Regression
Lecture notes for CSCI 5454

Sears Merritt & Abigail Jacobs

April 25, 2012

1 Introduction

Regression analysis is the study of the relationship between independent (**features**) and dependent (**targets**) variables[1]. Usually our goal is to predict the unknown targets of some features, given that we have a sample of features and their corresponding targets (**training set**), although sometimes we may simply be interested in measuring the strength of the relationship between features and targets. To do this, we would like to estimate a function (**hypothesis function**) that maps the features in our training set to the corresponding targets. As shown in Figure 2, this is done by using the training set and a learning algorithm to train a hypothesis function. Then using this function, we may map new features whose actual targets are unknown to their estimated targets.

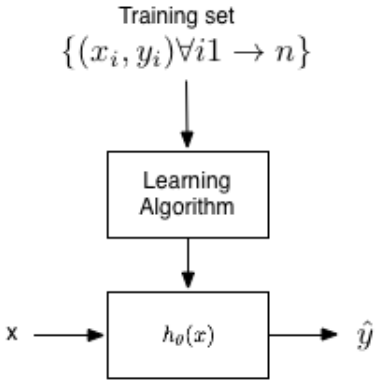

Figure 1: The architecture of performing linear regression

Examples of regression include predicting prices given a set of observations, such as home prices given their location, square footage, number of rooms, etc. or the price of a stock given a set of financial statements, coded geopolitical events, commodity prices, etc. Figure 2 shows the linear regression of number of bedrooms onto median home prices in the city of Boston, Massachusetts[2].

In this lecture we will discuss a type of regression where the relationship between features and targets is assumed to be linear. First we will introduce notation used to mathematically describe

[1] In regression, the relationship between features and targets is continuous. When the relationship is discrete, we call this analysis classification.

[2] The data set is taken from the StatLib library which is maintained at Carnegie Mellon University. A copy of the data can be found at `http://archive.ics.uci.edu/ml/datasets/Housing`.

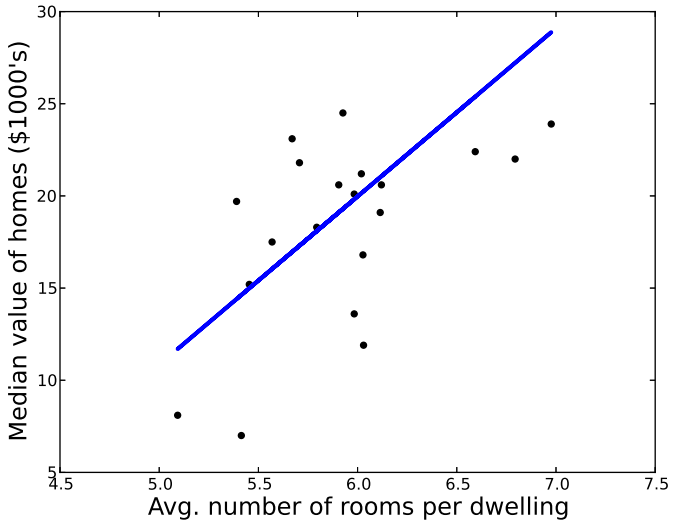

Figure 2: Simple regression of average number of rooms in a home onto the medium value of the home.

the concepts throughout the rest of this lecture. Following this we will introduce linear regression where we will analyze its mathematical foundations as well as efficient algorithms used to optimize its cost function. Then we will discuss regularization, which allows for the relaxation of some of the assumptions in ordinary least squares (OLS).

2 Notation

We adopt a notation set defined by Professor Andrew Ng in his second lecture on machine learning [6]. A training example is a tuple of features and targets denoted as

$$(x^{(i)}, y^{(i)}). \tag{1}$$

The training set is the set of training examples, denoted as

$$\{(x^{(i)}, y^{(i)}); \forall i = 1 \rightarrow m\}, \tag{2}$$

where i is the i^{th} training example in the set. We denote the hypothesis function, $h_\theta(x)$ as

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + ... + \theta_m x_m = \sum_{i=0}^{m} \theta_i x_i \tag{3}$$

where the θ_i's are the parameters we are interested in estimating such that we obtain a "good" mapping of features to targets. We will define "good" in the proceeding section. In matrix form h_θ is equivalent to

$$h_\theta(x) = \theta^T x. \tag{4}$$

3 Linear regression

Linear regression was invented and used heavily prior to the availability of computers because it was a simple model with nice analytical properties whose results could be calculated by hand. Even though we live in an age where the price of computation is low, there are still good reasons to study linear regression. First, the model is simple, and thus there are efficient methods for computation. Second, in some cases a linear model often times outperforms more complex models due to small or noisy data sets.

Given a training set, we would like to estimate a "good" linear hypothesis function that maps features to targets. With a few assumptions that we will discuss later, we can characterize the hypothesis function using the cost function

$$J(\theta) = \frac{1}{2}\sum_{i=0}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2. \tag{5}$$

$J(\theta)$ describes the sum of squared differences between the estimate produced by $h_\theta(x^{(i)})$ and the observed value $y^{(i)}$. It is sometimes called the sum of squared residuals. Intuitively, a good hypothesis function is one that produces values that are as close to the observed as possible, which mathematically minimizes $J(\theta)$, which can be though of as a measure of the error in our model. Therefore, we would like to find the set of parameters θ that minimizes $J(\theta)$. This is approach is called ordinary least squares (OLS). Before we move onto to describing and analyzing an algorithm that finds the optimal θ, let us discuss the probabilistic assumptions behind this function and then using these assumptions derive it.

