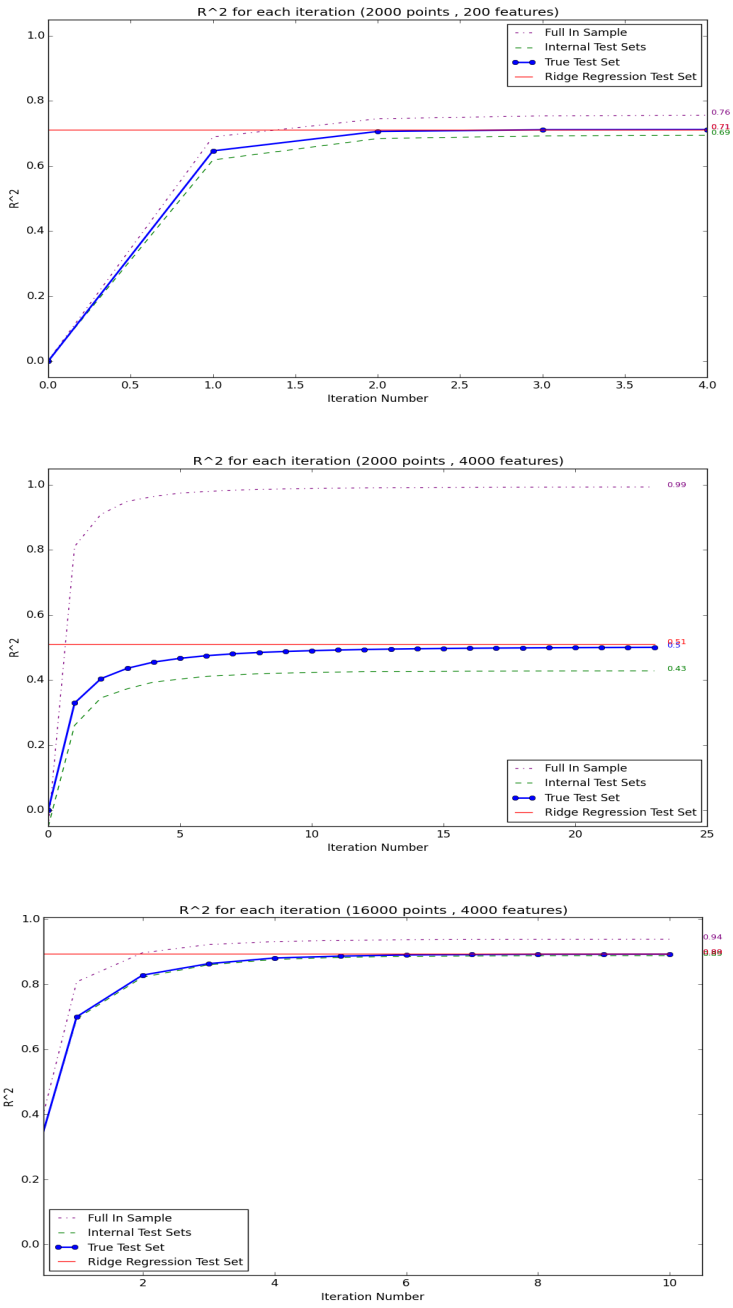

Figure 6.1: Impatient Learning convergence when using a linear prediction function with squared error (i.e. L_2) loss function, trained on synthetic linear data with gaussian additive label noise and points drawn from an isometric multi-variate gaussian. The blue bold line shows R^2 for the Impatient Learning on an entirely withheld test set, the dashed green line shows the total validation set error E (which is smooth and concave as desired), the dashed-and-dotted purple line shows the in sample error made by the entire learner, and the thin red horizontal line shows Ridge Regression performance (with automatic leave-one-out cross-validation parameter selection), which is approached by the blue bold line as desired.

Impatient Learning Number of Iterations

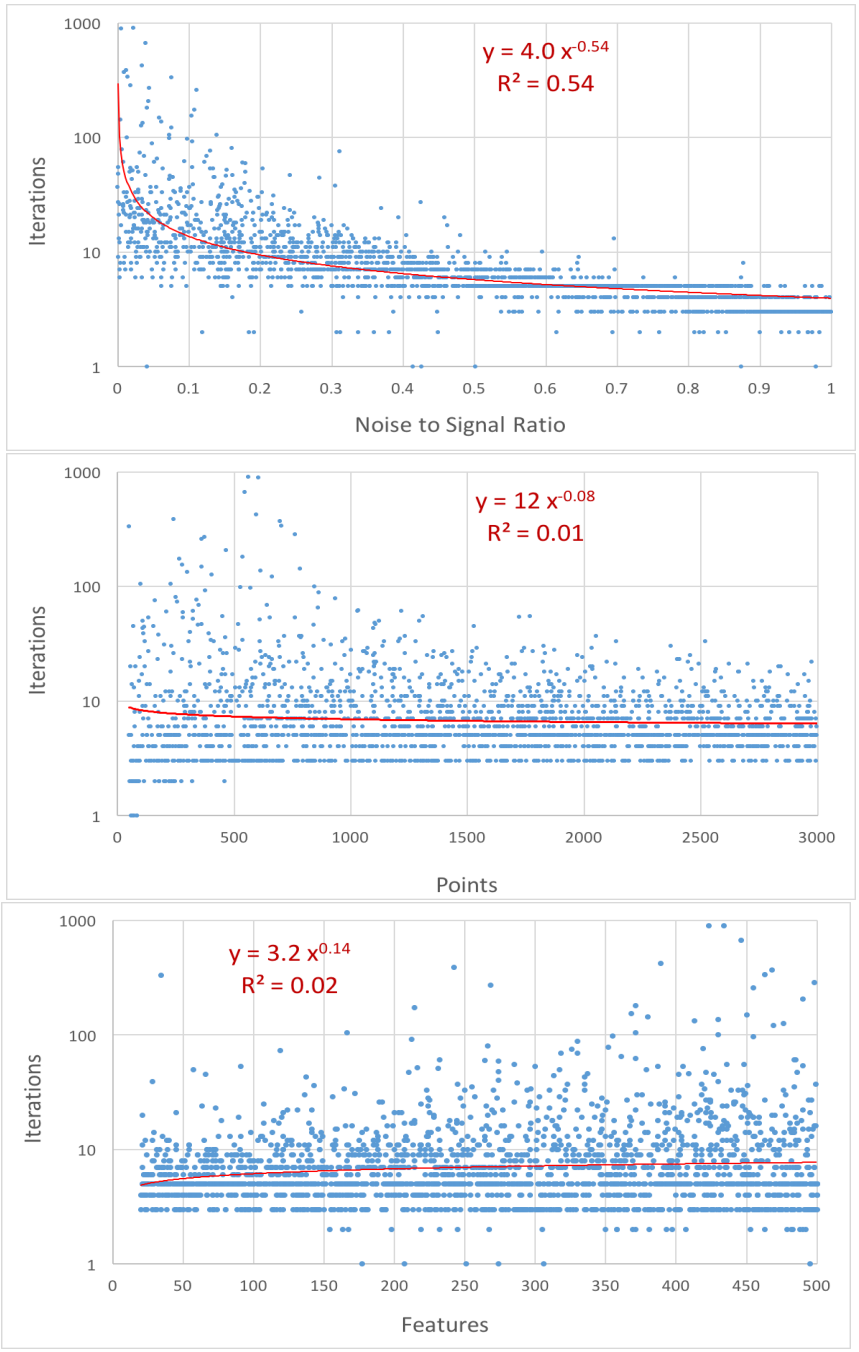

Figure 6.2: These charts show log plots of the number of iterations used by the Impatient Learning algorithm with a linear prediction function and squared error loss function, trained on many synthetic data sets, constructing from a linear model with random coefficients, gaussian additive label noise, and points drawn from an isometric multi-variate gaussian. The number of training points, number of features, and noise to signal ratio for the synthetic cases (which are used as the x-axes) were chosen at random in each test case, so that these three variables are statistically independent. The red lines show "best fit" monomial curves that were fit to the data, and the equations and R^2 values for these curves is shown as well. Only the top chart admits a meaningfully strong relationship.

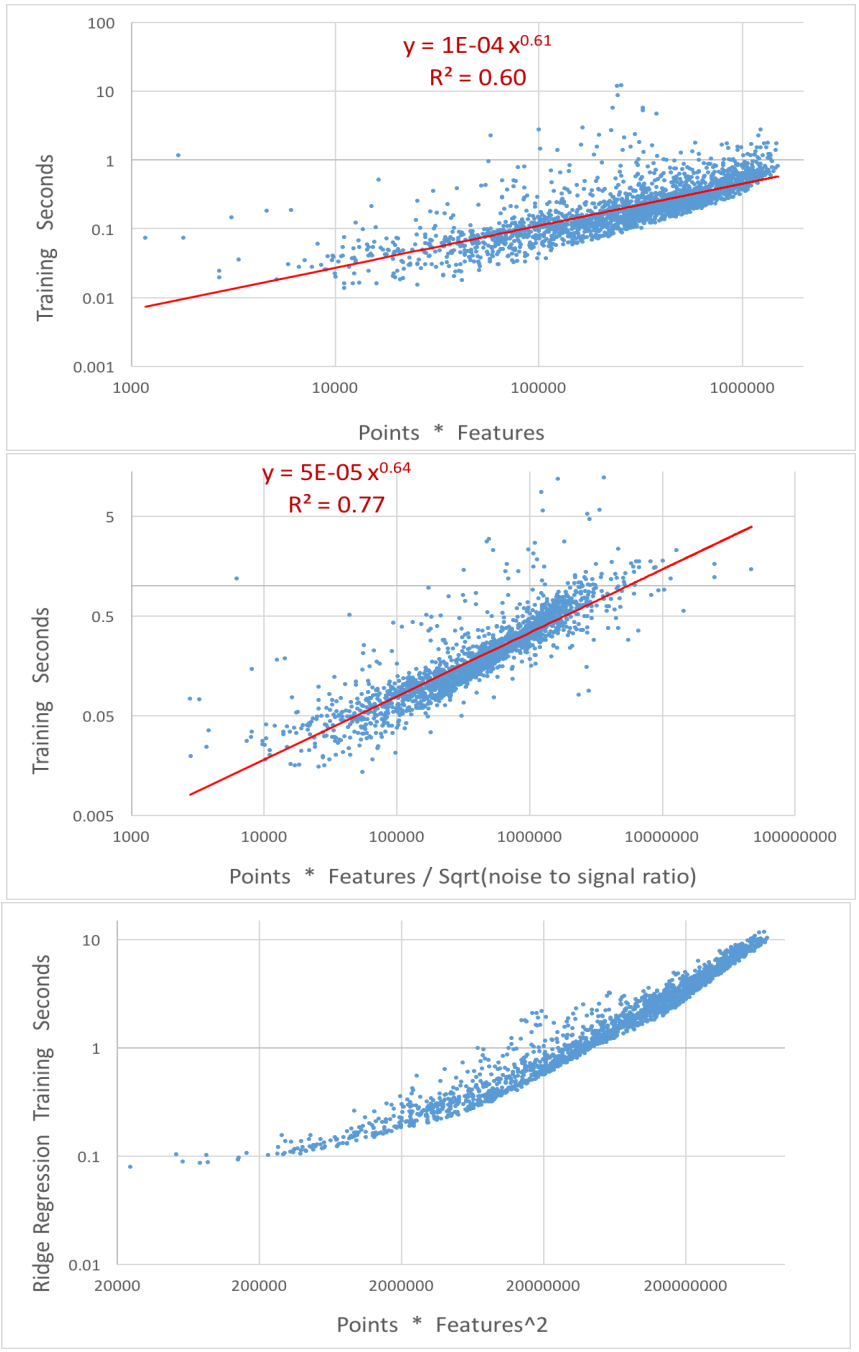

Figure 6.3: Using the same randomly sampled learning problems from Figure 6.2, these log-log plots show (a) the number of training seconds for Impatient Learning vs. points times features, (b) the number of training seconds for impatient Learning vs. points times features divided by the noise to signal ratio of the additive label noise, and (c) the number of training seconds for Ridge Regression vs. points times the square of features.

Part IV

Theory

Chapter 7

Relative Deviation and Unbounded Loss Bounds

Our work in Chapters 3 and 4 on outliers points to an interesting theoretical question: if extreme outliers can destroy the performance of algorithms like Ridge Regression, could just the possibility of such outliers ruin our ability to prove that algorithms can learn effectively? Examining common approaches to bounding the prediction error in terms of the training error does not give one hope, since as we will see they generally assume bounded loss functions. Worse still, this bound on the loss function generally enters into the resulting inequality, so if it is too large, the inequalities derived may be vacuous.

Fortunately, this limitation is not fundamental. In this Chapter we investigate how one can successfully analyze the ability of machine learning algorithms to generalize when a bounded loss function cannot be assumed. Our approach, using relative deviation bounds, fills in a number of gaps in existing proofs in the literature.

7.1 Introduction

Most generalization bounds in learning theory hold only for bounded loss functions. This includes standard VC-dimension bounds (Vapnik, 1998), Rademacher complexity (Koltchinskii and Panchenko, 2000a; Bartlett et al., 2002a; Koltchinskii and Panchenko, 2002; Bartlett and Mendelson, 2002) or local Rademacher complexity bounds (Koltchinskii, 2006; Bartlett et al., 2002b), as well as most other bounds based on other complexity terms. This assumption is typically unrelated to the statistical nature of the problem considered but it is convenient since when the loss functions are uniformly bounded, standard tools such as Hoeffding's inequality (Hoeffding, 1963; Azuma, 1967), McDiarmid's inequality (McDiarmid, 1989), or Talagrand's concentration inequality (Talagrand, 1994) apply.

There are however natural learning problems where the boundedness assumption does not hold. This includes unbounded regression tasks where the target labels are not uniformly bounded, and a variety of applications such as sample bias correction (Dudík et al., 2006; Huang et al., 2006; Cortes et al., 2008; Sugiyama et al., 2008; Bickel et al., 2007), domain adaptation (Ben-David et al., 2007; Blitzer et al., 2008; Daumé III and Marcu, 2006; Jiang and Zhai, 2007; Mansour et al., 2009; Cortes and Mohri, 2013), or the analysis of boosting (Dasgupta and Long, 2003), where the importance weighting technique is used (Cortes et al., 2010a). It is therefore critical to derive learning guarantees that hold for these scenarios and the general case of unbounded loss functions.

When the class of functions is unbounded, a single function may take arbitrarily large values with arbitrarily small probabilities. This is probably the main challenge in deriving uniform convergence bounds for unbounded losses. This problem can be avoided by assuming the existence of an envelope, that is a single non-negative function with a finite expectation lying above the absolute value of the loss of every function in the hypothesis set (Dudley, 1984; Pollard, 1984; Dudley, 1987; Pollard, 1989; Haussler, 1992), an alternative assumption

similar to Hoeffding's inequality based on the expectation of a hyperbolic function, a quantity similar to the moment-generating function, is used by Meir and Zhang (2003). However, in many problems, e.g., in the analysis of importance weighting even for common distributions, there exists no suitable envelope function (Cortes et al., 2010a). Instead, the second or some other αth-moment of the loss seems to play a critical role in the analysis. Thus, instead, we will consider here the assumption that some αth-moment of the loss functions is bounded as in Vapnik (1998, 2006b).

This chapter presents in detail two-sided generalization bounds for unbounded loss functions under the assumption that some αth-moment of the loss functions, $\alpha > 1$, is bounded. The proof of these bounds makes use of relative deviation generalization bounds in binary classification, which we also prove and discuss in detail. Much of the results and material we present is not novel and the chapter has therefore a survey nature. However, our presentation is motivated by the fact that the proofs given in the past for these generalization bounds were either incorrect or incomplete.

We now discuss in more detail prior results and proofs. One-side relative deviation bounds were first given by Vapnik (1998), later improved by a constant factor by Anthony and Shawe-Taylor (1993). These publications and several others have all relied on a lower bound on the probability that a binomial random variable of m trials exceeds its expected value when the bias verifies $p > \frac{1}{m}$. This also later appears in Vapnik (2006a) and implicitly in other publications referring to the relative deviations bounds of Vapnik (1998). To the best of our knowledge, no actual proof of this inequality was ever given in the past in the machine learning literature before our recent work (Greenberg and Mohri, 2013). One attempt was made to prove this lemma in the context of the analysis of some generalization bounds (Jaeger, 2005), but unfortunately that proof is not sufficient to support the general case needed for the proof of the relative deviation bound of Vapnik (1998).

We present the proof of two-sided relative deviation bounds in detail using the recent

results of Greenberg and Mohri (2013). The two-sided versions we present, as well as several consequences of these bounds, appear in Anthony and Bartlett (1999). However, we could not find a full proof of the two-sided bounds in any prior publication. Our presentation shows that the proof of the other side of the inequality is not symmetric and cannot be immediately obtained from that of the first side inequality. Additionally, this requires another proof related to the binomial distributions given by Greenberg and Mohri (2013).

Relative deviation bounds are very informative guarantees in machine learning of independent interest, regardless of the key role they play in the proof of unbounded loss learning bounds. They lead to sharper generalization bounds whose right-hand side is expressed as the interpolation of a $O(1/m)$ term and a $O(1/\sqrt{m})$ term that admits as a multiplier the empirical error or the generalization error. In particular, when the empirical error is zero, this leads to faster rate bounds. We present in detail the proof of this type of results as well as that of several others of interest (Anthony and Bartlett, 1999). Let us mention that, in the form presented by Vapnik (1998), relative deviation bounds suffer from a discontinuity at zero (zero denominator), a problem that also affects inequalities for the other side and which seems not to have been rigorously treated by previous work. Our proofs and results explicitly deal with this issue.

We use relative deviations bounds to give the full proofs of two-sided generalization bounds for unbounded losses with finite moments of order α, both in the case $1 < \alpha \leq 2$ and the case $\alpha > 2$. One-sided generalization bounds for unbounded loss functions were first given by Vapnik (1998, 2006b) under the same assumptions and also using relative deviations. The one-sided version of our bounds for the case $1 < \alpha \leq 2$ coincides with that of (Vapnik, 1998, 2006b) modulo a constant factor, but the proofs given by Vapnik in both books seem to be incorrect.[1] The core component of our proof is based on a different technique using

[1] In (Vapnik, 1998)[p.204-206], statement (5.37) cannot be derived from assumption (5.35), contrary to what is claimed by the author, and in general it does not hold: the first integral in (5.37) is restricted to a sub-domain and is thus smaller than the integral of (5.35). Furthermore, the main statement claimed in

Hölder's inequality. We also present some more explicit bounds for the case $1 < \alpha \leq 2$ by approximating a complex term appearing in these bounds. The one-sided version of the bounds for the case $\alpha > 2$ are also due to Vapnik (1998, 2006b) with similar questions about the proofs.[2] In that case as well, we give detailed proofs using the Cauchy-Schwarz inequality in the most general case where a positive constant is used in the denominator to avoid the discontinuity at zero. These learning bounds can be used directly in the analysis of unbounded loss functions as in the case of importance weighting (Cortes et al., 2010a).

The remainder of this chapter is organized as follows. In Section 7.2, we briefly introduce some definitions and notation used in the next sections. Section 7.3 presents in detail relative deviation bounds as well as several of their consequences. Next, in Section 7.4 we present generalization bounds for unbounded loss functions under the assumption that the moment of order α is bounded first in the case $1 < \alpha \leq 2$ (Section 7.4.1), then in the case $\alpha > 2$ (Section 7.4.2).

7.2 Preliminaries

We consider an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, which in the particular case of binary classification is $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$, or a measurable subset of $\mathbb{R}$ in regression. We denote by D a distribution over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. For a sample S of size m drawn from D^m, we will denote by $\widehat{D}$ the corresponding empirical distribution, that is the distribution corresponding to drawing a point from S uniformly at random. Throughout this chapter, H denotes a hypothesis of functions mapping from $\mathcal{X}$ to $\mathcal{Y}$. The loss incurred by hypothesis

Section (5.6.2) is not valid. In (Vapnik, 2006b)[p.200-202], the author invokes the *Lagrange method* to show the main inequality, but the proof steps are not mathematically justified. Even with our best efforts, we could not justify some of the steps and strongly believe the proof not to be correct. In particular, the way function z is concluded to be equal to one over the first interval is suspicious and not rigorously justified.

[2]Several of the comments we made for the case $1 < \alpha \leq 2$ hold here as well. In particular, the author's proof is not based on clear mathematical justifications. Some steps seem suspicious and are not convincing, even with our best efforts to justify them.

$h \in H$ at $z \in \mathcal{Z}$ is denoted by $L(h, z)$. L is assumed to be non-negative, but not necessarily bounded. We denote by $\mathcal{L}(h)$ the expected loss or generalization error of a hypothesis $h \in H$ and by $\widehat{\mathcal{L}}_S(h)$ its empirical loss for a sample S:

$$\mathcal{L}(h) = \mathop{\mathbb{E}}_{z \sim D}[L(h, z)] \qquad \widehat{\mathcal{L}}_S(h) = \mathop{\mathbb{E}}_{z \sim \widehat{D}}[L(h, z)]. \tag{7.1}$$

For any $\alpha > 0$, we also use the notation $\mathcal{L}_\alpha(h) = \mathbb{E}_{z \sim D}[L^\alpha(h, z)]$ and $\widehat{\mathcal{L}}_\alpha(h) = \mathbb{E}_{z \sim \widehat{D}}[L^\alpha(h, z)]$ for the αth moments of the loss. When the loss L coincides with the standard zero-one loss used in binary classification, we equivalently use the following notation

$$R(h) = \mathop{\mathbb{E}}_{z=(x,y) \sim D}[1_{h(x) \neq y}] \qquad \widehat{R}_S(h) = \mathop{\mathbb{E}}_{z=(x,y) \sim \widehat{D}}[1_{h(x) \neq y}]. \tag{7.2}$$

We will sometimes use the shorthand x_1^m to denote a sample of $m > 0$ points $(x_1, \ldots, x_m) \in \mathcal{X}^m$. For any hypothesis set H of functions mapping $\mathcal{X}$ to $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$ and sample x_1^m, we denote by $\mathbb{S}_H(x_1^m)$ the number of distinct dichotomies generated by H over that sample and by $\Pi_m(H)$ the growth function:

$$\mathbb{S}_H(x_1^m) = \text{Card}\left(\left\{(h(x_1), \ldots, h(x_m)) \colon h \in H\right\}\right) \tag{7.3}$$

$$\Pi_m(H) = \max_{x_1^m \in \mathcal{X}^m} \mathbb{S}_H(x_1^m). \tag{7.4}$$

7.3 Relative deviation bounds

In this section we prove a series of relative deviation learning bounds which we use in the next section for deriving generalization bounds for unbounded loss functions. We will assume throughout the chapter, as is common in much of learning theory, that each expression of the form $\sup_{h \in H}[...]$ is a measurable function, which is not guaranteed when H is not a countable

set. This assumption holds nevertheless in most common applications of machine learning.

