Statistical Machine Learning Notes 2

Empirical Risk Minimization and Optimization

Instructor: Justin Domke

1 Empirical Risk Minimization

Empirical Risk Minimization is a fancy sounding name for a very simple concept. Supervised learning can usually be seen as picking one function f from a set of possible functions F. An obvious questions is, how can we tell a good function f from a bad one? These notes introduce a general framework that applies for

First, we introduce the concept of a **loss function** L. Given some particular pair of inputs $\mathbf{x}$ and outputs $\mathbf{y}$,

$$L(f(\mathbf{x}), \mathbf{y})$$

tells us how much it "hurts" to make the prediction $f(\mathbf{x})$ when the true output is $\mathbf{y}$.

Now, let us define the (true) **risk**

$$R_{\text{true}}(f) = E_p\big[L(f(\mathbf{x}), \mathbf{y})\big] = \int\int p(\mathbf{x}, \mathbf{y}) L(f(\mathbf{x}), \mathbf{y}) d\mathbf{x} d\mathbf{y}.$$

Here p is the *true* distribution over the inputs $\mathbf{x}$ and $\mathbf{y}$. The risk measures how much, on average, it hurts to use f as our prediction algorithm.

This can all be made clear by considering an example. Suppose we want to fit a function for predicting if it will rain or not. The input $\mathbf{x}$ will be the sky: CLEAR, CLOUDY, or MIXED. The output $\mathbf{y}$ will be either, RAIN (when it rains) or NOPE (when it doesn't rain). The loss function is now a function $L : \{\text{RAIN}, \text{NOPE}\}^2 \to \Re$.

What loss function is appropriate? It is important to realize that this cannot be answered by math. The loss function depends on the priorities of the user. For example, if you are a person who really hates getting wet, but doesn't particularly mind carrying an umbrella on a clear day, you might use a loss function like:

$L_{\text{hate-rain}}(Y_1, Y_2)$

$Y_1 \backslash Y_2$	RAIN	NOPE
RAIN	0	1
NOPE	25	0

Meaning you hate getting rained on 25 times as much as carrying an umbrella on a clear day. On the other hand, someone who loses things frequently might hate carrying an umbrella on a clear day. They might have a loss function more like:

$L_{\text{hate-umbrellas}}(Y_1, Y_2)$

$Y_1 \backslash Y_2$	RAIN	NOPE
RAIN	0	1
NOPE	1	0

Now, let's suppose that the true distribution p is given as follows:

$p(\mathbf{x}, \mathbf{y})$

$\mathbf{x} \backslash \mathbf{y}$	RAIN	NOPE
CLEAR	0	1/4
CLOUDY	1/4	0
MIXED	1/6	1/3

Let's consider two possible prediction functions

$$f_1(\mathbf{x}) = \begin{cases} \text{CLEAR} & \text{NOPE} \\ \text{CLOUDY} & \text{RAIN} \\ \text{MIXED} & \text{NOPE} \end{cases}$$

$$f_2(\mathbf{x}) = \begin{cases} \text{CLEAR} & \text{NOPE} \\ \text{CLOUDY} & \text{RAIN} \\ \text{MIXED} & \text{RAIN} \end{cases}$$

If we use $L_{\text{hate-rain}}$, it is easy to calculate that $R(f_1) = 1/6 \cdot 25$, and $R(f_2) = 1/6 \cdot 1$, and so f_2 has the lower risk. Meanwhile, if we use $L_{\text{hate-umbrellas}}$, we can calculate $R(f_1) = 1/6 \cdot 1$, and $R(f_2) = 1/3 \cdot 1$, and so f_1 has the lower risk.

So, it sounds like the thing to do is to pick f to minimize the risk. Trouble is, that is impossible. To calculate the risk, we would need to know the true distribution p. We don't have the true distribution– if we did, we wouldn't be doing machine learning. So, what should we do?

Since the data D comes from p, we should be able to get a reasonable approximation

$$E_p L(f(\mathbf{x}), \mathbf{y}) \approx \frac{1}{N} \sum_{\hat{\mathbf{x}}, \hat{\mathbf{y}}} L(f(\hat{\mathbf{x}}), \hat{\mathbf{y}}). \tag{1.1}$$

The right hand side of Eq. 1.1 is called the **empirical risk**.

$$R(f) = \frac{1}{N} \sum_{\hat{\mathbf{x}}, \hat{\mathbf{y}}} L(f(\hat{\mathbf{x}}), \hat{\mathbf{y}}).$$

Picking the function f^* that minimizes it is known as **empirical risk minimization**.

$$f^* = \arg\min_{f \in F} R(f)$$

Two points of caution here: First, how well the empirical risk approximates the true risk, obviously, will depend on the amount of data. Second, notice that empirical risk minimization has no safeguards to resist overfitting. When used in practice, it is usually necessary to perform some sort of model selection or regularization to make empirical risk minimization generalize well to new data. We will talk about these things much more later on, when discussing specific classifiers.

2 Convex Optimization

The rest of these notes discus specific algorithms for optimization. From now on, $f(\mathbf{x})$ denotes a generic function we want to optimize. Notice that this is different from the previous section, which used $f(\mathbf{x})$ to denote a possible predictor. The clash of notation is unfortunate, but both of these usages are very standard (and used in the supplementary readings).