First, let us assume that our training set comes from the function

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \tag{6}$$

where ϵ is an error term used to explain unmodeled effects in our data. Further, let our data be IID and distributed as a Gaussian random variable with $\mu = 0$ and σ^2, $\mathcal{N}(0, \sigma^2)$[3] thus:

$$\Pr(y^{(i)}|x^{(i)}; \theta) = \mathcal{N}(0, \sigma^2) \tag{7}$$

$$\Pr(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \tag{8}$$

Using these assumptions[4] we can write the likelihood function for θ as

$$\mathcal{L}(\theta|y, x) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \tag{9}$$

In order to find the set of parameters θ which maximizes the probability of observing this sequence of data we maximize the likelihood function (**maximum likelihood**). Analytically and computationally, it is easier to maximize this function if we take the log.

$$\ell(\theta|y, x) = \log\left(\prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\right) \tag{10}$$

[3] If the error terms are independent and homoskedastic (constant variance) and the features are uncorrelated (lack of multicolinearity) then OLS produces the optimal linear model of the data by the Gauss-Markov theorem.

[4] If the error terms are not Gaussian, but instead come from a distribution in the exponential family, we can still use least squares via generalized linear models (GLMs).

Simplifying the equation above yields:

$$\ell(\theta|y,x) = \sum_{i=1}^{m} -\log(\sqrt{2\pi\sigma}) + \sum_{i=1}^{m} \log\left(\exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\right) \tag{11}$$

$$\ell(\theta|y,x) = \sum_{i=1}^{m} -\log(\sqrt{2\pi\sigma}) + \sum_{i=1}^{m} \left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \tag{12}$$

$$\ell(\theta|y,x) = -m\log(\sqrt{2\pi\sigma}) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2 \tag{13}$$

Thus, when we maximize this function with respect to θ we have the least squares cost function introduced in Equation 5.

In order to find the θ that produces the best fit, or minimizes the sum of squared errors between our hypothesis function and our known targets we can simply take the derivative of our function, set it equal to 0 and solve for θ[5]. In order to reduce the amount of algebra and complexity in this derivation, let us assume there is only a single θ and one training example.

$$J(\theta) = \frac{1}{2}(h_\theta(x) - y)^2 = \frac{1}{2}(\theta x - y)^2 \tag{14}$$

$$\frac{d}{d\theta}J(\theta) = \frac{d}{d\theta}\frac{1}{2}(\theta x - y)^2 \tag{15}$$

$$\frac{d}{d\theta}J(\theta) = (\theta x - y)\cdot\frac{d}{d\theta}(\theta x - y) \tag{16}$$

$$\frac{d}{d\theta}J(\theta) = (\theta x - y)x \tag{17}$$

$$(\theta x - y)x = 0 \tag{18}$$

$$\theta = y(x)^{-1} \tag{19}$$

Equivalently in matrix form we have[6]:

$$\theta(X^T X) = X^T \mathbf{y} \tag{20}$$

$$\theta = X^T \mathbf{y}(X^T X)^{-1} \tag{21}$$

Equation 21 is known as the normal equations and is the analytical solution to θ. We can use this equation to final the optimal θ that minimizes our original cost function. There are alternatives to the normal equations, such as the gradient descent algorithm, which has already been discussed in class.

For alternative machine learning oriented explanations of OLS and regression in general, consider reading the following texts [2], [4]. For a succinct review of relevant topics on probability and statistics, see [9].

How well does the OLS estimate fit the data (**goodness of fit**)? Often times we use the coefficient of determination (R^2) to measure the goodness of fit. The value indicates how much variability is explained by the model and how much is unexplained, and is the ratio of explained variance to total variance. Therefore $0 \leq R^2 \leq 1$, with a larger value indicating a better fit.

$$R^2 = \frac{\sum_{i=1}^{m}(y - \hat{y})^2}{\sum_{i=1}^{m}(y - \bar{y})^2} \tag{22}$$

[5]This works because our cost function is convex and thus the local optimum at 0 is equivalent to the global optimum. For a brief introduction to convex optimization, refer to `http://en.wikipedia.org/wiki/Convex_optimization`

[6]For a full derivation, see [6].

3.1 Run time and space requirements

The run time of OLS is a function of the complexity of matrix operations involved in analyzing the normal equations. A singular value decomposition method runs in polynomial time, $O(np^2)$ where p is the number of features and n is the size of the training set, assuming $n \geq p$[7]. The algorithm requires constant space $O(1)$, equal to the size of the vector of parameters θ.

3.2 Alternative minimization methods

Gradient descent is another algorithm that can be used to minimize $J(\theta)$ and is often times used when the form of the training set of not . As we saw in our simulated annealing lecture, gradient descent works by iterating through the solution landscape and selecting the next point that has the steepest descent, or negative gradient. Gradient descent is guaranteed to find a local minimum. If the function is convex, then it is guaranteed to find the global optimum. As we saw in the optimization lecture, gradient descent works as follows:

$$\theta_i := \theta_{i-1} - \alpha \frac{d}{d\theta} J(\theta) \tag{23}$$

where α is the cooling parameter, or learning rate and controls how large the steps are in the algorithm. The parameter controls the convergence time of the algorithm. A value too large may skip over the optimal solution, while a value too small may make the algorithm take excessively long to converge.