We start with the proof of a symmetrization lemma (Lemma 9) originally presented by Vapnik (1998), which is used by Anthony and Shawe-Taylor (1993). These publications and several others have all relied on a lower bound on the probability that a binomial random variable of m trials exceeds its expected value when the bias verifies $p > \frac{1}{m}$. To our knowledge, no rigorous proof of this fact was ever provided in the literature in the full generality needed. The proof of this result was recently given by Greenberg and Mohri (2013).

Lemma 8 (Greenberg and Mohri (2013))**.** *Let X be a random variable distributed according to the binomial distribution $B(m, p)$ with m a positive integer (the number of trials) and $p > \frac{1}{m}$ (the probability of success of each trial). Then, the following inequality holds:*

$$\mathbb{P}\Big[X \geq \mathbb{E}[X]\Big] > \frac{1}{4}, \tag{7.5}$$

where $\mathbb{E}[X] = mp$.

The lower bound is never reached but is approached asymptotically when $m = 2$ as $p \to \frac{1}{2}$ from the right.

Our proof of Lemma 9 is more concise than that of Vapnik (1998). Furthermore, our statement and proof handle the technical problem of discontinuity at zero ignored by previous authors. The denominator may in general become zero, which would lead to an undefined result. We resolve this issue by including an arbitrary positive constant τ in the denominator in most of our expressions.

For the proof of the following result, we will use the function F defined over $(0, +\infty) \times (0, +\infty)$ by $F\colon (x, y) \mapsto \frac{x-y}{\sqrt[\alpha]{\frac{1}{2}[x+y+\frac{1}{m}]}}$. By Lemma 12, $F(x, y)$ is increasing in x and decreasing in y.

Lemma 9. *Let $1 < \alpha \leq 2$. Assume that $m\epsilon^{\frac{\alpha}{\alpha-1}} > 1$. Then, for any hypothesis set H and*

any $\tau > 0$, the following holds:

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}>\epsilon\right]\leq 4\mathbb{P}_{S,S'\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_{S'}(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}}>\epsilon\right].$$

Proof. We give a concise version of the proof given by (Vapnik, 1998). We first show that the following implication holds for any $h \in H$:

$$\left(\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}>\epsilon\right)\wedge\left(\widehat{R}_{S'}(h)>R(h)\right)\Rightarrow F(\widehat{R}_{S'}(h),\widehat{R}_S(h))>\epsilon. \tag{7.6}$$

The first condition can be equivalently rewritten as $\widehat{R}_S(h) < R(h) - \epsilon(R(h)+\tau)^{\frac{1}{\alpha}}$, which implies

$$\widehat{R}_S(h) < R(h) - \epsilon R(h)^{\frac{1}{\alpha}} \qquad \text{and} \qquad \epsilon^{\frac{\alpha}{\alpha-1}} < R(h), \tag{7.7}$$

since $\widehat{R}_S(h) \geq 0$. Assume that the antecedent of the implication (7.6) holds for $h \in H$. Then, in view of the monotonicity properties of function F (Lemma 12), we can write:

$$\begin{aligned}
F(\widehat{R}_{S'}(h),\widehat{R}_S(h)) &\geq F(R(h),R(h)-\epsilon R(h)^{\frac{1}{\alpha}}) && (\widehat{R}_{S'}(h)>R(h) \text{ and 1st ineq. of (7.7)})\\
&= \frac{R(h)-(R(h)-\epsilon R(h)^{\frac{1}{\alpha}})}{\sqrt[\alpha]{\frac{1}{2}[2R(h)-\epsilon R(h)^{\frac{1}{\alpha}}+\frac{1}{m}]}}\\
&\geq \frac{\epsilon R(h)^{\frac{1}{\alpha}}}{\sqrt[\alpha]{\frac{1}{2}[2R(h)-\epsilon^{\frac{\alpha}{\alpha-1}}+\frac{1}{m}]}} && (\text{2nd ineq. of (7.7)})\\
&> \frac{\epsilon R(h)^{\frac{1}{\alpha}}}{\sqrt[\alpha]{\frac{1}{2}[2R(h)]}}=\epsilon, && (m\epsilon^{\frac{\alpha}{\alpha-1}}>1)
\end{aligned}$$

which proves (7.6). Now, by definition of the supremum, for any $\eta > 0$, there exists $h_0 \in H$ such that

$$\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}-\frac{R(h_0)-\widehat{R}_S(h_0)}{\sqrt[\alpha]{R(h_0)+\tau}}\leq\eta. \tag{7.8}$$

Using the definition of h_0 and implication (7.6), we can write

$$\begin{aligned}
&\mathbb{P}_{S,S'\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_{S'}(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}}>\epsilon\right]\\
&\geq \mathbb{P}_{S,S'\sim D^m}\left[\frac{\widehat{R}_{S'}(h_0)-\widehat{R}_S(h_0)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h_0)+\widehat{R}_{S'}(h_0)+\frac{1}{m}]}}>\epsilon\right] && \text{(by def. of sup)}\\
&\geq \mathbb{P}_{S,S'\sim D^m}\left[\left(\frac{R(h_0)-\widehat{R}_S(h_0)}{\sqrt[\alpha]{R(h_0)+\tau}}>\epsilon\right)\wedge\Big(R_{S'}(h_0)>R(h_0)\Big)\right] && \text{(implication (7.6))}\\
&= \mathbb{P}_{S\sim D^m}\left[\frac{R(h_0)-\widehat{R}_S(h_0)}{\sqrt[\alpha]{R(h_0)+\tau}}>\epsilon\right]\mathbb{P}_{S'\sim D^m}\left[R_{S'}(h_0)>R(h_0)\right] && \text{(independence).}
\end{aligned}$$

We now show that this implies the following inequality

$$\mathbb{P}_{S,S'\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_{S'}(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}}>\epsilon\right]\geq\frac{1}{4}\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}>\epsilon+\eta\right], \tag{7.9}$$

by distinguishing two cases. If $R(h_0)>\epsilon^{\frac{\alpha}{\alpha-1}}$, since $\epsilon^{\frac{\alpha}{\alpha-1}}>\frac{1}{m}$, by Theorem 8 the inequality $\mathbb{P}_{S'\sim D^m}\left[R_{S'}(h_0)>R(h_0)\right]>\frac{1}{4}$ holds, which yields immediately (7.9). Otherwise we have $R(h_0)\leq\epsilon^{\frac{\alpha}{\alpha-1}}$. Then, by (7.7), the condition $\frac{R(h_0)-\widehat{R}_S(h_0)}{\sqrt[\alpha]{R(h_0)+\tau}}>\epsilon$ cannot hold for any sample $S\sim D^m$ which by (7.8) implies that the condition $\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}>\epsilon+\eta$ cannot hold for any sample $S\sim D^m$, in which case (7.9) trivially holds. Now, since (7.9) holds for all $\eta>0$, we can take the limit $\eta\to 0$ and use the right-continuity of the cumulative distribution to obtain

$$\mathbb{P}_{S,S'\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_{S'}(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}}>\epsilon\right]\geq\frac{1}{4}\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}>\epsilon\right],$$

which completes the proof of Lemma 9. □

Note that the factor of 4 in the statement of lemma 9 can be modestly improved by

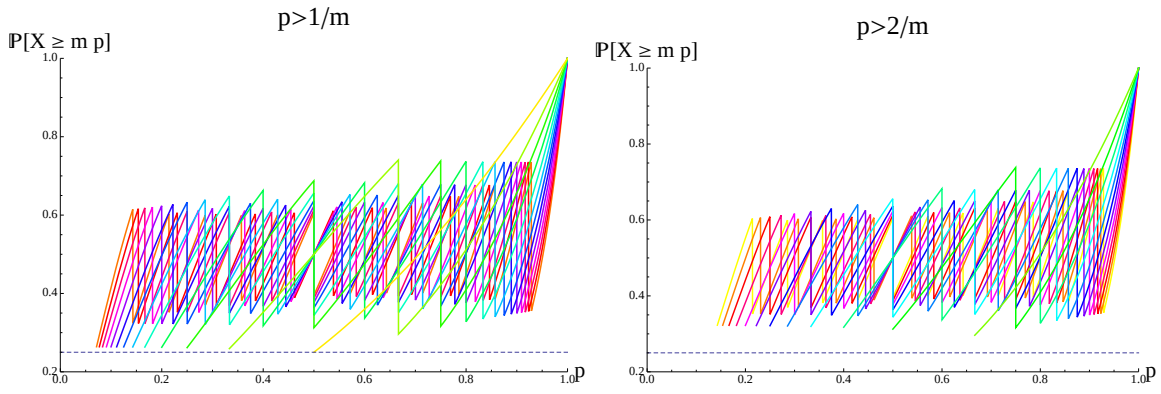

Figure 7.1: These plots depict $\mathbb{P}[X \geq \mathbb{E}[X]]$, the probability that a binomially distributed random variable X exceeds its expectation, as a function of the trial success probability p. The left plot shows only regions satisfying $p > \frac{1}{m}$ whereas the right plot shows only regions satisfying $p > \frac{2}{m}$. Each colored line corresponds to a different number of trials, $m = 2, 3, \ldots, 14$. The dashed horizontal line at $\frac{1}{4}$ represents the value of the lower bound used in the proof of lemma 9.

changing the condition assumed from $\epsilon^{\frac{\alpha}{\alpha-1}} > \frac{1}{m}$ to $\epsilon^{\frac{\alpha}{\alpha-1}} > \frac{k}{m}$ for constant values of $k > 1$. This leads to a slightly better lower bound on $\mathbb{P}_{S' \sim D^m}\left[R_{S'}(h_0) > R(h_0)\right]$, e.g. 3.375 rather than 4 for $k = 2$, at the expense of not covering cases where the number of samples m is less than $\frac{k}{\epsilon^{\frac{\alpha}{\alpha-1}}}$. For some values of k, e.g. $k = 2$, covering these cases is not needed for the proof of our main theorem (Theorem 11) though. However, this does not seem to simplify the critical task of proving a lower bound on $\mathbb{P}_{S' \sim D^m}\left[R_{S'}(h_0) > R(h_0)\right]$, that is the probability that a binomial random variable $B(m, p)$ exceeds its expected value when $p > \frac{k}{m}$. One might hope that restricting the range of p in this way would help simplify the proof of a lower bound on the probability of a binomial exceeding its expected value. Unfortunately, our analysis of this problem and proof (Greenberg and Mohri, 2013) suggest that this is not the case since the regime where p is small seems to be the easiest one to analyze for this problem.

The result of Lemma 9 is a one-sided inequality. The proof of a similar result (Lemma 11) with the roles of $R(h)$ and $\widehat{R}_S(h)$ interchanged makes use of the following theorem.

Lemma 10 (Greenberg and Mohri (2013))**.** *Let X be a random variable distributed according*

to the binomial distribution $B(m,p)$ with m a positive integer and $p < 1 - \frac{1}{m}$. Then, the following inequality holds:

$$\mathbb{P}\Big[X \leq \mathbb{E}[X]\Big] > \frac{1}{4}, \tag{7.10}$$

where $\mathbb{E}[X] = mp$.

The proof of the following lemma (Lemma 11) is novel.[3] While the general strategy of the proof is similar to that of Lemma 9, there are some non-trivial differences due to the requirement $p < 1 - \frac{1}{m}$ of Theorem 10. The proof is not symmetric as shown by the details given below.

Lemma 11. *Let $1 < \alpha \leq 2$. Assume that $m\epsilon^{\frac{\alpha}{\alpha-1}} > 1$. Then, for any hypothesis set H and any $\tau > 0$ the following holds:*

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_S(h) - R(h)}{\sqrt[\alpha]{\widehat{R}_S(h)+\tau}} > \epsilon\right] \leq 4\mathbb{P}_{S,S'\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_{S'}(h) - \widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}} > \epsilon\right]$$

Proof. Proceeding in a way similar to the proof of Lemma 9, we first show that the following implication holds for any $h \in H$:

$$\left(\frac{\widehat{R}_S(h) - R(h)}{\sqrt[\alpha]{\widehat{R}_S(h)+\tau}} > \epsilon\right) \wedge \Big(R(h) \geq \widehat{R}_{S'}(h)\Big) \Rightarrow F(\widehat{R}_S(h), \widehat{R}_{S'}(h)) > \epsilon. \tag{7.11}$$

The first condition can be equivalently rewritten as $R(h) < \widehat{R}_S(h) - \epsilon(\widehat{R}_S(h)+\tau)^{\frac{1}{\alpha}}$, which implies

$$R(h) < \widehat{R}_S(h) - \epsilon\widehat{R}_S(h)^{\frac{1}{\alpha}} \qquad \text{and} \qquad \epsilon^{\frac{\alpha}{\alpha-1}} < \widehat{R}_S(h), \tag{7.12}$$

since $\widehat{R}_S(h) \geq 0$. Assume that the antecedent of the implication (7.11) holds for $h \in H$.

[3] A version of this lemma is stated in (Boucheron et al., 2005), but no proof is given.

Then, in view of the monotonicity properties of function F (Lemma 12), we can write:

$$\begin{aligned}
F(\widehat{R}_S(h), \widehat{R}_{S'}(h)) &\geq F(\widehat{R}_S(h), R(h)) && (R(h) \geq \widehat{R}_{S'}(h)) \\
&\geq F(\widehat{R}_S(h), \widehat{R}_S(h) - \epsilon \widehat{R}_S(h)^{\frac{1}{\alpha}}) && \text{(1st ineq. of (7.12))} \\
&= \frac{\widehat{R}_S(h) - (\widehat{R}_S(h) - \epsilon \widehat{R}_S(h)^{\frac{1}{\alpha}})}{\sqrt[\alpha]{\frac{1}{2}[2\widehat{R}_S(h) - \epsilon \widehat{R}_S(h)^{\frac{1}{\alpha}} + \frac{1}{m}]}} \\
&\geq \frac{\epsilon R(h)^{\frac{1}{\alpha}}}{\sqrt[\alpha]{\frac{1}{2}[2R(h) - \epsilon^{\frac{\alpha}{\alpha-1}} + \frac{1}{m}]}} && \text{(2nd ineq. of (7.12))} \\
&> \frac{\epsilon R(h)^{\frac{1}{\alpha}}}{\sqrt[\alpha]{\frac{1}{2}[2R(h)]}} = \epsilon, && (m\epsilon^{\frac{\alpha}{\alpha-1}} > 1)
\end{aligned}$$

which proves (7.11). For the application of Theorem 10 to a hypothesis h, the condition $R(h) < 1 - \frac{1}{m}$ is required. Observe that this is implied by the assumptions $\widehat{R}_S(h) \geq \epsilon^{\frac{\alpha}{\alpha-1}}$ and $m\epsilon^{\frac{\alpha}{\alpha-1}} > 1$:

$$R(h) < \widehat{R}_S(h) - \epsilon \sqrt[\alpha]{\widehat{R}_S(h)} \leq 1 - \epsilon\, \epsilon^{\frac{1}{\alpha-1}} = 1 - \epsilon^{\frac{\alpha}{\alpha-1}} < 1 - \frac{1}{m}.$$

The rest of the proof proceeds nearly identically to that of Lemma 9. □

In the statements of all the following results, the term $\mathbb{E}_{x_1^{2m} \sim D^{2m}}[\mathbb{S}_H(x_1^{2m})]$ can be replaced by the upper bound $\Pi_{2m}(H)$ to derive simpler expressions. By Sauer's lemma (Sauer, 1972; Vapnik and Chervonenkis, 1971), the VC-dimension d of the family H can be further used to bound these quantities since $\Pi_{2m}(H) \leq \left(\frac{2em}{d}\right)^d$ for $d \leq 2m$. The first inequality of the following theorem was originally stated and proven by Vapnik (1998, 2006b), later by Anthony and Shawe-Taylor (1993) (in the special case $\alpha = 2$) with a somewhat more favorable constant, in both cases modulo the incomplete proof of the symmetrization and the technical issue related to the denominator taking the value zero, as already pointed out. The second inequality of the theorem and its proof are novel. Our proofs benefit from the improved

analysis of Anthony and Shawe-Taylor (1993).

Theorem 11. *For any hypothesis set H of functions mapping a set $\mathcal{X}$ to $\{0,1\}$, and any fixed $1 < \alpha \leq 2$ and $\tau > 0$, the following two inequalities hold:*

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}} > \epsilon\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right)$$
$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H}\frac{\widehat{R}_S(h)-R(h)}{\sqrt[\alpha]{\widehat{R}_S(h)+\tau}} > \epsilon\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right).$$

Proof. We first consider the case where $m\epsilon^{\frac{\alpha}{\alpha-1}} \leq 1$, which is not covered by Lemma 9. We can then write

$$4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left[\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right] \geq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left[\frac{-1}{2^{\frac{\alpha+2}{\alpha}}}\right] > 1,$$

for $1 < \alpha \leq 2$. Thus, the bounds of the theorem hold trivially in that case. On the other hand, when $m\epsilon^{\frac{\alpha}{\alpha-1}} \geq 1$, we can apply Lemma 9 and Lemma 11. Therefore, to prove theorem 11, it is sufficient to work with the symmetrized expression $\sup_{h\in H}\frac{\widehat{R}_{S'}(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h)+\widehat{R}_{S'}(h)+\frac{1}{m}]}}$, rather than working directly with our original expressions $\sup_{h\in H}\frac{R(h)-\widehat{R}_S(h)}{\sqrt[\alpha]{R(h)+\tau}}$ and $\sup_{h\in H}\frac{\widehat{R}(h)-R_S(h)}{\sqrt[\alpha]{\widehat{R}(h)+\tau}}$. To upper bound the probability that the symmetrized expression is larger than ϵ, we begin by introducing a vector of Rademacher random variables $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$, where the σ_i are independent, identically distributed random variables each equally likely to take the value $+1$

or -1. Using the shorthand x_1^{2m} for $(x_1, \ldots, x_{2m})$, we can then write

$$\begin{aligned}
&\mathbb{P}_{S,S' \sim D^m}\left[\sup_{h \in H} \frac{\widehat{R}_{S'}(h) - \widehat{R}_S(h)}{\sqrt[\alpha]{\frac{1}{2}[\widehat{R}_S(h) + \widehat{R}_{S'}(h) + \frac{1}{m}]}} > \epsilon\right] \\
&\qquad = \mathbb{P}_{x_1^{2m} \sim D^{2m}}\left[\sup_{h \in H} \frac{\frac{1}{m}\sum_{i=1}^m (h(x_{m+i}) - h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^m (h(x_{m+i}) + h(x_i)) + 1]}} > \epsilon\right] \\
&\qquad = \mathbb{P}_{x_1^{2m} \sim D^{2m}, \boldsymbol{\sigma}}\left[\sup_{h \in H} \frac{\frac{1}{m}\sum_{i=1}^m \sigma_i (h(x_{m+i}) - h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^m (h(x_{m+i}) + h(x_i)) + 1]}} > \epsilon\right] \\
&\qquad = \mathop{\mathbb{E}}_{x_1^{2m} \sim D^{2m}}\left[\mathbb{P}_{\boldsymbol{\sigma}}\left[\sup_{h \in H} \frac{\frac{1}{m}\sum_{i=1}^m \sigma_i (h(x_{m+i}) - h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^m (h(x_{m+i}) + h(x_i)) + 1]}} > \epsilon \,\middle|\, x_1^{2m}\right]\right].
\end{aligned}$$

Now, for a fixed x_1^{2m}, we have $\mathbb{E}_{\boldsymbol{\sigma}}\left[\frac{\frac{1}{m}\sum_{i=1}^m \sigma_i (h(x_{m+i}) - h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^m (h(x_{m+i}) + h(x_i)) + 1]}}\right] = 0$, thus, by Hoeffding's inequality, we can write

$$\begin{aligned}
&\mathbb{P}_{\boldsymbol{\sigma}}\left[\frac{\frac{1}{m}\sum_{i=1}^m \sigma_i (h(x_{m+i}) - h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^m (h(x_{m+i}) + h(x_i))]}} > \epsilon \,\middle|\, x_1^{2m}\right] \\
&\qquad \leq \exp\left(\frac{-[\sum_{i=1}^{2m} (h(x_{m+i}) + h(x_i)) + 1]^{\frac{2}{\alpha}} m^{\frac{2(\alpha-1)}{\alpha}} \epsilon^2}{2^{\frac{\alpha+2}{\alpha}} \sum_{i=1}^m (h(x_{m+i}) - h(x_i))^2}\right) \\
&\qquad \leq \exp\left(\frac{-[\sum_{i=1}^{2m} (h(x_{m+i}) + h(x_i))]^{\frac{2}{\alpha}} m^{\frac{2(\alpha-1)}{\alpha}} \epsilon^2}{2^{\frac{\alpha+2}{\alpha}} \sum_{i=1}^m (h(x_{m+i}) - h(x_i))^2}\right).
\end{aligned}$$

Since the variables $h(x_i)$, $i \in [1, 2m]$, take values in $\{0, 1\}$, we can write

$$\begin{aligned}
\sum_{i=1}^m (h(x_{m+i}) - h(x_i))^2 &= \sum_{i=1}^m h(x_{m+i}) + h(x_i) - 2h(x_{m+i})h(x_i) \\
&\leq \sum_{i=1}^m h(x_{m+i}) + h(x_i) \leq \Big[\sum_{i=1}^m h(x_{m+i}) + h(x_i)\Big]^{\frac{2}{\alpha}},
\end{aligned}$$

where the last inequality holds since $\alpha \leq 2$ and the sum is either zero or greater than or

equal to one. In view of this identity, we can write

$$\mathbb{P}_{\boldsymbol{\sigma}}\left[\frac{\frac{1}{m}\sum_{i=1}^{m}\sigma_i(h(x_{m+i})-h(x_i))}{\sqrt[\alpha]{\frac{1}{2m}[\sum_{i=1}^{m}(h(x_{m+i})+h(x_i))]}}>\epsilon\,\middle|\,x_1^{2m}\right]\leq\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right).$$

We note now that the supremum over $h\in H$ in the left-hand side expression in the statement of our theorem need not be over all hypothesis in H: without changing its value, we can replace H with a smaller hypothesis set where only one hypothesis remains for each unique binary vector $(h(x_1),h(x_2),\ldots,h(x_{2m}))$. The number of such hypotheses is $\mathbb{S}_H(x_1^{2m})$, thus, by the union bound, the following holds:

$$\mathbb{P}_{\boldsymbol{\sigma}}\left[\sup_{h\in H}\frac{\sum_{i=1}^{m}\sigma_i(h(x_{m+i})-h(x_i))}{\sqrt[\alpha]{\frac{1}{2}[\sum_{i=1}^{m}(h(x_{m+i})+h(x_i))]}}>\epsilon\,\middle|\,x_1^{2m}\right]\leq\mathbb{S}_H(x_1^{2m})\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right).$$

The result follows by taking expectations with respect to x_1^{2m} and applying Lemma 9 and Lemma 11 respectively. □

Corollary 5. *Let $1<\alpha\leq 2$ and let H be a hypothesis set of functions mapping $\mathcal{X}$ to $\{0,1\}$. Then, for any $\delta>0$, each of the following two inequalities holds with probability at least $1-\delta$:*

$$R(h)-\widehat{R}_S(h)\leq 2^{\frac{\alpha+2}{2\alpha}}\sqrt[\alpha]{R(h)}\sqrt{\frac{\log\mathbb{E}[\mathbb{S}_H(x_1^{2m})]+\log\frac{4}{\delta}}{m^{\frac{2(\alpha-1)}{\alpha}}}}$$
$$\widehat{R}(h)-R_S(h)\leq 2^{\frac{\alpha+2}{2\alpha}}\sqrt[\alpha]{\widehat{R}(h)}\sqrt{\frac{\log\mathbb{E}[\mathbb{S}_H(x_1^{2m})]+\log\frac{4}{\delta}}{m^{\frac{2(\alpha-1)}{\alpha}}}}.$$

Proof. The result follows directly from Theorem 11 by setting δ to match the upper bounds and taking the limit $\tau\to 0$. □

For $\alpha = 2$, the inequalities become

$$R(h) - \widehat{R}_S(h) \leq 2\sqrt{R(h)\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}} \tag{7.13}$$

$$\widehat{R}_S(h) - R(h) \leq 2\sqrt{\widehat{R}(h)\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}}, \tag{7.14}$$

with the familiar dependency $O\left(\sqrt{\frac{\log(m/d)}{m/d}}\right)$. The advantage of these relative deviations is clear. For small values of $R(h)$ (or $\widehat{R}(h)$) these inequalities provide tighter guarantees than standard generalization bounds. Solving the corresponding second-degree inequalities in $\sqrt{R(h)}$ or $\sqrt{\widehat{R}(h)}$ leads to the following results.