A generic optimization problem is of the form

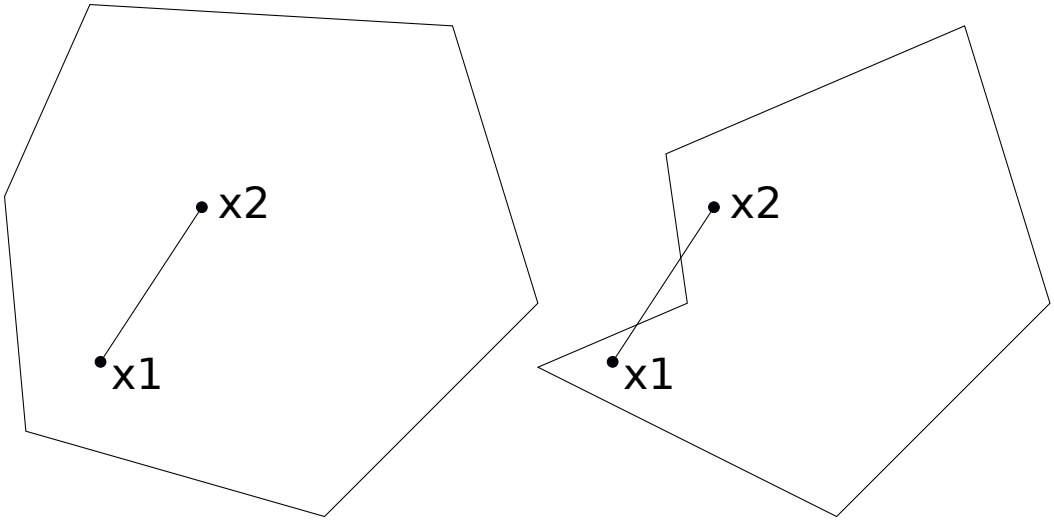
$$\begin{aligned} \min_{\mathbf{x}} . \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{x} \in C. \end{aligned} \tag{2.1}$$

That is, one should minimize the function f, subject to the constraint that $\mathbf{x}$ is in the set C. The trouble with a problem like this is that it is very easy to write down optimization problems like in Eq. 2.1 that are essentially impossible to solve. Convex optimization is a subset of optimizations that, roughly speaking, we can solve.

A set C is **convex** if for $\mathbf{x}_1$,$\mathbf{x}_2 \in C$,

$$\theta \mathbf{x}_1 + (1 - \theta)\mathbf{x}_2 \in C, \, 0 \leq \theta \leq 1.$$

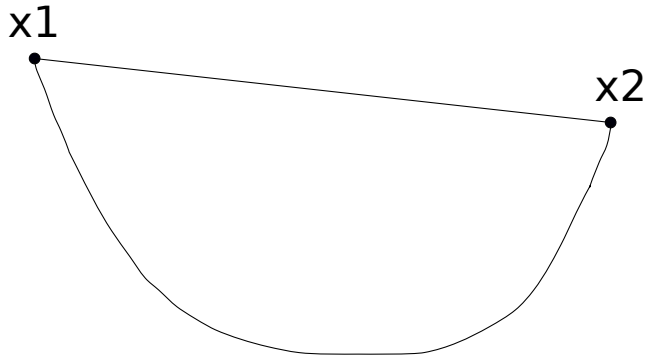
This has a simple geometrical interpretation: Given any two points in the set, the line segment joining them must be in the set. So the set below on the left is convex, while the one on the right is not.



Now, a function $f(\mathbf{x})$ is **convex** over C if for $\mathbf{x}_1$,$\mathbf{x}_2 \in C$,

$$f(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) \leq \theta f(\mathbf{x}_1) + (1-\theta)f(\mathbf{x}_2),\ 0 \leq \theta \leq 1.$$

Geometrically, this means the line joining any two function values must lie above the function itself.



Notice that a linear function is convex.

A function is called **strictly convex** if the inequality is strictly satisfied for all points not at the ends.

$$f(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) < \theta f(\mathbf{x}_1) + (1-\theta)f(\mathbf{x}_2),\ 0 < \theta < 1.$$

A linear function is *not* strictly convex.

Convex optimization problems can usually be solved reliably. The basic reason is that if you identify a local minima in a convex optimization problem, there is no possibility that some other, better, minima exists elsewhere.

Statistical Machine Learning Notes 3

Linear Methods

Instructor: Justin Domke

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 $\sum_{\hat{\mathbf{y}},\hat{\mathbf{x}}} \left(\hat{y} - f(\hat{\mathbf{x}})\right)^2$, or perhaps the sum of absolute differences $\sum_{\hat{\mathbf{y}},\hat{\mathbf{x}}} |\hat{y} - f(\hat{\mathbf{x}})|$? 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 $\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{(\hat{\mathbf{y}},\hat{\mathbf{x}})} L(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{\mathbf{y}}) + \lambda h(\mathbf{w})$.
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	$L(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{\mathbf{y}}) = \left(\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\right)^2$
Least Absolute Deviation	$L(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{\mathbf{y}}) = \left\lvert\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\right\rvert$

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

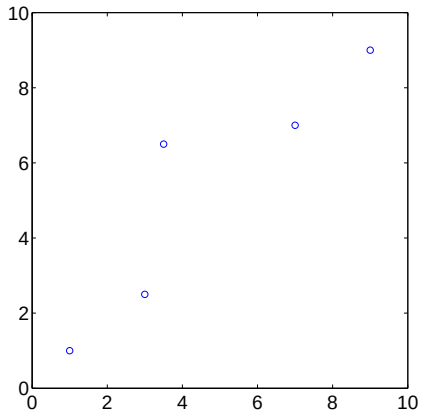
$$L_{\text{lsq}}(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{y}) = \big(\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\big)^2.$$

