Statistical Machine Learning Notes 4

Linear Methods

Instructor: Justin Domke

Contents

1 Introduction

Linear methods are the workhorse of statistics and machine learning. The fundamental idea of linear methods is deceptively simple: a linear map from inputs to outputs:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$$

How much could there be to say about such a simple idea? There are many issues relating to using linear methods in practice.

- **Loss Functions.** Suppose we are doing linear regression. Do we want to fit f to minimize the sum of squares differences $\hat{\mathbb{E}}\big(Y - f(\mathbf{X})\big)^2$, or perhaps the sum of absolute differences $\hat{\mathbb{E}}|\hat{y} - f(\hat{\mathbf{x}})|$? It also turns out to be very natural to adapt linear methods for the purpose of classification, with other loss functions.

- **Regularization.** Unless you have a huge amount of data, it is usually necessary to "regularize" the model. This is done by adding a penalty $\lambda h(\mathbf{w})$ to the empirical risk when fitting to favor "small" $\mathbf{w}$. Depending on how you define "small" different regularizers can be used.

- **Algorithms.** Linear methods are frequently the method of choice when we have very large datasets. In this setting, it is important that we have a fast algorithm for finding the best $\mathbf{w}$. The fastest algorithms are based on specific loss functions and specific types of regularization.

- **Cross-Validation.** We saw in the first notes how important it is to properly penalize complexity to prevent overfitting. When doing regularization, this amounts to properly selecting the penalty parameter λ alluded to above. The obvious way of doing this is to pick a bunch of possible values (say, $0.1, 0.2, ..., 10$) and fit the weights for each. The only problem with this is that it is rather computationally expensive. In some cases, clever "regularization path" algorithms can be designed that, once they are complete, can give you the weights corresponding to any particular λ very quickly.

- **Doing nonlinear stuff with linear methods.** There are several ways to adapt linear methods to do apparently nonlinear things. (For example, linear methods were used to fit the polynomials in the first set of notes). In addition, linear methods form the base for more advanced methods, such as neural networks and Support Vector Machines.

2 Generic Linear Method

We can phrase all of the linear methods we will discuss as specializations of one framework.

Generic Linear Method:

1. Input
   - The training data $D = \{(\hat{\mathbf{y}}, \hat{\mathbf{x}})\}$.
   - A loss function L.

   - A regularization function $h(\mathbf{w})$
   - A regularization constant λ.

2. Optimize, to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} \hat{\mathbb{E}} L(\mathbf{w}\cdot\mathbf{X}, Y) + \lambda h(\mathbf{w})$.

3. Output $f(\mathbf{x}) = \mathbf{w}^* \cdot \mathbf{x}$.

Depending on what the training data is, what the loss function is, what regularizer you use, this framework can become many different specific methods, both for regression, and for classification. As well as choosing all these, there are many possible algorithms for performing the optimization in step 2. All of this flexibility means there are a huge variety of linear methods. In this class, we just

3 Linear Regression

To begin these notes, we will phrase different regression methods in terms of the *optimization problems* that they solve. We will worry about the actual *algorithms* for performing the optimizations later on.

name	function
Least Squares	$L(\mathbf{w}\cdot\mathbf{x}, y) \;=\; \big(\mathbf{w}\cdot\mathbf{x} - y\big)^2$
Least Absolute Deviation	$L(\mathbf{w}\cdot\mathbf{x}, y) \;=\; \lvert\mathbf{w}\cdot\mathbf{x} - y\rvert$

We begin with the simplest method, linear regression. The classic method, **least squares regression**, fits the weights $\mathbf{w}$ to minimize the sum of squares errors. So our loss function is

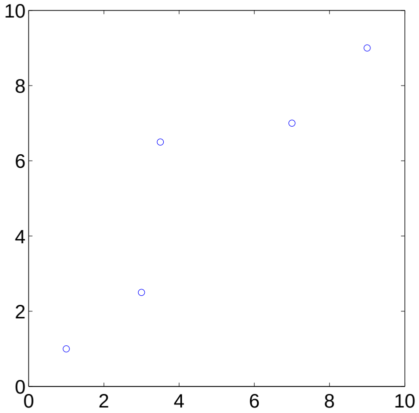
$$L_{\text{lsq}}(\mathbf{w}\cdot\mathbf{x}, y) = \big(\mathbf{w}\cdot\mathbf{x} - y\big)^2.$$

If we have a dataset $D = \{(\hat{y}, \hat{\mathbf{x}})\}$, we seek $\mathbf{w}$ to minimize the empirical risk

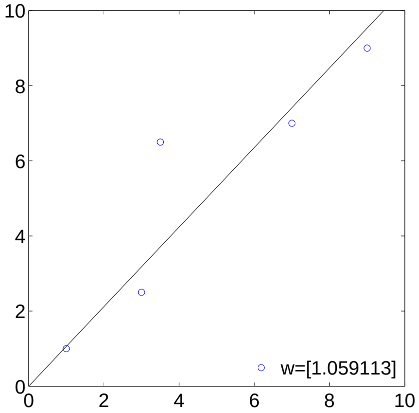
$$R(\mathbf{w}) = \hat{\mathbb{E}} L_{\text{lsq}}(\mathbf{w}\cdot\mathbf{X}, Y) = \hat{\mathbb{E}}\big(\mathbf{w}\cdot\mathbf{X} - Y\big)^2.$$

It is worth looking at an example. Consider the dataset

$\hat{x}$	$\hat{y}$
1	1
3	2.5
3.5	6.5
7	7
9	9



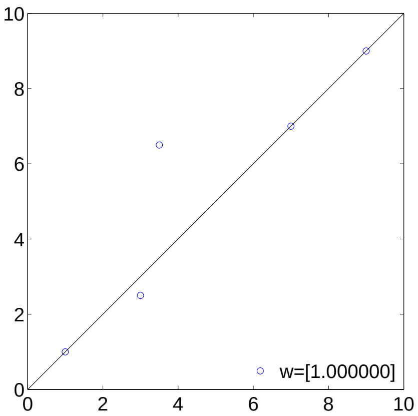
The least-squares solution is $w^* = 1.059113$.



However, of course, we don't have to minimize the sum of squares. We could also minimize, for example, the sum of absolute differences, called **least absolute deviation regression**,

$$L_{\text{lad}}\left(\mathbf{w}\cdot\mathbf{x},y\right) = |\mathbf{w}\cdot\mathbf{x} - y|.$$

Minimizing this on the above training set, we see something very interesting. The solution is for the weights to be *exactly* one.



Of course, we could create many other loss functions. Which is right? The answer, as ever, is that the "best" loss function depends on the priorities of the user.

4 Regularization

Suppose that we have a dataset with 20 points, with inputs of dimension 7. If we fit a linear regression model as above, with less than 3 points per parameter, we are essentially guaranteed to overfit. How can we deal with this? The standard solution for preventing overfitting is **regularization**. Since we know that the empirical risk will underestimate the true risk, add a penalty to complex functions f to try to compensate for this[1]. Then, instead of doing

$$\min_f R(\mathbf{w}),$$

$$R(\mathbf{w}) = \hat{\mathbb{E}} L(\mathbf{w} \cdot \mathbf{X}, Y),$$

instead do

$$\min_f R(\mathbf{w}) + \lambda h(\mathbf{w}). \tag{4.1}$$

Notice that we can rephrase this unconstrained linear optimization as an constrained optimization. For any given λ there is a c such that the same solution can be obtained from

[1] This sounds rather ad-hoc. To some degree it is, though we will see more principled methods when we talk about learning theory.

$$\begin{aligned} \min_f \quad & R(\mathbf{w}) \\ \text{s.t} \quad & h(\mathbf{w}) \leq c. \end{aligned} \tag{4.2}$$

(**Warning:** It is probably easier to convince yourself of this than slog through the following details. Nevertheless, here is a proof sketch if you want it. First, consider the solution f^* from the first optimization (Eq. 4.1) corresponding to some λ. It is not hard to see that f^* will also be a solution to the second optimization (Eq. 4.2) if $c = h(f^*)$. (If there is some other f' with $h(f') \leq c$, but a better objective value than f^*, then this would also be a better solution to the first optimization– which is impossible.))

The following table shows a variety of regularizers. In the context of linear regression, we will consider three regularizers: the number of nonzero weights (the l_0 norm), the sum of absolute values of weights (the l_1 norm), and the sum of squares of weights (the l_2 norm *squared*[2]). The elastic net and l_∞ regularizers are shown for variety.

name	function			
l_0 / Best Subset	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_0$	$=$	$\sum_i I[w_i \neq 0]$
l_1 / Lasso	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_1$	$=$	$\sum_i \lvert w_i\rvert$
l_2^2 / Ridge	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_2^2$	$= \mathbf{w}\cdot\mathbf{w} =$	$\sum_i w_i^2$
Elastic Net	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_1 + \alpha\lvert\mathbf{w}\rvert_2^2$	$=$	$\sum_i \lvert w_i\rvert + \alpha\sum_i w_i^2$
l_∞	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_\infty$	$=$	$\max_i \lvert w_i\rvert$

The l_0 penalty may be the easiest to understand. If we are doing least-squares regression, the weights will be selected by

$$\begin{aligned} \min_{\mathbf{w}} \quad & \hat{\mathbb{E}}(\mathbf{w}\cdot\mathbf{X} - Y)^2 \\ \text{s.t} \quad & \sum_i I[w_i \neq 0] \leq c. \end{aligned} \tag{4.3}$$

This just says to find the weights minimizing the squared error, subject to the restriction that only c of the weights can be nonzero. Once the nonzero indices of $\mathbf{w}$ are chosen, finding the actual weights is essentially nonregularized least squares. This is very natural– if we don't have enough data to fit all of the parameters, just fit as many as we can afford.

Though natural, this is not so frequently done. There are two basic reasons for this, one computational and one statistical.

[2] The weights that results from the two are the same, but for different regularization constants. (A constant of C for ridge is equivalent to a constant of $\sqrt{C}$ for l_2.) Using the l_2 norm complicates a lot of algebra.

- The computational issue is that Eq. 4.3 is a highly nonconvex optimization, meaning we cannot use techniques like gradient descent or Newton's method to solve it. With d variables, there are 2^d subsets, meaning a brute-force approach is also impractical when d is big. There are clever branch-and-bound techniques that are faster than a full search, but even these cannot tackle problems with $d > 50$ or so. There are also obvious heuristics that start with $\mathbf{w} = 0$, and greedily add indices, or start with the unregularized solution and greedily (stingily?) subtract indices. These will provide suboptimal solutions to the above *optimization problem*, but they won't necessarily give worse *results on test data*. (Why? Think about the first notes.)

- The statistical issue is that subset selection can be *unstable*. Sometimes a tiny change in the data leads to a large change in the estimated weights. For example, if we might find $\mathbf{w} = (1.1, 0, 2.0)$, but a small change in x_2 leads to it being selected in favor of x_1, with the results of $\mathbf{w} = (0, 1.2, 2.5)$, say.

Least-squares regression with ridge penalty is called **ridge regression**. This is the most popular method. In the unconstrained representation, one looks for the weights $\mathbf{w}$ that minimize

$$\hat{\mathbb{E}}\big(\mathbf{w} \cdot \mathbf{X} - Y\big)^2 + \lambda \mathbf{w} \cdot \mathbf{w},$$

while in the constrained representation, one seeks the weights with the minimum squared error, subject to the constraint that $|\mathbf{w}|_2^2 \leq c$. This essentially means that the weights are constrained to lie in an origin-centered sphere of some radius.

Ridge regression has neither of the problems mentioned above for best-subset regression. The weights can be found quite efficiently, and are stable with respect to changes in the data. The main downside, of course, is that if you want weights with some of the indices set to zero, ridge regression doesn't do it.

Least-squares regression with l_1 regularization is called **lasso regression**. This finds the weights $\mathbf{w}$ that minimize

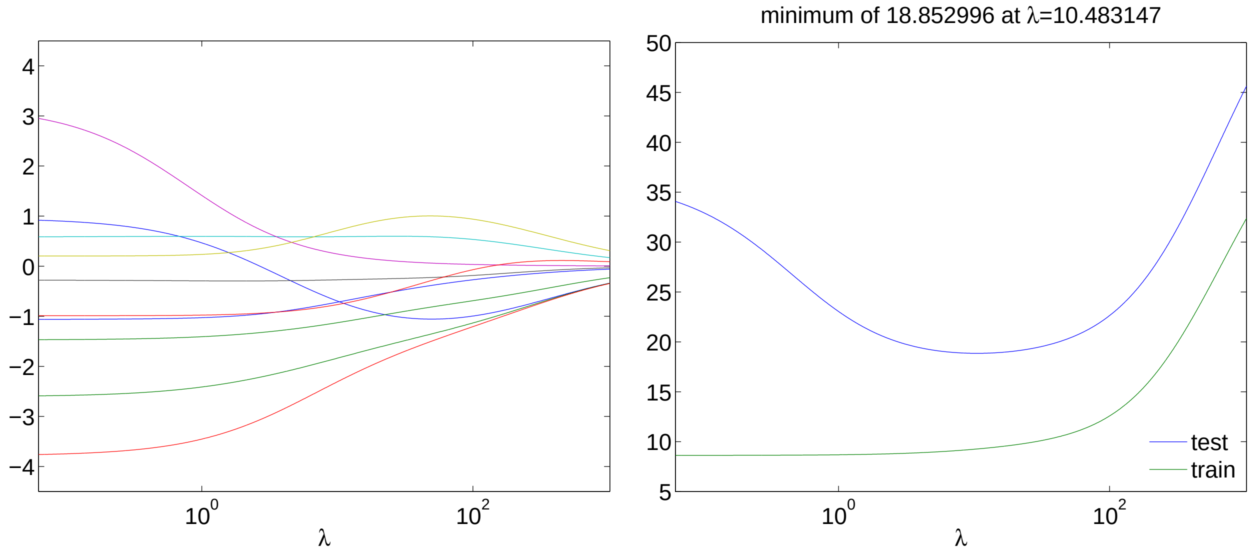
$$\hat{\mathbb{E}}\big(\mathbf{w} \cdot \mathbf{X} - Y\big)^2 + \lambda |\mathbf{w}|_1. \tag{4.4}$$

The behavior of lasso regression is something between ridge and best subset. Like best subset, for high enough λ, it does induce sparsity in $\mathbf{w}$. However, Eq. 4.4 represents a convex optimization, and so it is possible to design reliable and efficient algorithms to find $\mathbf{w}$, even with very high dimensionality.