Corollary 6. *Let $1 < \alpha \leq 2$ and let H be a hypothesis set of functions mapping $\mathcal{X}$ to $\{0, 1\}$. Then, for any $\delta > 0$, each of the following two inequalities holds with probability at least $1 - \delta$:*

$$R(h) \leq \widehat{R}_S(h) + 2\sqrt{\widehat{R}_S(h)\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}} + 4\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}$$

$$\widehat{R}_S(h) \leq R(h) + 2\sqrt{R(h)\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}} + 4\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}.$$

Proof. The second-degree inequality corresponding to (7.13) can be written as

$$\sqrt{R(h)}^2 - 2\sqrt{R(h)}u - \widehat{R}_S(h) \leq 0,$$

with $u = \sqrt{\frac{\log \mathbb{E}[\mathbb{S}_H(x_1^{2m})] + \log \frac{4}{\delta}}{m}}$, and implies $\sqrt{R(h)} \leq u + \sqrt{u^2 + \widehat{R}_S(h)}$. Squaring both sides

gives:

$$R(h) \leq \left[u + \sqrt{u^2 + \widehat{R}_S(h)}\right]^2 = u^2 + 2u\sqrt{u^2 + \widehat{R}_S(h)} + u^2 + \widehat{R}_S(h)$$
$$\leq u^2 + 2u\left(\sqrt{u^2} + \sqrt{\widehat{R}_S(h)}\right) + u^2 + \widehat{R}_S(h)$$
$$= 4u^2 + 2u\sqrt{\widehat{R}_S(h)} + \widehat{R}_S(h).$$

The second inequality can be proven in the same way from (7.14). □

The learning bounds of the corollary make clear the presence of two terms: a term in $O(1/m)$ and a term in $O(1/\sqrt{m})$ which admits as a factor $\widehat{R}_S(h)$ or $R(h)$ and which for small values of these terms can be more favorable than standard bounds. Theorem 11 can also be used to prove the following relative deviation bounds.

The following theorem and its proof assuming the result of Theorem 11 were given by Anthony and Bartlett (1999).

Theorem 12. *For all $0 < \epsilon < 1$, $\nu > 0$, the following inequalities hold:*

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H} \frac{R(h) - \widehat{R}_S(h)}{R(h) + \widehat{R}(h) + \nu} > \epsilon\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})] \exp\left(\frac{-m\nu\epsilon^2}{2(1-\epsilon^2)}\right)$$
$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H} \frac{\widehat{R}_S(h) - R(h)}{R(h) + \widehat{R}(h) + \nu} > \epsilon\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})] \exp\left(\frac{-m\nu\epsilon^2}{2(1-\epsilon^2)}\right).$$

Proof. We prove the first statement, the proof of the second statement is identical modulo the permutation of the roles of $R(h)$ and $\widehat{R}_S(h)$. To do so, it suffices to determine $\epsilon' > 0$ such that

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H} \frac{R(h) - \widehat{R}_S(h)}{R(h) + \widehat{R}(h) + \nu} > \epsilon\right] \leq \mathbb{P}_{S\sim D^m}\left[\sup_{h\in H} \frac{R(h) - \widehat{R}_S(h)}{\sqrt[\alpha]{R(h) + \tau}} > \epsilon'\right],$$

since we can then apply theorem 11 with $\alpha = 2$ to bound the right-hand side and take the limit as $\tau \to 0$ to eliminate the τ-dependence. To find such a choice of ϵ', we begin by

observing that for any $h \in H$,

$$\frac{R(h) - \widehat{R}_S(h)}{R(h) + \widehat{R}(h) + \nu} \leq \epsilon \Leftrightarrow R(h) \leq \frac{1+\epsilon}{1-\epsilon}\widehat{R}_S(h) + \frac{\epsilon}{1-\epsilon}\nu. \tag{7.15}$$

Assume now that $\frac{R(h)-\widehat{R}_S(h)}{\sqrt{R(h)+\tau}} \leq \epsilon'$ for some $\epsilon' > 0$, which is equivalent to $R(h) \leq \widehat{R}_S(h) + \epsilon'\sqrt{R(h)+\tau}$. We will prove that this implies (7.15). To show that, we distinguish two cases, $R(h) + \tau \leq \mu^2\epsilon'^2$ and $R(h) + \tau > \mu^2\epsilon'^2$, with $\mu > 1$. The first case implies the following:

$$R(h) + \tau \leq \mu^2\epsilon'^2 \Rightarrow R(h) \leq \widehat{R}_S(h) + \epsilon'\sqrt{\mu^2\epsilon'^2} \Leftrightarrow R(h) \leq \widehat{R}_S(h) + \mu\epsilon'^2.$$

The second case $R(h) + \tau > \mu^2\epsilon'^2$ is equivalent to $\epsilon' < \frac{\sqrt{R(h)+\tau}}{\mu}$ and implies

$$\epsilon' < \frac{\sqrt{R(h)+\tau}}{\mu} \Rightarrow R(h) \leq \widehat{R}_S(h) + \frac{R(h)+\tau}{\mu} \Leftrightarrow R(h) \leq \frac{\mu}{\mu-1}\widehat{R}_S(h) + \frac{\tau}{\mu-1}.$$

Observe now that since $\frac{\mu}{\mu-1} > 1$, both cases imply

$$R(h) \leq \frac{\mu}{\mu-1}\widehat{R}_S(h) + \frac{\tau}{\mu-1} + \mu\epsilon'^2. \tag{7.16}$$

We now choose ϵ' and μ to make (7.16) match (7.15) by setting $\frac{\mu}{\mu-1} = \frac{1+\epsilon}{1-\epsilon}$ and $\frac{\tau}{\mu-1} + \mu\epsilon'^2 = \frac{\epsilon}{1-\epsilon}\nu$, which gives:

$$\mu = \frac{1+\epsilon}{2\epsilon} \qquad \epsilon'^2 = \frac{2\epsilon^2(\nu - 2\tau)}{1-\epsilon^2}.$$

With these choices, the following inequality holds for all $h \in H$:

$$\frac{R(h) - \widehat{R}_S(h)}{\sqrt{R(h)+\tau}} \leq \epsilon' \Rightarrow \frac{R(h) - \widehat{R}_S(h)}{R(h) + \widehat{R}(h) + \nu} \leq \epsilon,$$

which concludes the proof. □

The following corollary was given by Anthony and Bartlett (1999).

Corollary 7. *For all $\epsilon > 0$, $v > 0$, the following inequality holds:*

$$\mathbb{P}_{S\sim D^m}\left[\sup_{h\in H} R(h) - (1+v)\widehat{R}_S(h) > \epsilon\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left(\frac{-mv\epsilon}{4(1+v)}\right).$$

Proof. Observe that

$$\frac{R(h) - \widehat{R}_S(h)}{R(h) + \widehat{R}(h) + \nu} > \epsilon \Leftrightarrow R(h) - \widehat{R}_S(h) > (R(h) + \widehat{R}(h) + \nu)\epsilon \Leftrightarrow R(h) > \frac{1+\epsilon}{1-\epsilon}\widehat{R}(h) + \frac{\epsilon\nu}{1-\epsilon}.$$

To derive the statement of the corollary from that of Theorem 12, we identify $\frac{1+\epsilon}{1-\epsilon}$ with $1+v$, which gives $\epsilon(2+v) = v$, that is we choose $\epsilon = \frac{v}{2+v}$, and similarly identify $\frac{\epsilon\nu}{1-\epsilon}$ with ϵ', that is $\epsilon' = \frac{\frac{v}{2+v}}{\frac{2}{2+v}}\nu = \frac{v}{2}\nu$, thus we choose $\nu = \frac{2}{v}\epsilon'$. With these choices of ϵ' and ν, the coefficient in the exponential appearing in the bounds of Theorem 12 can be rewritten as follows: $\frac{v\epsilon^2}{2(1-\epsilon^2)} = \frac{2\epsilon'}{2v}\frac{v^2}{\frac{4v+4}{(2+v)^2}} = \frac{\epsilon' v}{v}\frac{v^2}{4(v+1)} = \frac{\epsilon' v}{4(v+1)}$, which concludes the proof. □

The result of Corollary 7 is remarkable since it shows that a fast convergence rate of $O(1/m)$ can be achieved provided that we settle for a slightly larger value than the empirical error, one differing by a fixed factor $(1+v)$. The following is an immediate corollary when $\widehat{R}_S(h) = 0$, where we take $v \to \infty$.

Corollary 8. *For all $\epsilon > 0$, $v > 0$, the following inequality holds:*

$$\mathbb{P}_{S\sim D^m}\left[\exists h \in H\colon R(h) > \epsilon \wedge \widehat{R}_S(h) = 0\right] \leq 4\,\mathbb{E}[\mathbb{S}_H(x_1^{2m})]\exp\left(\frac{-m\epsilon}{4}\right).$$

This is the familiar fast rate convergence result for separable cases.

7.4 Generalization bounds for unbounded losses

In this section we will make use of the relative deviation bounds given in the previous section to prove generalization bounds for unbounded loss functions under the assumption that the moment of order α of the loss is bounded. We will start with the case $1 < \alpha \leq 2$ and then move on to considering the case when $\alpha > 2$. As already indicated earlier, the one-sided version of the results presented in this section were given by Vapnik (1998) with slightly different constants, but the proofs do not seem to be correct or complete. The second statements in all these results (other side of the inequality) are new. Our proofs for both sets of results are new.

7.4.1 Bounded moment with $1 < \alpha \leq 2$

Our first theorem reduces the problem of deriving a relative deviation bound for an unbounded loss function with $\mathcal{L}_\alpha(h) = \mathbb{E}_{z\sim D}[L(h,z)^\alpha] < +\infty$ for all $h \in H$, to that of relative deviation bound for binary classification. To simplify the presentation of the results, in what follows we will use the shorthand $\mathbb{P}[L(h,z) > t]$ instead of $\mathbb{P}_{z\sim D}[L(h,z) > t]$, and similarly $\widehat{\mathbb{P}}[L(h,z) > t]$ instead of $\mathbb{P}_{z\sim \widehat{D}}[L(h,z) > t]$.

Theorem 13. *Let $1 < \alpha \leq 2$, $0 < \epsilon \leq 1$, and $0 < \tau^{\frac{\alpha-1}{\alpha}} < \epsilon^{\frac{\alpha}{\alpha-1}}$. For any loss function L (not necessarily bounded) and hypothesis set H such that $\mathcal{L}_\alpha(h) < +\infty$ for all $h \in H$, the following two inequalities hold:*

$$\mathbb{P}\left[\sup_{h\in H} \frac{\mathcal{L}(h) - \widehat{\mathcal{L}}_S(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} > \Gamma(\alpha,\epsilon)\,\epsilon\right] \leq \mathbb{P}\left[\sup_{h\in H, t\in\mathbb{R}} \frac{\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t]}{\sqrt[\alpha]{\mathbb{P}[L(h,z) > t] + \tau}} > \epsilon\right]$$
$$\mathbb{P}\left[\sup_{h\in H} \frac{\mathcal{L}(h) - \widehat{\mathcal{L}}_S(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} > \Gamma(\alpha,\epsilon)\,\epsilon\right] \leq \mathbb{P}\left[\sup_{h\in H, t\in\mathbb{R}} \frac{\widehat{\mathbb{P}}[L(h,z) > t] - \mathbb{P}[L(h,z) > t]}{\sqrt[\alpha]{\widehat{\mathbb{P}}[L(h,z) > t] + \tau}} > \epsilon\right],$$

with $\Gamma(\alpha,\epsilon) = \frac{\alpha-1}{\alpha}(1+\tau)^{\frac{1}{\alpha}} + \frac{1}{\alpha}\left(\frac{\alpha}{\alpha-1}\right)^{\alpha-1}(1 + \left(\frac{\alpha-1}{\alpha}\right)^\alpha \tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}}\left[1 + \frac{\log(1/\epsilon)}{\left(\frac{\alpha}{\alpha-1}\right)^{\alpha-1}}\right]^{\frac{\alpha-1}{\alpha}}$.

Proof. We prove the first statement. The second statement can be shown in a very similar way. Fix $1 < \alpha \leq 2$ and $\epsilon > 0$ and assume that for any $h \in H$ and $t \geq 0$, the following holds:

$$\frac{\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t]}{\sqrt[\alpha]{\mathbb{P}[L(h,z) > t] + \tau}} \leq \epsilon. \tag{7.17}$$

We show that this implies that for any $h \in H$, $\frac{\mathcal{L}(h) - \widehat{\mathcal{L}}_S(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} \leq \Gamma(\alpha, \epsilon)\epsilon$. By the properties of the Lebesgue integral, we can write

$$\begin{aligned}
\mathcal{L}(h) &= \mathop{\mathbb{E}}_{z \sim D}[L(h,z)] = \int_0^{+\infty} \mathbb{P}[L(h,z) > t]\, dt \\
\widehat{\mathcal{L}}(h) &= \mathop{\mathbb{E}}_{z \sim \widehat{D}}[L(h,z)] = \int_0^{+\infty} \widehat{\mathbb{P}}[L(h,z) > t]\, dt,
\end{aligned}$$

and, similarly,

$$\mathcal{L}_\alpha(h) = \mathcal{L}_\alpha(h) = \int_0^{+\infty} \mathbb{P}[L^\alpha(h,z) > t]\, dt = \int_0^{+\infty} \alpha t^{\alpha-1} \mathbb{P}[L(h,z) > t]\, dt.$$

In what follows, we use the notation $I_\alpha = \mathcal{L}_\alpha(h) + \tau$. Let $t_0 = s I_\alpha^{\frac{1}{\alpha}}$ and $t_1 = t_0 \left[\frac{1}{\epsilon}\right]^{\frac{1}{\alpha-1}}$ for $s > 0$. To bound $\mathcal{L}(h) - \widehat{\mathcal{L}}(h)$, we simply bound $\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t]$ by $\mathbb{P}[L(h,z) > t]$ for large values of t, that is $t > t_1$, and use inequality (7.17) for smaller values of t:

$$\begin{aligned}
\mathcal{L}(h) - \widehat{\mathcal{L}}(h) &= \int_0^{+\infty} \mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t]\, dt \\
&\leq \int_0^{t_1} \epsilon \sqrt[\alpha]{\mathbb{P}[L(h,z) > t] + \tau}\, dt + \int_{t_1}^{+\infty} \mathbb{P}[L(h,z) > t]\, dt.
\end{aligned}$$

For relatively small values of t, $\mathbb{P}[L(h,z) > t]$ is close to one. Thus, we can write

$$\begin{aligned}\mathcal{L}(h) - \widehat{\mathcal{L}}(h) &\leq \int_0^{t_0} \epsilon \sqrt[\alpha]{1+\tau}\, dt + \int_{t_0}^{t_1} \epsilon \sqrt[\alpha]{\mathbb{P}[L(h,z) > t] + \tau}\, dt + \int_{t_1}^{+\infty} \mathbb{P}[L(h,z) > t] dt \\ &= \int_0^{+\infty} f(t) g(t)\, dt,\end{aligned}$$

with

$$f(t) = \begin{cases} \gamma_1 I_\alpha^{\frac{\alpha-1}{\alpha^2}} \epsilon \sqrt[\alpha]{1+\tau} & \text{if } 0 \leq t \leq t_0 \\ \gamma_2 \left[\alpha t^{\alpha-1} (\mathbb{P}[L(h,z) > t] + \tau)\right]^{\frac{1}{\alpha}} \epsilon & \text{if } t_0 < t \leq t_1 \\ \gamma_2 \left[\alpha t^{\alpha-1} \mathbb{P}[L(h,z) > t]\right]^{\frac{1}{\alpha}} \epsilon & \text{if } t_1 < t. \end{cases} \quad g(t) = \begin{cases} \frac{1}{\gamma_1 I_\alpha^{\frac{\alpha-1}{\alpha^2}}} & \text{if } 0 \leq t \leq t_0 \\ \frac{1}{\gamma_2 (\alpha t^{\alpha-1})^{\frac{1}{\alpha}}} & \text{if } t_0 < t \leq t_1 \\ \frac{\mathbb{P}[L(h,z)>t]^{\frac{\alpha-1}{\alpha}}}{\gamma_2 (\alpha t^{\alpha-1})^{\frac{1}{\alpha}}} \frac{1}{\epsilon} & \text{if } t_1 < t, \end{cases}$$

where γ_1, γ_2 are positive parameters that we shall select later. Now, since $\alpha > 1$, by Hölder's inequality,

$$\mathcal{L}(h) - \widehat{\mathcal{L}}(h) \leq \left[\int_0^{+\infty} f(t)^\alpha\, dt\right]^{\frac{1}{\alpha}} \left[\int_0^{+\infty} g(t)^{\frac{\alpha}{\alpha-1}}\, dt\right]^{\frac{\alpha-1}{\alpha}}.$$

The first integral on the right-hand side can be bounded as follows:

$$\begin{aligned}\int_0^{+\infty} f(t)^\alpha\, dt &= \int_0^{t_0} (1+\tau)(\gamma_1 I_\alpha^{\frac{\alpha-1}{\alpha^2}} \epsilon)^\alpha\, dt + \gamma_2^\alpha \epsilon^\alpha \tau \int_{t_0}^{t_1} \alpha t^{\alpha-1} dt + \gamma_2^\alpha \int_{t_0}^{+\infty} \alpha t^{\alpha-1} \mathbb{P}[L(h,z) > t] \epsilon^\alpha\, dt \\ &\leq (1+\tau) \gamma_1^\alpha I_\alpha^{\frac{\alpha-1}{\alpha}} t_0 \epsilon^\alpha + \gamma_2^\alpha \epsilon^\alpha \tau (t_1^\alpha - t_0^\alpha) + \gamma_2^\alpha \epsilon^\alpha I_\alpha \\ &\leq (\gamma_1^\alpha (1+\tau) s + \gamma_2^\alpha (1 + s^\alpha (1/\epsilon)^{\frac{\alpha}{\alpha-1}} \tau)) \epsilon^\alpha I_\alpha \\ &\leq (\gamma_1^\alpha (1+\tau) s + \gamma_2^\alpha (1 + s^\alpha \tau^{\frac{1}{\alpha}})) \epsilon^\alpha I_\alpha.\end{aligned}$$

Since $t_1/t_0 = (1/\epsilon)^{\frac{1}{\alpha-1}}$, the second one can be computed and bounded following

$$\begin{aligned}
\int_0^{+\infty} g(t)^{\frac{\alpha}{\alpha-1}}\,dt &= \int_0^{t_0} \frac{dt}{\gamma_1^{\frac{\alpha}{\alpha-1}} I_\alpha^{\frac{1}{\alpha}}} + \int_{t_0}^{t_1} \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}} \alpha^{\frac{1}{\alpha-1}}} \frac{dt}{t} + \int_{t_1}^{+\infty} \frac{\mathbb{P}[L(h,z)>t]}{\gamma_2^{\frac{\alpha}{\alpha-1}} \alpha^{\frac{1}{\alpha-1}} \epsilon^{\frac{\alpha}{\alpha-1}} t} dt \\
&= \frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \int_{t_1}^{+\infty} \frac{\alpha t^{\alpha-1}\mathbb{P}[L(h,z)>t]}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha\epsilon)^{\frac{\alpha}{\alpha-1}} t^\alpha} dt \\
&\leq \frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \int_{t_1}^{+\infty} \frac{\alpha t^{\alpha-1}\mathbb{P}[L(h,z)>t]}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha\epsilon)^{\frac{\alpha}{\alpha-1}} t_1^\alpha} dt \\
&\leq \frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{I_\alpha}{\gamma_2^{\frac{\alpha}{\alpha-1}} (\alpha\epsilon)^{\frac{\alpha}{\alpha-1}} s^\alpha I_\alpha(\frac{1}{\epsilon})^{\frac{\alpha}{\alpha-1}}} \\
&= \frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}}} \left(\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{1}{\alpha^{\frac{\alpha}{\alpha-1}} s^\alpha} \right).
\end{aligned}$$