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

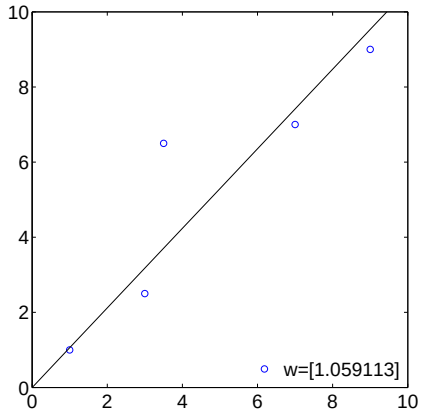
$$R(\mathbf{w}) = \sum_{\hat{y},\hat{\mathbf{x}}} L_{\text{lsq}}(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{y}) = \sum_{\hat{y},\hat{\mathbf{x}}} \big(\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\big)^2.$$

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



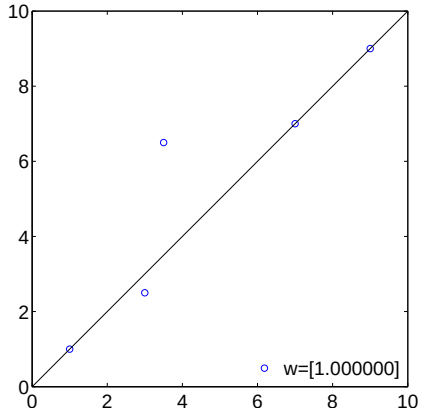
The least-squares solution is $w^* = 1.059113$.



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**,

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

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 minimizing (over f)

$$R(\mathbf{w}) = \sum_{\hat{y},\hat{\mathbf{x}}} L(\mathbf{w} \cdot \hat{\mathbf{x}}, \hat{y}),$$

instead minimize

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

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

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

(**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 corresponding to some λ. It is not hard to see that f^* will also be a solution to the second optimization 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_∞	$h(\mathbf{w}) =$	$\lvert\mathbf{w}\rvert_\infty$	$=$	$\max_i \lvert w_i\rvert$

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

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

$$\begin{aligned} \min_{\mathbf{w}} \quad & \sum_{\hat{y},\hat{\mathbf{x}}} (\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y})^2 \\ \text{s.t} \quad & \sum_i I[w_i \neq 0] \leq c. \end{aligned} \tag{4.1}$$

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.

- The computational issue is that Eq. 4.1 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*. (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

$$\sum_{\hat{y},\hat{\mathbf{x}}} \left(\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\right)^2 + \lambda \mathbf{w} \cdot \mathbf{w},$$

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

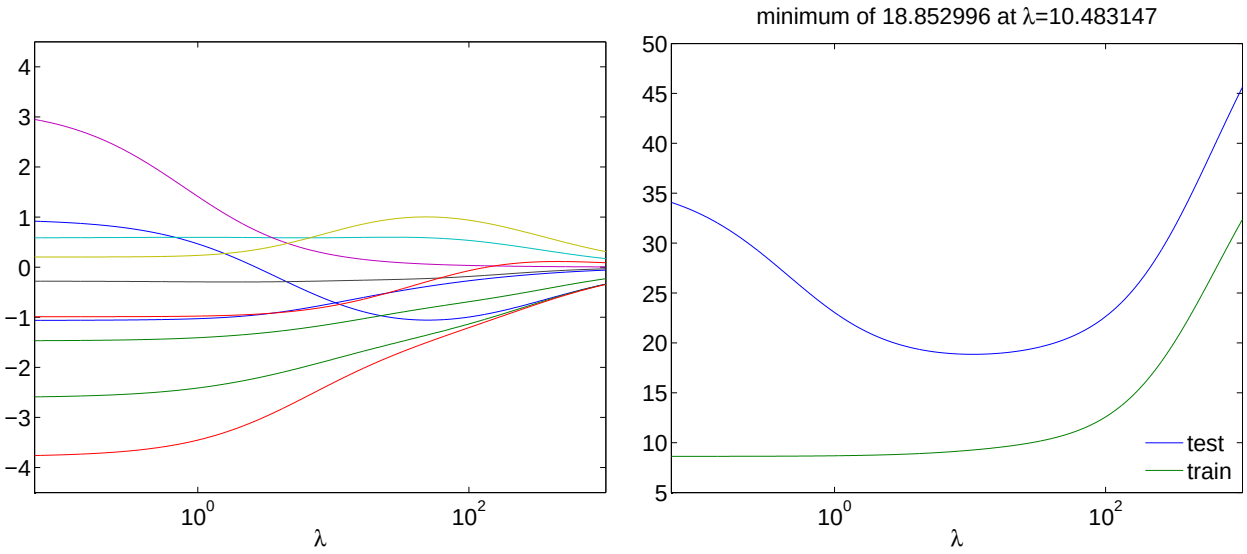
$$\sum_{\hat{y},\hat{\mathbf{x}}} \left(\mathbf{w} \cdot \hat{\mathbf{x}} - \hat{y}\right)^2 + \lambda |\mathbf{w}|_1. \tag{4.2}$$

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. 4.2 represents a convex optimization, and so it is possible to design reliable and efficient algorithms to find $\mathbf{w}$, even with very high dimensionality.