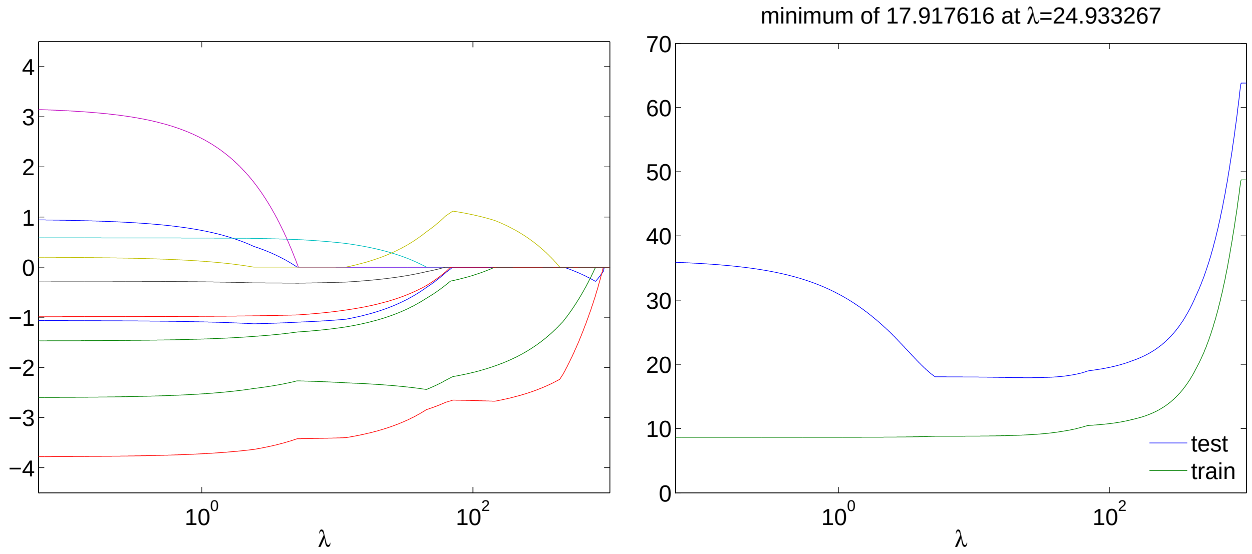
The figures below compare ridge and lasso regression on a "famous" dataset. In this dataset, y is the MPG of a vehicle, while the inputs $\mathbf{x}$ encode the number of cylinders, the engine displacement, , the horsepower, the vehicle weight, acceleration speed, and origin (North America, Europe, or Japan)[3].

[3] This dataset also includes model year, but this was not used in these figures.

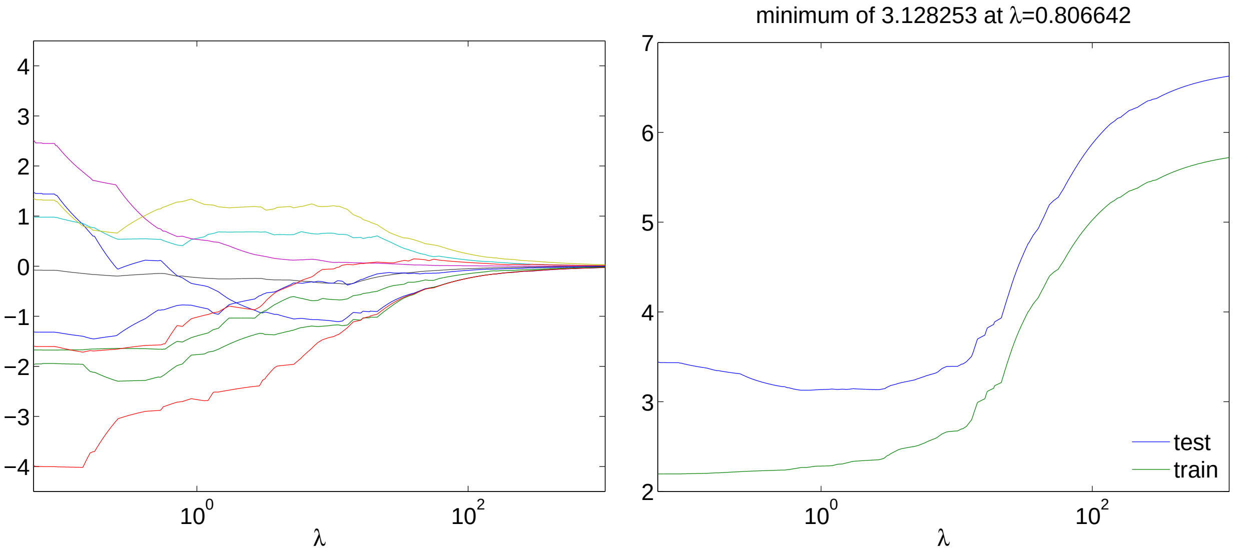
Least Squares Regression + Ridge Penalty



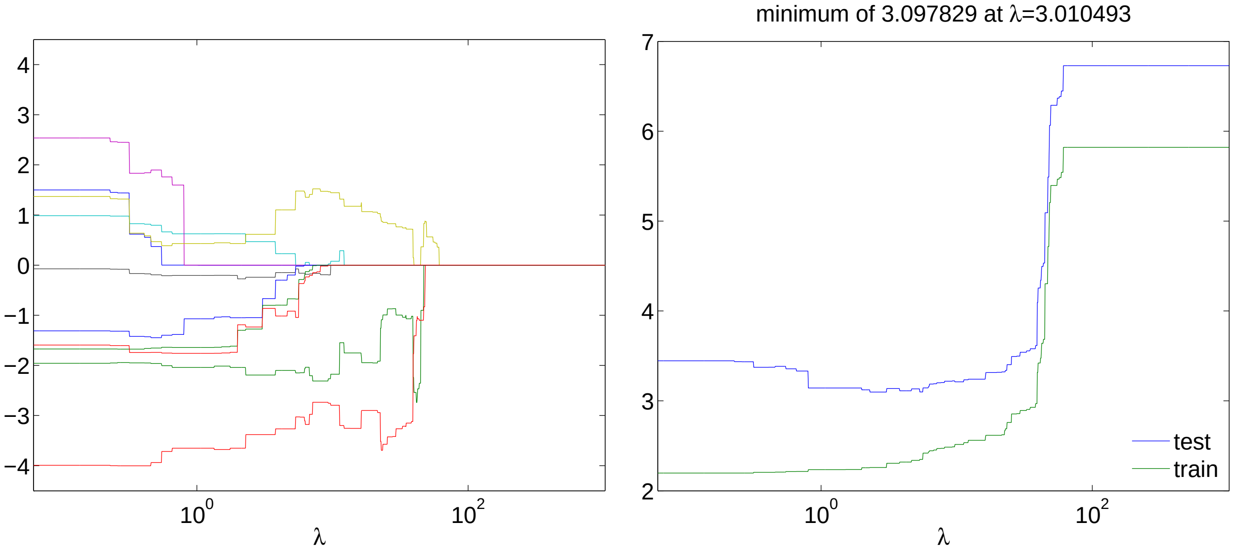
Least Squares Regression + Lasso Penalty



Least Absolute Deviation Regression + Ridge Penalty

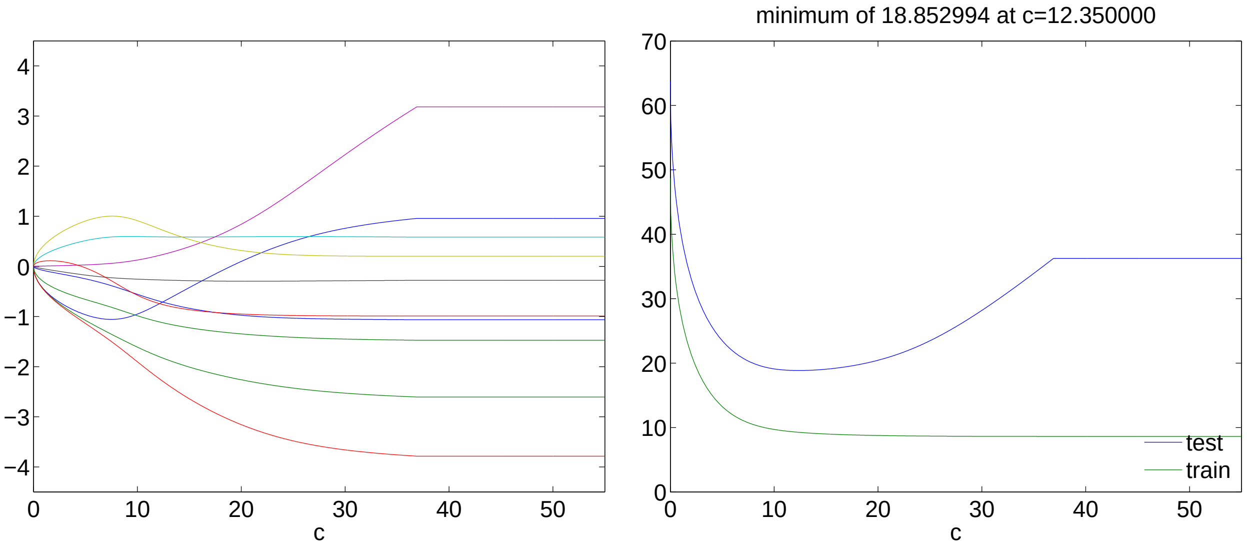


Least Absolute Deviation Regression + Lasso Penalty

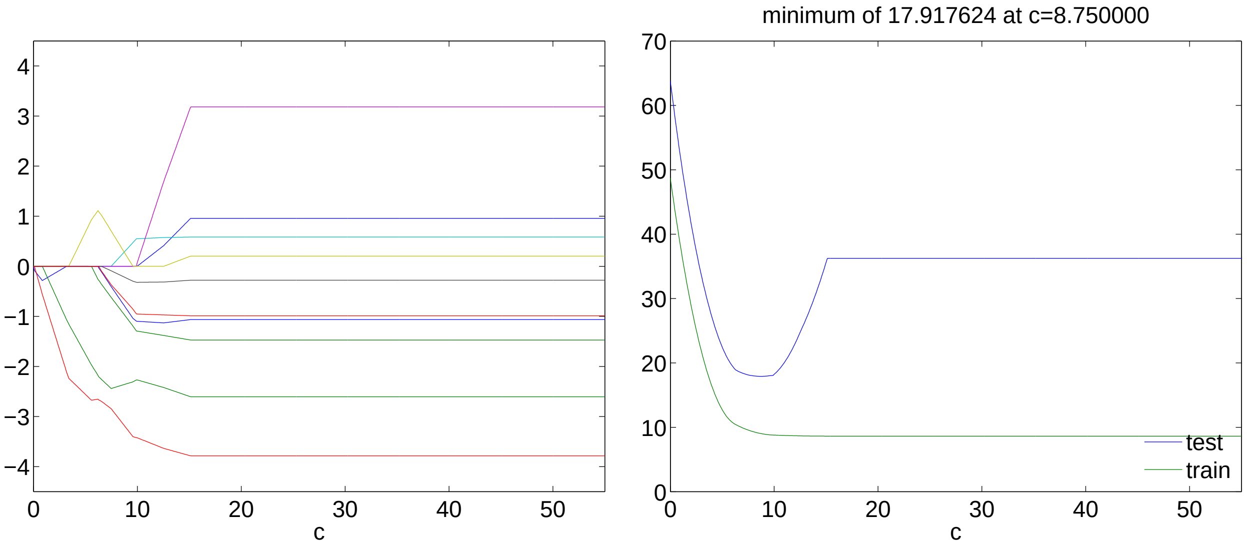


The next two pages show the same thing, but in terms of the constrained formulation $\min_{\mathbf{w}} R(\mathbf{w})$ s.t. $h(\mathbf{w}) \leq c$. If we watch carefully, we see that these figures sweep through the same paths as $c : 0 \to \infty$ as the previous figures do as $\lambda : \infty \to 0$. Note also that above a certain c, nothing changes. This occurs for $c \geq h(\mathbf{w}^*)$, where $\mathbf{w}^*$ is the unregularized solution.