There are a few variants of gradient descent used in linear regression: batch, online, and stochastic. Batch gradient descent works well when there is a medium to large training set while online gradient descent works well when the training set is small and is growing as a function of time. Batch gradient descent works by iterating through each training example and computing θ according to update rule. The online variant runs the update rule each time a new training example is entered into the system. Finally, when the training set is very large, stochastic gradient is used. It works by performing the computation one training example at a time but halts once a convergence criteria has been satisfied. Often times this criteria is satisfied long before the entire training set has been analyzed.

4 Regularization

In this class, we have been focusing on solving problems efficiently by taking advantage of the structure of the problem. Here, we manipulate the structure of the problem in two ways: the first, using math, to turn polynomial time algorithms into linear time analytic solutions; the second, we manipulate our goal to bring us more quickly to types of solutions that we are more interested in.

By showing that the maximum likelihood solution is equivalent to OLS, we have already seen an example of using analytic results to find the line of best fit. In addition, that also shows us how we find the answer within a particular framework, maximum likelihood, where we have assumed the *form* of the model, but we are searching for the parameter values that are most likely to have generated the data.

However, there may be reasons to distrust the OLS model, or rather, we may have different goals. Then here we are searching a solution space for a "good" solution, where we might want to consider other goals past the maximum likelihood solution. Our data may have qualities that recall the "rugged landscape" we explored of the Ising model using simulated annealing: that is, there

[7] A Python implementation can be found here: `http://scikit-learn.org/stable/modules/linear_model.html`

may be many local optima describing models that predict points similar to the data we train on. By manipulating the score function and the shape of the solution landscape, we can make certain solutions easier to find to fit more different goals. While the least squares landscape is convex and thus theoretically easy to traverse, qualities in our data $\mathbf{X}$ may cause the behavior of the OLS to break if our assumptions are violated.

A basic way to confront these issues, assuming the general linear form of the model, is regularization by *adding an additional penalty term to the original score function that introduces new information.* By penalizing our score in addition to minimizing squared errors, we may introduce certain restrictions by introducing other information to the score function, that may then induce certain classes of solutions or emphasize solutions holding other properties. This idea is called **regularization**.

4.1 Changing our goals for the "best" solution: Looking past OLS

The sections that follow confront some of the issues that may lead us to search for a slightly different solution than the OLS/maximum likelihood solution. To address these issues of over-fitting, ill-posed problems, noisy data, and ideas about parsimony or structure, we can change the structure of the score function. This allows us to smooth the solution space and to make the desired types of models have higher scores than under the maximum likelihood framework alone. In different settings, we can think of this as a way to manipulate the solution space to avoid ugliness from the data; alternatively, we can see it as a way to induce a Bayesian approach (where we have some knowledge or preference for certain results based on prior information); to apply Occam's razor (to induce solutions with the most parsimonious models[8]); and to directly avoid over-fitting, which is especially desirable when we have a small amount of data, a large number of input variables, or both.

4.1.1 Avoiding over-fitting

A basic motivation for moving away from simply the maximum likelihood solution would be to avoid over-fitting. If we train a model that effectively "memorizes" our data, we have accomplished the goal of finding a model that reproduces our data with extremely high probability; on the other hand, it can be reasonable to assume that this model may not provide a very good prediction of future points, whether interpolating between points, extrapolating to the future, or resampling from the model.

Additionally, this amount of memorization of the data means that any slight adjustment - a new data point from the same model, a small amount of noise, etc. - can dramatically change the solution, which means that the solution space is not as smooth as we would like. Ugly solution spaces can be used to define ill-posed problems, where the solution is either nonexistent, not unique, or unreasonably sensitive to the data.[9] We are likely to stumble into this scenario with a lot of noise, not enough data, and/or a large feature set, particularly if many of those features are irrelevant to the target.

Similarly, consider trying to fit the linear model where the features are defined as a polynomial, i.e., where $\mathbf{y} = \theta^T \mathbf{X}$, where each feature $x_i = x_1^i$. Then the model we are trying to fit is given by the m-th degree polynomial: $y = \theta_m x^m + \theta_{m-1} x^{m-1} + \ldots + \theta_1 x + \theta_0$. As we increase m from 0

[8] For more on Occam's razor, see, e.g., `http://en.wikipedia.org/wiki/Occam%27s_razor#Science_and_the_scientific_method`

[9] The definition of ill-posed problems follows directly from the definition of a well-posed problem, taken from `http://en.wikipedia.org/wiki/Well-posed_problem`, last accessed April 23, 2012

to the size of the data n, we will move from models that roughly summarize the data ($m = 0$ will find the constant average of the data; $m = 1$ will be the basic linear regression; $m = 2$ quadratic, etc.) to models that explicitly describe all of the data ($m = n$). It seems unlikely that such a specific model to the data - while it will produce the data with extremely high probability - will be representative of either the points between the data (**interpolation**) or outside of that range (**extrapolation**).

As we increase the degree of the polynomial, the weights θ will begin to fluctuate and explode in size, as the requirements to create a *smooth* curve that captures all of the data will be highly sensitive to this increase. Of course, it is possible that we could fit the true linear model that the data was generated from, but this is only the OLS/maximum likelihood solution we would land upon if there was zero noise. Instead we will continue to fit higher degree polynomials that move towards exactly explaining the data, with extreme fluctuations in θ to preserve smoothness. Then new training data or small amounts of noise will dramatically change the θ found, and as m approaches n, the size of the θ_i's will increase in size. Then we will want some way to constrain θ to stay in a "reasonable" range of models.