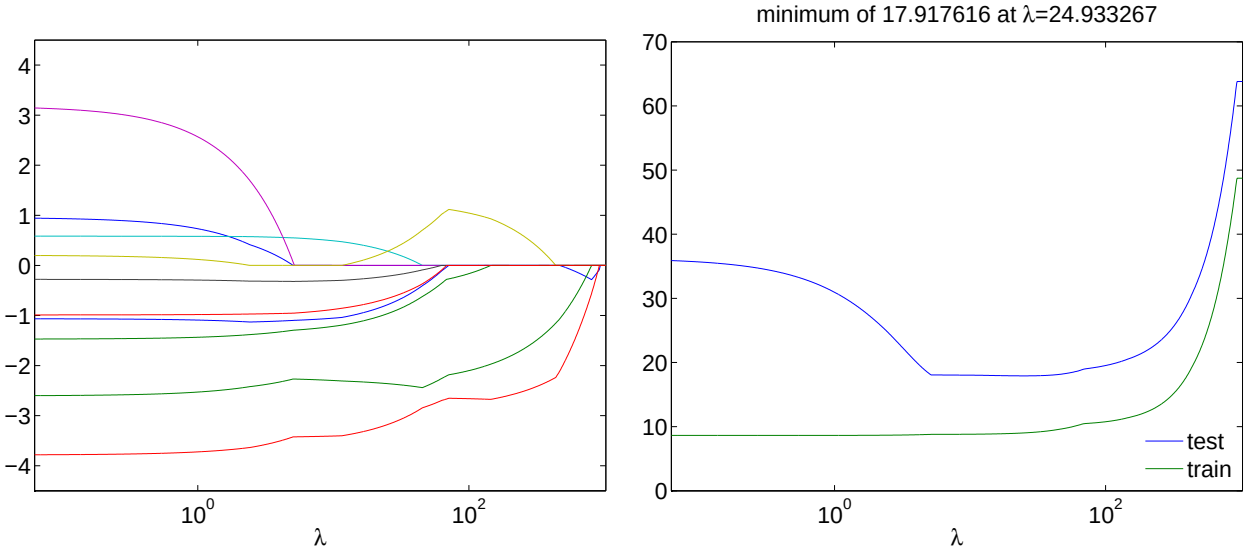
The figures below compare ridge and lasso regression on a "famous" dataset. In this dataset, $\hat{y}$ is the MPG of a vehicle, while the inputs encode the number of cylinders, the engine displacement, , the horsepower, the vehicle weight, acceleration speed, and origin (North America, Europe, or Japan)[3].

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

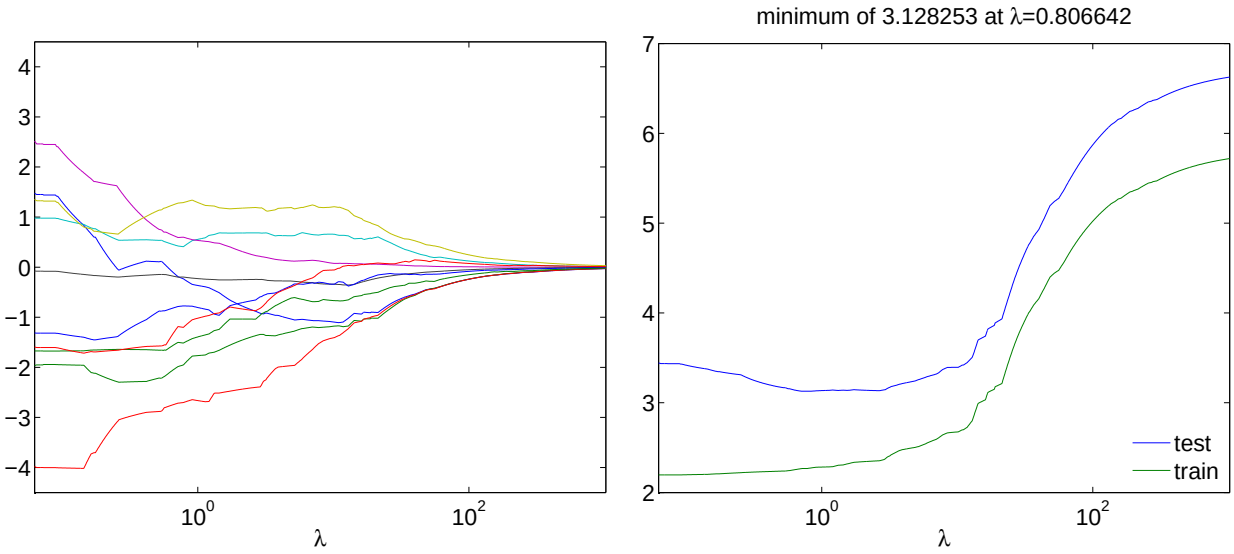
Least Squares Regression + Ridge Penalty



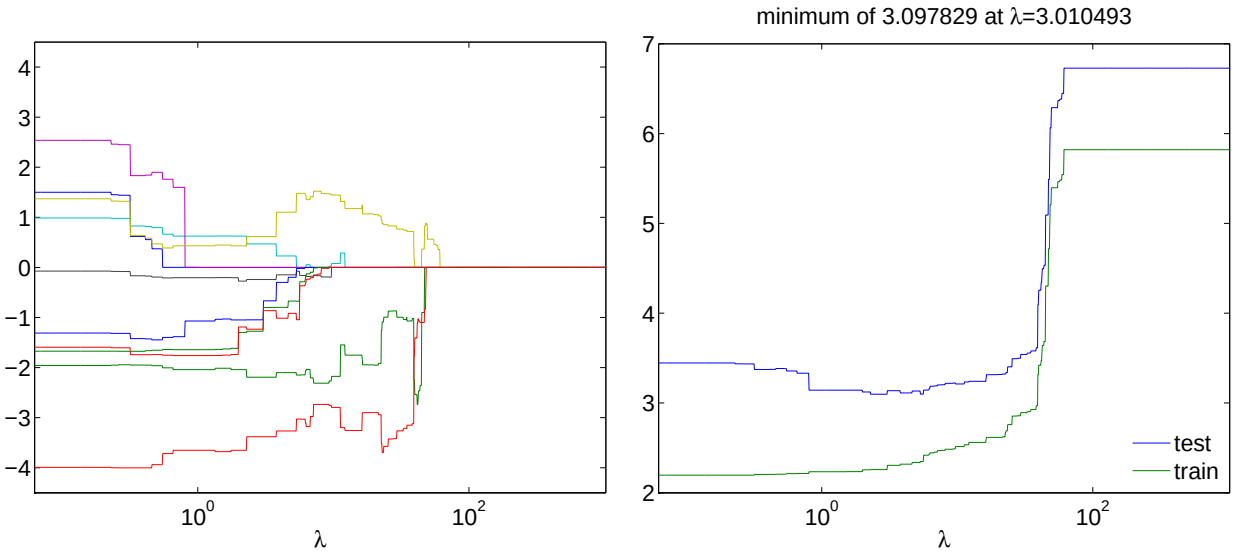
Least Squares Regression + Lasso Penalty