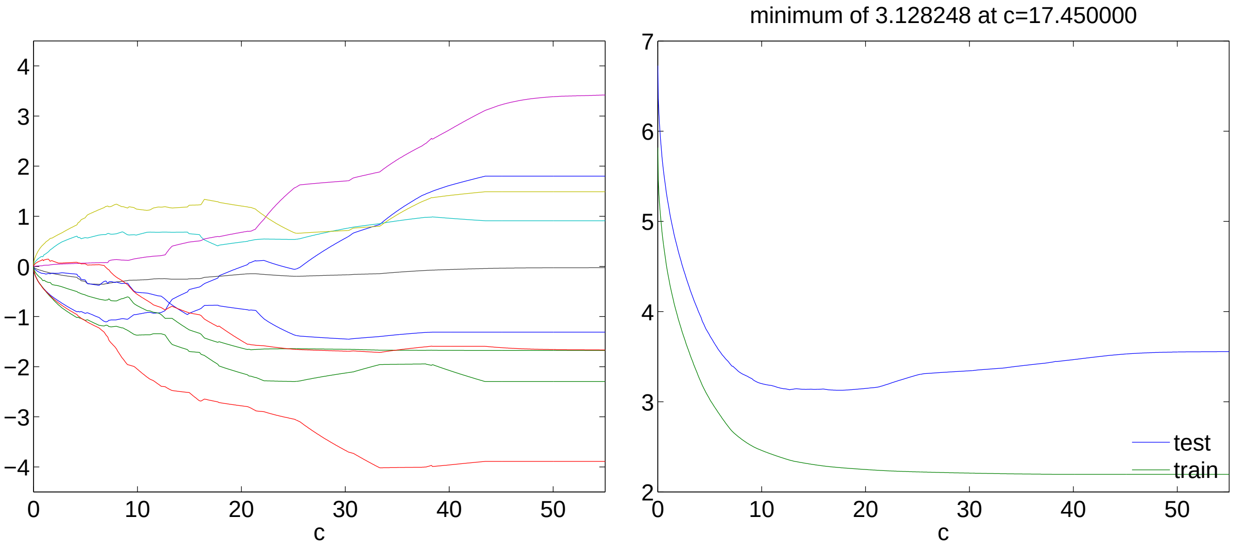
Least Squares Regression + Ridge Penalty



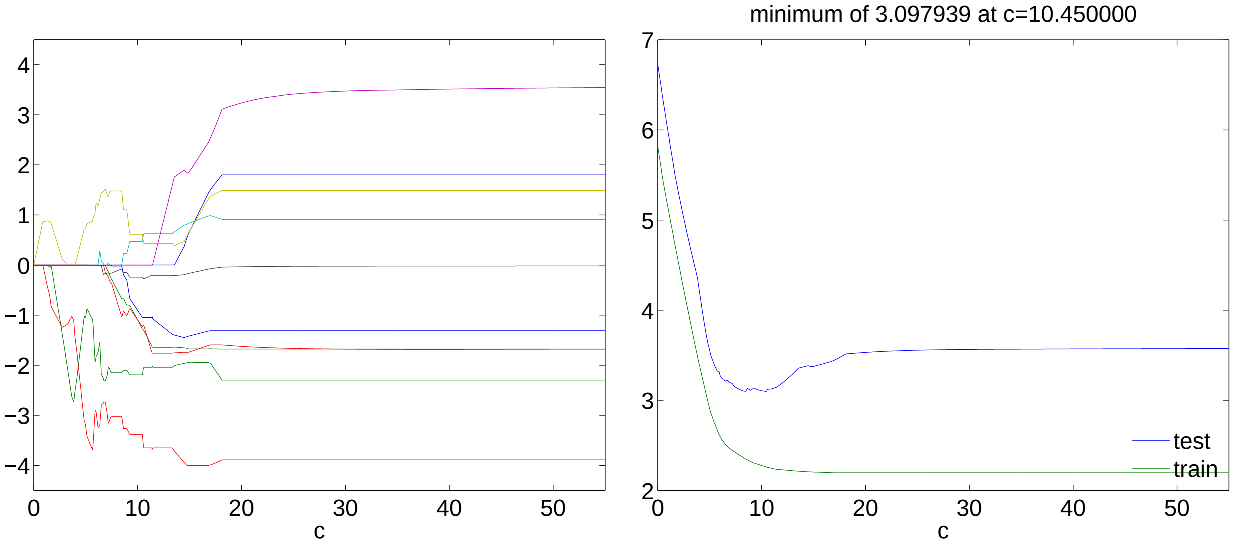
Least Squares Regression + Lasso Penalty



Least Absolute Deviation Regression + Ridge Penalty

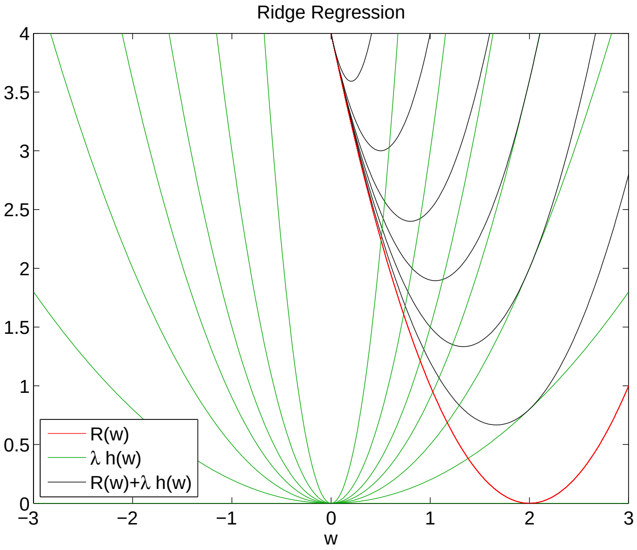


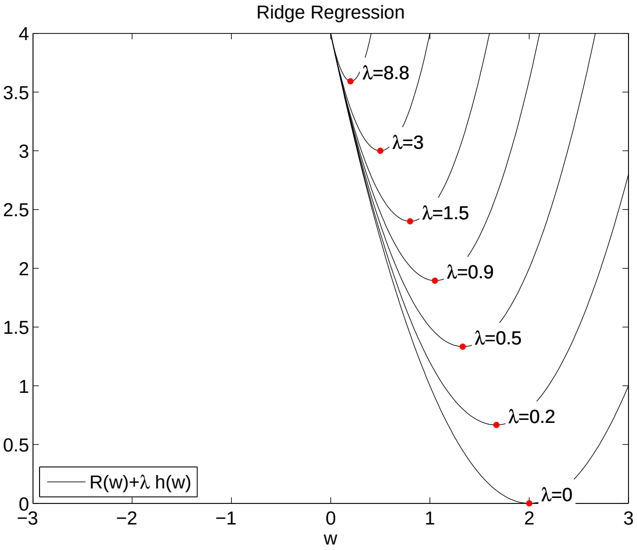
Least Absolute Deviation Regression + Lasso Penalty



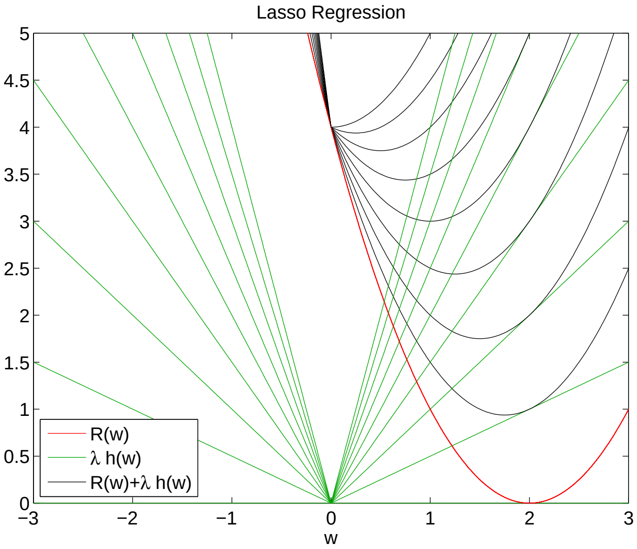
Naturally, for larger λ, all the weights go towards zero. For ridge regression they go towards zero smoothly, only reaching zero as $\lambda \to \infty$. For the lasso, on the other hand, the weights hit zero for a finite λ.

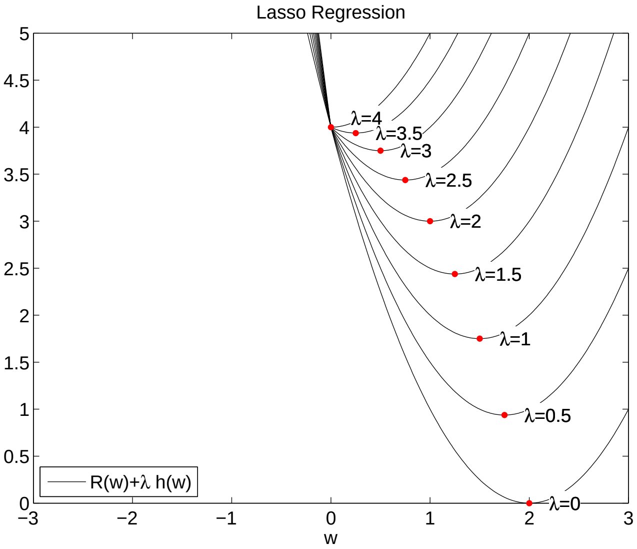
To understand this, let's picture the situation where the input x only has one dimension, and so we are fitting a single weight w. Let's start with ridge regression. The following figures show the empirical risk, and regularized risk for a variety of λ. We can see that $R(w)+\lambda h(w)$ is always just a quadratic function. For $\lambda = 0$ it is equivalent to the pure risk, minimized by $w = 2$. As λ gets larger, the regularization term slowly comes to dominate. However, until λ is infinite, the minimum of the quadratic will always be slightly greater than zero.





Compare this to the same picture for Lasso regression.





We can see that the discontinuity in the regularizer means that that for large enough λ, $R(w) + \lambda h(w)$ develops a "kink" at zero. For any λ greater than this, the optimum will be exactly $w = 0$.

5 Geometrical visualization of linear regression

So, when doing regularized linear regression, we have to pick two things:

1. $R(\mathbf{w})$: How to measure how closely $\mathbf{w} \cdot \mathbf{x}$ matches to y. (Least squares, Least Absolute Deviation)

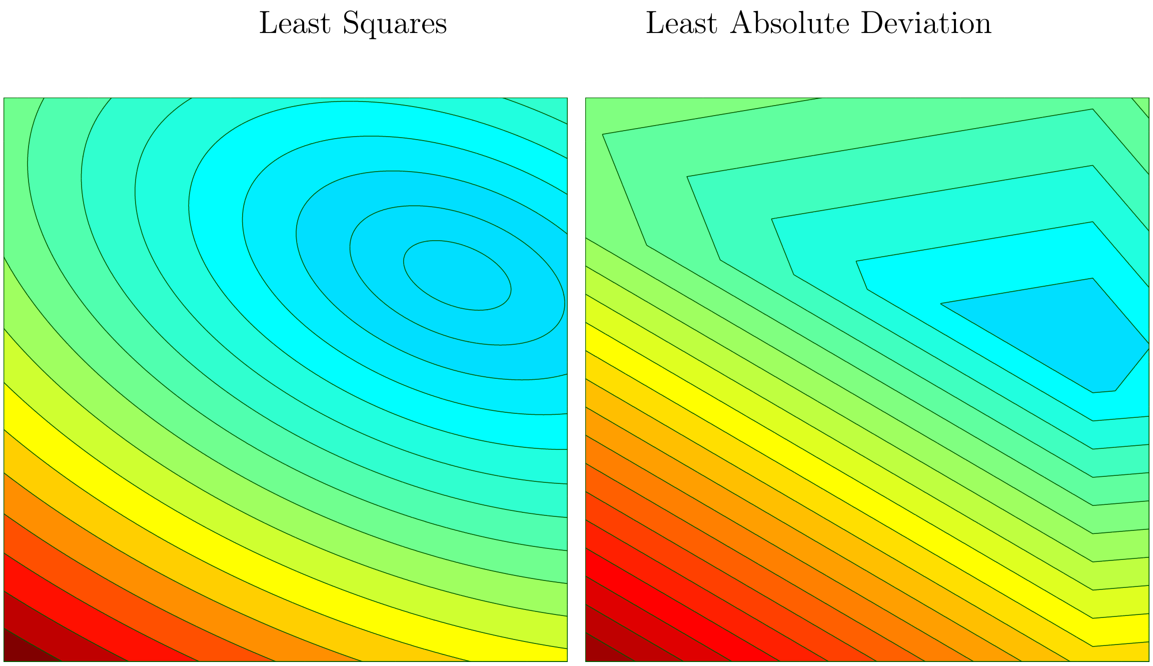
2. $h(\mathbf{w})$: How to penalize complexity. (Ridge, Lasso)

In a perfect world, we would like both of these to be small. Of course, we must ultimately choose how to trade-off between the two. With out specifying a trade-off parameter, we cannot choose the "best" solution $\mathbf{w}$. However, we still can rule our most! Why? Consider a set of weights $\mathbf{w}^*$. If there exists another set of weights $\mathbf{w}'$ such that $R(\mathbf{w}') < R(\mathbf{w}^*)$ and $h(\mathbf{w}') \leq h(\mathbf{w}^*)$, then say that $\mathbf{w}'$ *dominates* $\mathbf{w}^*$, meaning that $\mathbf{w}'$ is as good as $\mathbf{w}^*$ in every way, and strictly better in at least one. The set weights not dominated in this way are the only possible solutions. This is a general concept known as Pareto optimality.

This section visualises the different possible choices of R and h on a small two-dimensional dataset, along with the set of weights that result from different tradeoffs between the two.

$\hat{\mathbf{x}}$	$\hat{y}$
(1, 5)	2
(4, 0)	4
(2, 4)	2
(0, 3)	5

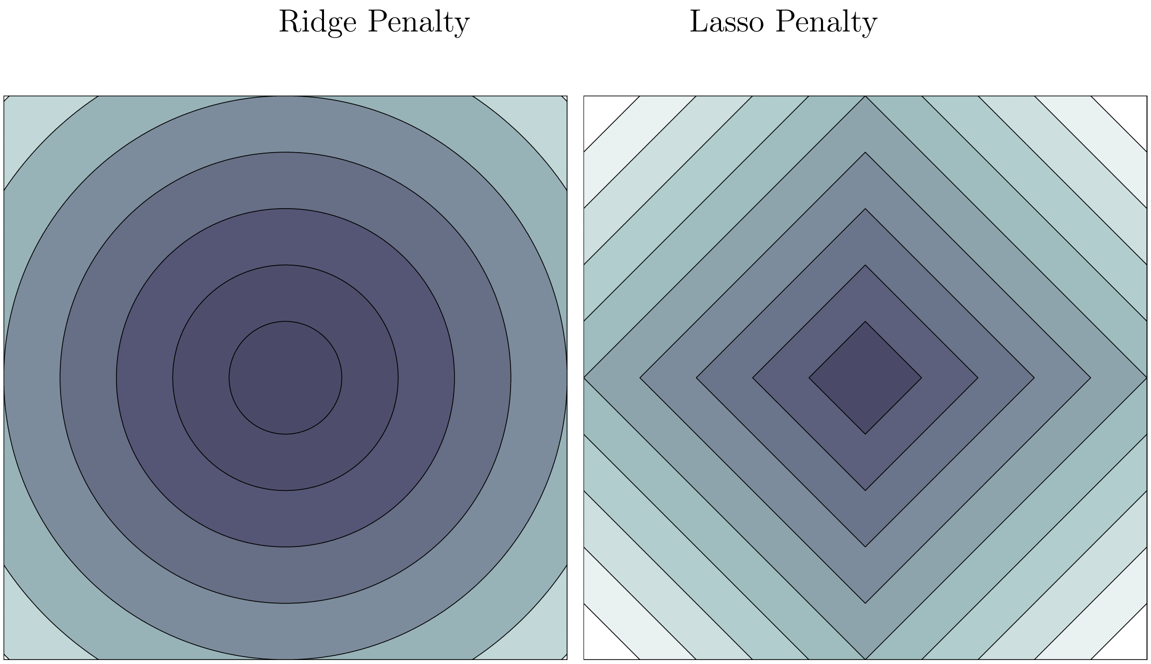
The first set of figures show $R(\mathbf{w})$ for both the least-squares loss, and the least-absolute-deviation loss. The least-squares loss is simply a quadratic in the space of $\mathbf{w}$, while least-absolute-deviation is a more complex piecewise linear function. Notice that the minima are in similar, but not identical positions.