Combining the bounds obtained for these integrals yields directly

$$\begin{aligned}
&\mathcal{L}(h) - \widehat{\mathcal{L}}(h) \\
&\leq \left[(\gamma_1^\alpha (1+\tau) s + \gamma_2^\alpha (1 + s^\alpha \tau^{\frac{1}{\alpha}})) \epsilon^\alpha I_\alpha \right]^{\frac{1}{\alpha}} \left[\frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}}} \left(\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{1}{\alpha^{\frac{\alpha}{\alpha-1}} s^\alpha} \right) \right]^{\frac{\alpha-1}{\alpha}} \\
&= (\gamma_1^\alpha (1+\tau) s + \gamma_2^\alpha (1 + s^\alpha \tau^{\frac{1}{\alpha}}))^{\frac{1}{\alpha}} \left[\frac{s}{\gamma_1^{\frac{\alpha}{\alpha-1}}} + \frac{1}{\gamma_2^{\frac{\alpha}{\alpha-1}}} \left(\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{1}{\alpha^{\frac{\alpha}{\alpha-1}} s^\alpha} \right) \right]^{\frac{\alpha-1}{\alpha}} \epsilon I_\alpha^{\frac{1}{\alpha}}.
\end{aligned}$$

Observe that the expression on the right-hand side can be rewritten as $\|\mathbf{u}\|_\alpha \|\mathbf{v}\|_{\frac{\alpha}{\alpha-1}}\, \epsilon I_\alpha^{\frac{1}{\alpha}}$ where the vectors $\mathbf{u}$ and $\mathbf{v}$ are defined by $\mathbf{u} = (\gamma_1(1+\tau)^{\frac{1}{\alpha}} s^{\frac{1}{\alpha}}, \gamma_2(1+s^\alpha\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}})$ and $\mathbf{v} = (v_1, v_2) = \left(\frac{s^{\frac{\alpha-1}{\alpha}}}{\gamma_1}, \frac{1}{\gamma_2} \left[\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{1}{\alpha^{\frac{\alpha}{\alpha-1}} s^\alpha} \right]^{\frac{\alpha-1}{\alpha}} \right)$. The inner product $\mathbf{u}\cdot\mathbf{v}$ does not depend on γ_1 and γ_2 and by the properties of Hölder's inequality can be reached when $\mathbf{u}$ and the vector $\mathbf{v}' = (v_1^{\frac{1}{\alpha-1}}, v_2^{\frac{1}{\alpha-1}})$ are collinear. γ_1 and γ_2 can be chosen so that $\det(\mathbf{u},\mathbf{v}') = 0$, since this condition can be rewritten as

$$s^{\frac{1}{\alpha}}(1+\tau)^{\frac{1}{\alpha}} \frac{\gamma_1}{\gamma_2^{\frac{1}{\alpha-1}}} \left[\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}} \log\frac{1}{\epsilon} + \frac{1}{\alpha^{\frac{\alpha}{\alpha-1}} s^\alpha} \right]^{\frac{1}{\alpha}} - s^{\frac{1}{\alpha}}(1+s^\alpha\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}} \frac{\gamma_2}{\gamma_1^{\frac{1}{\alpha-1}}} = 0, \qquad (7.18)$$

or equivalently,

$$\left(\frac{\gamma_1}{\gamma_2}\right)^{\frac{\alpha}{\alpha-1}} \Big[\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}}\log\frac{1}{\epsilon}+\frac{1}{\alpha^{\frac{\alpha}{\alpha-1}}s^\alpha}\Big]^{\frac{1}{\alpha}} - (1+s^\alpha\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}} = 0. \tag{7.19}$$

Thus, for such values of γ_1 and γ_2, the following inequality holds:

$$\mathcal{L}(h) - \widehat{\mathcal{L}}(h) \leq (\mathbf{u}\cdot\mathbf{v})\,\epsilon I_\alpha^{\frac{1}{\alpha}} = f(s)\,\epsilon I_\alpha^{\frac{1}{\alpha}},$$

with

$$\begin{aligned} f(s) &= (1+\tau)^{\frac{1}{\alpha}}s + (1+s^\alpha\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}}\Big[\frac{1}{(\alpha-1)\alpha^{\frac{1}{\alpha-1}}}\log\frac{1}{\epsilon}+\frac{1}{\alpha^{\frac{\alpha}{\alpha-1}}s^\alpha}\Big]^{\frac{\alpha-1}{\alpha}} \\ &= (1+\tau)^{\frac{1}{\alpha}}s + \frac{(1+s^\alpha\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}}}{\alpha}\Big[\frac{\alpha}{(\alpha-1)}\log\frac{1}{\epsilon}+\frac{1}{s^\alpha}\Big]^{\frac{\alpha-1}{\alpha}}. \end{aligned}$$

Setting $s = \frac{\alpha-1}{\alpha}$ yields the statement of the theorem. □

The next corollary follows immediately by upper bounding the right-hand side of the learning bounds of theorem 13 using theorem 11. It provides learning bounds for unbounded loss functions in terms of the growth functions in the case $1 < \alpha \leq 2$.

Corollary 9. *Let $\epsilon < 1$, $1 < \alpha \leq 2$, and $0 < \tau^{\frac{\alpha-1}{\alpha}} < \epsilon^{\frac{\alpha}{\alpha-1}}$. For any loss function L (not necessarily bounded) and hypothesis set H such that $\mathcal{L}_\alpha(h) < +\infty$ for all $h \in H$, the following inequalities hold:*

$$\begin{aligned} \mathbb{P}\Bigg[\sup_{h\in H}\frac{\mathcal{L}(h)-\widehat{\mathcal{L}}(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h)+\tau}} > \Gamma(\alpha,\epsilon)\epsilon\Bigg] &\leq 4\,\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right) \\ \mathbb{P}\Bigg[\sup_{h\in H}\frac{\widehat{\mathcal{L}}(h)-\mathcal{L}(h)}{\sqrt[\alpha]{\widehat{\mathcal{L}}_\alpha(h)+\tau}} > \Gamma(\alpha,\epsilon)\epsilon\Bigg] &\leq 4\,\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]\exp\left(\frac{-m^{\frac{2(\alpha-1)}{\alpha}}\epsilon^2}{2^{\frac{\alpha+2}{\alpha}}}\right), \end{aligned}$$

where Q is the set of functions $Q = \{z \mapsto 1_{L(h,z)>t} \mid h \in H, t \in \mathbb{R}\}$, and $\Gamma(\alpha,\epsilon) =$

$\frac{\alpha-1}{\alpha}(1+\tau)^{\frac{1}{\alpha}} + \frac{1}{\alpha}\left(\frac{\alpha}{\alpha-1}\right)^{\alpha-1}(1+\left(\frac{\alpha-1}{\alpha}\right)^{\alpha}\tau^{\frac{1}{\alpha}})^{\frac{1}{\alpha}}\left[1+\frac{\log(1/\epsilon)}{\left(\frac{\alpha}{\alpha-1}\right)^{\alpha-1}}\right]^{\frac{\alpha-1}{\alpha}}$.

The following corollary gives the explicit result for $\alpha = 2$.

Corollary 10. *Let $\epsilon < 1$ and $0 < \tau < \epsilon^4$. For any loss function L (not necessarily bounded) and hypothesis set H such that $\mathcal{L}_2(h) < +\infty$ for all $h \in H$, the following inequalities hold:*

$$\mathbb{P}\left[\sup_{h\in H}\frac{\mathcal{L}(h)-\widehat{\mathcal{L}}(h)}{\sqrt{\mathcal{L}_2(h)+\tau}} > \Gamma(2,\epsilon)\epsilon\right] \leq 4\ \mathbb{E}[\mathbb{S}_Q(z_1^{2m})]\exp\left(\frac{-m\epsilon^2}{4}\right)$$
$$\mathbb{P}\left[\sup_{h\in H}\frac{\widehat{\mathcal{L}}(h)-\mathcal{L}(h)}{\sqrt{\widehat{\mathcal{L}}_2(h)+\tau}} > \Gamma(2,\epsilon)\epsilon\right] \leq 4\ \mathbb{E}[\mathbb{S}_Q(z_1^{2m})]\exp\left(\frac{-m\epsilon^2}{4}\right),$$

with $\Gamma(2,\epsilon) = \left(\frac{\sqrt{1+\tau}}{2} + \sqrt{1+\frac{1}{4}\sqrt{\tau}}\sqrt{1+\frac{1}{2}\log\frac{1}{\epsilon}}\right)$ and Q the set of functions $Q = \{z \mapsto 1_{L(h,z)>t} \mid h \in H, t \in \mathbb{R}\}$.

Corollary 11. *Let L be a loss function (not necessarily bounded) and H a hypothesis set such that $\mathcal{L}_2(h) < +\infty$ for all $h \in H$. Then, for any $\delta > 0$, with probability at least $1-\delta$, each of the following inequalities holds for all $h \in H$:*

$$\mathcal{L}(h) \leq \widehat{\mathcal{L}}_S(h) + 2\sqrt{\mathcal{L}_2(h)}\sqrt{\frac{2\log\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]+\log\frac{1}{\delta}}{m}}\Gamma_0\left(2, 2\sqrt{\frac{2\log\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]+\log\frac{1}{\delta}}{m}}\right)$$
$$\widehat{\mathcal{L}}_S(h) \leq \mathcal{L}(h) + 2\sqrt{\widehat{\mathcal{L}}_2(h)}\sqrt{\frac{2\log\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]+\log\frac{1}{\delta}}{m}}\Gamma_0\left(2, 2\sqrt{\frac{2\log\mathbb{E}[\mathbb{S}_Q(z_1^{2m})]+\log\frac{1}{\delta}}{m}}\right),$$

where Q is the set of functions $Q = \{z \mapsto 1_{L(h,z)>t} \mid h \in H, t \in \mathbb{R}\}$ and $\Gamma_0(2,\epsilon) = \frac{1}{2} + \sqrt{1+\frac{1}{2}\log\frac{1}{\epsilon}}$.

Proof. For any $\epsilon > 0$, let $f(\epsilon) = \Gamma_0(2,\epsilon)\epsilon$. Then, by Corollary 10,

$$\mathbb{P}\left[\sup_{h\in H}\frac{\mathcal{L}(h)-\widehat{\mathcal{L}}(h)}{\sqrt{\mathcal{L}_2(h)+\tau}} > \epsilon\right] \leq 4\ \mathbb{E}[\mathbb{S}_Q(z_1^{2m})]\exp\left(\frac{-m[f^{-1}(\epsilon)]^2}{4}\right).$$

Setting the right-hand side to ϵ and using inversion yields immediately the first inequality. The second inequality is proven in the same way. □

Observe that, modulo the factors in Γ_0, the bounds of the corollary admit the standard $(1/\sqrt{m})$ dependency and that the factors in Γ_0 are only logarithmic in m.

7.4.2 Bounded moment with $\alpha > 2$

This section gives two-sided generalization bounds for unbounded losses with finite moments of order α, with $\alpha > 2$. As for the case $1 < \alpha < 2$, the one-sided version of our bounds coincides with that of Vapnik (1998, 2006b) modulo a constant factor, but, here again, the proofs given by Vapnik in both books seem to be incorrect.

Proposition 1. *Let $\alpha > 2$. For any loss function L (not necessarily bounded) and hypothesis set H such that $0 < \mathcal{L}_\alpha(h) < +\infty$ for all $h \in H$, the following two inequalities hold:*

$$\int_0^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]}dt \leq \Psi(\alpha) \sqrt[\alpha]{\mathcal{L}_\alpha(h)} \quad \textit{and} \quad \int_0^{+\infty} \sqrt{\widehat{\mathbb{P}}[L(h,z) > t]}dt \leq \Psi(\alpha) \sqrt[\alpha]{\widehat{\mathcal{L}}_\alpha(h)},$$

where $\Psi(\alpha) = \left(\frac{1}{2}\right)^{\frac{2}{\alpha}} \left(\frac{\alpha}{\alpha-2}\right)^{\frac{\alpha-1}{\alpha}}$.

Proof. We prove the first inequality. The second can be proven in a very similar way. Fix $\alpha > 2$ and $h \in H$. As in the proof of Theorem 13, we bound $\mathbb{P}[L(h,z) > t]$ by 1 for t close to 0, say $t \leq t_0$ for some $t_0 > 0$ that we shall later determine. We can write

$$\int_0^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]}dt \leq \int_0^{t_0} 1dt + \int_{t_0}^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]}dt = \int_0^{+\infty} f(t)g(t)dt,$$

with functions f and g defined as follows:

$$f(t) = \begin{cases} \gamma I_\alpha^{\frac{\alpha-1}{2\alpha}} & \text{if } 0 \leq t \leq t_0 \\ \alpha^{\frac{1}{2}} t^{\frac{\alpha-1}{2}} \mathbb{P}[L(h,z) > t]^{\frac{1}{2}} & \text{if } t_0 < t. \end{cases} \quad g(t) = \begin{cases} \frac{1}{\gamma I_\alpha^{\frac{\alpha-1}{2\alpha}}} & \text{if } 0 \leq t \leq t_0 \\ \frac{1}{\alpha^{\frac{1}{2}} t^{\frac{\alpha-1}{2}}} & \text{if } t_0 < t, \end{cases}$$

where $I_\alpha = \mathcal{L}_\alpha(h)$ and where γ is a positive parameter that we shall select later. By the Cauchy-Schwarz inequality,

$$\int_0^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]} dt \leq \left(\int_0^{+\infty} f(t)^2 dt \right)^{\frac{1}{2}} \left(\int_0^{+\infty} g(t)^2 dt \right)^{\frac{1}{2}}.$$

Thus, we can write

$$\begin{aligned} &\int_0^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]} dt \\ &\qquad \leq \left(\gamma^2 I_\alpha^{\frac{\alpha-1}{\alpha}} t_0 + \int_{t_0}^{+\infty} \alpha t^{\alpha-1} \mathbb{P}[L(h,z) > t] dt \right)^{\frac{1}{2}} \left(\frac{t_0}{\gamma^2 I_\alpha^{\frac{\alpha-1}{\alpha}}} + \int_{t_0}^{+\infty} \frac{1}{\alpha t^{\alpha-1}} dt \right)^{\frac{1}{2}} \\ &\qquad \leq \left(\gamma^2 I_\alpha^{\frac{\alpha-1}{\alpha}} t_0 + I_\alpha \right)^{\frac{1}{2}} \left(\frac{t_0}{\gamma^2 I_\alpha^{\frac{\alpha-1}{\alpha}}} + \frac{1}{\alpha(\alpha-2) t_0^{\alpha-2}} \right)^{\frac{1}{2}}. \end{aligned}$$

Introducing t_1 with $t_0 = I_\alpha^{1/\alpha} t_1$ leads to

$$\begin{aligned} \int_0^{+\infty} \sqrt{\mathbb{P}[L(h,z) > t]} dt &\leq \left(\gamma^2 I_\alpha t_1 + I_\alpha \right)^{\frac{1}{2}} \left(\frac{t_1}{\gamma^2 I_\alpha^{\frac{\alpha-2}{\alpha}}} + \frac{1}{\alpha(\alpha-2) t_1^{\alpha-2} I_\alpha^{\frac{\alpha-2}{\alpha}}} \right)^{\frac{1}{2}} \\ &\leq \left(\gamma^2 t_1 + 1 \right)^{\frac{1}{2}} \left(\frac{t_1}{\gamma^2} + \frac{1}{\alpha(\alpha-2) t_1^{\alpha-2}} \right)^{\frac{1}{2}} I_\alpha^{\frac{1}{\alpha}}. \end{aligned}$$

We now seek to minimize the expression $(\gamma^2 t_1 + 1)^{\frac{1}{2}} \left(\frac{t_1}{\gamma^2} + \frac{1}{\alpha(\alpha-2) t_1^{\alpha-2}} \right)^{\frac{1}{2}}$, first as a function of γ. This expression can be viewed as the product of the norms of the vectors $\mathbf{u} = (\gamma t_1^{\frac{1}{2}}, 1)$ and $\mathbf{v} = (\frac{t_1^{\frac{1}{2}}}{\gamma}, \frac{1}{\sqrt{\alpha(\alpha-2)} t_1^{\frac{\alpha-2}{2}}})$, with a constant inner product (not depending on γ). Thus, by the

properties of the Cauchy-Schwarz inequality, it is minimized for collinear vectors and in that case equals their inner product:

$$\mathbf{u} \cdot \mathbf{v} = t_1 + \frac{1}{\sqrt{\alpha(\alpha-2)} t_1^{\frac{\alpha-2}{2}}}.$$

Differentiating this last expression with respect to t_1 and setting the result to zero gives the minimizing value of t_1: $(\frac{2}{\alpha-2}\sqrt{\alpha(\alpha-2)})^{-\frac{2}{\alpha}} = \left(\frac{1}{2}\sqrt{\frac{\alpha-2}{\alpha}}\right)^{\frac{2}{\alpha}}$. For that value of t_1,

$$\mathbf{u} \cdot \mathbf{v} = \left(1 + \frac{2}{\alpha-2}\right) t_1 = \frac{\alpha}{\alpha-2}\left(\frac{1}{2}\sqrt{\frac{\alpha-2}{\alpha}}\right)^{\frac{2}{\alpha}} = \left(\frac{1}{2}\right)^{\frac{2}{\alpha}}\left(\frac{\alpha-2}{\alpha}\right)^{\frac{1-\alpha}{\alpha}},$$

which concludes the proof. □

Theorem 14. *Let $\alpha > 2$, $0 < \epsilon \leq 1$, and $0 < \tau \leq \epsilon^2$. Then, for any loss function L (not necessarily bounded) and hypothesis set H such that $\mathcal{L}_\alpha(h) < +\infty$ and $\widehat{\mathcal{L}}_\alpha(h) < +\infty$ for all $h \in H$, the following two inequalities hold:*

$$\mathbb{P}\left[\sup_{h\in H} \frac{\mathcal{L}(h) - \widehat{\mathcal{L}}(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} > \Lambda(\alpha)\epsilon\right] \leq \mathbb{P}\left[\sup_{h\in H, t\in\mathbb{R}} \frac{\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t]}{\sqrt{\mathbb{P}[L(h,z) > t] + \tau}} > \epsilon\right]$$

$$\mathbb{P}\left[\sup_{h\in H} \frac{\widehat{\mathcal{L}}(h) - \mathcal{L}(h)}{\sqrt[\alpha]{\widehat{\mathcal{L}}_\alpha(h) + \tau}} > \Lambda(\alpha)\epsilon\right] \leq \mathbb{P}\left[\sup_{h\in H, t\in\mathbb{R}} \frac{\widehat{\mathbb{P}}[L(h,z) > t] - \mathbb{P}[L(h,z) > t]}{\sqrt{\widehat{\mathbb{P}}[L(h,z) > t] + \tau}} > \epsilon\right],$$

where $\Lambda(\alpha) = \left(\frac{1}{2}\right)^{\frac{2}{\alpha}}\left(\frac{\alpha}{\alpha-2}\right)^{\frac{\alpha-1}{\alpha}} + \frac{\alpha}{\alpha-1}\tau^{\frac{\alpha-2}{2\alpha}}$.

Proof. We prove the first statement since the second one can be proven in a very similar way. Assume that $\sup_{h,t} \frac{\mathbb{P}[L(h,z)>t] - \widehat{\mathbb{P}}[L(h,z)>t]}{\sqrt{\mathbb{P}[L(h,z)>t]+\tau}} \leq \epsilon$. Fix $h \in H$, let $J = \int_0^{+\infty} \sqrt{\mathbb{P}\left[L(h,z) > t\right]}\, dt$ and $\nu = \mathcal{L}_\alpha(h)$. By Markov's inequality, for any $t > 0$, $\mathbb{P}[L(h,z) > t] = \mathbb{P}[L^\alpha(h,z) > t^\alpha] \leq$

$\frac{\mathcal{L}_\alpha(h)}{t^\alpha} = \frac{\nu}{t^\alpha}$. Using this inequality, for any $t_0 > 0$, we can write

$$\begin{aligned}
\mathcal{L}(h) - \widehat{\mathcal{L}}(h) &= \int_0^{+\infty} (\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t])\, dt \\
&= \int_0^{t_0} (\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t])\, dt + \int_{t_0}^{+\infty} (\mathbb{P}[L(h,z) > t] - \widehat{\mathbb{P}}[L(h,z) > t])\, dt \\
&\leq \epsilon \int_0^{t_0} \sqrt{\mathbb{P}[L(h,z) > t] + \tau}\, dt + \int_{t_0}^{+\infty} \mathbb{P}[L(h,z) > t]\, dt \\
&\leq \epsilon \int_0^{t_0} (\sqrt{\mathbb{P}[L(h,z) > t]} + \sqrt{\tau})\, dt + \int_{t_0}^{+\infty} \frac{\nu}{t^\alpha}\, dt \\
&\leq \epsilon J + \epsilon \sqrt{\tau} t_0 + \frac{\nu}{(\alpha-1) t_0^{\alpha-1}}.
\end{aligned}$$

Choosing t_0 to minimize the right-hand side yields $t_0 = \left(\frac{\nu}{\epsilon\sqrt{\tau}}\right)^{\frac{1}{\alpha}}$ and gives

$$\mathcal{L}(h) - \widehat{\mathcal{L}}(h) \leq \epsilon J + \frac{\alpha}{\alpha - 1} \nu^{\frac{1}{\alpha}} (\epsilon\sqrt{\tau})^{\frac{\alpha-1}{\alpha}}.$$

Since $\tau \leq \epsilon^2$, $(\epsilon\sqrt{\tau})^{\frac{\alpha-1}{\alpha}} = \left[\epsilon \tau^{\frac{1}{2(\alpha-1)}} \tau^{\frac{\alpha-2}{2(\alpha-1)}}\right]^{\frac{\alpha-1}{\alpha}} \leq \left[\epsilon \epsilon^{\frac{1}{(\alpha-1)}} \tau^{\frac{\alpha-2}{2(\alpha-1)}}\right]^{\frac{\alpha-1}{\alpha}} = \epsilon \tau^{\frac{\alpha-2}{2\alpha}}$. Thus, by Proposition 1, the following holds:

$$\frac{\mathcal{L}(h) - \widehat{\mathcal{L}}(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} \leq \epsilon \Psi(\alpha) \frac{\nu^{\frac{1}{\alpha}}}{(\nu+\tau)^{\frac{1}{\alpha}}} + \frac{\alpha}{\alpha-1} \epsilon \tau^{\frac{\alpha-2}{2\alpha}} \frac{\nu^{\frac{1}{\alpha}}}{(\nu+\tau)^{\frac{1}{\alpha}}} \leq \epsilon \Psi(\alpha) + \frac{\alpha}{\alpha-1} \epsilon \tau^{\frac{\alpha-2}{2\alpha}},$$

which concludes the proof. □

Combining Theorem 14 with Theorem 11 leads immediately to the following two results.