4.1.2 Handling outliers

For the same reasons that we want to avoid over-fitting of the data, we also want to avoid letting a few points dramatically change the model we fit. Just as over-fitting led us to models where some noise, or excluding some data, could change the solution found significantly, outliers are points that specifically, if excluded, would change the solution significantly.

OLS can also make us unreasonably sensitive to these outliers, where a few points - possibly drawn from some other model, or containing some measurement error - are not informative to or representative of the relationship between the features and the target. These points can strongly influence the OLS solution, as the true model that generated the rest of the points will have large least-squares error from these points. As a result, a model that "compromises" between the true model and the unreasonable outliers will be found by the OLS method. This means a regularization method that could de-incentivize reducing the least squares of the outliers would find a solution more similar to the true model, while better ignoring the outliers.[10] I show an example of this in Section 5.2.

4.1.3 Identifying related features

In real-world applications of regression, it is extremely likely that we use features that are in some way correlated. This directly violates our independence assumption in developing OLS; in practice, this changes the shape of the equal-likelihood solutions in the solution space. That is, the solutions sets of θ that have possibly very different values of θ_i, θ_j for correlated features x_i, x_j may have equal likelihood. Then small amounts of noise or more data might "bump" us to a far away solution: then we want to be able to focus on a stable solution, as this might not be well-posed. I discuss this in more detail in Section 5.1: this was one of the motivations for developing ridge regression.

There are many, many examples of where this might appear in practice. One motivating example could be to predict test scores, in particular, the SAT test taken by many high schoolers. If we want to predict SAT scores, we might look at the following model:

$$SAT\ score = \theta_2(family\ income) + \theta_1(zip\ code) + \theta_0 \tag{24}$$

[10] There are reasons to not want to ignore outliers, but they are beyond the scope of this course.

However, we know that income and location will be correlated (it is not a stretch to say that income is not evenly geographically distributed). Then both might add some explanatory power to predicting SAT scores in a similar way; then we can move between values of weights θ_1 and θ_2 that provide similar explanatory power of SAT scores, by "exchanging" weight in one to the other.

Most extremely, if there are two features that are the same but for a constant factor, or if a feature is a linear combination of the other one, we have the problem of multicollinearity, or just collinearity. This could happen if, for example, we have $income = \theta_2(\text{height in inches}) + \theta_1(\text{height in cm}) + \theta_0$ or $y = \theta_3(x_1 + x_2) + \theta_2 x_2 + \theta_1 x_1 + \theta_0$. While this is obvious in these toy examples, in large or automatically generated feature sets or features of unknown origin, we may end up with features that are combined in this way. In this case, OLS will not fail in a way that is necessarily obvious. Then we would want a way to identify and exclude redundant features, or at least come to solutions that do this for us. We will see an example of a regularization term that does this in Section 6.3.

4.1.4 Sparsity from noisy data

Similarly, the presence of noise in the data (an exceptionally reasonable assumption) can make some input variables seem more meaningful than they might actually be to the model: if we want to eliminate some input variables, we might no longer be able to assume that the OLS solution will find their coefficients to be exactly zero. Then we must use some nonzero threshold to eliminate variables that were found to have some input. In science, we may want to find the simplest models with high explanatory power. To do so, we would want the most parsimonious model such that we make claims about the smallest number of features. Alternatively, we may have some outside additional information about what the coefficients "should" be, such as a known relationship (or non-relationship) between two input variables or inputs and output variables.[11]

For example, if we wanted to predict some disease outcome using gene sequence data, we might begin by using the entire set of all genes as possible predictors: then we create a feature for each gene. However, it might be that only *in combination* do some genes have some correlation with the disease outcome; then we might want to include all pairs (or triplets, or sets of k, etc.) of genes to try and predict. Then not only is it likely that we have more features than training data points, it is highly likely that due to noise alone that we will recognize some irrelevant features as relevant. In addition, biologically, it is unlikely that we would have so many contributors to such a biological phenomenon (disease) that all were necessary or useful for prediction. The first problem is known as the problem where $p > n$ or "large p, small n", i.e., the number of features outnumbers the data; and despite the "age of big data" this is still an extremely common problem and active area of research in statistics. In all of those cases, we hope that we would be able to fit the model that appropriately assigns θ_j to zero for the irrelevant features.

4.1.5 Domain knowledge and heuristics; looking for a "reasonable model"

The disease outcomes example has already motivated some encouragement for incorporating domain knowledge and searching for solutions that include as few features as is necessary (according to some threshold). It can be the sparsity induces the most parsimonious model, and the way we find that is by manipulating our solution space to better support finding the desired type of solution. Then we would want to be able to incorporate some kind of restriction on the types of solutions we considered, such that we are able to find, or more likely to find, solutions of the desired form.

[11] Note that we are already imposing a specific relationship structure by using the linear model.

As we have seen, this can arise through issues of noise, over-fitting, outliers, etc., but we can be driven by scientific or engineering goals, by Occam's razor, a Bayesian perspective, or the computational or practical need to want to exclude certain features (that are, e.g., expensive to measure, collect, or access). Similarly, we might prefer general models with high predictive power in certain domains, where we are willing to accept errors in a certain direction, or tolerating certain errors as "cheaper" than including expensive features. All of this is extremely relevant to data analysis in practical applications, and this has become ever more so as computation has become cheaper. Fortunately, we can address many of these issues by manipulating our standard OLS/maximum likelihood score function to manipulate shape of the solution space. This can help us find the types of solutions that address these issues while still remaining "close" to our original maximum likelihood solution. A basic version of such regularization is known as ridge regression.