Least Squares

Least Absolute Deviation

The next set of figures shows $h(\mathbf{w})$ for both the Ridge and Lasso penalties. These are both centered at $(0, 0)$, but differ elsewhere.

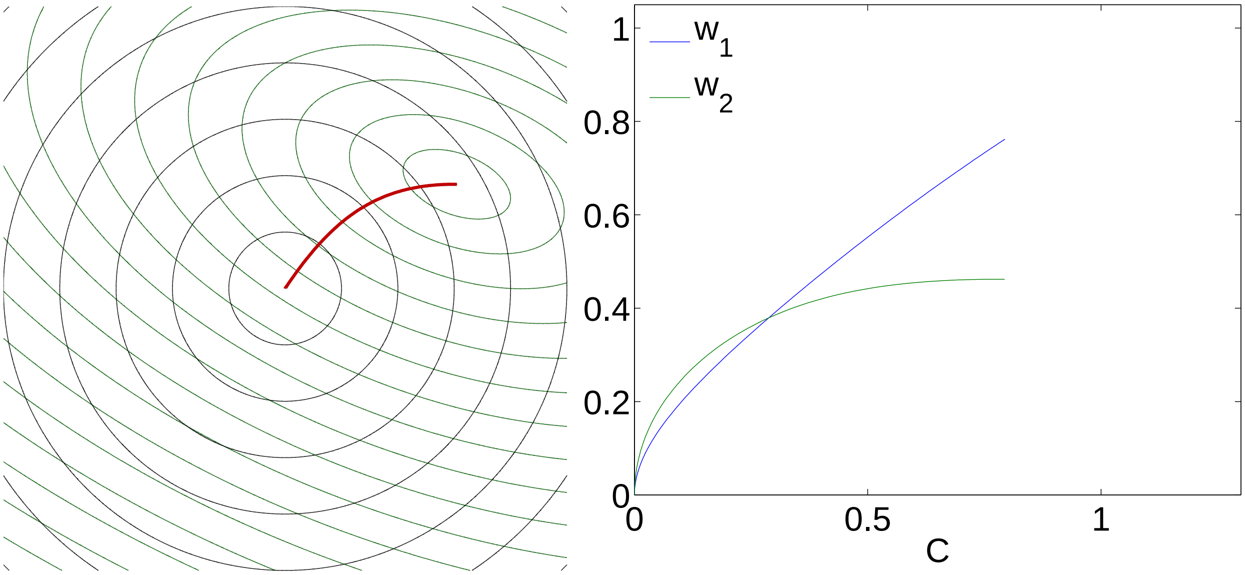


Ridge Penalty

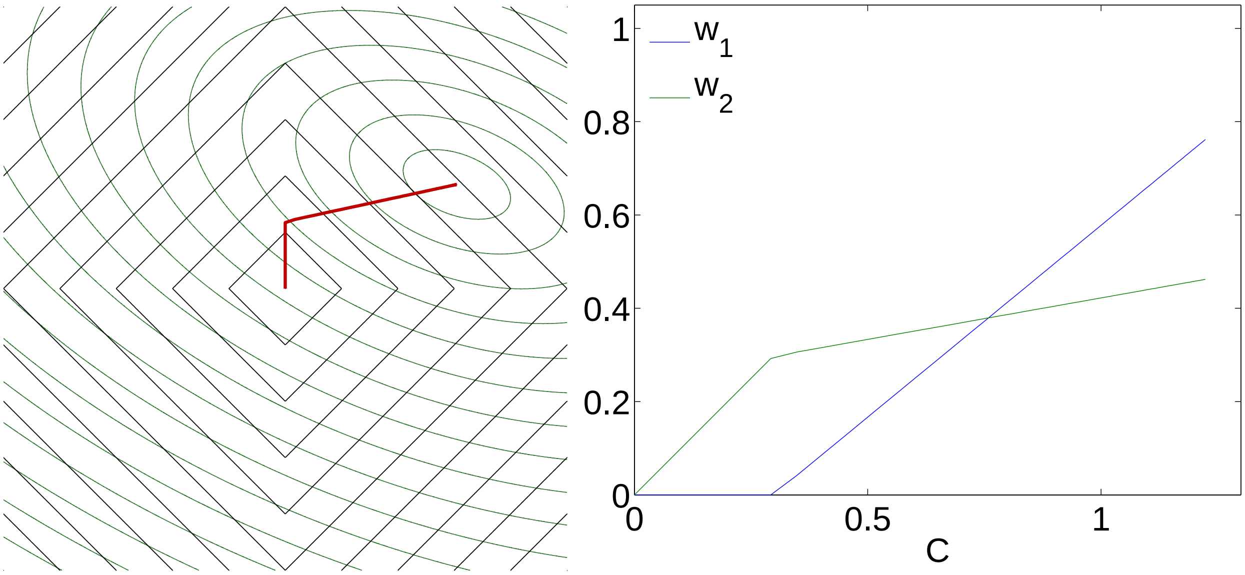
Lasso Penalty

The next figures show each combination of the above risks (in green) along with the penalties (in black). The red curve shows the possible weights in each situation. To understand these figures, think of the set $\{\mathbf{w} : h(\mathbf{w}) \leq c\}$. This will be all $\mathbf{w}$ inside one black curve– a curve close to the center for small c, further for large c. Inside this set, find the $\mathbf{w}$ that minimizes $R(\mathbf{w})$, meaning find the point that inside the smallest green curve. Repeating this process for all C generates all the possible weights.

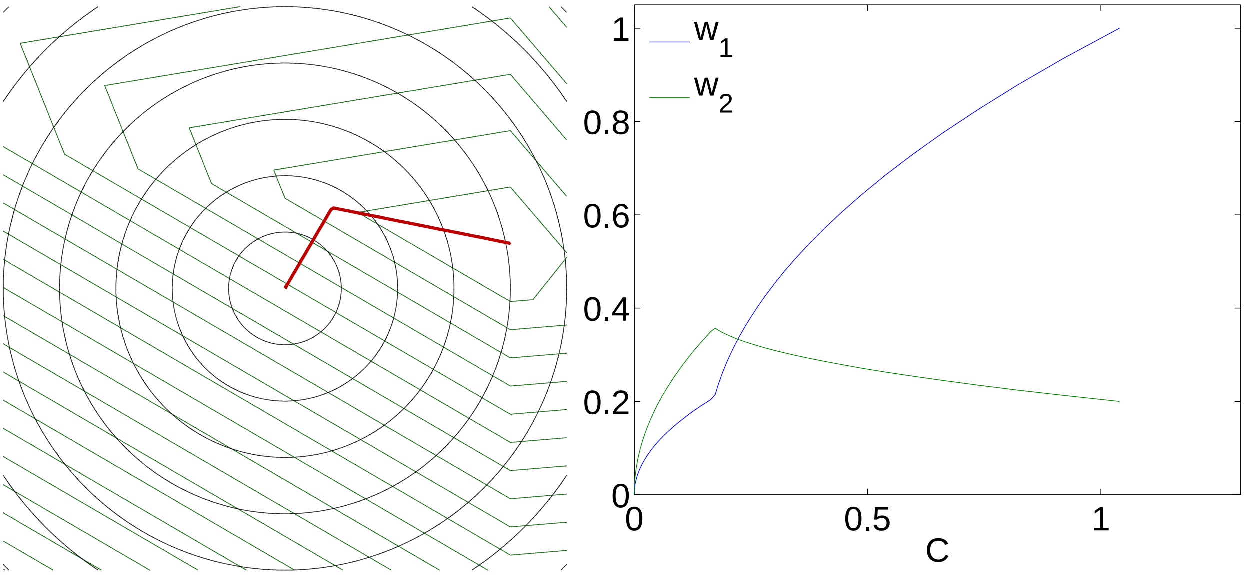
Least Squares + Ridge Penalty



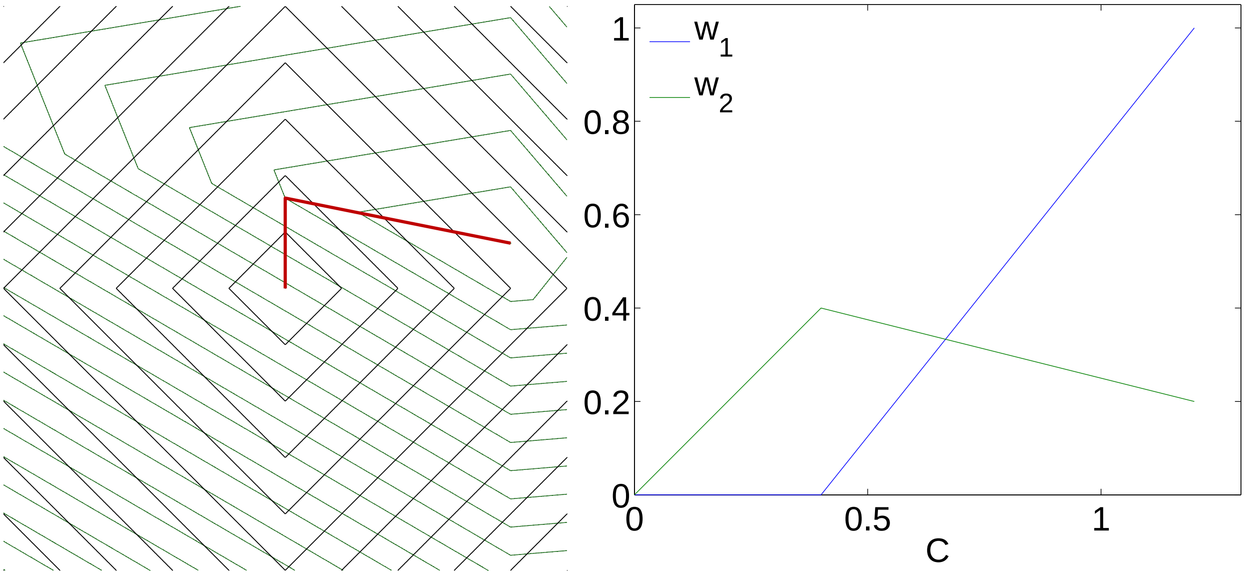
Least Squares + Lasso Penalty



Least Absolute Deviation + Ridge Penalty



Least Absolute Deviation + Lasso Penalty



6 Linear Classification

Consider the binary[4] classification problem: take real-valued vectors $\mathbf{x}$ as input as before, but try to predict an output y that is either -1 or $+1$. At first glance, linear functions don't appear to apply: $\mathbf{w}\cdot\mathbf{x}$ will be a general real number, not -1 or $+1$. The basic idea of linear classification is to predict y to be the *sign* of $\mathbf{w}\cdot\mathbf{x}$. As we will see, some methods also use the magnitude of $\mathbf{w}\cdot\mathbf{x}$ to give the confidence of the prediction.

name	function		
0-1 Loss	$L_{0-1}(\mathbf{w}\cdot\mathbf{x}, y)$	$=$	$I[y\,\mathbf{w}\cdot\mathbf{x} > 0]$
Hinge	$L_{\text{hinge}}(\mathbf{w}\cdot\mathbf{x}, y)$	$=$	$\big(1 - y\,\mathbf{w}\cdot\mathbf{x}\big)_+$
Logistic	$L_{\log}(\mathbf{w}\cdot\mathbf{x}, y)$	$=$	$\log(1+\exp(-y\,\mathbf{w}\cdot\mathbf{x}))$

The most obvious loss function is the **0-1 loss**, so called because it is 0 if $\mathbf{w}$ classifies a point correctly, and 1 if it misclassifies it.

$$L_{0-1}(\mathbf{w}\cdot\mathbf{x}, y) = I[y\,\mathbf{w}\cdot\mathbf{x} > 0]$$

This loss is almost never used. The major reason is that leads to a non-convex (in fact non-differentiable) empirical risk.

A more tractable option is the **hinge loss**. This is defined by

$$L_{\text{hinge}}(\mathbf{w}\cdot\mathbf{x}, y) = \big(1 - y\,\mathbf{w}\cdot\mathbf{x}\big)_+,$$

where $(a)_+ = \max(0, a)$. To understand this, consider the case when $\mathbf{w}$ classifies the point correctly, and with high confidence. Then, $\hat{y}\mathbf{w}\cdot\hat{\mathbf{x}}$ will be large, and so the loss is zero. Now, consider the case that $\mathbf{w}$ classifies the point *incorrectly.* Now the loss is positive, and becomes larger with higher confidence.

As shown in the figure below, the hinge loss is an upper-bound on the 0-1 loss. Intuitively, you can see that there is no way to make the hinge loss closer to the 0-1 loss, with out losing convexity.

The final loss we will consider is the **logistic loss**.

$$L_{\log}(\mathbf{w}\cdot\mathbf{x}, y) = \log(1+\exp(-y\,\mathbf{w}\cdot\mathbf{x})). \tag{6.1}$$

[4] Real-world classification problems are usually non-binary. However, binary problems are easier to visualize and have somewhat simpler math. There are extensions of everything discussed here to multi-class problems.