Corollary 12. *Let $\alpha > 2$, $0 < \epsilon \leq 1$, and $0 < \tau \leq \epsilon^2$. Then, for any loss function L (not necessarily bounded) and hypothesis set H such that $\mathcal{L}_\alpha(h) < +\infty$ and $\widehat{\mathcal{L}}_\alpha(h) < +\infty$ for all*

$h \in H$, the following two inequalities hold:

$$\mathbb{P}\left[\sup_{h\in H} \frac{\mathcal{L}(h) - \widehat{\mathcal{L}}(h)}{\sqrt[\alpha]{\mathcal{L}_\alpha(h) + \tau}} > \Lambda(\alpha)\epsilon\right] \leq 4\ \mathbb{E}[\mathbb{S}_Q(z_1^{2m})] \exp\left(\frac{-m\epsilon^2}{4}\right)$$
$$\mathbb{P}\left[\sup_{h\in H} \frac{\widehat{\mathcal{L}}(h) - \mathcal{L}(h)}{\sqrt[\alpha]{\widehat{\mathcal{L}}_\alpha(h) + \tau}} > \Lambda(\alpha)\epsilon\right] \leq 4\ \mathbb{E}[\mathbb{S}_Q(z_1^{2m})] \exp\left(\frac{-m\epsilon^2}{4}\right),$$

where $\Lambda(\alpha) = \left(\frac{1}{2}\right)^{\frac{2}{\alpha}} \left(\frac{\alpha}{\alpha-2}\right)^{\frac{\alpha-1}{\alpha}} + \frac{\alpha}{\alpha-1}\tau^{\frac{\alpha-2}{2\alpha}}$ *and where Q is the set of functions* $Q = \{z \mapsto 1_{L(h,z)>t} \mid h \in H, t \in \mathbb{R}\}$.

In the following result, $\mathrm{Pdim}(G)$ denotes the pseudo-dimension of a family of real-valued functions G (Pollard, 1984, 1989; Vapnik, 1998), which coincides with the VC-dimension of the corresponding thresholded functions:

$$\mathrm{Pdim}(G) = \mathrm{VCdim}\Big(\big\{(x,t) \mapsto 1_{(g(x)-t)>0} \colon g \in G\big\}\Big). \tag{7.20}$$

Corollary 13. *Let $\alpha > 2$, $0 < \epsilon \leq 1$. Let L be a loss function (not necessarily bounded) and H a hypothesis set such that $\mathcal{L}_\alpha(h) < +\infty$ for all $h \in H$, and $d = \mathrm{Pdim}(\{z \mapsto L(h,z) \mid h \in H\}) < +\infty$. Then, for any $\delta > 0$, with probability at least $1-\delta$, each of the following inequalities holds for all $h \in H$:*

$$\mathcal{L}(h) \leq \widehat{\mathcal{L}}(h) + 2\Lambda(\alpha)\sqrt[\alpha]{\mathcal{L}_\alpha(h)}\sqrt{\frac{d\log\frac{2em}{d} + \log\frac{4}{\delta}}{m}}$$
$$\widehat{\mathcal{L}}(h) \leq \mathcal{L}(h) + 2\Lambda(\alpha)\sqrt[\alpha]{\widehat{\mathcal{L}}_\alpha(h)}\sqrt{\frac{d\log\frac{2em}{d} + \log\frac{4}{\delta}}{m}}$$

where $\Lambda(\alpha) = \left(\frac{1}{2}\right)^{\frac{2}{\alpha}} \left(\frac{\alpha}{\alpha-2}\right)^{\frac{\alpha-1}{\alpha}}$.

7.5 Conclusion

We presented a series of results for relative deviation bounds used to prove generalization bounds for unbounded loss functions. These learning bounds can be used in a variety of applications to deal with the more general unbounded case. The relative deviation bounds are of independent interest and can be further used for a sharper analysis of guarantees in binary classification and other tasks.

7.6 Lemmas in support of Section 7.3

Lemma 12. *Let $1 < \alpha \leq 2$ and for any $\eta > 0$, let $f\colon (0,+\infty)\times(0,+\infty) \to \mathbb{R}$ be the function defined by $f\colon (x,y) \mapsto \frac{x-y}{\sqrt[\alpha]{x+y+\eta}}$. Then, f is a strictly increasing function of x and a strictly decreasing function of y.*

Proof. f is differentiable over its domain of definition and for all $(x,y) \in (0,+\infty)\times(0,+\infty)$,

$$\frac{\partial f}{\partial x}(x,y) = \frac{(x+y+\eta)^{\frac{1}{\alpha}} - \frac{x-y}{\alpha}(x+y+\eta)^{\frac{1}{\alpha}-1}}{(x+y+\eta)^{\frac{2}{\alpha}}} = \frac{\frac{\alpha-1}{\alpha}x + \frac{\alpha+1}{\alpha}y + \eta}{(x+y+\eta)^{1+\frac{1}{\alpha}}} > 0$$
$$\frac{\partial f}{\partial y}(x,y) = \frac{-(x+y+\eta)^{\frac{1}{\alpha}} - \frac{x-y}{\alpha}(x+y+\eta)^{\frac{1}{\alpha}-1}}{(x+y+\eta)^{\frac{2}{\alpha}}} = -\frac{\frac{\alpha+1}{\alpha}x + \frac{\alpha-1}{\alpha}y + \eta}{(x+y+\eta)^{1+\frac{1}{\alpha}}} < 0.$$

□

Chapter 8

Tight lower bound on the probability of a binomial exceeding its expectation

As discussed in Chapter 7, proofs, which apply relative deviation bounds to demonstrate the generalization ability of learning algorithms with unbounded loss functions, make use of a certain fact about binomial distributions, about the probability of a binomial random variable exceeding its expected value. Though it has been stated many times in the machine learning literature, oddly we were unable to find a sufficiently general proof of this fact in existing papers. Therefore in this chapter we provide such a proof. It is perhaps surprisingly complex given the simple nature of its conclusion and the basic nature of its subject. The discrete and (for some parameter values) highly asymmetric nature of the binomial distribution creates considerable complications.

8.1 Motivation

This chapter presents a tight lower bound on the probability that a binomial random variable exceeds its expected value. If the binomial distribution were symmetric around its mean, such a bound would be trivially equal to 1/2. And indeed, when the number of trials m for a binomial distribution is large, and the probability p of success on each trial is not too close to 0 or to 1, the binomial distribution is approximately symmetric. With p is fixed, and m sufficiently large, the de Moivre-Laplace theorem tells us that we can approximate the binomial distribution with a normal distribution. But, when p is close to 0 or 1, or the number of trials m is small, substantial asymmetry around the mean can arise, as illustrated by Figure 8.1, which shows the binomial distribution for different values of m and p.

The lower bound we prove has been invoked several times in the machine learning literature, starting with work on relative deviation bounds by Vapnik (1998), where it is stated without proof. Relative deviation bounds are useful bounds in learning theory that provide more insight than the standard generalization bounds because the approximation error is scaled by the square root of the true error. In particular, they lead to sharper bounds for empirical risk minimization, and play a critical role in the analysis of the generalization bounds for unbounded loss functions (Cortes et al., 2010b).

This binomial inequality is mentioned and used again without proof or reference in Anthony and Shawe-Taylor (1993), where the authors improve the original work of Vapnik (1998) on relative deviation bounds by a constant factor. The same claim later appears in Vapnik (2006a) and implicitly in other publications referring to the relative deviations bounds of Vapnik (1998).

To the best of our knowledge, there is no publication giving an actual proof of this inequality in the machine learning literature. Our search efforts for a proof in the statistics literature were also unsuccessful. Instead, some references suggest in fact that such a proof is

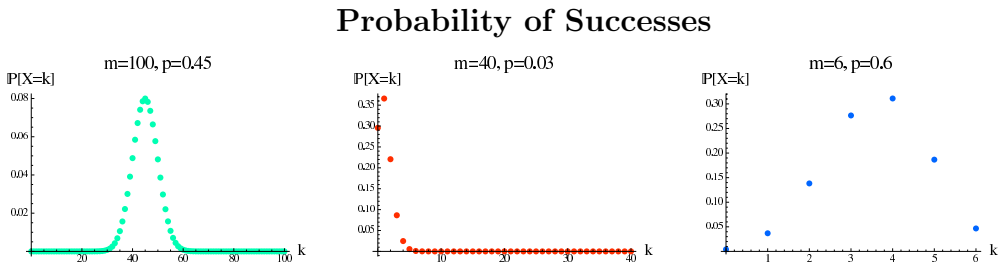

Figure 8.1: Plots of the probability of getting different numbers of successes k, for the binomial distribution $B(m, p)$, shown for three different values of m, the number of trials, and p, the probability of a success on each trial. Note that in the second and third image, the distribution is clearly not symmetrical around its mean.

indeed not available. In particular, we found one attempt to prove this result in the context of the analysis of some generalization bounds (Jaeger, 2005), but the proof is not sufficient to show the general case needed for the proof of Vapnik (1998), and only pertains to cases where the number of Bernoulli trials is 'large enough'. A concise proof of this inequality for the special case where $p \leq \frac{1}{2}$ (Rigollet and Tong, 2011) was also recently brought to our attention. However that proof technique does not seem to readily extend to the general case. We also derived an alternative straightforward proof for the same special case using Slud's inequality following a suggestion of Luc Devroye (private communication). However, the proof of the general case turned out to be more challenging. Our proof therefore seems to be the first rigorous justification of this inequality, which is needed, among other things, for the analysis of relative deviation bounds in machine learning.

In Section 8.2, we start with some preliminaries and then give the presentation of our main result. In Section 8.3, we give a detailed proof of the inequality.

8.2 Main Result

The following is the standard definition of a binomial distribution.

Definition 1. A random variable X is said to be distributed according to the binomial

distribution with parameters m (the number of trials) and p (the probability of success on each trial), if for $k = 0, 1, \ldots, m$ we have

$$\mathbb{P}[X = k] = \binom{m}{k} p^k (1-p)^{m-k}. \tag{8.1}$$

The binomial distribution with parameters m and p is denoted by $B(m, p)$. It has mean mp and variance $mp(1-p)$.

The following theorem is the main result of this chapter.

Theorem 15. *For any positive integer m and any probability p such that $p > \frac{1}{m}$, let X be a random variable distributed according to $B(m, p)$. Then, the following inequality holds:*

$$\mathbb{P}\big[X \geq \mathbb{E}\,[X]\,\big] > \frac{1}{4}, \tag{8.2}$$

where $\mathbb{E}[X] = mp$ is the expected value of X.

The lower bound is never reached but is approached asymptotically when $m = 2$ as $p \to \frac{1}{2}$ from the right. Note that when $m = 2$, the case $p = \frac{1}{2}$ is excluded from consideration, due to our assumption $p > \frac{1}{m}$. In words, the theorem says that a coin that is flipped a fixed number of times always has a probability of more than 1/4 of getting at least as many heads as the expected value of the number of heads, as long as the coin's chance of getting a head on each flip is not so low that the expected value is less than or equal to 1. The inequality is tight, as illustrated by Fig. 8.2.

8.3 Proof

Our proof is based on the following series of lemmas and corollaries and makes use of Camp-Paulson's normal approximation to the binomial cumulative distribution function

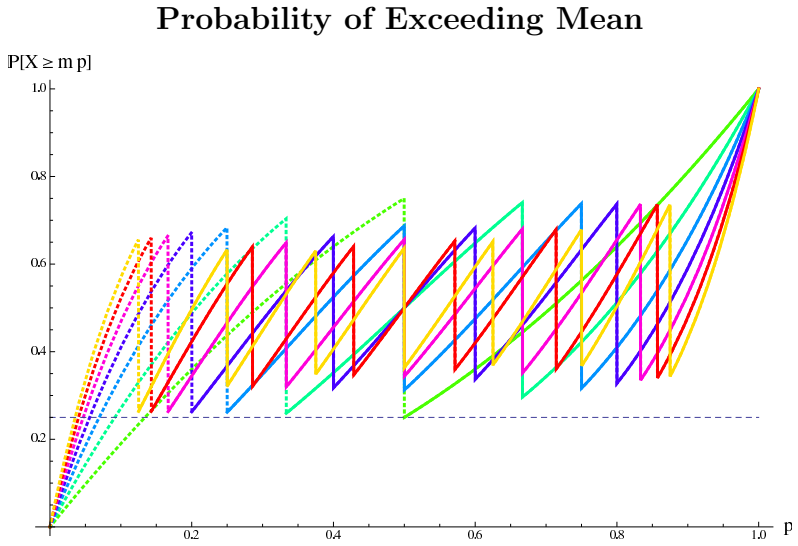

Figure 8.2: This plot depicts $\mathbb{P}[X \geq \mathbb{E}[X]]$, the probability that a binomially distributed random variable X exceeds its expectation, as a function of the trial success probability p. Each colored line corresponds to a different number of trials, $m = 2, 3, \ldots, 8$. Each colored line is dotted in the region where $p \leq \frac{1}{m}$, and solid in the region that our proof pertains to, where $p > \frac{1}{m}$. The dashed horizontal line at $\frac{1}{4}$ represents the value of the lower bound. Our theorem is equivalent to saying that for all positive integers m (not just the values of m shown in the plot), the solid portions of the colored lines never cross below the dashed horizontal line. As can be seen from the figure, the lower bound is nearly met for many of values of m.

(Johnson et al., 1995, 2005; Lesch and Jeske, 2009). We start with a lower bound that reduces the problem to a simpler one.

Lemma 13. *For all $k = 1, 2, \ldots, m-1$ and $p \in (\frac{k}{m}, \frac{k+1}{m}]$, the following inequality holds:*

$$\mathbb{P}_{X \sim B(m,p)}[X \geq \mathbb{E}[X]] \geq \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \geq k+1].$$

Proof. Let X be a random variable distributed according to $B(m, p)$ and let $F(m, p)$ denote $\mathbb{P}[X \geq \mathbb{E}[X]]$. Since $\mathbb{E}[X] = mp$, $F(m, p)$ can be written as the following sum:

$$F(m,p) = \sum_{j=\lceil mp \rceil}^{m} \binom{m}{j} p^j (1-p)^{m-j}.$$

We will consider the smallest value that $F(m,p)$ can take for $p \in (\frac{1}{m}, 1]$ and m a positive integer. Observe that if we restrict p to be in the half open interval $I_k = (\frac{k-1}{m}, \frac{k}{m}]$, which represents a region between the discontinuities of $F(m,p)$ which result from $\lceil mp \rceil$, then we have $mp \in (k-1, k]$ and so $\lceil mp \rceil = k$. Thus, we can write

$$\forall p \in I_k,\ \forall k = 0, 1, \ldots, m-1 \quad F(m,p) = \sum_{j=k}^{m} \binom{m}{j} p^j (1-p)^{m-j}.$$

The function $p \mapsto F(m,p)$ is differentiable for all $p \in I_k$ and its differential is

$$\frac{\partial F(m,p)}{\partial p} = \sum_{j=k}^{m} \binom{m}{j} (1-p)^{m-j-1} p^{j-1} (j - mp).$$

Furthermore, for $p \in I_k$, we have $k \geq mp$, therefore $j \geq mp$ (since in our sum $j \geq k$), and so $\frac{\partial F(m,p)}{\partial p} \geq 0$. The inequality is in fact strict when $p \neq 0$ and $p \neq 1$ since the sum must have at least two terms and at least one of these terms must be positive. Thus, the function $p \mapsto F(m,p)$ is strictly increasing within each I_k. In view of that, the value of $F(m,p)$ for $p \in I_{k+1}$ is lower bounded by $\lim_{p \to (\frac{k}{m})^+} F(m,p)$, which is given by

$$\lim_{p \to (\frac{k}{m})^+} F(m,p) = \sum_{j=k+1}^{m} \binom{m}{j} \left(\frac{k}{m}\right)^j \left(1 - \frac{k}{m}\right)^{m-j} = \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \geq k+1].$$

Therefore, for $k = 1, 2, \ldots, m-1$, whenever $p \in (\frac{k}{m}, \frac{k+1}{m}]$ we have

$$F(m,p) \geq \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \geq k+1]. \qquad \square$$

Corollary 14. *For all $p \in (\frac{1}{m}, 1)$, the following inequality holds:*

$$\mathbb{P}_{X \sim B(m,p)}[X \geq \mathbb{E}[X]] \geq 1 - \max_{k \in \{1, \ldots, m-1\}} \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \leq k].$$

Proof. By Lemma 13, the following inequality holds

$$\mathbb{P}_{X\sim B(m,p)}[X \geq \mathbb{E}[X]] \geq \min_{k\in\{1,..,m-1\}} \mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \geq k+1].$$

The right-hand side is equivalent to

$$\min_{k\in\{1,..,m-1\}} 1 - \mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] = 1 - \max_{k\in\{1,...,m-1\}} \mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k]$$

which concludes the proof. □

In view of Corollary 14, in order to prove our main result, it suffices that we upper bound the expression

$$\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] = \sum_{j=0}^{k} \binom{m}{j} \left(\frac{k}{m}\right)^j \left(1-\frac{k}{m}\right)^{m-j}$$

by $\frac{3}{4}$ for all integers $m \geq 2$ and $1 \leq k \leq m-1$. Note that the case $m = 1$ is irrelevant since the inequality $p > \frac{1}{m}$ assumed for our main result cannot hold in that case, due to p being a probability. The case $k = 0$ can also be ignored since it corresponds to $p \leq \frac{1}{m}$. Finally, the case $k = m$ is irrelevant, since it corresponds to $p > 1$. We note, furthermore, that when $p = 1$ that immediately gives $\mathbb{P}_{X\sim B(m,p)}[X \geq \mathbb{E}[X]] = 1 \geq \frac{1}{4}$.

Now, we introduce some lemmas which will be used to prove our main result.

Lemma 14. *The following inequality holds for all $k = 1, 2, \ldots, m-1$:*

$$\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] \leq \Phi\left[\frac{\beta_k\theta + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k + \gamma_{m,k}}}\right] + \frac{0.007}{\sqrt{1-\frac{1}{m}}},$$

where $\Phi\colon x \mapsto \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{\frac{-s^2}{2}} ds$ is the cumulative distribution function for the standard normal

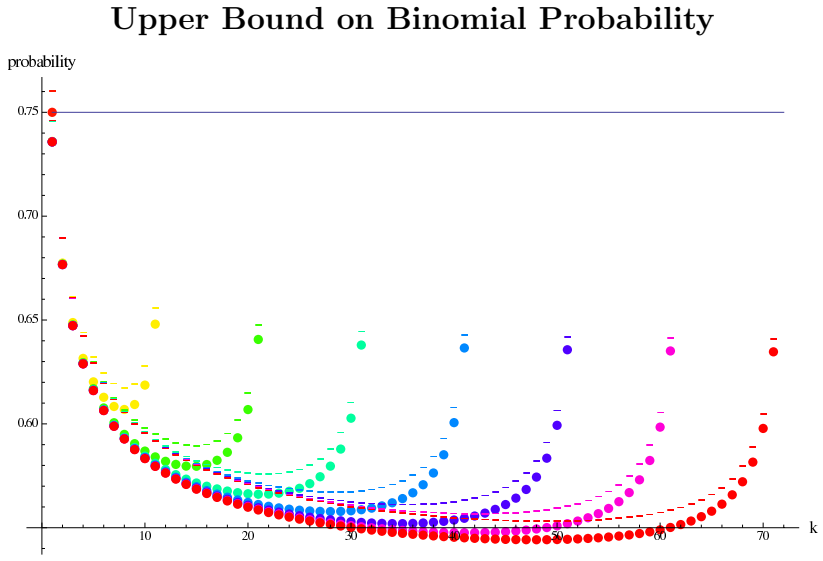

Figure 8.3: For m=2, 22, ..., 72 and $1 \leq k \leq m-1$, this plot depicts the values of $\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k]$ as colored dots (with one color per choice of m), against the values of the bound from Lemma 14, which are shown as short horizontal lines of matching color. The upper bound that we need to demonstrate, $\frac{3}{4}$, is shown as a blue horizontal line.

distribution and β_k, $\gamma_{m,k}$, *and* θ *are defined as follows:*

$$\beta_k = \frac{1}{1+k}\left(1+\frac{1}{k}\right)^{2/3} \quad \gamma_{m,k} = \frac{1}{m-k}, \quad \theta = \frac{17}{3\ 2^{1/3}} - 3\ 2^{1/3} \approx 0.71787.$$

Proof. Our proof makes use of Camp-Paulson's normal approximation to the binomial cumulative distribution function (Johnson et al., 1995, 2005; Lesch and Jeske, 2009), which helps us reformulate the bound sought in terms of the normal distribution. The Camp-Paulson approximation improves on the classical normal approximation by using a non-linear transformation. This is useful for modeling the asymmetry that can occur in the binomial distribution. The Camp-Paulson bound can be stated as follows (Johnson et al., 1995, 2005):

$$\left|\mathbb{P}_{X\sim B(m,p)}[X \leq j] - \Phi\left[\frac{c-\mu}{\sigma}\right]\right| \leq \frac{0.007}{\sqrt{mp(1-p)}}$$

where

$$c = (1-b)r^{1/3}, \quad \mu = 1-a, \quad \sigma = \sqrt{br^{2/3}+a},$$
$$a = \frac{1}{9m-9j}, \quad b = \frac{1}{9j+9}, \quad r = \frac{(j+1)(1-p)}{mp-jp}.$$

Plugging in the definitions for all of these variables yields

$$\Phi\left[\frac{c-\mu}{\sigma}\right] = \Phi\left[\frac{\left(1-\frac{1}{9}\frac{1}{j+1}\right)\left(\frac{1}{p}\frac{(j+1)(1-p)}{m-j}\right)^{1/3} + \frac{1}{9}\frac{1}{m-j} - 1}{\sqrt{\frac{1}{9}\frac{1}{j+1}\left(\frac{1}{p}\frac{(j+1)(1-p)}{m-j}\right)^{2/3} + \frac{1}{9}\frac{1}{m-j}}}\right].$$