5 Ridge regression

Previously, we saw that we could find the best linear unbiased estimator of the data using OLS, where we find the least-squares solution. There we have our cost function, or score function, be the sum of squared residuals generated by the (linear) model:[12]

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \tag{25}$$

To restrict the solution space via regularization, we can introduce a simple penalty based on the values of the parameters θ.

Adding a simple penalty based on the size of the weights, we can add a quadratic regularization term, which is more widely known as ridge regression in statistics:

$$\begin{aligned} J(\theta) &= \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \frac{\lambda}{2}||\theta||^2 & (26) \\ &= \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \frac{\lambda}{2}\theta^T\theta & (27) \end{aligned}$$

This has been reintroduced to the literature several times over, originally as Tikhonov regularization and popularly brought into statistics by Hoerl and Kennard (1970) in [5]. [13]

Note that in general, we will want to exclude the constant $\theta_0 = w_0$, otherwise we can dramatically change the solution space by simple translation through space of the data.[4] Consider two sets of data generated by the models $y = 50x + 1000$ and $y = 50x + 1$. For any order polynomial that is at least linear, we would hope to recover parameters close to $\theta_1 = 50$ and $\theta_0 = 1000$ and $\theta_0 = 1$, respectively. If we are fitting a linear model with a penalty on the size of θ_0, then the amount of error we tolerate, and the set of values $\hat{\theta}$ we find, will change based on this translation. To make results comparable, we would want to work with normalized data, i.e., transform the data to have zero mean (and possibly unit variance).

Here, the penalty term can be used to trace contours, or ridges, in the solution space of the (unregularized) contour function, such that ridge regression finds the ridge (using $\theta^T\theta$) that minimizes the square error.

[12] Note that the factor of $\frac{1}{2}$ does not change the structure of the solution space or the solutions found.

A note on notation: in general, we use θ to represent the true values of the parameters and $\hat{\theta}$ to be our estimates of those parameters. However, in the score functions, we will often use the value "θ" too hold for any value of $\hat{\theta}$. This is as opposed to their use in expected value functions, where we find the expected value of an estimate $\hat{\theta}$ to be related to the true value θ. In addition, this notation is related to the idea that we **assume** the linear form of the model, such that the data was truly generated by some $\mathbf{X}\theta$ and some noise, thus there is some notion of a true value of θ.

[13] `http://en.wikipedia.org/wiki/Tikhonov_regularization`, last accessed April 23, 2012

5.1 The penalty in ridge regression: motivation and alternate forms

As we will see in the general form of the penalty term, there are simpler penalties that can be enforced to limit the search space. In addition, we can consider dropping input variables to choose the optimal subset, which would reduce the dimensionality of the space to search (assuming we have *already* selected the best subset out of 2^p possibilities).

Ridge regression directly addresses a weakness of using OLS on non-orthogonal $\mathbf{X^T X}$, i.e., on input data that are possibly related. In the case of fitting a polynomial, it is obvious that x_1 and x_1^2 are related; in almost every applied setting when working with real-world data, one works with input variables that are correlated. In the example we saw earlier, we saw that the number of rooms in a house was correlated with the price of the dwelling. We can easily imagine including other variables, such as total square footage or neighborhood indicators, that might be reasonable predictors of the price of the dwelling. However, we would also likely observe that the total square feet is related to the number of rooms, just as certain neighborhoods might have larger properties (possibly as well as or in spite of higher property prices). Even though square footage might "go with" the number of rooms, this does not necessarily mean that we want to pick only one of them for the regression. [14]

Hoerl & Kennard, in introducing ridge regression, note that while OLS gives unbiased estimates of θ, it does not give unbiased estimates of $\theta^T\theta$ when the data is not orthogonal. Note that for the linear model $\mathbf{y} = \mathbf{X}\theta + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$, the noise term is normally distributed with mean zero and variance σ^2. Then the OLS estimate of θ, $\hat{\theta}_{OLS} = \hat{\theta}$ can be described directly: $\hat{\theta} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}$, which is unbiased, i.e., the expected value[15] of $\hat{\theta}$ is θ. However, the expected value of $\hat{\theta}^T\hat{\theta}$ is

$$E[\hat{\theta}^T\hat{\theta}] = \theta^T\theta + \sigma^2 Tr(X^T X)^{-1} \tag{28}$$

This means that for orthogonal $\mathbf{X}$, we just have $E[\hat{\theta}^T\hat{\theta}] = \theta^T\theta + \sigma^2/n$, which tends to the true value of $\theta^T\theta$ as we see more data and n increases. However, for the more common condition of non-orthogonal $\mathbf{X}$, this means that our estimate $\hat{\theta}$ is likely to be far from the true θ. Then by restricting the size of our estimates of $\theta^T\theta$, we hope that we can stay "closer" to the true value.

The form of the ridge regression penalty creates contours such that in the unregularized score function (i.e., sum of squares) surface, while we are necessarily moving away from the maximum likelihood solution, we do so in the direction of the shortest length of the regression vector, as discussed in [5]. This makes sense: we are choosing by limiting the length of the vector θ by weighting by $\sum_{i=1}^{m} \theta_i^2 = \theta^T\theta$. This means we are limiting the impact of non-orthogonal $\mathbf{X}$.