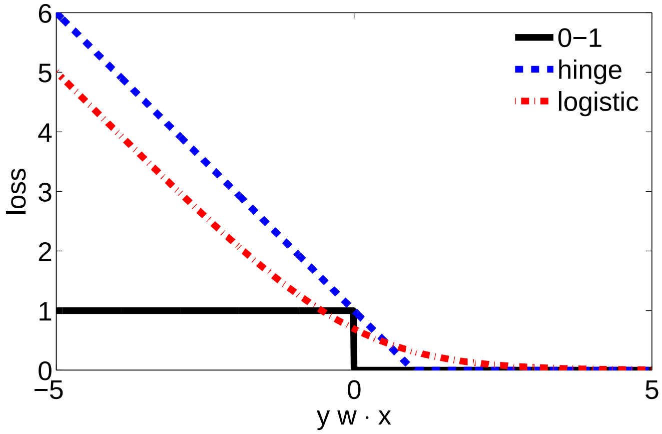
Unlike the hinge loss, this is (twice) differentiable, which makes it somewhat easier to optimize. For historical reasons linear classification with the logistic loss is typically called **logistic regression** even though we are doing classification, not regression. The logistic loss was originated from a probabilistic perspective. Given an input $\mathbf{x}$, we can think of the model as predicting a probability that y is either -1 or $+1$ by

$$p(y|\mathbf{x}) = \frac{1}{1 + \exp(-y\,\mathbf{w} \cdot \mathbf{x})}.$$

You should check that $p(-1|\mathbf{x}) + p(+1|\mathbf{x}) = 1$ for all $\mathbf{x}$. If we use the negative log likelihood $-\log p(\hat{y}|\hat{\mathbf{x}})$ as our loss function, and substitute this expression for p, the results simplify into Eq. 6.1. We are free, however, to ignore this probabilistic perspective, and just think of the logistic loss as an approximation to the 0-1 loss.

Warning: You need to be careful when reading about logistic regression in the literature, as there are different variations of "logistic regression", and no standard terminology to differentiate them. For example, a different loss function results when $y \in \{0, 1\}$ rather than $\{-1, 1\}$. Logistic regression can also be extended to the multi-label setting where $y \in \{0, 1, ..., L\}$, again in several different ways.

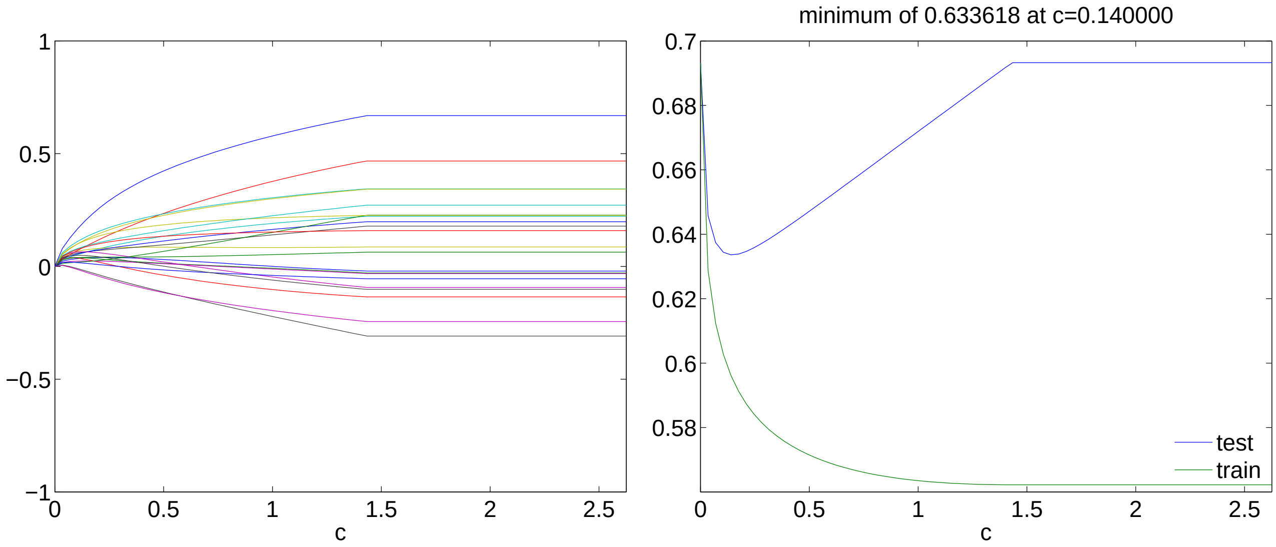
Since all the above loss functions only depend on $\hat{y}\,\mathbf{w} \cdot \hat{\mathbf{x}}$, we can compare them by plotting them as a function of this quantity.



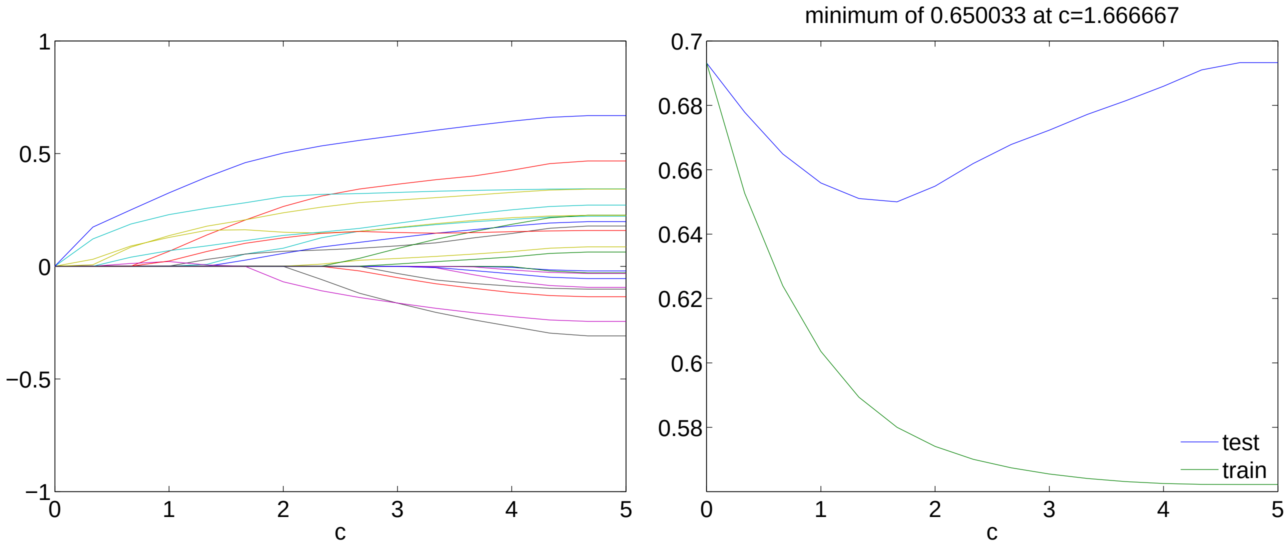
The figures below show the results of fitting linear models on the "SPECT heart" dataset[5]. Notice that in these plots, we perform validation in terms of the same loss used for training. If we are only using the logistic or hinge loss for computational convenience, we could also validate in terms of the 0-1 loss, since we search though all the regularization strengths explicitly.

[5] http://archive.ics.uci.edu/ml/datasets/SPECT+Heart

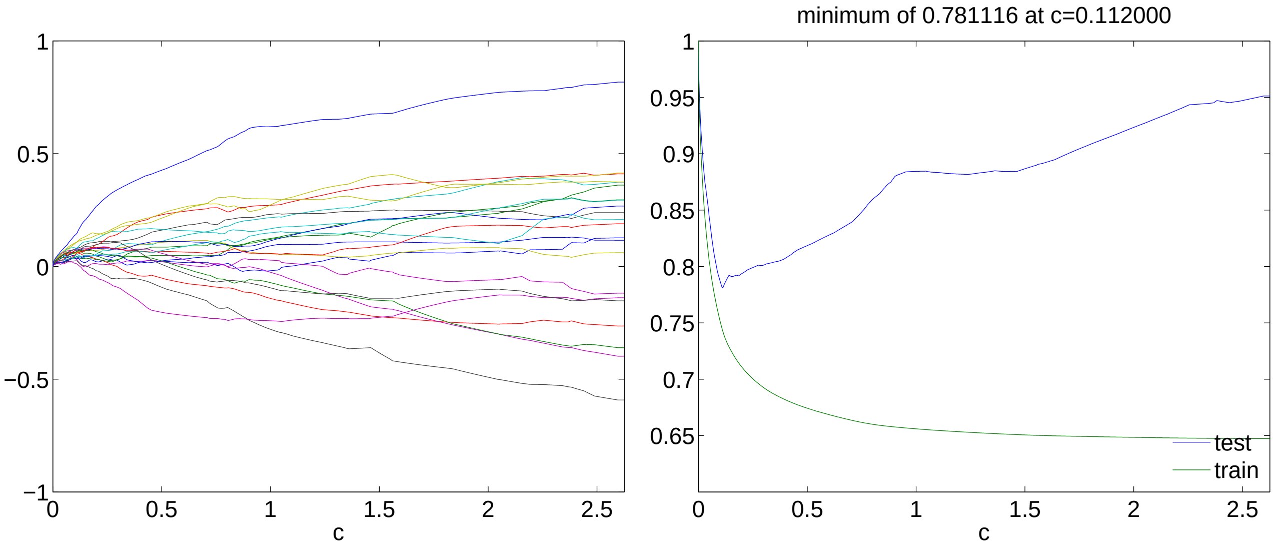
Logistic Loss + Ridge Penalty



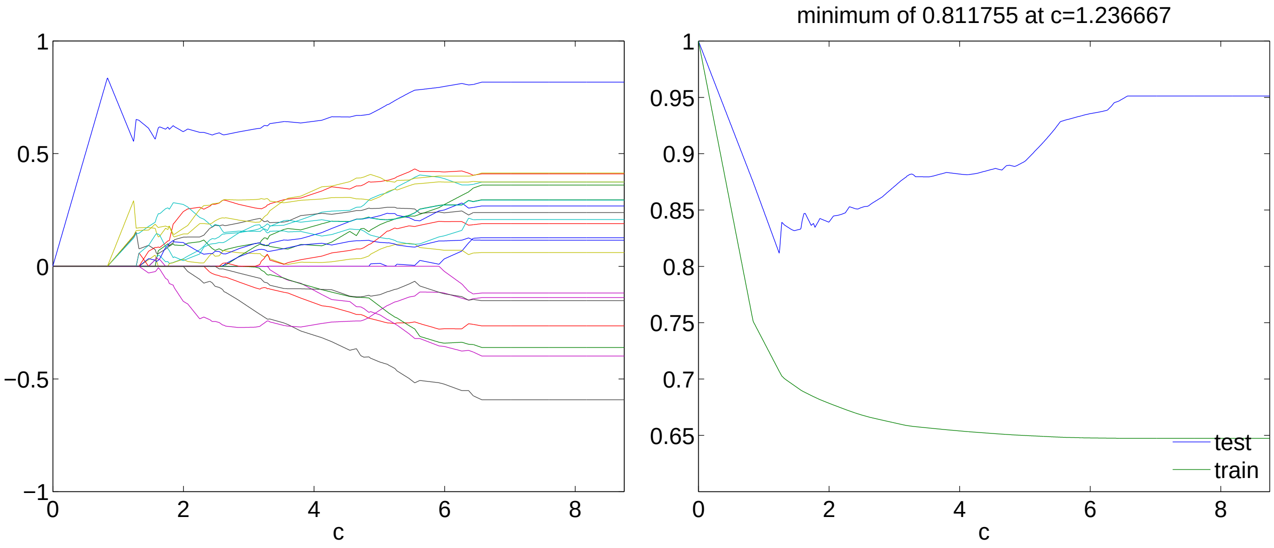
Logistic Loss + Lasso Penalty



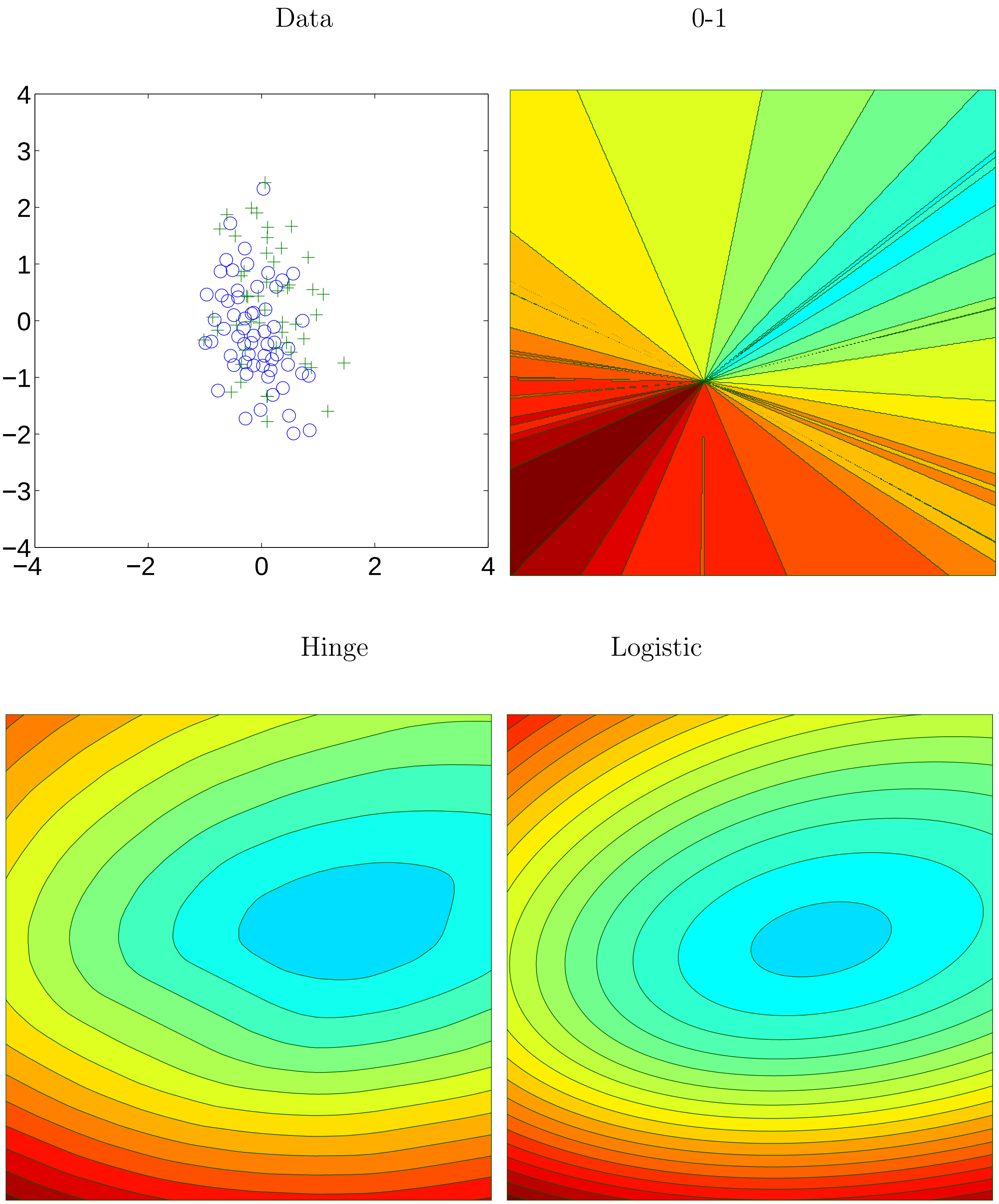
Hinge Loss + Ridge Penalty



Hinge Loss + Lasso Penalty



Here is an example dataset, along with the empirical risks for all three of the above loss functions. Here, for the sake of visualization, $(0, 0)$ is towards the lower-left.