Least Absolute Deviation Regression + Ridge Penalty

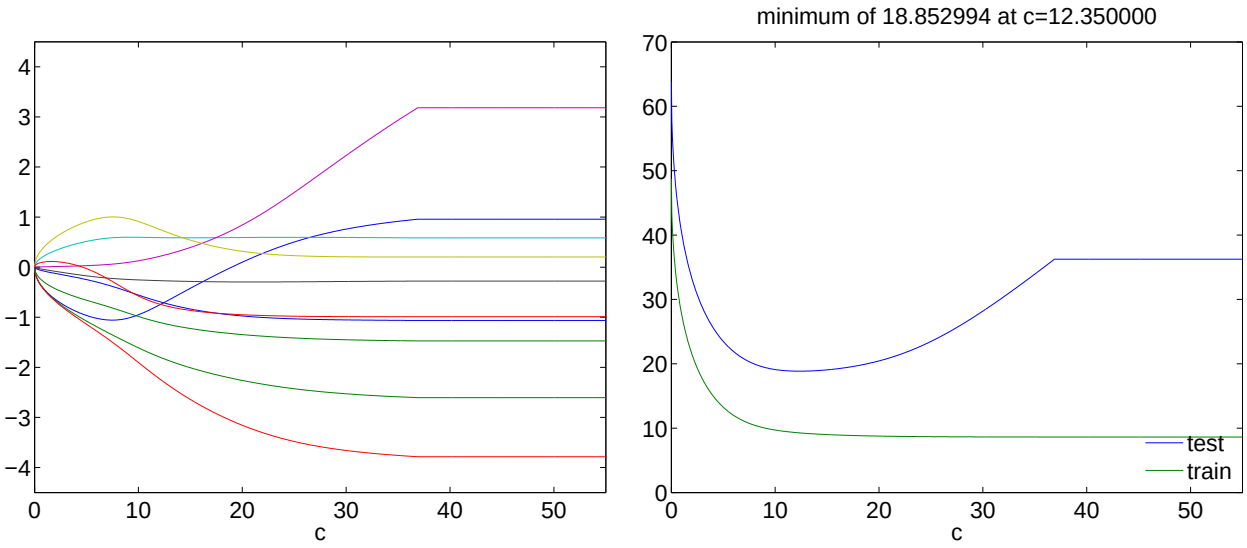


Least Absolute Deviation Regression + Lasso Penalty

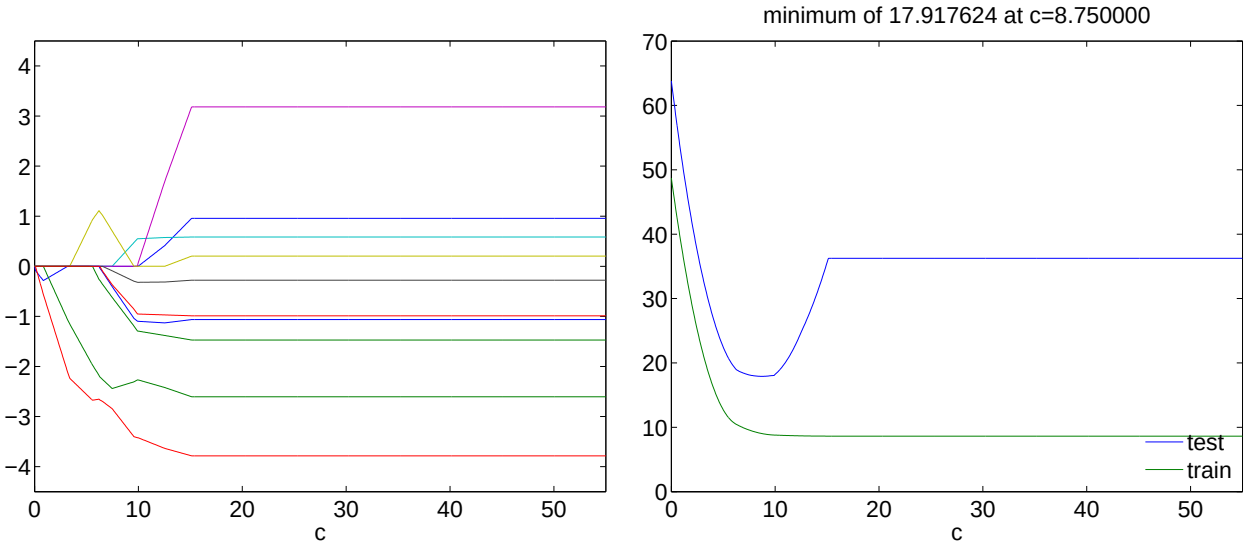


The next two pages show the same thing, but in terms of the constrained formulation $\min_{\mathbf{w}} R(\mathbf{w})$ s.t. $h(\mathbf{w}) \leq c$. If we watch carefully, we see that these figures sweep through the same paths as $c : 0 \to \infty$ as the previous figures do as $\lambda : \infty \to 0$.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