In addition, we can rearrange our equations of the OLS equations with the ridge regression penalty to give the estimates of

$$\hat{\theta} = [\mathbf{X^T X} + \lambda \mathbf{I}]^{-1}\mathbf{X^T y} \tag{29}$$

for $\lambda \geq 0$ and $\mathbf{I}$ the identity matrix. [5] Then we can interpret the penalty through this matrix $k\mathbf{I}$, either from a Bayesian point of view[16] or to avoid singularity if $\mathbf{X^T X}$ is not invertible. [5], [1]

[14] We do, however, want to avoid at all costs any input variables that are exact linear combinations of other input variables. For example, if we include the age of the house in years, we do not want to also include the age of the house in months. This is called multicollinearity, or just collinearity, and it's bad news.

[15] For a definition of expected value, see, e.g., `http://en.wikipedia.org/wiki/Expected_value` or `http://mathworld.wolfram.com/ExpectationValue.html`.

[16] If we assume the (mean zero, variance σ_x not necessarily 1) data came from a multivariate normal distribution, this is the most likely solution, with λ a function of σ_x and the variance of θ. By `http://en.wikipedia.org/wiki/Tikhonov_regularization` and [1]

Other considerations We have largely ignored the setting of the **tuning parameter** λ. One consideration is the "bias-variance" tradeoff, where for a larger λ, we accept a larger bias and smaller variance, and vice versa for smaller λ. [2], [5] In practice, we will want to choose the tuning parameter by cross-validation (see, e.g., [4] for a useful discussion of cross-validation).

5.2 Ridge regression: Example

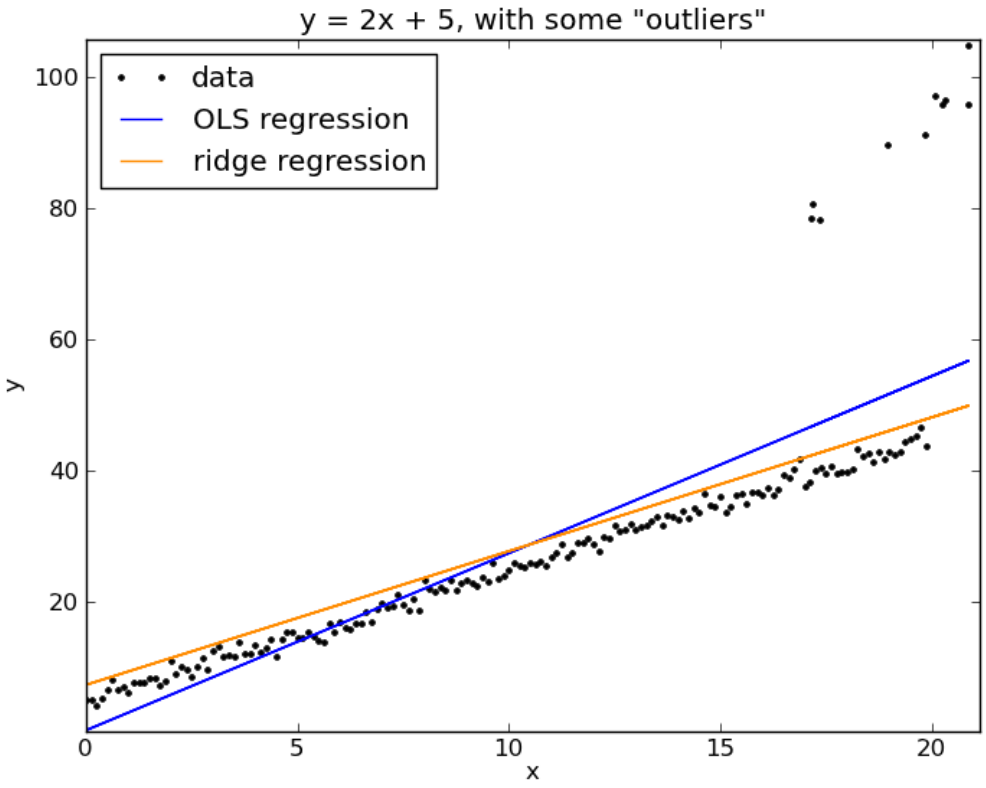

Figure 3: Ridge regression and standard OLS on (mostly) linear data.

I run ridge regression and standard OLS on an artificial set of data; the results are in Figure 3 and the code is provided in section 9. The original model,

$$y = 2x + 5 + \epsilon, \tag{30}$$

with noise ϵ with unit variance, was used to produce 160 data points. An additional 10 data points are produced by the model $y = 0.18x^2 + 25 + \epsilon_1$, $\epsilon_1 \sim N(0, 4)$.

The model predicted by OLS tries to capture all of the points, and does a poor job at fitting any of the data: it finds the linear model $y = 2.70x + 0.47$. The model found using ridge regression systematically overestimates the main set of points generated by the linear model, but overall finds a closer fit to the original: $y = 2.04x + 7.39$.

6 Regularization: general form & LASSO

6.1 General form

We can generalize the goal of **model selection** or **model comparison**, where we might want to change the structure of the solution space to make a certain set of solutions easier to find or more likely based on different goals.