Data

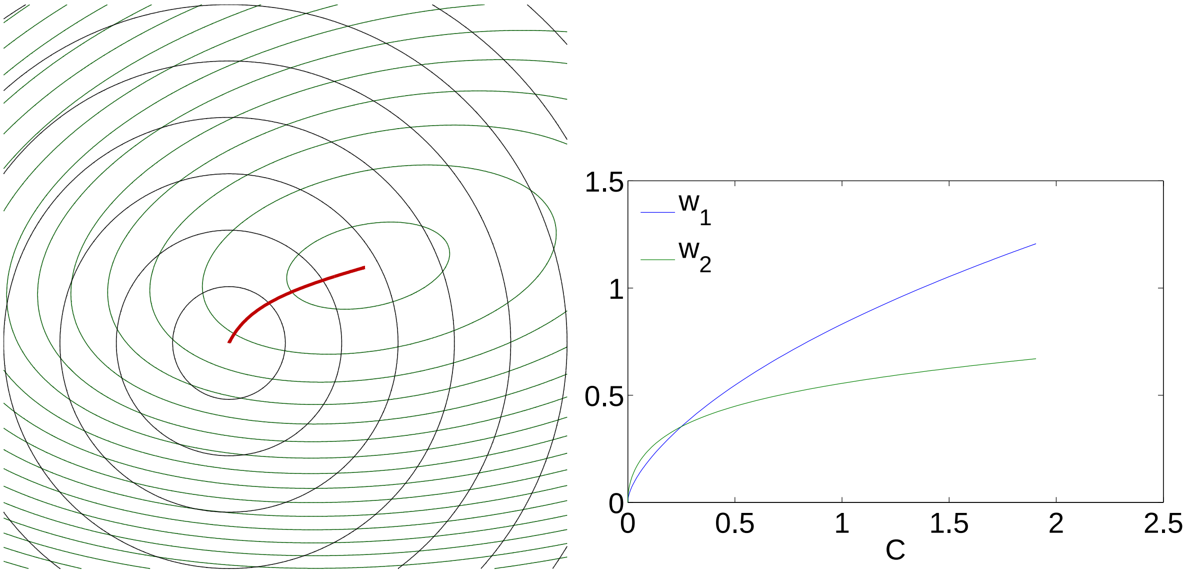
0-1

Hinge

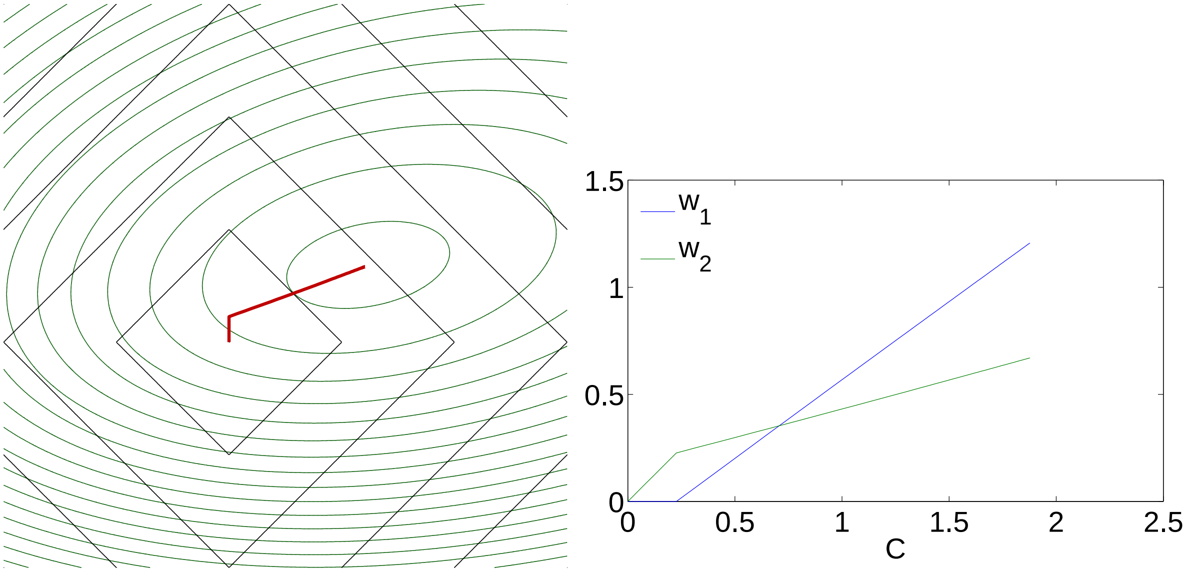
Logistic

The 0-1 loss is clearly unsmooth. Below we show the full regularization path for the hinge and logistic risks, with the ridge and lasso penalties. As we see above, the logistic risk closely resembles the least-squares risk we saw above. So, it is not too surprising that the regularization paths also look similar.

Logistic + Ridge Penalty

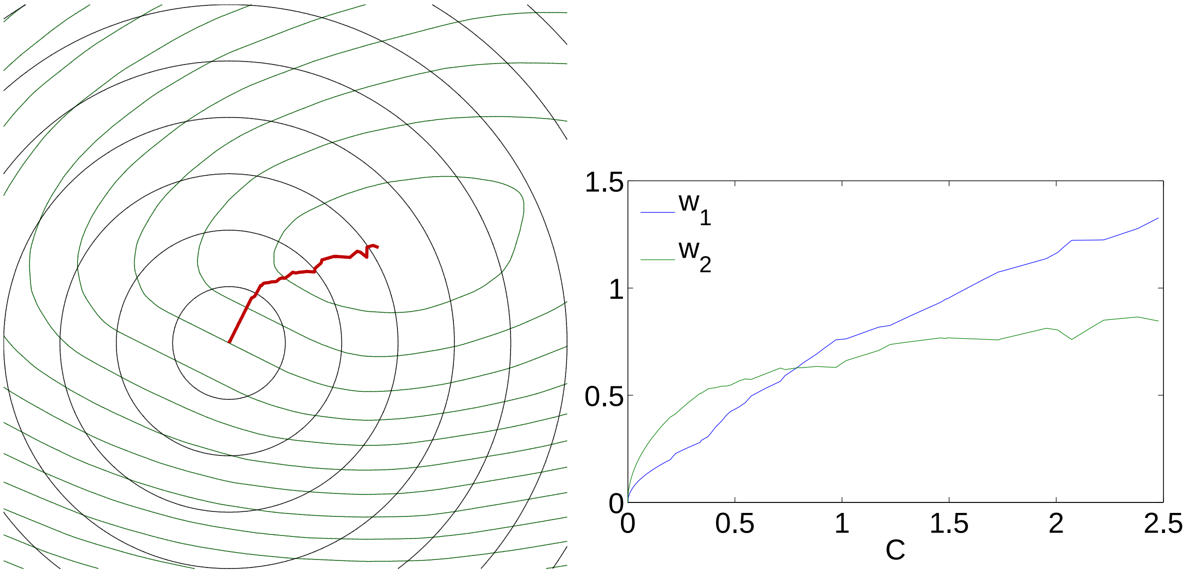


Logistic + Lasso Penalty

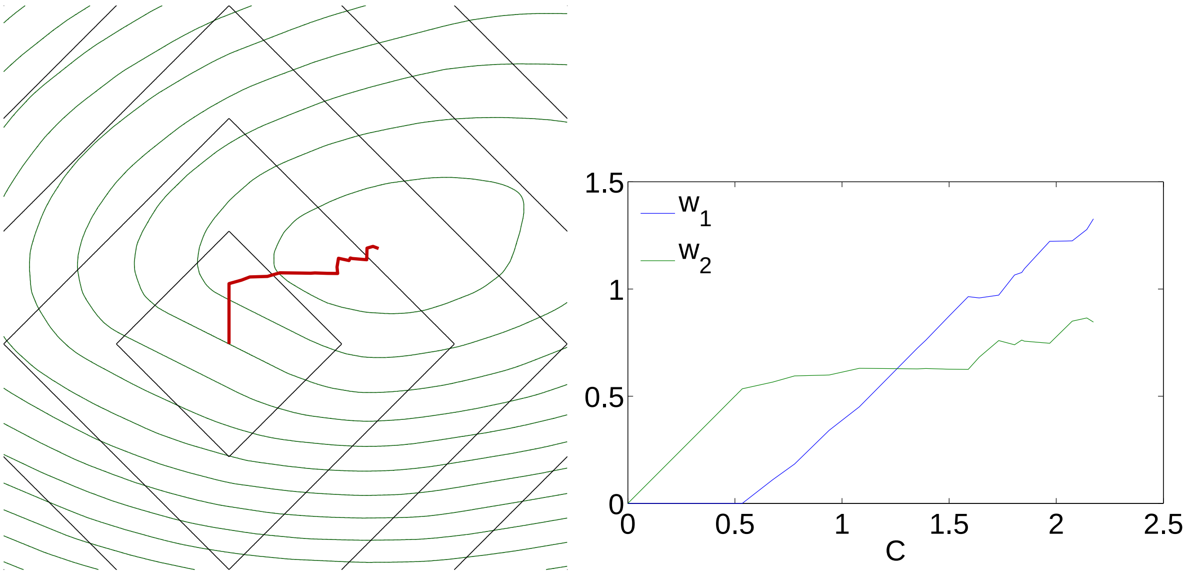


The hinge loss, however, gives quite complicated curves.

Hinge + Ridge Penalty



Hinge + Lasso Penalty



7 Ridge regression and random noise

It turns out that linear regression with a ridge penalty can be seen as equivalent to adding some random noise to the data. In a single dimension, consider the normal distribution

$$P(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{X^2}{2\sigma^2}).$$

The key observation is the following

$$\mathop{\mathbb{E}}_{X \sim P(X)} \left[(wX)^2\right] = \sigma^2 w^2.$$

(You can check this by doing the integral.)

Now, suppose that we are doing linear regression with a single input variable. If we take our dataset, and we append a bunch of random data taken from a normal distribution, along with outputs of $Y = 0$, we will asymptotically approach the results of ridge regularization.

If we are to do this, there are two things that we can control:

1. The number of random data that we append to the dataset.
2. The variance σ of the gaussian that we draw the data from.

Similarly, there are two issues that we have to deal with:

1. How good is the randomized approximation of ridge regularization?
2. How strong is the ridge penalty?

The right way to deal with this is to use the number of data to achieve an approximation quality, and to set σ to control the amount of penalty. That is, use *as much* data as is computationally convenient, and then set the variance of the data to achieve the desired degree of penalization.

8 Algorithms for linear regression and classification

Above, we have talked about what optimization problems different loss functions and regularizers produce, but we have not talked about how to solve them. A very general strategy would be to apply a "stock" convex optimization solver, which will work for any combination of the above losses and regularization penalties (except the 0-1 classification loss). The main disadvantage of this is that it can be quite slow. Research is still ongoing to find faster algorithms tailored to specific situations. We will discuss just a few examples here.

First, consider unregularized least-squares regression. The goal is to find $\mathbf{w}$ to minimize

$$R(\mathbf{w}) = \sum_{\hat{y},\hat{\mathbf{x}}} \big(\mathbf{w}\cdot\hat{\mathbf{x}} - \hat{y}\big)^2.$$

If we set the gradient $dR/d\mathbf{w}$ to zero, we find

$$\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{\mathbf{x}}^T\mathbf{w} - \sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y} = \mathbf{0},$$

and so

$$\boxed{\mathbf{w} = \big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{\mathbf{x}}^T\big)^{-1}\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big).}$$

Thus, the weights can be recovered by solving one linear system. Now, suppose that we were doing ridge regression. The goal is now to minimize

$$R(\mathbf{w}) + \lambda h(\mathbf{w}) = \sum_{\hat{y},\hat{\mathbf{x}}} \big(\mathbf{w}\cdot\hat{\mathbf{x}} - \hat{y}\big)^2 + \lambda\frac{1}{2}\mathbf{w}\cdot\mathbf{w}.$$

Again, if we want to minimize this with respect to $\mathbf{w}$, we can just set the derivative to zero. This gives us

$$\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{\mathbf{x}}^T\mathbf{w} - \sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y} + \lambda\mathbf{w} = \mathbf{0},$$

with the solution

$$\boxed{\mathbf{w} = \big(\lambda I + \sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{\mathbf{x}}^T\big)^{-1}\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big).}$$

Again, we can recover the weights by solving one linear system. However, what if we want to solve this for a large range of different λ, say $\lambda = .01, .02, ..., 10$? Does this mean we have to suffer the computational expense of solving 1000 linear systems? Actually, through appropriate cleverness, we can avoid this and essentially solve the system for *all* the λ in the same expense as solving for one. First, recall the definition of Singular Value Decomposition.