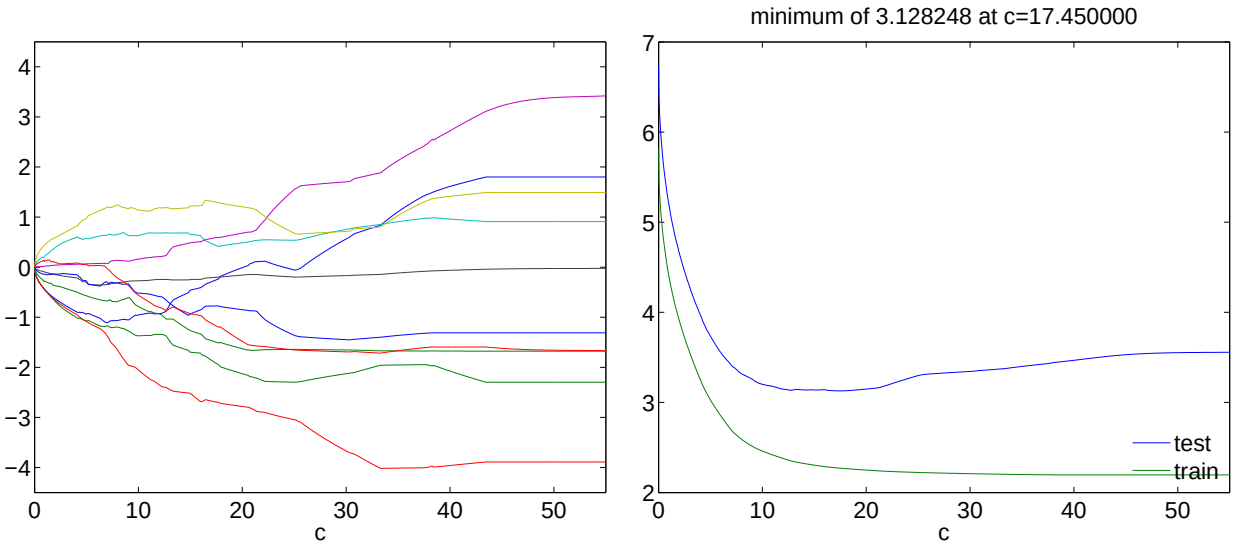
Least Squares Regression + Ridge Penalty



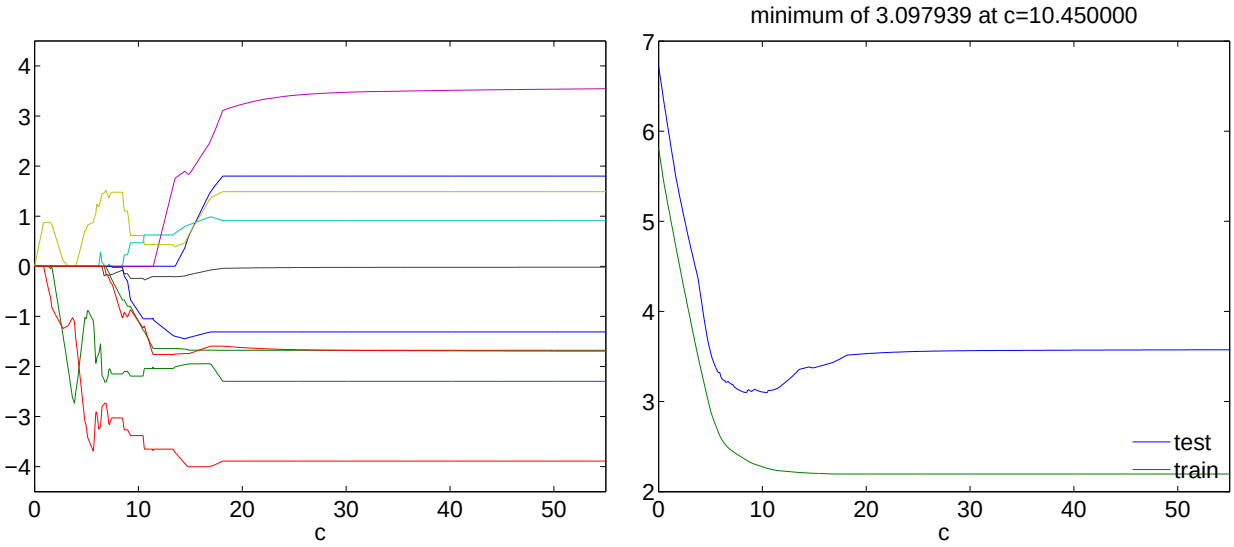
Least Squares Regression + Lasso Penalty