Applying this bound to the case of interest for us where $p = \frac{k}{m}$ and $j = k$, yields

$$\frac{c-\mu}{\sigma} = \frac{\alpha_k + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k + \gamma_{m,k}}},$$

with $\alpha_k = \left(1+\frac{1}{k}\right)^{1/3}\left(3 - \frac{1}{3}\frac{1}{1+k}\right) - 3$, and with $\beta_k = \frac{1}{1+k}(1+\frac{1}{k})^{2/3}$. Thus, we can write

$$\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] \leq \Phi\left[\frac{\alpha_k + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k+\gamma_{m,k}}}\right] + \frac{0.007}{\sqrt{k\left(1-\frac{k}{m}\right)}}. \tag{8.3}$$

To simplify this expression, we will first upper bound α_k in terms of β_k. To do so, we consider the ratio

$$\frac{\alpha_k}{\beta_k} = \frac{(1+\frac{1}{k})^{1/3}(3-\frac{1}{3}\frac{1}{1+k}) - 3}{\frac{1}{1+k}(1+\frac{1}{k})^{2/3}} = \frac{3[(1+\frac{1}{k})^{1/3}-1]}{\frac{1}{1+k}(1+\frac{1}{k})^{2/3}} - \frac{1}{3(1+\frac{1}{k})^{1/3}}.$$

Let $\lambda = (1+\frac{1}{k})^{1/3}$, which we can rearrange to write $\frac{1}{1+k} = \frac{\lambda^3-1}{\lambda^3}$, with $\lambda \in (1, 2^{1/3}]$. Then, the

ratio can be rewritten as follows:

$$\frac{\alpha_k}{\beta_k} = \frac{3\lambda^3[(\lambda-1]}{(\lambda^3-1)\lambda^2} - \frac{1}{3\lambda} = \frac{3\lambda}{1+\lambda+\lambda^2} - \frac{1}{3\lambda}.$$

The expression is differentiable and its differential is given by

$$\begin{aligned}\frac{d}{d\lambda}\frac{\alpha_k}{\beta_k} &= \frac{3(1+\lambda+\lambda^2) - 3\lambda(2\lambda+1)}{(1+\lambda+\lambda^2)^2} + \frac{1}{3\lambda^2} \\ &= \frac{(1-\lambda^2)}{(1+\lambda+\lambda^2)^2} + \frac{1}{3\lambda^2} = \frac{-8(\lambda-1)^4 - 30(\lambda-1)^3 - 30(\lambda-1)^2 + 9}{3\lambda^2\left(1+\lambda+\lambda^2\right)^2}.\end{aligned}$$

For $\lambda \in (1, 2^{\frac{1}{3}}]$, $\lambda - 1 \leq 2^{\frac{1}{3}} - 1 \leq 0.26$, thus, the following inequality holds:

$$8(\lambda-1)^4 + 30(\lambda-1)^3 + 30(\lambda-1)^2 \leq 8(2^{\frac{1}{3}}-1)^4 + 30(2^{\frac{1}{3}}-1)^3 + 30(2^{\frac{1}{3}}-1)^2 \approx 2.59 < 9.$$

Thus, the derivative is positive, so $\frac{\alpha_k}{\beta_k}$ is an increasing function of λ on the interval $(1, 2^{\frac{1}{3}}]$ and its maximum is reached for $\lambda = 2^{\frac{1}{3}}$. For that choice of λ, the ratio can be written

$$\frac{3\ 2^{\frac{1}{3}}}{1+2^{\frac{1}{3}}+2^{\frac{2}{3}}} - \frac{1}{3\ 2^{\frac{1}{3}}} = \frac{17}{3\ 2^{\frac{1}{3}}} - 3\ 2^{\frac{1}{3}} = \theta \approx 0.717874,$$

which upper bounds $\frac{\alpha_k}{\beta_k}$. Since $\Phi[x]$ is a strictly increasing function, using $\alpha_k \leq \theta\beta_k$ yields

$$\Phi\left[\frac{\alpha_k + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k+\gamma_{m,k}}}\right] \leq \Phi\left[\frac{\beta_k\theta + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k+\gamma_{m,k}}}\right]. \tag{8.4}$$

We now bound the term $\frac{0.007}{\sqrt{k(1-\frac{k}{m})}}$. The quadratic function $k \mapsto k(1-\frac{k}{m})$ for $k = 1, 2, \ldots, m-1$, achieves its minimum at $k = 1$, giving $k(1-\frac{k}{m}) \geq (1-\frac{1}{m})$. Thus, in view of (8.3) and (8.4), we can write

$$\mathbb{P}\left[B(m, \frac{k}{m}) \leq k\right] \leq \Phi\left[\frac{\beta_k\theta + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k+\gamma_{m,k}}}\right] + \frac{0.007}{\sqrt{1-\frac{1}{m}}}.$$

This concludes the proof. □

As illustrated by Fig. 8.3, the upper bound provided by Lemma 14 closely approximates the true value.

Lemma 15. *Let* $\beta_k = \frac{1}{1+k}\left(1+\frac{1}{k}\right)^{2/3}$ *and* $\gamma_{m,k} = \frac{1}{m-k}$ *for* $m > 1$ *and* $k = 1, 2, \ldots, m-1$. *Then, the following inequality holds for* $\theta = \frac{17}{3\ 2^{\frac{1}{3}}} - 3\ 2^{\frac{1}{3}}$*:*

$$\Phi\left[\frac{\beta_k\theta + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k + \gamma_{m,k}}}\right] \leq \Phi\left[\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}}\right].$$

Proof. Since for $k = 1, 2, \ldots, m-1$, we have $\frac{1}{m-1} \leq \gamma_{m,k} \leq 1$, the following inequality holds:

$$\frac{\beta_k\theta + \frac{1}{3}\gamma_{m,k}}{\sqrt{\beta_k + \gamma_{m,k}}} \leq \max_{\gamma\in[0,1]} \frac{\beta_k\theta + \frac{1}{3}\gamma}{\sqrt{\beta_k + \gamma}}.$$

Since $\beta_k = \frac{1}{1+k}\left(1+\frac{1}{k}\right)^{2/3} > 0$, the function $\phi\colon \gamma \mapsto \frac{\beta_k\theta+\frac{1}{3}\gamma}{\sqrt{\beta_k+\gamma}}$ is continuously differentiable for $\gamma \in [0,1]$. Its derivative is given by $\phi'(\gamma) = \frac{\gamma+\beta_k(2-3\theta)}{6(\beta_k+\gamma)^{3/2}}$. Since $2 - 3\theta \approx -0.1536 < 0$, $\phi'(\gamma)$ is non-negative if and only if $\gamma \geq \beta_k(3\theta - 2)$. Thus, $\phi(\gamma)$ is decreasing for $\gamma < \beta_k(3\theta-2)$ and increasing for values of γ larger than that threshold. That implies that the shape of the graph of $\phi(\gamma)$ is such that the function's value is maximized at the end points. So $\max_{\gamma\in[0,1]}\phi(\gamma) = \max(\phi(0), \phi(1)) = \max(\sqrt{\beta_k}\theta, \frac{\beta_k\theta+\frac{1}{3}}{\sqrt{\beta_k+1}})$. The inequality $\sqrt{\beta_k}\theta \leq \frac{\beta_k\theta+\frac{1}{3}}{\sqrt{\beta_k+1}}$ holds if and only if $\beta_k(\beta_k+1)\theta^2 \leq (\beta_k\theta+\frac{1}{3})^2$, that is if $\beta_k \leq \frac{1}{\theta(9\theta-6)} \approx 3.022$. But since β_k is a decreasing function of k, it has $\beta_1 \approx 0.7937$ as its upper bound, and so this necessary requirement always holds. That means that the maximum value of $\phi(\gamma)$ for $\gamma \in [0,1]$ occurs at $\gamma = 1$, yielding the upper bound $\frac{\beta_k\theta+\frac{1}{3}}{\sqrt{\beta_k+1}}$, which concludes the proof. □

Corollary 15. *The following inequality holds for all* $m \geq 2$ *and* $k \geq 2$*:*

$$\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] \leq 0.7152.$$

Proof. By Lemmas 14-15, we can write

$$\mathbb{P}_{X\sim B(m,\frac{k}{m})}[X \leq k] \leq \Phi\left[\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}}\right] + \frac{0.007}{\sqrt{1-\frac{1}{m}}}.$$

Furthermore $\beta_k = \frac{1}{1+k}\left(1+\frac{1}{k}\right)^{2/3}$ is a decreasing function of k. Therefore, for $k \geq 2$, it must always be within the range $\beta_k \in [\lim_{k\to\infty}\beta_k, \beta_2] = \left[0, \frac{1}{2^{2/3}3^{1/3}}\right] \approx [0, 0.43679]$, which implies

$$\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}} \leq \max_{k\geq 2}\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}} \leq \max_{\beta\in[0,\beta_2]}\frac{\beta\theta + \frac{1}{3}}{\sqrt{\beta + 1}}.$$

The derivative of the differentiable function $g\colon \beta \mapsto \frac{\beta\theta+\frac{1}{3}}{\sqrt{\beta+1}}$ is given by $g'(\beta) = \frac{3(\beta+2)\theta-1}{6(\beta+1)^{3/2}}$. We have that $3(\beta+2)\theta - 1 \geq 6\theta - 1 \geq 6\times .717 - 1 > 0$, thus $g'(\beta) \geq 0$. Hence, the maximum of $g(\beta)$ occurs at β_2, where $g(\beta_2)$ is slightly smaller than 0.53968. Thus, we can write

$$\Phi\left[\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}}\right] \leq \Phi\left[\max_{\beta\in[0,\beta_2]}\frac{\beta\theta + \frac{1}{3}}{\sqrt{\beta + 1}}\right] \leq \Phi[0.53968] < 0.7053.$$

Now, $m \mapsto \frac{0.007}{\sqrt{1-\frac{1}{m}}}$ is a decreasing of function of m, thus, for $m \geq 2$ it is maximized at $m = 2$, yielding $\frac{0.007}{\sqrt{1-\frac{1}{m}}} \leq 0.0099$. Hence, the following holds:

$$\Phi\left[\frac{\beta_k\theta + \frac{1}{3}}{\sqrt{\beta_k + 1}}\right] + \frac{0.007}{\sqrt{1-\frac{1}{m}}} \leq 0.7053 + 0.0099 = 0.7152,$$

as required. □

The case $k = 1$ is addressed by the following lemma.

Lemma 16. *Let X be a random variable distributed according to $B(m, \frac{1}{m})$. Then, the following equality holds for any $m \geq 2$:*

$$\mathbb{P}\left[X \leq 1\right] \leq \frac{3}{4}.$$

Proof. For $m \geq 2$, define the function ρ by

$$\rho(m) = \mathbb{P}\left[X \leq 1\right] = \sum_{j=0}^{1} \binom{m}{j} \left(\frac{1}{m}\right)^j \left(1 - \frac{1}{m}\right)^{m-j} = \left(1 - \frac{1}{m}\right)^m + \left(1 - \frac{1}{m}\right)^{m-1}.$$

The value of the function for $m = 2$ is given by $\rho(2) = \left(1 - \frac{1}{2}\right)^2 + \left(1 - \frac{1}{2}\right)^{2-1} = \frac{3}{4}$. Thus, to prove the result, it suffices to show that ρ is non-increasing for $m \geq 2$. The derivative of ρ is given for all $m \geq 2$ by

$$\rho'(m) = (m-1)^{m-1} m^{-m} \left(2 + (2m-1) \log\left[1 - \frac{1}{m}\right]\right).$$

Thus, for $m \geq 2$, $\rho'(m) \leq 0$ if and only if $2 + (2m-1) \log\left[1 - \frac{1}{m}\right] \leq 0$. Now, for $m \geq 2$, using the first three terms of the expansion $-\log\left[1 - \frac{1}{m}\right] = \sum_{k=1}^{\infty} \frac{1}{k} \frac{1}{m^k}$, we can write

$$-(2m-1) \log\left[1 - \frac{1}{m}\right] \geq (2m-1)\left(\frac{1}{m} + \frac{1}{2m^2} + \frac{1}{3m^3}\right) = 2 + \frac{1}{6m^2} - \frac{1}{3m^3} \geq 2,$$

where the last inequality follows from $\frac{1}{6m^2} - \frac{1}{3m^3} \geq 0$ for $m \geq 2$. This shows that $\rho'(m) \leq 0$ for all $m \geq 2$ and concludes the proof. □

We now complete the proof of our main result, by combining the previous lemmas and corollaries.

of Theorem 15. By Corollary 14, we can write

$$\begin{aligned}
&\mathbb{P}_{X \sim B(m,p)}[X \geq \mathbb{E}[X]] \\
&\geq 1 - \max_{k \in \{1,\ldots,m-1\}} \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \leq k] \\
&= 1 - \max\left\{\mathbb{P}_{X \sim B(m, \frac{1}{m})}[X \leq 1], \max_{k \in \{2,\ldots,m-1\}} \mathbb{P}_{X \sim B(m, \frac{k}{m})}[X \leq k]\right\} \\
&\geq 1 - \max\{\frac{3}{4}, 0.7152\} = 1 - \frac{3}{4} = \frac{1}{4},
\end{aligned}$$

where the last inequality holds by Corollary 15 and Lemma 16. □

Corollary 16. *For any positive integer m and any probability p such that $p < 1 - \frac{1}{m}$, let X be a random variable distributed according to $B(m, p)$. Then, the following inequality holds:*

$$\mathbb{P}\big[X \leq \mathbb{E}\,[X]\,\big] > \frac{1}{4}, \tag{8.5}$$

where $\mathbb{E}[X] = mp$ is the expected value of X.

Proof. Let $G(m, p)$ be defined as

$$G(m,p) \equiv \mathbb{P}\big[X \leq \mathbb{E}\,[X]\,\big] = \sum_{j=0}^{\lfloor mp \rfloor} \binom{m}{j} p^j (1-p)^{m-j}$$

and let $F(m, p)$ be defined as before as

$$F(m,p) \equiv \mathbb{P}\big[X \geq \mathbb{E}\,[X]\,\big] = \sum_{j=\lceil mp \rceil}^{m} \binom{m}{j} p^j (1-p)^{m-j}.$$

Then, we can write, for $q = 1 - p$,

$$\begin{aligned} G(m,p) &= G(m, 1-q) \\ &= \sum_{j=0}^{\lfloor m(1-q) \rfloor} \binom{m}{j} (1-q)^j q^{m-j} \\ &= \sum_{t=m-\lfloor m(1-q) \rfloor}^{m} \binom{m}{m-t} (1-q)^{m-t} q^t \\ &= \sum_{t=\lceil mq \rceil}^{m} \binom{m}{t} (1-q)^{m-t} q^t \\ &= F(m,q) = F(m, 1-p) > \frac{1}{4} \end{aligned}$$

with the inequality at the end being an application of Theorem 15, which holds so long as

$q > \frac{1}{m}$, or equivalently, so long as $p < 1 - \frac{1}{m}$. □

8.4 Conclusion

We presented a rigorous justification of an inequality needed for the proof of relative deviations bounds in machine learning theory. A number of first attempts to find such a proof in the literature, or to prove this result ourselves, indicated that the inequality is not straightforward. Nevertheless, a simpler proof of this inequality is likely possible, and we may present such a simpler result in the future.

Acknowledgment

We thank Luc Devroye for discussions about the topic of this work.

Chapter 9

Dependent Rademacher Complexity and Hypothesis Set Stability

In Chapter 7 we explored one limitation of classical machine learning generalization inequalities, namely their dependence on the boundedness of the loss function. Here, we explore another limitation, which is that they generally require us to either view our machine learning algorithm as selecting its hypothesis from a fixed data independent hypothesis set, or to view the algorithm as considering only a single data dependent hypothesis. The former constraint characterizes Rademacher Complexity bounds, and the latter, Uniform Stability bounds.

In this chapter we construct a generalization inequality which allows for situations where the hypothesis set considered by our algorithm is data dependent. This case is actually surprisingly common, given that many algorithms set a free parameter by using k-fold cross-validation, which effectively makes the hypothesis set under consideration depend on the data. This work also applies to Chapter 2, which discusses Weight Functions that reduce the influence of some data points, Chapter 3, which applies Weight Functions in the context of

Ridge Regression, and Chapter 5, which is about unusual choices of regularization. In the first two cases we need our weighting for each data point to be data dependent, and in the latter case, we may want our generalized choice of regularization to be based on the data, so these are all potential applications of the theory of data dependent hypothesis sets. As we will see however, the bound we introduce in this chapter is much more general than these examples, as it encompasses both Rademacher Complexity and Uniform Stability as special cases.

9.1 Introduction

In machine learning theory, Rademacher Complexity (Bartlett and Mendelson, 2003; Koltchinskii and Panchenko, 2000b) is a powerful tool for measuring the potential that an algorithm has to overfit the training data. It is calculated from the set H of hypothesis functions from which an algorithm selects its hypothesis. In this chapter, we introduce D-Rademacher Complexity, a generalization of the usual Rademacher Complexity. It allows us to measure the complexity of a set of hypothesis functions, even when that set of hypothesis functions is dependent on the training sample, S (hence the "D" stands for "Dependent"). Formally, D-Rademacher Complexity is not defined on sets of functions, H, as in the usual case, but rather, on functions H_S that map the sample S into a set of functions. It is a strict generalization of the ordinary notion of Rademacher Complexity, which becomes apparent when we use $H_S = H$, in other words, when the function that maps the sample into a set of hypothesis functions does not actually depend on the sample.

Uniform Stability is another powerful tool for analyzing a learning algorithm's potential to overfit. Its view of machine learning algorithms is as maps from a training set S into a hypothesis function h_S (Bousquet and Elisseeff, 2002). In this chapter, we introduce

Hypothesis Set Stability, a generalization of the usual notion of Uniform Stability. It allows us to think of a learning algorithm as mapping the sample, S, into a set of hypotheses, H_S, rather than into a single hypothesis, h_S.

By generalizing Rademacher Complexity to allow for the possibility of the hypothesis set being data dependent[1], and by generalizing the Uniform Stability to allow for the possibility of having sets of hypotheses rather than just a single hypothesis, we show that these two popular theories can be fused into a single, overarching theory. We then apply this new theory to derive a generalization bound for data-dependent hypothesis sets, which relates a learning algorithm's training set error to its average error on new, unseen data. By allowing for hypothesis sets that have one element or many, that are data independent or highly data dependent, this bound encompasses both the standard Rademacher Complexity generalization bound and the standard Uniform Stability bound as special cases.

In practice, data dependent hypothesis sets are common. They occur, for instance, when a learning algorithm has a free parameter that is set using the training data, such as via k-fold cross-validation. Furthermore, just as Rademacher Complexity and Uniform Stability can both be used to analyze the same algorithm, this more general theory can be used to analyze one algorithm from multiple perspectives. The bounds introduced here could very likely be improved and tightened in future work, but this chapter provides a starting point

[1]We note that sometimes Rademacher Complexity is already viewed as a "data dependent" measure of complexity, because the Empirical Rademacher Complexity depends on the sample. But that is not the same type of data dependence that we are talking about here, as we are referring to the hypothesis set itself being data dependent. What's more, one may be tempted to simply fix the sample S in advance to form a static hypothesis set, and then apply the usual Empirical Rademacher Complexity bound. But because generalization bounds are probabilistic bounds with respect to a random draw of the sample S, it is unclear what the probability is being taken over if the sample is fixed. Finally, it is worth noting that the standard proof of the Rademacher Complexity bound fails in the data dependent hypothesis set case at the moment one attempts to introduce Rademacher random variables, and the standard proof of the Empirical Rademacher Complexity bound therefore also fails, since it is usually derived from the standard (non-empirical) Rademacher Complexity bound.

for a unified theory of Rademacher Complexity and Uniform Stability.

Note that, technically speaking, a data dependent hypothesis set H_S is a function that maps any set $S \in \mathcal{X}^m$ into a set of functions containing (at minimum) all of the hypothesis functions our machine learning algorithm could select when trained on the sample S. However, when it is clear from context, we will often use H_S to mean the set of hypothesis functions that results from applying this function to the sample S, just as $f(x)$ is often used to refer to both the function f and its output when the value x is used as an input. It is also important to observe that a data dependent hypothesis set, H_S, does not actually correspond to a specific algorithm, but rather, to a set of algorithms. In particular, it refers to the set of all algorithms that, when trained on sample S, always output a single hypothesis function that is contained in the set H_S.

9.1.1 Example Hypothesis Set

To make the notion of a data dependent hypothesis set concrete, we now give an example. Throughout this chapter, we will assume that each point in our training sample, S, is drawn independently from the same fixed distribution D, and is limited to some domain $\mathcal{X}$. We write $S = \{s_1, s_2, \ldots, s_m\}$, and we think of each s_i as having two parts. We write $s_i = (X^i, y_i)$, with X^i being one training point in a matrix X of such points, and y_i being the corresponding training label from a column vector of training labels y. We also need to select an associated loss function L for our learning problem, which measures the error that is realized when predicting point s using hypothesis $h \in H_S$, written $L(h, s)$.

As an example, consider a data dependent hypothesis set H_S that consists of all functions that are non-negative linear combinations of exactly m multi-variate gaussians, all sharing a positive definite covariance matrix Σ, where one gaussian is centered at each of the points X^i, and where each gaussian is multiplied by its corresponding training label. That means that, for positive numbers c_i, each hypothesis h in our hypothesis set takes the form:

$$h(x) = \sum_{i=1}^{m} c_i \, y_i \, e^{-\frac{1}{2}(x-X^i)^\intercal \Sigma^{-1}(x-X^i)}$$

and our entire data dependent hypothesis set is

$$H_S = \{h(x) \mid \forall \; c_i \geq 0 \;\&\; \Lambda \succ 0\}.$$

Then, to measure error, suppose we have the squared loss function, $L(h, s_i) = (h(X^i) - y_i)^2$. Note that the hypothesis set depends directly on the sample points $s_i = (X^i, y_i)$, and so is indeed data dependent. Note also that for each training sample, S, the definition above describes a set of hypotheses, not just a single hypothesis, due to the undetermined non-negative values c_i and the positive definite matrix Λ. Therefore this case can neither be analyzed using Rademacher Complexity (since there is data dependence) nor using Uniform Stability (since the case refers to an entire set of hypotheses, not just a single hypothesis). We could, however, apply the former theory if we discard information and switch to the larger data independent hypothesis set $\bigcup_{S \in \mathcal{X}^m} H_S$, and we could apply the latter theory if we add extra information to specify precisely how the c_i and Σ vary as a function of S, so that we are considering only a single, data dependent hypothesis function. One advantage of the generalized theory proposed in this chapter is that it applies to cases such as this one directly, without needing to discard or specify extra information.