This is a factorization of a matrix A into the product of three matrices, $A = USV^T$. SVD is useful because U, S, and V are all our favorite kinds of matrices. U and V are orthonormal, and S is diagonal. It turns out that for symmetric matrices, $U = V$. Now, what happens if we have an SVD of the matrix $\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{\mathbf{x}}^T$? We can do a bunch of algebra, and get

$$\begin{aligned}
\mathbf{w} &= \big(\lambda I + USU^T\big)^{-1}\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big) \\
&= \big(\lambda UU^T + USU^T\big)^{-1}\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big) \\
&= \big(U\big(\lambda I + S\big)U^T\big)^{-1}\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big) \\
&= U\big(\lambda I + S\big)^{-1}U^T\big(\sum_{\hat{y},\hat{\mathbf{x}}} \hat{\mathbf{x}}\hat{y}\big). \qquad (8.1)
\end{aligned}$$

This last line is great news. Since the two matrices inside the inverse are diagonal, the inverse is trivial to compute. Thus, for any given λ, we have more or less a “closed-form” solution for the weights. Computing the SVD to obtain S and U will be far more expensive than doing the matrix-vector multiplies necessary to recover $\mathbf{w}$ from Eq. 8.1. This is an example of what is called a “regularization path” algorithm, because it computes the entire “path” of $\mathbf{w}$ as λ changes. There are some recent algorithms that can compute regularization paths in some other situations, for example lasso regularization.

Unfortunately, the above analysis above basically only works for ridge regression. What we did above is set the derivative of the risk (or regularized risk) with respect to $\mathbf{w}$ to zero, and then solve. However many of the losses (least absolute deviation, hinge) and regularizers (lasso) we discussed are non-differentiable, meaning this strategy cannot work. The logistic loss, while smooth, does not admit a simple closed-form solution. For the logistic loss under ridge regularization, we can solve the problem quite well using standard unconstrained optimization tools (gradient descent or Newton's method). For almost all the other cases, we have to deal with non-differentiablity. We discuss some of the solutions here:

1. **Quadratic programming**. Any of the combination of the least-squares, least absolute deviation, and hinge losses and the ridge, lasso, elastic net, and l_∞ regularization penalties can be reformulated as a quadratic program or QP, for which specialized solvers have been developed. A QP is a quadratic objective, to be minimized under linear equality and inequality constraints, for example,

$$\begin{aligned}
\min_{\mathbf{x}} \quad & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \qquad (8.2) \\
\text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\
& E\mathbf{x} = \mathbf{d}.
\end{aligned}$$

When Q is positive semidefinite (meaning $\mathbf{x}^T Q \mathbf{x} \geq 0$ for all $\mathbf{x}$) these problems can be solved reliably. Notice that here we have written the problem as a function of a single variable $\mathbf{x}$, but problems involving multiple variables are fundamentally the same. For example, if we have a problem involving variables $\mathbf{y}$ and $\mathbf{z}$, we can re-express it in the form of Eq. 8.2 by writing $\mathbf{x} = (\mathbf{y}, \mathbf{z})$, and creating Q, $\mathbf{c}$, etc. correspondingly. Most standard QP software requires that problems be input in the "standard form" of Eq. 8.2, which can be inconvenient. In this class, we will not worry about transforming to standard form. Any problem with a quadratic objective and linear constraints is a QP.

Let's consider an example, least absolute deviation regression under the ridge penalty. This is

$$\min_{\mathbf{w}} \quad \sum_i |y_i - \mathbf{w} \cdot \mathbf{x}_i| + \lambda \mathbf{w} \cdot \mathbf{w},$$

which doesn't immediately look like a QP. However, if we define z_i, we can constrain it to be at least as large as $|y_i - \mathbf{w} \cdot \mathbf{x}_i|$ by enforcing that $z_i \geq y_i - \mathbf{w} \cdot \mathbf{x}_i$ and $z_i \geq -y_i + \mathbf{w} \cdot \mathbf{x}_i$. Then, noting that $\lambda \mathbf{w} \cdot \mathbf{w} = \mathbf{w}^T (\lambda I) \mathbf{w}$, we have the problem

$$\begin{aligned} \min_{\mathbf{w},\mathbf{z}} \quad & \mathbf{w}^T (\lambda I) \mathbf{w} + \mathbf{1}^T \mathbf{z}, \\ \text{s.t.} \quad & z_i \geq y_i - \mathbf{w} \cdot \mathbf{x}_i \\ & z_i \geq \mathbf{w} \cdot \mathbf{x}_i - y_i. \end{aligned}$$

(Here z_i is free to be larger than $|y_i - \mathbf{w} \cdot \mathbf{x}_i|$, but since the problem is to minimize $\mathbf{1}^T \mathbf{z}$, there is no profit in doing this at the solution.) In general, QP solvers are quite robust, so tranforming a problem into this form is reliable. However, in many cases faster methods are possible. (This was the method used to generate the figures in these notes, some of which took several hours to run.)

2. **Projected Gradient Descent.** Recall from the optimization notes that projected gradient descent solves a problem of the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in C, \end{aligned}$$

by iteratively taking a gradient step and then projecting back into the constraint set C. This method is attractive when there happens to exist a fast algorithm for *projecting* onto the set C, that is finding $\min_{\mathbf{x} \in C} ||\mathbf{x} - \mathbf{y}||$. This can be profitably applied to problems with ridge, lasso, or l_∞ regularization, where we use $C = \{\mathbf{w} : h(\mathbf{w}) \leq c\}$. The methods for ridge and l_∞ regularization are pretty simple, while the lasso is less obvious. It was also recently shown[6] that this can be done for elastic net regularization.

[6] http://www.cs.berkeley.edu/~jduchi/projects/proj_elastic_net.pdf

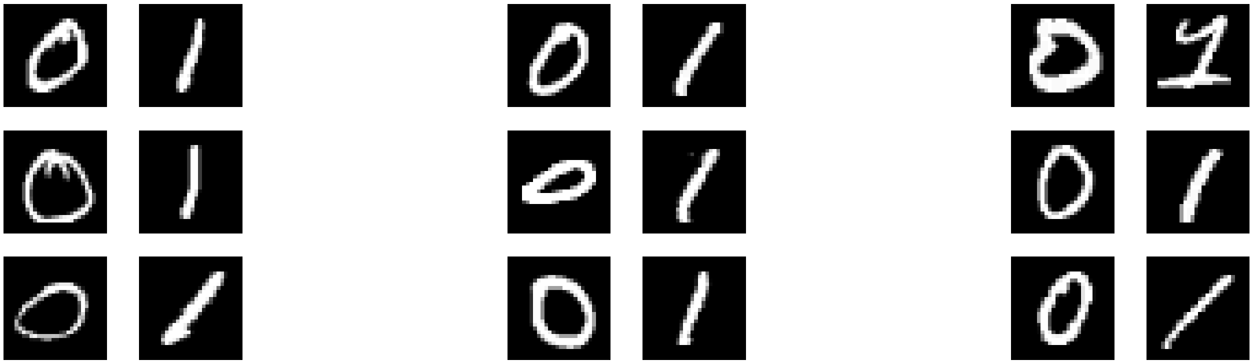
3. **Coordinate Descent.** The basic idea of coordinate descent is to minimize some function $f(\mathbf{x})$ by iterating through the "coordinates" x_i of $\mathbf{x}$ and repeatedly setting x_i to minimize f.

$$x_i \leftarrow \arg\min_{x_i'} f(x_1, x_2, ..., x_{i-1}, x_i', x_{i+1}, ..., x_N).$$

   This is usually most attractive when the minimizing over a single variable like this can be done quickly and exactly. Like gradient descent, coordinate descent doesn't usually converge at the fastest rate, in terms of the number of iterations. Also like gradient descent, it can still be faster than more sophisticated optimization methods when each iteration can be completed much more quickly. Some care needs to be taken to prove that coordinate descent will converge to the optimum value[7].

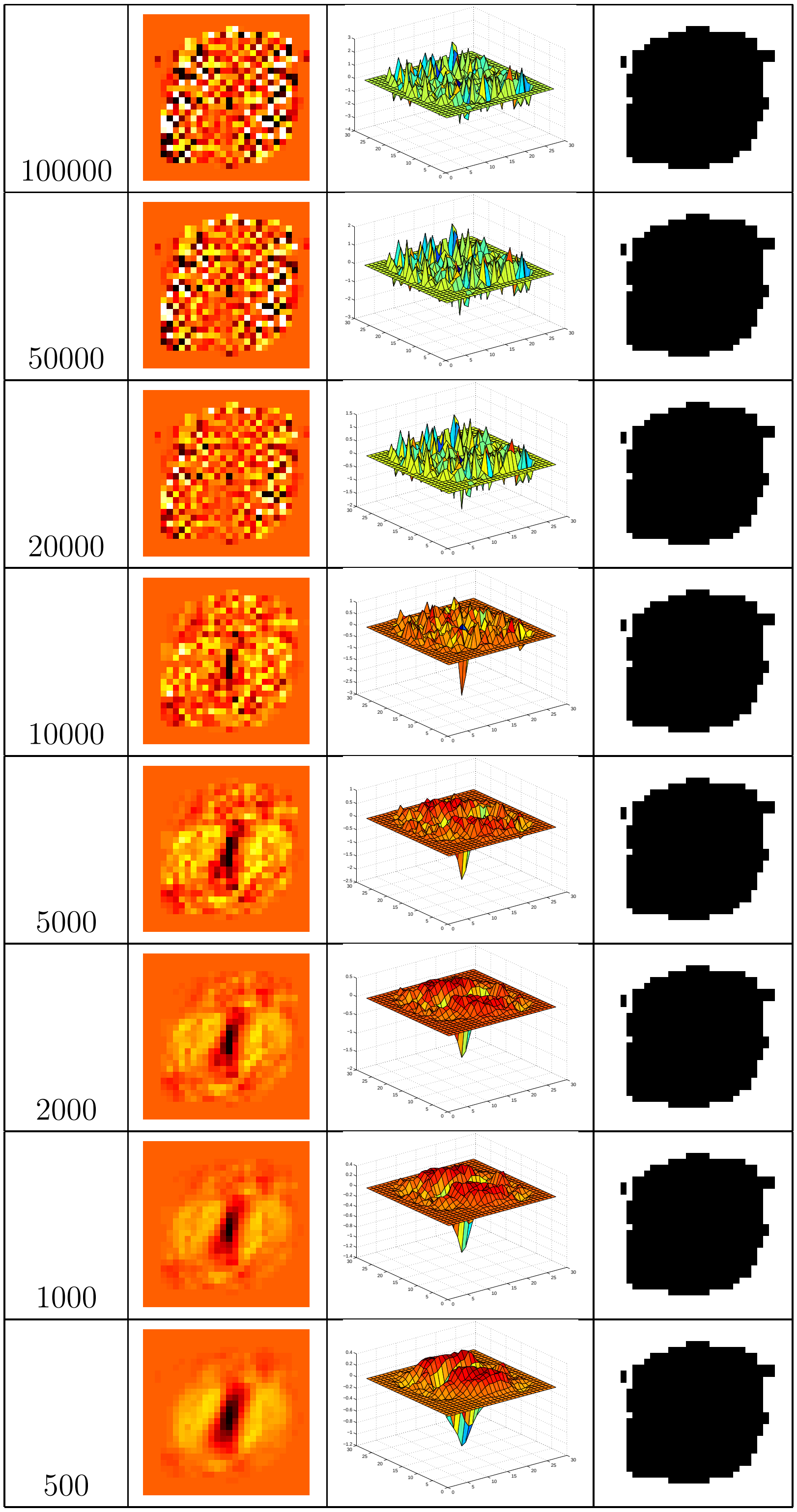
9 MNIST Demo

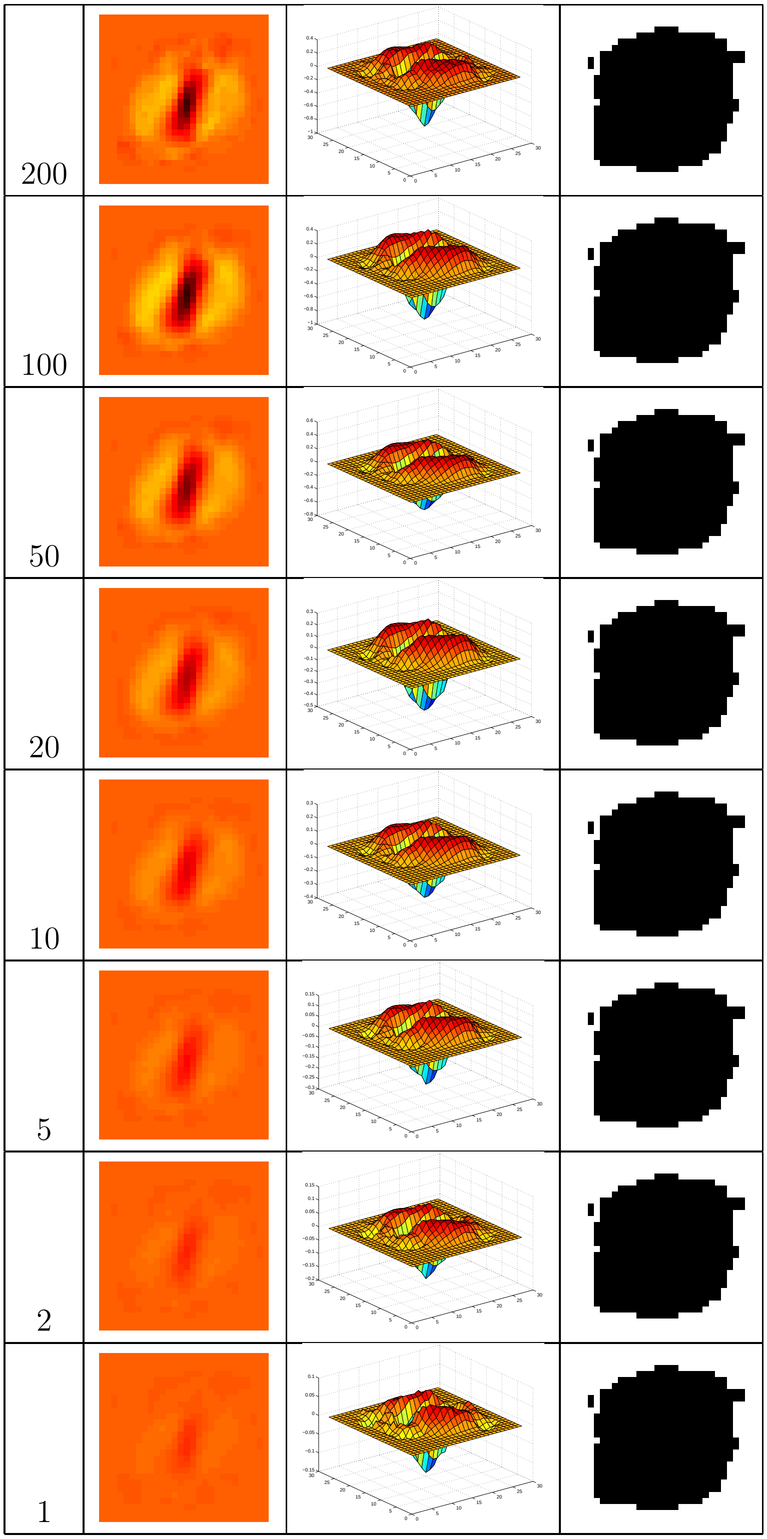
This section fits a binary classifier using the hinge loss to the MNIST dataset. Here, we consider the binary task of distinguishing 1s and 0s.