Least Absolute Deviation Regression + Ridge Penalty

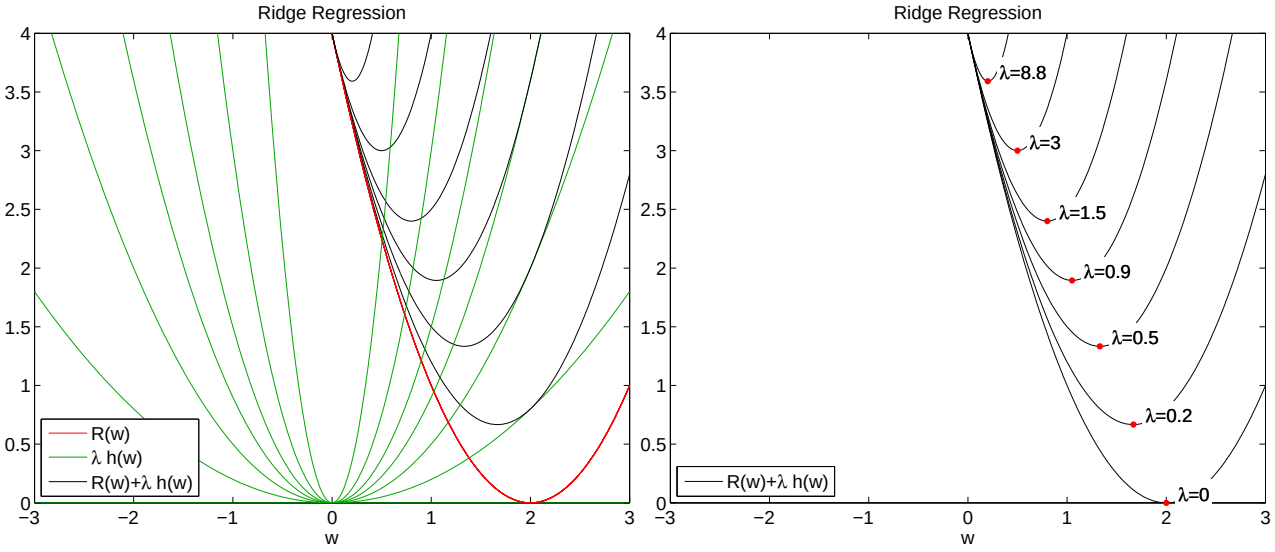


Least Absolute Deviation Regression + Lasso Penalty

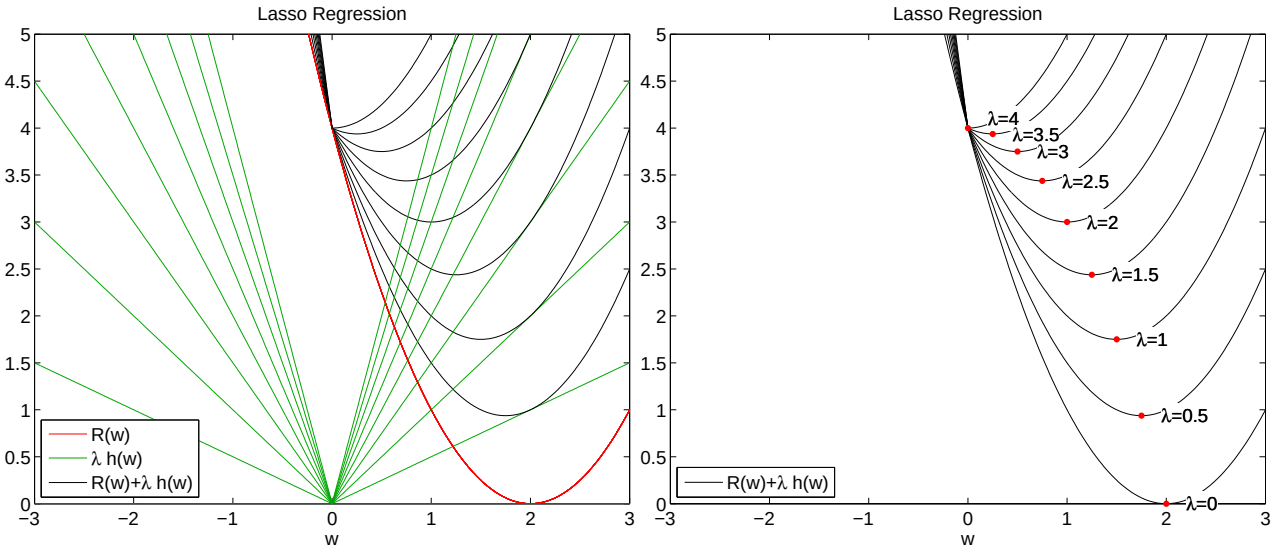


Naturally, for larger λ, all the weights go towards zero. For ridge regression they go towards zero smoothly, only reaching zero as $\lambda \to \infty$. For the lasso, on the other hand, the weights hit zero for a finite λ.

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)

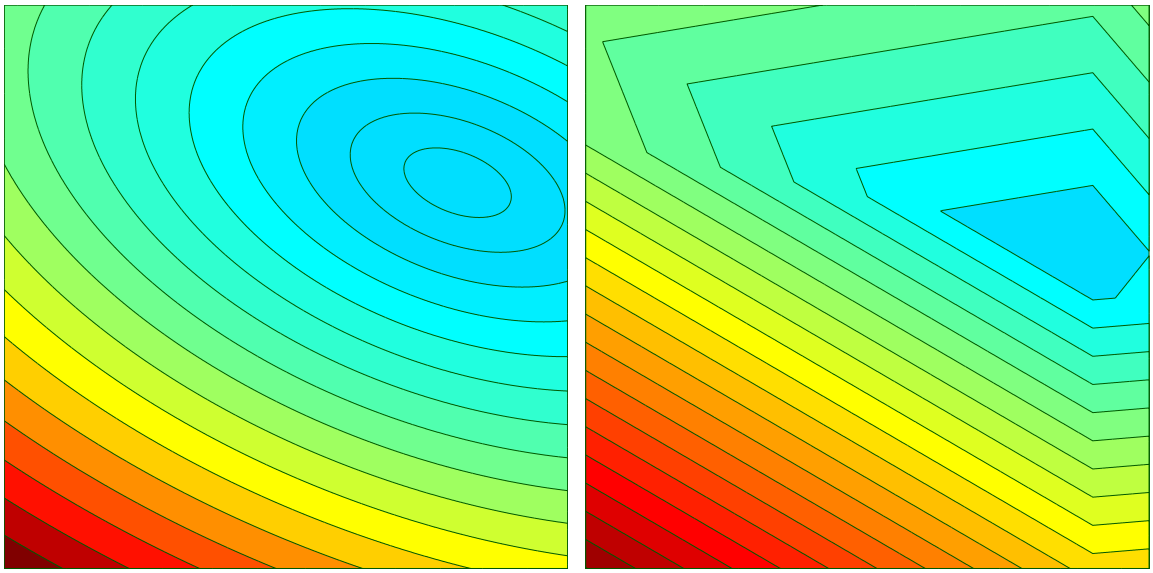
In a perfect world, we would like both of these to be small. Of course, we must ultimately choose how to trade-off between the two. With out specifying a trade-off parameter, we cannot choose the "best" solution $\mathbf{w}$. However, we still can rule our most! Why? Consider a set of weights $\mathbf{w}^*$. 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