As we discussed before, the penalty term can be used to trace contours, or ridges, in the solution space of the (unregularized) contour function, such that ridge regression finds the ridge (using $\theta^T\theta$) that minimizes the square error. If we change the form of the penalty term, we can change the shape of the ridges, such that we are more likely to induce certain kinds of solutions. [2]

Then we can consider the general class of regularizers, which have the form, following [2]:

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \frac{\lambda}{2}||\theta||^q \tag{31}$$

As we can see, $q = 2$ recovers ridge regression. For $q = 1$, this gives the score function for the popular "LASSO" regularization technique, due to Tibshirani [7], which will be discussed further. A schematic showing the contours we trace out by the $q = 0, 1, 2$ norms is shown in Figure 4.

6.2 Controlling the number of features: $q = 0$

We can consider the class of "$q = 0$" regularizers to be those that implement a constant penalty (as a factor of the number of nonzero-weight features) on the desired term. We can choose to think of the Akaike Information Criterion or Bayesian Information Criterion (AIC or BIC, respectively; see [4]), as belonging to this class. These are constant penalties (only by the number of features included with nonzero weight), but they may (e.g.) factor in the amount of data being considered. However, we will not discuss these in any more depth here.

6.3 LASSO for (shrinkage and) sparsity: $q = 1$

The case of $q = 1$ as above is widely known as "LASSO" in statistics and machine learning. The score function is then given by:

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) - \frac{\lambda}{2}\sum_{i=1}^{m}|\theta_i| \tag{32}$$

The structure of LASSO is partly motivated by the **subset selection** problem. Consider the linear model $y = \theta_m x_m + \theta_{m-1}x_{m-1} + \ldots + \theta_1 x_1 + \theta_0$, in which some set of $\{x_j\}$ have no predictive power of y. Then it should be that this model should be both just as good as excluding those $\{x_j\}$ and, in fact, equivalent to having the weights $\{\theta_j\}$ all set to zero. We know that if there is an optimal subset of the current m input variables, there are 2^m such sets to consider; on the other hand, we know that by setting $\theta_j = 0$, we can effectively eliminate input variable x_j from the model, which we should be able to accomplish from our estimate of θ. Then rather than searching for the optimal subset that excludes exactly those $\{x_j\}$, we can manipulate the solution space such that it becomes easier for us to find solutions that set those $\{\theta_j\}$ to zero.

The LASSO penalty uses the L^1 norm[17], by including the term $||\theta||_1 = \sum_{i=1}^{m}|\theta_i|$ as a penalty. This will draw "ridges" that are more likely to induce solutions with some coefficients equal to

[17] For more on L^p norms, see, e.g. `http://en.wikipedia.org/wiki/Lp_space` or `http://mathworld.wolfram.com/L2-Norm.html` and `http://mathworld.wolfram.com/L1-Norm.html`

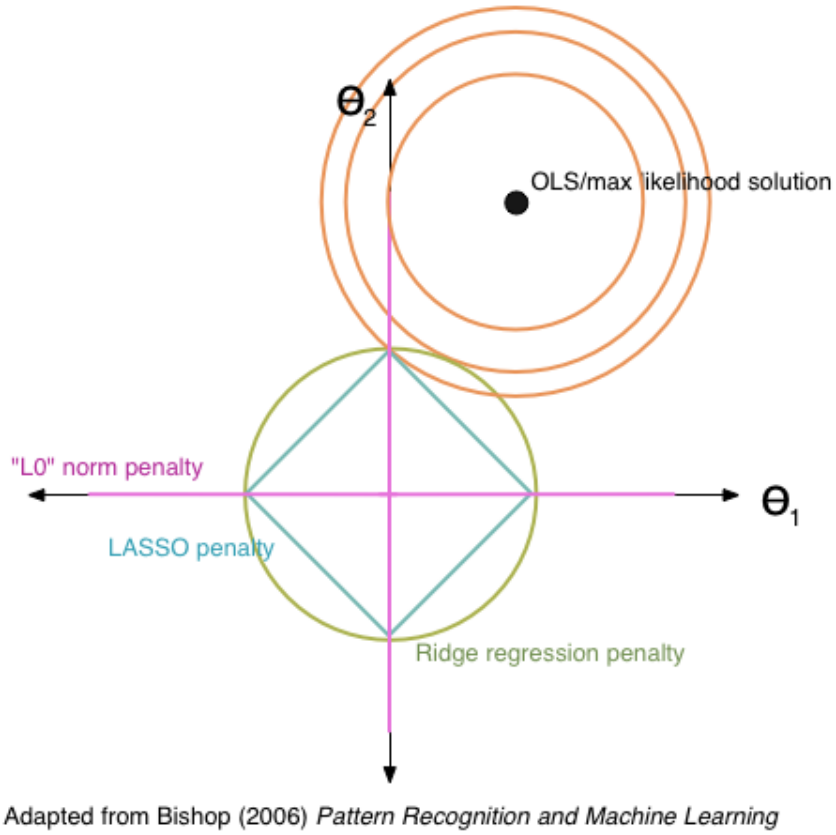

Figure 4: A schematic showing the state space of two weights θ. We can imagine moving further from the OLS solution towards solutions preferred by the type of the regularizer. The degree to which we move to each solution is dependent on the tuning parameter λ. As we can see, ridge regression simply follows the Euclidean norm, whereas $q = 1$ follows the absolute value of the sum, and $q = 0$ strictly prefers the axes (where the other θ_i will be set to zero

.

zero, whereas ridge regression will find similar solutions with small but nonzero coefficients, thus avoiding the subset selection problem. Then LASSO induces sparsity even on variables that are not irrelevant. In addition, if the model includes several highly correlated input variables, LASSO is more likely to exclude some of them (set $\theta_j = 0$) whereas ridge regression is more likely to share small values across all or most of them [2]. Then we can say that LASSO induces sparsity, while ridge regression simply encourages **shrinkage** of the estimated parameters.