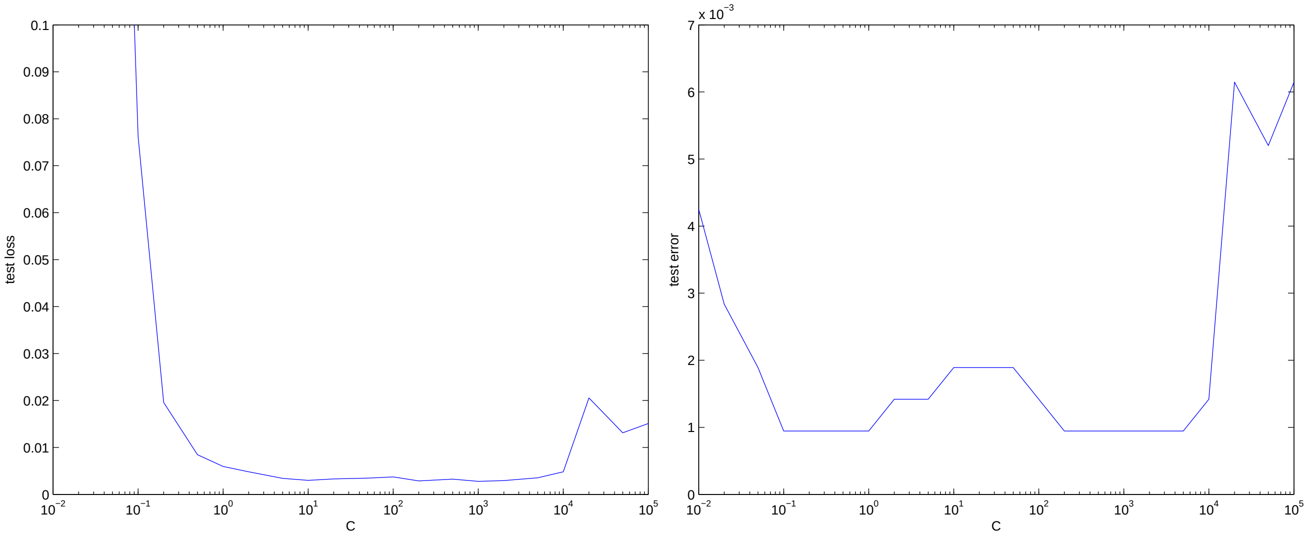
First, ridge regularization. The fist column shows the constant c constraining $h(\mathbf{w}) \leq c$.

[7] Tseng (1988, 2001) showed that coordinate descent converges for problems of the form $f(\mathbf{x}) = f_0(\mathbf{x}) + \sum_i f_i(x_i)$ when f_i are convex, and f_0 is differentiable and convex. Thus, coordinate descent can tolerate non-differentiability, *as long as it is along the directions of the coordinates.* (Of course, coordinate descent is not guaranteed *not* to converge on problems violating these conditions.)

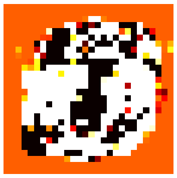
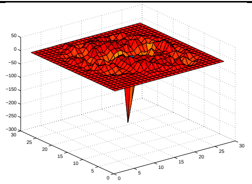
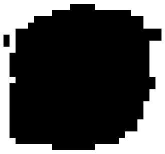
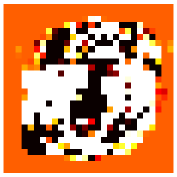
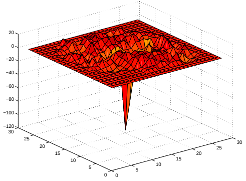
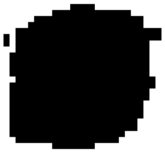
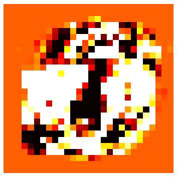
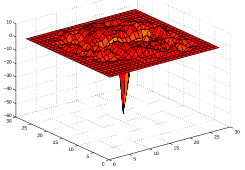
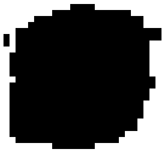
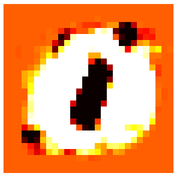
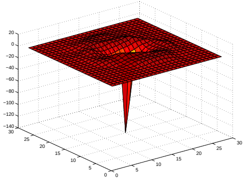
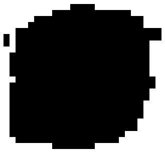
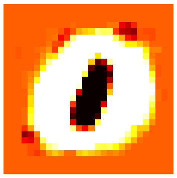
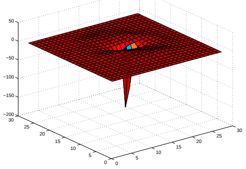
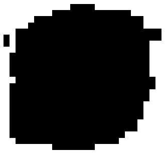
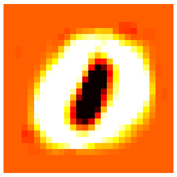
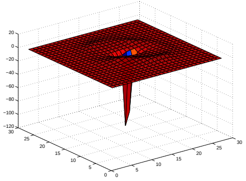
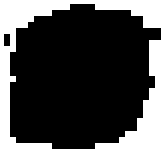
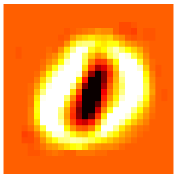
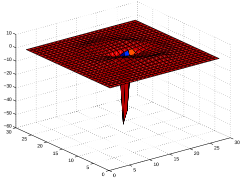
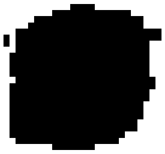
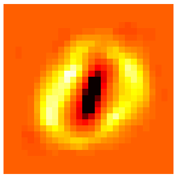
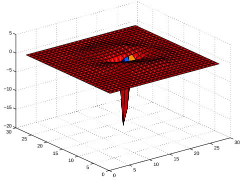
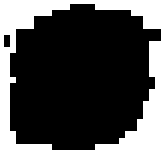

100000
50000
20000
10000
5000
2000
1000
500

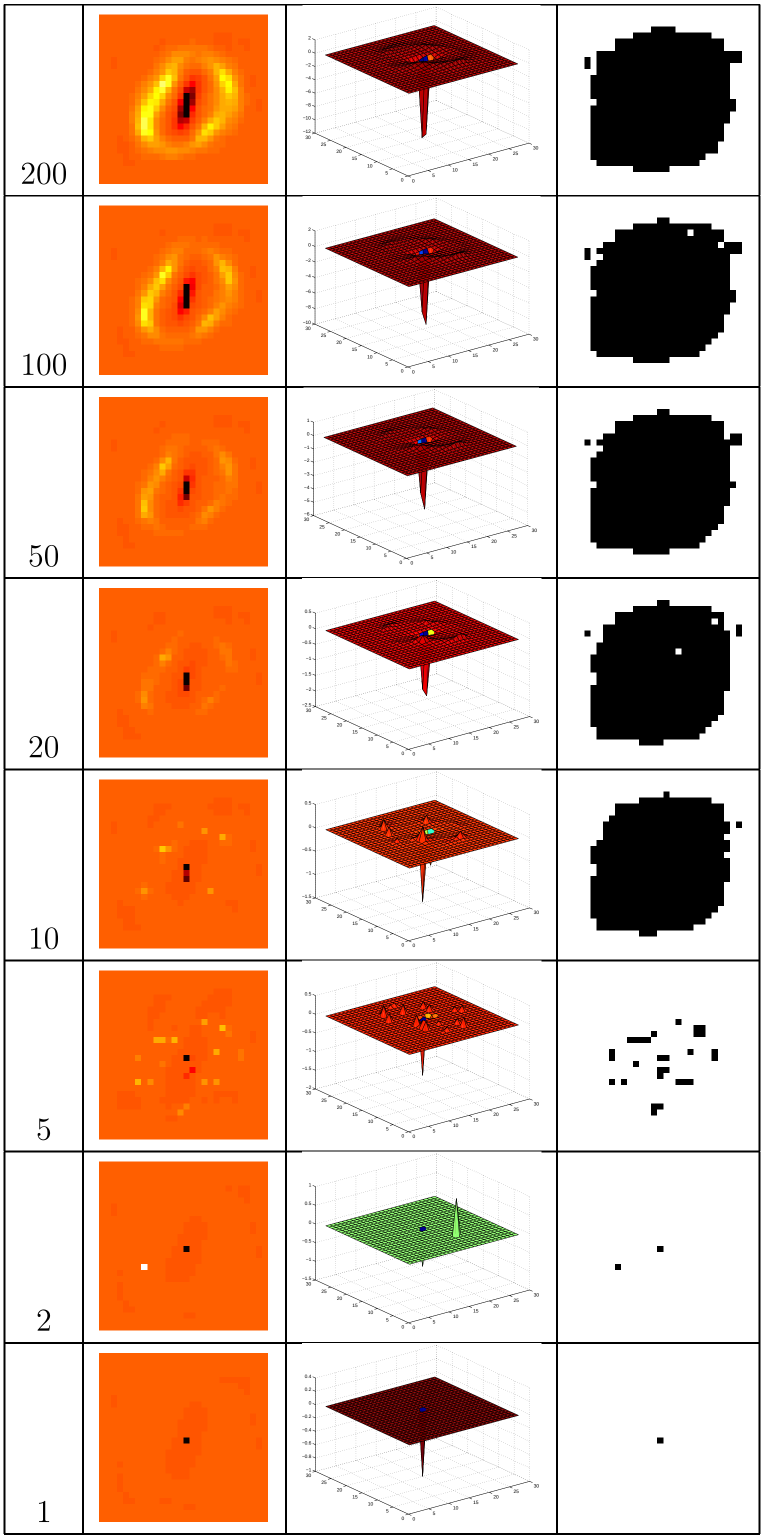


Now, we can plot the test loss and error as a function of the regularization constant.


test loss
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
C
x 10^{-3}
test error
7
6
5
4
3
2
1
0
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
C

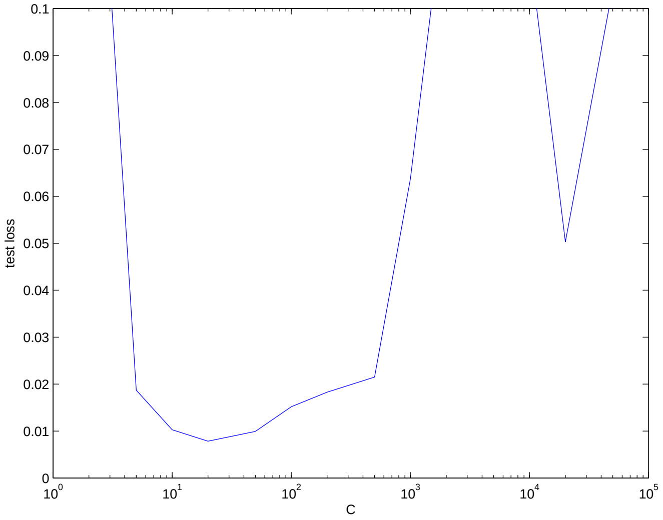
Next up, lasso regularization.

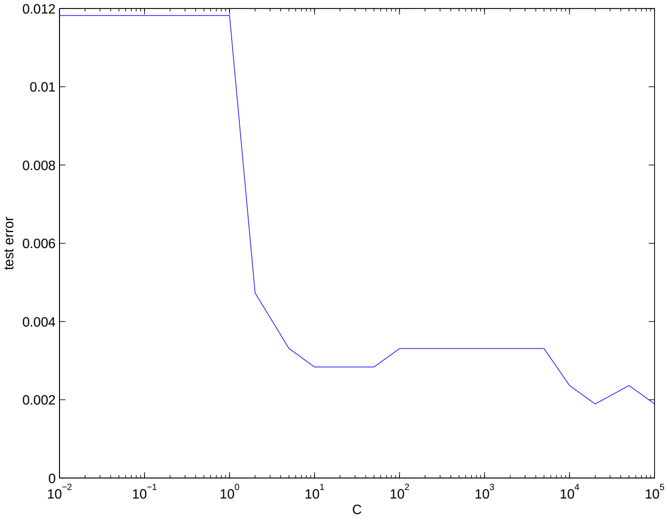
100000			
50000			
20000			
10000			
5000			
2000			
1000			
500			



The final result is somewhat amusing: the simplest way to differentiate 0s and 1s is simply to check if the center pixel is nonzero.

Again, we can plot the test loss and error as a function of the regularization constant.


test loss
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
C


test error
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
C