9.2 Two Perspectives

Except in trivial cases, there are always multiple ways to represent a machine learning algorithm's hypothesis set. At one extreme, we can think of its hypothesis set as not depending on the training data at all, by choosing a set of hypothesis functions H that fully encompasses the possible hypotheses that may be selected by the algorithm, as was illustrated in the example above. When we represent an algorithm in this manner, we can write a generalization bound in terms of the Rademacher Complexity, which tells us that the out of sample performance of the algorithm will be unlikely to exceed the training error by much, as long as the Rademacher Complexity is low (Mohri et al., 2012).

At the other extreme, we can imagine regarding the hypothesis set as just a single function $H_S = \{h_s\}$, but this function will of course then be highly dependent on the training sample, S. In this case, a generalization bound can be produced by analyzing the Uniform Stability of the function h_S (Bousquet and Elisseeff, 2002). This analysis is based on examining the maximum amount that predictions can change when the training sample S is replaced with S', a sample that is identical except for one point which has been changed.

To see both of the perspectives mentioned above, take Ridge Regression as an example. On the one hand, we can view this algorithm as having a fixed hypothesis set

$$H = \{h_w(x) = w^\intercal x \mid ||w||_2 \leq \epsilon\}$$

where ϵ is a fixed complexity constant. On the other hand, we can view it as having a single data dependent hypothesis

$$h_S(x) = (XX^\intercal + \lambda I)^{-1} Xy$$

where λ is a complexity constant which is a function of ϵ.

These perspectives are equally correct. But there are also infinite perspectives that lie between these two extremes, where we think of Ridge Regression as searching through a data dependent hypothesis set H_S, which is always a subset of the data independent hypothesis set H, but that always contains the data dependent hypothesis $h_S(x)$. We now proceed to present a theory that encompassed both extremes, as well as the perspectives in between.

9.3 Notation

Let L be our loss function of interest. let $L(h, s_i)$ be the loss of hypothesis $h \in H_S$ when applied to $s_i = (X^i, y_i) \in S$, where X^i is a training point and y_i is a corresponding training label. As before, we assume that each training point, s, is drawn from some fixed distribution D, and is restricted to the set $\mathcal{X}$. For the generalization error of a hypothesis h, we write:

$$\mathcal{L}(h) \equiv \mathop{\mathbb{E}}_{s \sim D}[L(h, s)]$$

where $\mathbb{E}_{s \sim D}[.]$, indicates the expected value taken over points s drawn from distribution D. For the empirical error achieved on a sample $S = \{s_1, \ldots, s_m\}$, we write:

$$\hat{\mathcal{L}}_{\mathcal{S}}(h) \equiv \frac{1}{m} \sum_{i=1}^{m} L(h, s_i).$$

For arbitrary sets of points Q and R each of size m, we define:

$$\eta(Q,R) \equiv \sup_{h \in H_Q} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{R}}(h).$$

which measures the most that the generalization error and the empirical error on sample R can differ for any hypothesis in set H_Q. Furthermore, we will sometimes use $\mathbb{E}_S[.]$ as a shorthand for $\mathbb{E}_{S \sim D^m}[.]$, and $\mathbb{E}_s[.]$ as a shorthand for $\mathbb{E}_{s \sim D}[.]$. We write $S \equiv \{s_1, \ldots, s_m\}$ when we need to refer to the individual points within the sample S. Likewise, when we need to refer to the individual points of another sample, R, we write $R \equiv \{r_1, \ldots, r_m\}$.

9.4 Definitions

Our main theorem of this chapter requires a handful of new definitions. We now generalize Rademacher Complexity and Uniform Stability, and introduce the idea of Error Radius.

9.4.1 Definition of D-Rademacher Complexity

Let $\mathcal{C}$, which we call the "D-Rademacher Complexity", be given by

$$\mathcal{C} \equiv \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right] \tag{9.1}$$

with $H_{S,R}$ being the "symmetrization" of H_S, which we define as

$$H_{S,R} \equiv \bigcup_{\substack{A \subset S \bigcup R \\ |A|=m}} H_A \tag{9.2}$$

and where the expectation $\mathbb{E}_{S,R}[.]$ is taken with respect to two samples, S and R, both of size m, with each point in both samples drawn independently from the distribution D, and

where each σ_i is an independent random variable taking $+1$ and -1 with equal probability, and the r_i are the points in sample R. This symmetrization is critical because, as will be seen below in the proof of our primary theorem, inserting Rademacher random variables into our formula is equivalent to moving points between two sets, and so our equation must be symmetric with respect to these two sets so that we can insert these Rademacher random variables without changing the value of our expression. In the typical case where hypothesis sets are not data dependent, symmetry is still needed but it is achieved more easily without needing to introduce $H_{S,R}$.

9.4.2 Definition of Hypothesis Set Stability

For H_S, we define the "Hypothesis Set Stability" constant $\mathcal{Q} \geq 0$, which is the minimum value such that, for all allowed S, and for all $h \in H_S$, and any S' that differs from S by changing just one point, there always exists an $h' \in H_{S'}$ such that:

$$|L(h, s) - L(h', s)| \leq \mathcal{Q} \quad \forall s \in \mathcal{X}. \tag{9.3}$$

9.4.3 Definition of Error Radius

Finally, thinking of H_S as a function from the sample S to a set of hypothesis functions, we define the "Error Radius" constant $\omega \geq 0$, which is the smallest constant such that for all allowed S, and for all $h_1, h_2 \in H_S$, we always have:

$$|L(h_1, s) - L(h_2, s)| \leq \omega \quad \forall s \in \mathcal{X}. \tag{9.4}$$

Note that for the remainder of this chapter we will assume that the loss function L only takes

values in $[0, \mathcal{M}]$. In such cases, $\mathcal{C}$, $\mathcal{Q}$, and ω are all less than or equal to $\mathcal{M}$.

9.5 Main Result

We now have the notation and definitions that we need to introduce the main result of this chapter. The remainder of the chapter will focus on proving this result, and discussing some of its special cases.

Theorem 16 (Generalization Bound for Data-Dependent Hypothesis Sets)**.** *Let H_S be a data dependent hypothesis set (i.e. a function mapping a sample S into a set of hypothesis functions), which contains in it (at minimum) all the possible hypothesis functions that a particular machine learning algorithm could select when given training sample S. Assume that S consist of m points (with labels). Let L be a loss function that only takes values in $[0, \mathcal{M}]$. Consider random draws of the set S, with each sample point drawn independently from the distribution D. Then, for each $h \in H_S$, with probability at least $1 - \delta$, the generalization error $\mathcal{L}(h)$ has the following relationship with the training set error $\hat{\mathcal{L}}_{\mathcal{S}}(h)$:*

$$\mathcal{L}(h) \leq \hat{\mathcal{L}}_{\mathcal{S}}(h) + \min\left\{2\mathcal{C},\ \mathcal{Q} + \omega\right\} + \frac{\mathcal{M} + 2m\mathcal{Q}}{\sqrt{2}}\sqrt{\frac{\ln(\frac{1}{\delta})}{m}} \tag{9.5}$$

9.5.1 Interpretation

Intuitively, Theorem 16 is saying that if $\mathcal{Q}$ falls like $\frac{1}{m}$, and we have either a small enough $\mathcal{C}$ or a small enough ω, then the prediction error of our machine learning algorithm on future data (drawn from the same distribution as our training data) is unlikely to be much worse than the error on the training data (for sufficiently large m). We can think of our parameters of interest as follows:

1. $\mathcal{C}$ measures how "complex" our data dependent hypothesis set H_S is, measured in terms

of how well the best functions from it it can fit random binary noise on average (when using all hypotheses that can be generated using subsamples of size m taken from a set of $2m$ points).

2. $\mathcal{Q}$ measures how "stable" H_S is, in terms of how much difference in error (on any point) there can be between a hypothesis in H_S and the closest hypothesis in $H_{S'}$ where S' differs by one point from S.

3. ω measure how "big" H_S is, in terms of how much difference in error (on any point) there can be for any two hypothesis in H_S.

Therefore, we can summarize the main theorem of this chapter as saying: as long as H_S is sufficiently "stable", and additionally it is either sufficiently "low in complexity" or sufficiently "small", then error on future data is unlikely to be much worse than error on the training set when the sample size is large enough.

9.5.2 Limitations

While Theorem 16 is more general than both the standard Rademacher Complexity bound and the standard Uniform Stability bound, it is important to note its limitations. It only applies to bounded loss functions, and will only give meaningful results when $\mathcal{Q}$ falls at a rate faster tha $1/\sqrt{m}$ with respect to the sample size m, though a rate of $1/m$ is needed to match conventional bounds. Fortunately, when the hypothesis set is data independent this is automatically true since we obtain $\mathcal{Q} = 0$, and in the case where the hypothesis set contains just one (data dependent) function, this requirement simply reduces to the same requirements of ordinary Uniform Stability. Finally, we note that if the hypothesis set is highly dependent on the sample, then the D-Rademacher Complexity value $\mathcal{C}$ could be excessively large, and when the hypothesis set contains many functions with different behavior, the Error Radius

ω may be excessively large. So while the bound can be applied to any data dependent hypothesis set with a bounded loss function, it will not yield a helpful bound in all cases. Though, as we shall see, in the special cases where the standard Rademacher Complexity is applicable, and the special cases where standard Uniform Stability apply, the bound in Theorem 16 exactly reproduces the standard bounds! So it is indeed a strict generalization of both of these standard theories.

9.6 Proof Outline

Our approach for deriving this theorem mirrors the proof of standard Rademacher Complexity bound from Mohri et al. (2012), while borrowing ideas from the proof of the standard Uniform Stability bound from Bousquet and Elisseeff (2002), but generalizes some of the ideas used in those proofs, and introduces new definitions. We start by showing that the expected value $\mathbb{E}[\eta(S, S)]$ is upper bounded by the quantity that we call the D-Rademacher Complexity. Next, we introduce Hypothesis Set Stability. Doing so will provide us with the conditions we need to apply McDiarmid's inequality, so that we can say that, with high probability, $\mathbb{E}[\eta(S, S)]$ is close to $\eta(S, S)$. After that, we will introduce the Error Radius, and use it to upper bound $\mathbb{E}[\eta(S, S)]$ a second time. Then we will bound $\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h)$ with $\eta(S, S)$, and bound $\eta(S, S)$ with $\mathbb{E}[\eta(S, S)]$, and finally use our upper bounds coming from the D-Rademacher Complexity and the Error Radius (separately) to bound $\mathbb{E}[\eta(S, S)]$, which will at last yield the generalization bound of interest.

9.7 D-Rademacher Complexity

Let $\sigma = (\sigma_1, \ldots, \sigma_m)$ be a vector of Rademacher random variables, defined to be independent and identically distributed, with each taking on +1 and -1 with equal probability. We

also introduce a new hypothesis set $H_{S,R}$ that depends on the two samples S and R (both of size m) in a symmetric fashion. We construct $H_{S,R}$, as in Thoerem 16, by taking the union of all data dependent hypothesis sets H_A that can be formed by letting A be constructed from points in S and R, restricting A to have size m.

We observe that in the special case where H is actually independent of the training set, we simply have $H_A = H$ and therefore $H_{S,R} = H$. Since $H_{S,R}$ is a strictly larger set than H_S, but is symmetric in S and R, it will enable us to introduce the Rademacher random variables into our upper bound on $\mathbb{E}_S[\eta(S, S)]$. The symmetry of the hypothesis set that it gives us with respect to S and R is helpful because introducing Rademacher random variables is equivalent to swapping an arbitrary number of points between S and R, and we need this to be able to happen without changing the value of the expression. We now proceed with our derivation of D-Rademacher Complexity, which we will use to upper bound $\mathbb{E}_S[\eta(S, S)]$, writing:

$$\mathop{\mathbb{E}}_{S}[\eta(S,S)] = \mathop{\mathbb{E}}_{S}[\sup_{h\in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h)]$$

$$= \mathop{\mathbb{E}}_{S}[\sup_{h\in H_S} \mathop{\mathbb{E}}_{R}[\hat{\mathcal{L}}_{\mathcal{R}}(h)] - \hat{\mathcal{L}}_{\mathcal{S}}(h)]$$

$$\leq \mathop{\mathbb{E}}_{S,R}[\sup_{h\in H_S} \hat{\mathcal{L}}_{\mathcal{R}}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h)]$$

$$= \mathop{\mathbb{E}}_{S,R}\left[\sup_{h\in H_S}\left(\frac{1}{m}\sum_{i=1}^{m} L(h, r_i) - \frac{1}{m}\sum_{i=1}^{m} L(h, s_i)\right)\right]$$

$$\leq \mathop{\mathbb{E}}_{S,R}\left[\sup_{h\in H_{S,R}}\left(\frac{1}{m}\sum_{i=1}^{m} L(h, r_i) - \frac{1}{m}\sum_{i=1}^{m} L(h, s_i)\right)\right]$$

$$= \mathop{\mathbb{E}}_{\sigma}\mathop{\mathbb{E}}_{S,R}\left[\frac{1}{m}\sup_{h\in H_{S,R}}\left(\sum_{i=1}^{m} \sigma_i(L(h, r_i) - L(h, s_i))\right)\right]$$

$$\leq \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) + \frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} (-\sigma_i) L(h, s_i) \right]$$

$$= \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right] + \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} (-\sigma_i) L(h, s_i) \right]$$

$$= 2 \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right] \equiv 2\mathcal{C}.$$

We use $\mathcal{C}$ above to denote the D-Rademacher Complexity, which we have also defined in Theorem 16. It is interesting to note that we can also write our D-Rademacher Complexity in terms of the usual Empirical Rademacher Complexity, $\hat{\mathcal{C}}_R[H]$, on sample R for fixed hypothesis set H, using:

$$\mathcal{C} = \mathop{\mathbb{E}}_{S,R} \left[\mathop{\mathbb{E}}_{\sigma} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right] \right] = \mathop{\mathbb{E}}_{S,R} \left[\hat{\mathcal{C}}_R[H_{S,R}] \right].$$

So the D-Rademacher Complexity is just the average Empirical Rademacher Complexity, with respect to two samples S and R simultaneously, on the symmetrized data dependent hypothesis set $H_{S,R}$.

Intuitively, the D-Rademacher Complexity measures how well, on average, the loss of the hypotheses in our data dependent hypothesis set are able to fit +1/-1 noise, when considering all hypothesis sets that can be created from m points out of a sample of $2m$ points. Considering hypothesis sets that use any m points out of $2m$ may not be the best we can do, and opens up the possibility in future work of finding a smaller alternative that serves the same purpose. For instance, perhaps there is a way to define the D-Rademacher Complexity as simply the average Empirical Rademacher Complexity, $\mathbb{E}_S \left[\hat{\mathcal{C}}_S[H_S] \right]$, without changing the bound in Theorem 16.

9.8 Hypothesis Set Stability

Our next step is to analyze the stability of $\eta(S, S)$ with respect to replacing S with S', where S' is identical to S but with one point changed. Without loss of generality, we assume that the point that was changed is the last one, s_m, replaced with some other point s'_m. This analysis will allow us to apply McDiarmid's inequality to show that $\mathbb{E}_S[\eta(S, S)]$ and $\eta(S, S)$ are close with high probability. We begin by writing:

$$\eta(S, S) - \eta(S', S') = (\eta(S, S) - \eta(S, S')) + (\eta(S, S') - \eta(S', S')) \tag{9.6}$$

We now bound the first term on the right hand side of the equation, using our assumption that $L(h, s) \in [0, \mathcal{M}]$:

$$\eta(S, S) - \eta(S, S') = \sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) - \sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}'}(h).$$

$$= \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \right) - \sup_{h \in H_S} \left(\mathcal{L}(h) - \frac{1}{m} \left(\sum_{i=1}^{m-1} L(h, s_i) + L(h, s'_m) \right) \right)$$

$$= \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \right) - \sup_{h \in H_S} \left(\mathcal{L}(h) - \frac{1}{m} \left(\sum_{i=1}^{m-1} L(h, s_i) + L(h, s_m) + L(h, s'_m) - L(h, s_m) \right) \right)$$

$$= \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \right) - \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) - \frac{1}{m} (L(h, s'_m) - L(h, s_m)) \right)$$

$$\leq \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \right) - \sup_{h \in H_S} \left(\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) - \sup_{\widetilde{h} \in H_S} \frac{1}{m} (L(\widetilde{h}, s'_m) - L(\widetilde{h}, s_m)) \right)$$

$$= \sup_{\widetilde{h} \in H_S} \frac{1}{m} (L(\widetilde{h}, s'_m) - L(\widetilde{h}, s_m)) \leq \frac{\mathcal{M}}{m}.$$

Next we bound the second term on the right hand side of Equation 9.6. By the definition of the supremum, for any $\eta > 0$ as small as we like, there will always exist some hypothesis

$h_\eta \in H_S$ such that

$$\sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}'}(h) \leq \eta + \mathcal{L}(h_\eta) - \hat{\mathcal{L}}_{\mathcal{S}'}(h_\eta),$$

which means that for all $h' \in H_{S'}$ we have:

$$\eta(S, S') - \eta(S', S') = \sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}'}(h) - \sup_{h \in H_{S'}} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}'}(h)$$

$$\leq \eta + \mathcal{L}(h_\eta) - \hat{\mathcal{L}}_{\mathcal{S}'}(h_\eta) - \sup_{h \in H_{S'}} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}'}(h)$$

$$\leq \eta + (\mathcal{L}(h_\eta) - \hat{\mathcal{L}}_{\mathcal{S}'}(h_\eta)) - (\mathcal{L}(h') - \hat{\mathcal{L}}_{\mathcal{S}'}(h'))$$

$$= \eta + (\mathcal{L}(h_\eta) - \mathcal{L}(h')) + (\hat{\mathcal{L}}_{\mathcal{S}'}(h') - \hat{\mathcal{L}}_{\mathcal{S}'}(h_\eta)).$$

We now apply the Hypothesis Set Stability constant $\mathcal{Q}$, as defined in Equation 9.3. It measures the most that the hypothesis set can change when one point in the sample changes. Intuitively, when $\mathcal{Q}$ is small it means that if S and S' differ only by one point, then for every hypothesis in H_S we should be able to find some "close" hypothesis in $H_{S'}$, where we are measuring closeness in terms of the greatest difference in errors the two hypotheses can achieve on any possible point s (which is not necessarily in S or S'). We call this Hypothesis Set Stability because it generalizes Uniform Stability, but it applies to an entire data dependent hypothesis set H_S (when thought of as a function from S to a set of hypothesis functions) rather than to a data dependent single hypothesis h_S. Note that for $h \in H_S$ and $h' \in H_{S'}$, $|L(h, s) - L(h', s)|$ is always bounded by $\mathcal{M}$, but this is not a tight enough bound for our purposes. If the stability constant $\mathcal{Q}$ does not decay with respect to m as fast as $\frac{1}{m}$, then though we will still get a generalization bound in the end, the term associated with McDiarmid's inequality will not fall as fast as the desired rate $\frac{1}{\sqrt{m}}$.

Applying Hypothesis Set Stability, combined with the fact that we are free to choose any

$h' \in H_{S'}$ in the equation we were deriving above, means that we can simply pick h' so that it is "close" to $h_\eta \in H_S$. Hence we have:

$$\hat{\mathcal{L}}_{\mathcal{S}'}(h') - \hat{\mathcal{L}}_{\mathcal{S}'}(h_\eta) = \frac{1}{m}\sum_{i=1}^{m}(L(h', s_i') - L(h_\eta, s_i')) \leq \frac{1}{m}\sum_{i=1}^{m}\mathcal{Q} = \mathcal{Q}$$

and

$$\mathcal{L}(h_\eta) - \mathcal{L}(h') = \mathop{\mathbb{E}}_{s\sim D}[L(h_\eta, s) - L(h', s)] \leq \mathop{\mathbb{E}}_{s\sim D}[\mathcal{Q}] = \mathcal{Q}.$$

Therefore:

$$\eta(S, S') - \eta(S', S') \leq \eta + 2\mathcal{Q}$$

but since this holds for all $\eta > 0$, we conclude that:

$$\eta(S, S') - \eta(S', S') \leq 2\mathcal{Q}.$$

We now, at last, are in a position to bound Equation 9.6. We have:

$$\eta(S, S) - \eta(S', S') = (\eta(S, S) - \eta(S, S')) + (\eta(S, S') - \eta(S', S'))$$

$$\leq \frac{\mathcal{M}}{m} + 2\mathcal{Q} = \frac{\mathcal{M} + 2m\mathcal{Q}}{m}.$$

Since starting with S and replacing one point to get S' is symmetrical to starting with S' and replacing one point to get S, we can write an identical bound for $\eta(S', S') - \eta(S, S)$ as we did for $\eta(S, S) - \eta(S', S')$, which lets us conclude that:

$$|\eta(S, S) - \eta(S', S')| \leq \frac{\mathcal{M} + 2m\mathcal{Q}}{m}.$$

This bound gives us the stability we need (with respect to a point being changed) to apply McDiarmid's inequality. This will let us conclude that $\eta(S, S)$ is unlikely to be much bigger than $\mathbb{E}_S[\eta(S, S)]$.

9.9 McDiarmid's Inequality

McDiarmid's inequality tells us in this context that scalar valued functions of randomly drawn points, which change by a bounded amount when one such point is dropped, will have a very quickly declining probability of being progressively farther from the mean. Of particular interest is the way this probability drops off as the sample size m increases, which we will see has the form $\frac{1}{\sqrt{m}}$.