Hence, it should be less surprising that LASSO stands for the "**least absolute shrinkage and selection operator**," as it combines subset selection (which is not robust but is created by sparsity) with shrinkage, as in ridge regression (which alone does not induce sparsity). LASSO is supposed to bridge these two goals, by finding solutions that approximate between subset selection and ridge regression. [7] For other extensions of LASSO in particular and some perspective on its direction in the statistics community, see [8].

6.3.1 Algorithms for LASSO

Ridge regression and the zero-"norm" regularizers benefit from analytical tools to find the optimal solutions. Algorithms for LASSO originally descended from the LARS algorithm; recent developments focus on stochastic gradient descent to find the optimal (see, e.g., [10]). These coordinate descent methods, such as "shooting," are now common in practice for LASSO; this has recently been parallelized (called "shotgun") [3].

In practice, stochastic gradient descent takes $O(kn\bar{p})$, for n the length of the data, p the number of parameters, and k the number of iterations [18]. Dependence on the number of iterations is related to issues of (rates of) convergence on desired solutions; very often in practice, we are willing to accept hits in computational time for convergence on the right solutions. Exploration of the convergence of these tools is beyond the scope of this course.

7 Regression, regularization, and prediction

This covers a large area of active research interest in statistics and machine learning, much of which has accelerated in the last two decades as access to both data and computing power has increased. Regression leads us into the set of prediction and classification tasks, and it plays an important role in modeling in the sciences. Regularization and other techniques to manipulate the score functions and solution spaces of regression tasks continue to be a point of active interest in these fields. Parametric and non-parametric methods for modeling and prediction and methods for model selection and comparison remain an active area of research.

Tradeoffs between computational concerns (runtime, space), statistical goals (bias, robustness, convergence), scientific goals and heuristics (sparsity, over-fitting), and "engineering" goals (minimizing errors in prediction) are an active area of concern, but recognizing when and how we encounter these tradeoffs outside of the standard "algorithmic" framework is important to developing algorithms in this area. Understanding when and how we can manipulate our problems to search for solutions that satisfy (or satisfice) different goals is necessary to make tools that are both widely applicable and computationally relevant.

[18] `http://scikit-learn.org/stable/modules/sgd.html#stochastic-gradient-descent-for-sparse-data`

8 Linear regression example code

```
#!/usr/bin/env python
#Code source: Jaques Grobler
#License: BSD
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model

#Load the diabetes dataset
houses = datasets.load_boston()

#Use only one feature
houses_X = houses.data[:,np.newaxis]
houses_X_temp = houses_X[:,:,5]


#Split the data into training/testing sets
houses_X_train = houses_X_temp[:-20]
houses_X_test  = houses_X_temp[-20:]

#Split the targets into training/testing sets
houses_y_train = houses.target[:-20]
houses_y_test  = houses.target[-20:]

#Create linear regression object
regr = linear_model.LinearRegression()

#Train the model using the training sets
regr.fit(houses_X_train, houses_y_train)

#Plot outputs
plt.scatter(houses_X_test, houses_y_test,  color='black')
plt.plot(houses_X_test, regr.predict(houses_X_test), color='blue', linewidth=3)
plt.xlabel('Avg. number of rooms per dwelling', fontsize=18)
plt.ylabel('Median value of homes ($1000\'s)', fontsize=18)
plt.savefig('boston_housing_regression.eps',format='eps')
```

9 Regularization example code

```
# source: Abigail Jacobs
from sklearn import linear_model
import numpy as np
from random import *
from pylab import *

xs = list()
ys = list()
m = 2
m0 = 5
for i in range(0,160):
  x = 0.125*i
  xs.append(x)
  y = m*x + m0 + randn()
  ys.append(y)

for i in range(0,10):
  xoff = randint(16,22) + randn()
```

```
    yoff = (0.09*m)*xoff*xoff + 25 + 4*randn()
    xs.append(xoff)
    ys.append(yoff)
xs = np.array(xs)
xs.resize(len(xs),1)
ys = np.array(ys)

regr = linear_model.LinearRegression()
regr.fit(xs,ys)
plot(xs,ys,'k.')
plot(xs,regr.predict(xs),color='blue')

# ridge cv
clf = linear_model.Ridge(alpha=2000.0)
clf.fit(xs,ys)
plot(xs,clf.predict(xs),color='darkorange')

legend(('data','OLS regression','ridge regression'),'upper left')
title('y = 2x + 5, with some "outliers"')
xlabel('x')
ylabel('y')

regr.coeff_
regr.predict(0)
clf.coef_
clf.predict(0)
```

References

[1] R. Angeles. Ridge+lasso notes. *Stat305 Lecture notes for Art Owens*, 2007.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, WA, USA, 2011.

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[5] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):pp. 55–67, 1970.

[6] A. Ng. Cs229 lecture notes. *CS229 Lecture notes*, 1(1):1–3, 2000.

[7] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Statist. Soc. B.*, 58(1):267–288, 1996.

[8] R. Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *J. Royal Statist. Soc. B.*, 73(3):273–282, 2011.

[9] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer Verlag, 2004.

[10] T. T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. *Annals of Applied Statistics*, 2(1):224–244, 2008.