Theorem 17 (McDiarmid's Inequality)**.** *Let V be some set, and let $v = (v_1, \ldots, v_m)$ with each $v_i \in V$. Furthermore, assume the v_i are drawn independently from each other according to some (possibly different) probability distributions. Let $F(v)$ be a scalar valued function. Now, if for every $v' \in V^m$ that is identical to $v \in V^m$ but with the ith value v_i changed to any other $v_i' \in V$ we have the property*

$$|F(v) - F(v')| \leq c_i,$$

then the probability of $F(v)$ exceeding its expected value $\mathbb{E}_v[F(v)]$ by more than ϵ satisfies

$$\mathbb{P}[F(v) - \mathbb{E}_v[F(v)] \geq \epsilon] \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^m c_i^2}}.$$

We now apply McDiarmid's inequality to $\eta(S, S)$ by using $v = (s_1, s_2, \ldots, s_m)$ and

$F(v) = \eta(S, S)$. Since we have shown that $|\eta(S, S) - \eta(S', S')| \leq \frac{\mathcal{M}+2m\mathcal{Q}}{m}$ when S and S' differ by only one point, we have $c_i = \frac{\mathcal{M}+2m\mathcal{Q}}{m}$, which yields

$$\mathbb{P}[\eta(S, S) - \mathbb{E}_S[\eta(S, S)] \geq \epsilon] \leq e^{\frac{-2m\epsilon^2}{(\mathcal{M}+2m\mathcal{Q})^2}}.$$

If we want $\eta(S, S)$ to exceed its expected value by more than ϵ with at most probability δ, then we set $\delta = e^{\frac{-2m\epsilon^2}{(\mathcal{M}+2m\mathcal{Q})^2}}$ and solve for ϵ, which yields:

$$\epsilon = \frac{\mathcal{M} + 2m\mathcal{Q}}{\sqrt{2}}\sqrt{\frac{\ln \frac{1}{\delta}}{m}}.$$

We can now state that, with probability at least $1 - \delta$, we have:

$$\eta(S, S) \leq \mathbb{E}_S[\eta(S, S)] + \frac{\mathcal{M} + 2m\mathcal{Q}}{\sqrt{2}}\sqrt{\frac{\ln \frac{1}{\delta}}{m}}. \tag{9.7}$$

9.10 Error Radius

Consider again the Error Radius constant ω, which was defined in Equation 9.4. It measures the "size" of a hypothesis set, on a scale from 0 to $\mathcal{M}$, in terms of how much any two hypothesis from H_S can differ in error when applied to any point, for all fixed samples S. To confirm that ω is indeed a measure of size, note that on the smallest hypothesis sets possible, that is, ones which only ever contain a single function, we obtain $\omega = 0$. What is more, if H_S is as large as possible for all samples S, by which we mean it contains all functions, then we obtain $\omega = \mathcal{M}$. Finally, we note that, if for data dependent hypothesis sets A_S and B_S, we have that $A_S \subset B_S$ for all samples S, then the Error Radius of A_S is less than or equal to the Error Radius of B_S.

Now, let us fix $\gamma > 0$, which cannot depend on the sample S. Then, let $z_\gamma(S)$ be a function of S such that, given any sample S as input, it always produces as its output a hypothesis function h_S^γ that satisfies:

$$\sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \leq \gamma + \mathcal{L}(h_S^\gamma) - \hat{\mathcal{L}}_{\mathcal{S}}(h_S^\gamma). \tag{9.8}$$

This function $z_\gamma(S)$ is possible to construct. To see why, consider that by the definition of the supremum, for any fixed $\gamma > 0$, and any fixed S, at least one function will satisfy Equation 9.8. Let Q_S^γ be the set of all functions that satisfy the equation for that particular S. Now let Q^γ be a set of sets, which contains as its elements all Q_S^γ, for all allowed S. Then, by the axiom of choice, there exists a "choice function" $c(A)$, which when applied to any set A that is an element of Q^γ, will output exactly one function that satisfies Equation 9.8. Then we can simply use $z_\gamma(S) \equiv c(Q_S^\gamma)$.

Now, let S^0 be the sample S but with the last point replaced with the point s_0, where s_0 is drawn independently and from the same distribution as the points in S. Then, using h_S^γ as defined above, we have:

$$\mathop{\mathbb{E}}_{S}[\eta(S,S)] = \mathop{\mathbb{E}}_{S}[\sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h)] \leq \mathop{\mathbb{E}}_{S}[\gamma + \mathcal{L}(h_S^\gamma) - \hat{\mathcal{L}}_{\mathcal{S}}(h_S^\gamma)]$$

$$= \gamma + \mathop{\mathbb{E}}_{S}[\mathcal{L}(h_S^\gamma)] - \mathop{\mathbb{E}}_{S}[\hat{\mathcal{L}}_{\mathcal{S}}(h_S^\gamma)] = \gamma + \mathop{\mathbb{E}}_{S}\mathop{\mathbb{E}}_{s_0}[L(h_S^\gamma, s_0)] - \mathop{\mathbb{E}}_{S}\mathop{\mathbb{E}}_{s_0}[L(h_{S^0}^\gamma, s_0)]$$

$$= \gamma + \mathop{\mathbb{E}}_{S}\mathop{\mathbb{E}}_{s_0}[L(h_S^\gamma, s_0) - L(h_{S^0}^\gamma, s_0)]]$$

Now, we apply Hypothesis Set Stability. Since S and S^0 only differ by one point, we know that there is some $\mathcal{Q}$ such that, for any $h_S^\gamma \in H_S$, there always exists an $h'_{S^0} \in H_{S^0}$ with

$$|L(h_S^\gamma, s) - L(h'_{S^0}, s)| \leq \mathcal{Q} \quad \forall s \in \mathcal{X}.$$

Intuitively, a small ω means that every pair of hypotheses in the same set, H_S, must be "close" to each other in the sense of their errors never being too far apart when applied to any point. Whereas a small $\mathcal{Q}$ means that for any samples S and S' that differ by only one point, it is the case that for any hypothesis in H_S there must be a "close" hypothesis in H'_S, again in the sense of their errors never being too far apart when applied to any point. Putting these two ideas together, we have that there exists an h'_{S^0} such that:

$$L(h^\gamma_S, s_0) - L(h^\gamma_{S^0}, s_0)$$

$$= (L(h^\gamma_S, s_0) - L(h'_{S^0}, s_0)) + (L(h'_{S^0}, s_0) - L(h^\gamma_{S^0}, s_0)) \leq \mathcal{Q} + \omega.$$

Therefore, we have:

$$\mathop{\mathbb{E}}_{S}[\eta(S, S)] \leq \gamma + \mathop{\mathbb{E}}_{S} \mathop{\mathbb{E}}_{s_0}[L(h^\gamma_S, s_0) - L(h^\gamma_{S^0}, s_0)]] \leq \gamma + \mathcal{Q} + \omega$$

but since γ can be chosen to be any positive number, this implies that

$$\mathop{\mathbb{E}}_{S}[\eta(S, S)] \leq \mathcal{Q} + \omega.$$

9.11 Completing the Proof

Putting together the result from McDiarmid's inequality with our two upper bounds for $\mathbb{E}_S[\eta(S,S)]$, along with the fact that $\forall h \in H_S$:

$$\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \leq \sup_{h \in H_S} \mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \equiv \eta(S,S),$$

we have

$$\mathcal{L}(h) - \hat{\mathcal{L}}_{\mathcal{S}}(h) \ \leq \ \eta(S,S) \ \leq \ \mathbb{E}_S[\eta(S,S)] + \epsilon$$

but we have shown that both

$$\mathbb{E}_S[\eta(S,S)] \leq \ 2\mathcal{C}$$

and

$$\mathbb{E}_S[\eta(S,S)] \leq \mathcal{Q} + \omega.$$

This gives us our generalization bound at last. It says that, if we are choosing the sample S at random, then with probability at least $1-\delta$:

$$\mathcal{L}(h) \leq \hat{\mathcal{L}}_{\mathcal{S}}(h) + \min\{2\mathcal{C},\ \mathcal{Q} + \omega\} + \frac{\mathcal{M} + 2m\mathcal{Q}}{\sqrt{2}} \sqrt{\frac{\ln \frac{1}{\delta}}{m}}.$$

9.12 Data Independent Hypothesis Set Case

Consider the case where $H_S = H$, that is, the hypothesis set is independent of the sample, which is precisely the setting where the standard Rademacher Complexity bound applies. This data independence of the hypothesis set means that $H_{S,R} = H$ and so

$$\mathcal{C} \equiv \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{S,R} \left[\frac{1}{m} \sup_{h \in H_{S,R}} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right] = \mathop{\mathbb{E}}_{\sigma} \mathop{\mathbb{E}}_{R} \left[\frac{1}{m} \sup_{h \in H} \sum_{i=1}^{m} \sigma_i L(h, r_i) \right]$$

which is just the usual Rademacher Complexity. Furthermore, in this case we have the stability constant $\mathcal{Q} = 0$, since the hypothesis sets H_S and $H_{S'}$ are both just H, so obviously for every hypothesis function $h \in H_S$ there is a function $h' \in H_{S'}$ identical to it. We also can of course use the fact that:

$$\min\{2\mathcal{C},\ \mathcal{Q} + \omega\} \leq 2\mathcal{C}.$$

Hence, in the case where the hypothesis set is data independent, our generalization bound directly implies that with probability at least $1 - \delta$,

$$\mathcal{L}(h) \leq \hat{\mathcal{L}}_{\mathcal{S}}(h) + 2\mathop{\mathbb{E}}_{\sigma}\mathop{\mathbb{E}}_{R}\left[\frac{1}{m}\sup_{h\in H}\sum_{i=1}^{m}\sigma_i L(h, r_i)\right] + \frac{\mathcal{M}}{\sqrt{2}}\sqrt{\frac{\ln\frac{1}{\delta}}{m}}.$$

which is precisely the standard Rademacher Complexity generalization bound for a fixed hypothesis set H and a bounded loss function! Hence, our proposed D-Rademacher Complexity bound really does generalize the typical bound.

9.13 Single Hypothesis Case

Consider now the case where $H_S = \{h_S\}$, that is, the hypothesis set always consists of just a single (data dependent) hypothesis function. Then, we have:

$$H_{S,R} \equiv \bigcup_{\substack{A\subset S\bigcup R\\ |A|=m}} H_A = \bigcup_{\substack{A\subset S\bigcup R\\ |A|=m}} \{h_A\}.$$

Therefore, in this case $H_{S,R}$ is a set of size $\binom{2m}{m} \approx \frac{1}{\sqrt{\pi}}\frac{4^m}{\sqrt{m}}$. While finite, this set is extremely large for even a modest sample size (e.g. it is on the order of 10^{11} for just $m = 20$). Here, the stability constant $\mathcal{Q}$ takes on the interpretation that when S and S' are any two valid

samples that differ by only one point,

$$|L(h_S, s) - L(h_{S'}, s)| \leq \mathcal{Q} \quad \forall s \in \mathcal{X}.$$

If $\mathcal{Q}$ falls like $\frac{1}{m}$ then this is equivalent to to saying that the algorithm represented by the data dependent hypothesis h_S is "stable", in the sense of Uniform Stability (Bousquet and Elisseeff, 2002). Hence, Hypothesis Set Stability is a generalization of the usual notion of Uniform Stability, extending it to allow a data dependent set of hypotheses, rather than just a single data dependent hypothesis.

What is more, we observe that when the hypothesis set consists of just a single function, then $\omega = 0$. We then write

$$\min\{2\mathcal{C},\ \mathcal{Q} + \omega\} \leq \mathcal{Q} + \omega\ = \mathcal{Q}$$

which gives us the following special case of our generalization bound. With probability at least $1 - \delta$, we have:

$$\mathcal{L}(h) \leq \hat{\mathcal{L}}_{\mathcal{S}}(h) + \mathcal{Q} + \frac{\mathcal{M} + 2m\mathcal{Q}}{\sqrt{2}}\sqrt{\frac{\ln \frac{1}{\delta}}{m}}$$

which precisely matches the typical Uniform Stability bound[2].

9.14 Conclusion

In this chapter we have investigated the situation where the set of hypothesis functions that a machine learning algorithm explores is allowed to vary based on the training data. We have introduced the D-Rademacher Complexity, a generalization of the ordinary Rademacher

[2]Note that sometimes the $\mathcal{Q}$ shown here will have a 2 in front in both places it appears in the bound, which has to do with thinking in terms of dropping a point from the sample rather than changing a point in the sample as we do here.

Complexity, to measure the complexity of such a data dependent hypothesis set. We have also introduced the Hypothesis Set Stability, a generalization of the ordinary Uniform Stability, to measure the stability of a data dependent hypothesis set with respect to changing one point in the sample. Finally, we have introduced the Error Radius, which measures the size of a data dependent hypothesis set in terms of how different the errors of hypotheses it contains can be.

These definitions allow us to construct a generalization bound that applies to data dependent hypothesis sets. This bound coincides with the standard Rademacher Complexity bound when the hypothesis set is fixed, and coincides with the standard stability bound when the hypothesis set always consists of just one function. By generalizing Rademacher Complexity and Uniform Stability, and then fusing them together to form an overarching theory, we enable a wider range of perspectives from which to study the generalization ability of machine learning algorithms.

Part V

Conclusion

Figure 9.1: The image above summarizes the answers that this thesis gives to the questions that it investigates in each chapter.

We have analyzed a wide variety of problems that occur when applying prediction algorithms to data, and when analyzing the theoretical performance of these algorithms. For data points that vary in quality, we have proposed an approach for designing functions that weight data appropriately. For handling outliers we have proposed an algorithm based on point re-weighting to make Ridge Regression robust, and an algorithm for clipping outliers in univariate data based on a criteria that will have almost no effect on normally distributed data. To incorporate prior information into regression while preventing overfitting, we have proposed a generalization to the standard Ridge Regression regularization penalty. To handle situations where linear or kernelized regression algorithms spend too long on training, and

so overfit or waste precious time converging past the point of usefulness, we have proposed a cross-validated early stopping based approach with desirable properties. For unbounded loss functions in regression problems, we have given analyses of generalization error using relative deviation bounds that address or correct previous gaps in the literature, as well as derived a necessary theorem about the binomial distribution. Finally, for machine learning algorithms that have hypothesis sets that depend on the training data, we have generalized and combined Rademacher Complexity and Uniform Stability so that they can be applied in that context.

It is our hope that the solutions proposed in this thesis will significantly improve predictions in real-world applications, as well as our theoretical understanding of prediction algorithms.

Bibliography

Anthony, M. and Bartlett, P. L. (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Anthony, M. and Shawe-Taylor, J. (1993). A result of Vapnik with applications. *Discrete Applied Mathematics*, 47:207 – 217.

Azuma, K. (1967). Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19(3):357–367.

Bartlett, P. L., Boucheron, S., and Lugosi, G. (2002a). Model selection and error estimation. *Machine Learning*, 48:85–113.

Bartlett, P. L., Bousquet, O., and Mendelson, S. (2002b). Localized Rademacher complexities. In *COLT*, volume 2375, pages 79–97. Springer-Verlag.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3.

Bartlett, P. L. and Mendelson, S. (2003). Rademacher and gaussian complexities: Risk bounds and structural results. *The Journal of Machine Learning Research*, 3:463–482.

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2007). Analysis of representations for domain adaptation. *NIPS*.

Ben-Gal, I. (2005). *Data Mining and Knowledge Discovery Handbook*.

Bickel, S., Brückner, M., and Scheffer, T. (2007). Discriminative learning for differing training and test distributions. In *ICML*, pages 81–88.

Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Wortman, J. (2008). Learning bounds for domain adaptation. *NIPS 2007*.

Blunck, H. and Vahrenhold, J. (2006). In-place randomized slope selection. In *Algorithms and Complexity*, pages 30–41. Springer.

Boucheron, S., Bousquet, O., and Lugosi, G. (2005). Theory of classification: a survey of recent advances. *ESAIM: Probability and Statistics*, 9:323–375.

Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*.

Brandt, M. J. (2012). Nasty data can still be real: A reply to ullrich and schluter. *Psychological Science*, 23(7):826–827.

Chatterjee, S. and Hadi, A. (1986). Influential observations, high leverage points, and outliers in linear regression. *Institute of Mathematical Statistics*.

Cortes, C., Mansour, Y., and Mohri, M. (2010a). Learning bounds for importance weighting. In *Advances in Neural Information Processing Systems (NIPS 2010)*, Vancouver, Canada. MIT Press.

Cortes, C., Mansour, Y., and Mohri, M. (2010b). Learning bounds for importance weighting. In *NIPS*, pages 442–450, Vancouver, Canada. MIT Press.

Cortes, C. and Mohri, M. (2013). Domain adaptation and sample bias correction theory and algorithm for regression. *Theoretical Computer Science*, 9474.

Cortes, C., Mohri, M., Riley, M., and Rostamizadeh, A. (2008). Sample selection bias correction theory. In *ALT*.

Dasgupta, S. and Long, P. M. (2003). Boosting with diverse base classifiers. In *COLT*.

Daumé III, H. and Marcu, D. (2006). Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126.

Dean, R. B. and Dixon, W. (1951). Simplified statistics for small numbers of observations. *Analytical Chemistry*, 23(4):636–638.

Draper, N. R. and Smith, H. (1981). *Applied regression analysis*. Wiley.

Dudík, M., Schapire, R. E., and Phillips, S. J. (2006). Correcting sample selection bias in maximum entropy density estimation. In *NIPS*.

Dudley, R. M. (1984). A course on empirical processes. *Lecture Notes in Mathematics*, 1097:2 – 142.

Dudley, R. M. (1987). Universal Donsker classes and metric entropy. *Annals of Probability*, 14(4):1306 – 1326.

Ein-Dor, P. and Feldmesser, J. (1987). Computer hardware data set.

Ellis, S. (1998). Instability of least squares, least absolute deviation and least median of squares linear regression. *Statistical Science*.

Gardner, W. A. (1984). Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Processing*, 6(2):113–133.

Golub, G. H., Heath, M., and Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223.

Gordon, G. and Tibshirani, R. (2012). Gradient descent revisited.

Greenberg, S. and Mohri, M. (2013). Tight lower bound on the probability of a binomial exceeding its expectation. *Technical Report 2013-957, Courant Institute, New York, New York*.

Greenberg, S., Mohri, M., and Tabak, E. (2016a). *Parameterized Weight Functions: making bad data count less*.

Greenberg, S., Mohri, M., and Tabak, E. (2016b). *Stabilized Ridge Regression*.

Grubbs, F. E. (1950). Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics*, pages 27–58.

Haussler, D. (1992). Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150.

Hodge, V. J. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Huang, J., Smola, A. J., Gretton, A., Borgwardt, K. M., and Schölkopf, B. (2006). Correcting sample selection bias by unlabeled data. In *NIPS*, volume 19, pages 601–608.

Huber, P. J. et al. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101.

Iglewicz, B. and Hoaglin, D. C. (1993). *How to detect and handle outliers*, volume 16. Asq Press.

Jaeger, S. A. (2005). Generalization bounds and complexities based on sparsity and clustering for convex combinations of functions from random classes. *Journal of Machine Learning Research*, 6:307–340.

Jiang, J. and Zhai, C. (2007). Instance Weighting for Domain Adaptation in NLP. In *ACL*.

Johnson, N., Kemp, A., and Kotz, S. (2005). *Univariate Discrete Distributions*. Number v. 3 in Wiley series in probability and Statistics. Wiley & Sons.

Johnson, N., Kotz, S., and Balakrishnan, N. (1995). *Continuous univariate distributions*. Number v. 2 in Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley & Sons.

Knorr, E., Ng, R., and Tucakov, V. (2000a). Distance-based outliers: algorithms and applications. *The VLDB Journal*.

Knorr, E. M., Ng, R. T., and Tucakov, V. (2000b). Distance-based outliers: algorithms and applications. *The VLDB Journal?The International Journal on Very Large Data Bases*, 8(3-4):237–253.

Koltchinskii, V. (2006). Local rademacher complexities and oracle inequalities in risk minimization. *Annals of Statistics*, 34(6).

Koltchinskii, V. and Panchenko, D. (2000a). Rademacher processes and bounding the risk of function learning. In *High Dimensional Probability II*, pages 443–459. Birkhäuser.

Koltchinskii, V. and Panchenko, D. (2000b). Rademacher processes and bounding the risk of function learning. In *High dimensional probability II*, pages 443–457. Springer.

Koltchinskii, V. and Panchenko, D. (2002). Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30.

Laurikkala, J., Juhola, M., Kentala, E., Lavrac, N., Miksch, S., and Kavsek, B. (2000). Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, pages 20–24. Citeseer.

Lesch, S. M. and Jeske, D. R. (2009). Some suggestions for teaching about normal approximations to poisson and binomial distribution functions. *The American Statistician*, 63(3):274–277.

Lichman, M. (2013). UCI machine learning repository.

Mansour, Y., Mohri, M., and Rostamizadeh, A. (2009). Domain adaptation: Learning bounds and algorithms. In *COLT*.

McDiarmid, C. (1989). On the method of bounded differences. *Surveys in Combinatorics*, 141(1):148–188.

Meir, R. and Zhang, T. (2003). Generalization Error Bounds for Bayesian Mixture Algorithms. *Journal of Machine Learning Research*, 4:839–860.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

Niu, Z., Shi, S., Sun, J., and He, X. (2011). A survey of outlier detection methodologies and their applications. In *Artificial intelligence and computational intelligence*, pages 380–387. Springer.

Onur, T. and Cetin, M. (2011). The comparing of s-estimator and m-estimators in linear regression. *Gazi University Journal of Science*, 24(4):747–752.

Pollard, D. (1984). *Convergence of Stochastic Processess*. Springer, New York.

Pollard, D. (1989). Asymptotics via empirical processes. *Statistical Science*, 4(4):341 – 366.

Prechelt, L. (1998). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer.

Rigollet, P. and Tong, X. (2011). Neyman-pearson classification, convexity and stochastic constraints. *Journal of Machine Learning Research*, 12:2831–2855.

Rousseeuw, P. and Yohai, V. (1984). Robust regression by means of s-estimators. In *Robust and nonlinear time series analysis*, pages 256–272. Springer.

Rousseeuw, P. J. and Croux, C. (1993). Alternatives to the median absolute deviation. *Journal of the American Statistical association*, 88(424):1273–1283.

Sauer, N. (1972). On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147.

Schlkopf, B., Herbrich, R., and Smola, A. (2001). A generalized representer theorem. *COLT/EuroCOLT*.

Stevens, J. (1984). Outliers and influential data points in regression analysis. *Psychological Bulletin*, 95(2):334–344.

Sugiyama, M., Nakajima, S., Kashima, H., von Bünau, P., and Kawanabe, M. (2008). Direct importance estimation with model selection and its application to covariate shift adaptation. In *NIPS*.

Talagrand, M. (1994). Sharper bounds for gaussian and empirical processes. *Annals of Probability*, 22(1):28–76.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley & Sons.

Vapnik, V. N. (2006a). *Estimation of Dependences Based on Empirical Data*. Springer-Verlag.

Vapnik, V. N. (2006b). *Estimation of Dependences Based on Empirical Data, second edition.* Springer, Berlin.

Vapnik, V. N. and Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264.

Weigend, A. S. and Rumelhart, D. E. (1992). *Generalization through minimal networks with application to forecasting.* Defense Technical Information Center.