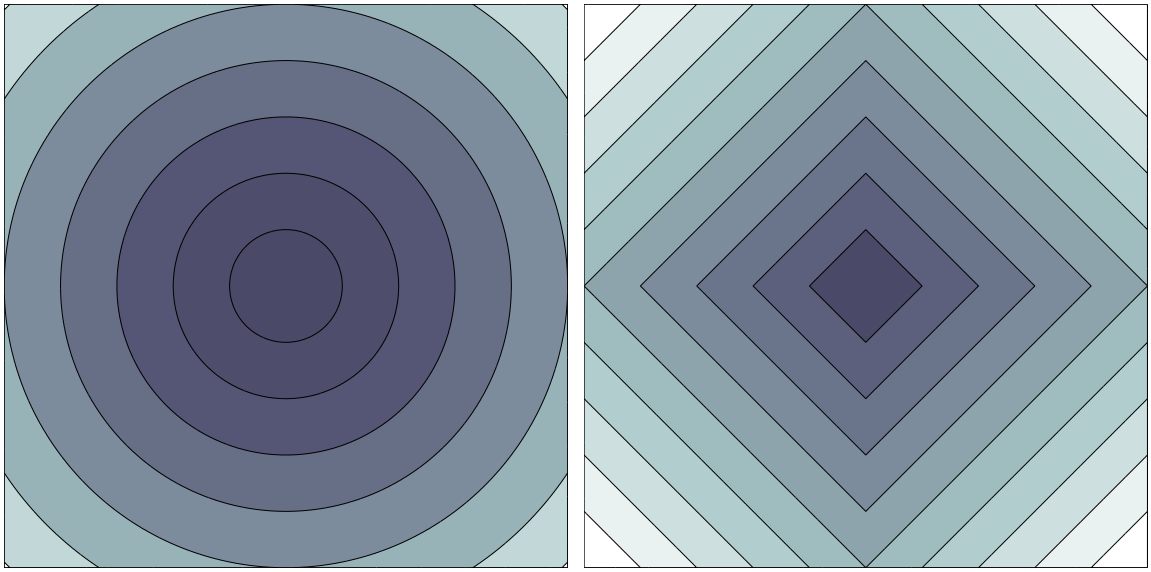
The first set of figures show $R(\mathbf{w})$ for both the least-squares loss, and the least-absolute-deviation loss. The least-squares loss is simply a quadratic in the space of $\mathbf{w}$, while least-absolute-deviation is a more complex piecewise linear function. Notice that the minima are in similar, 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.

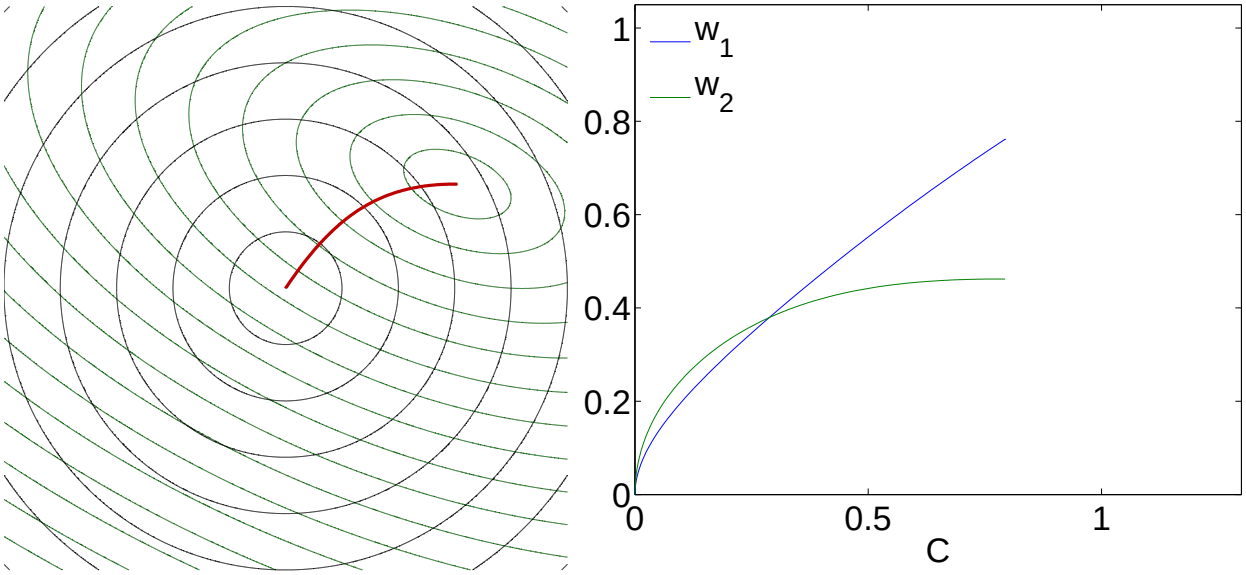
Ridge Penalty Lasso Penalty



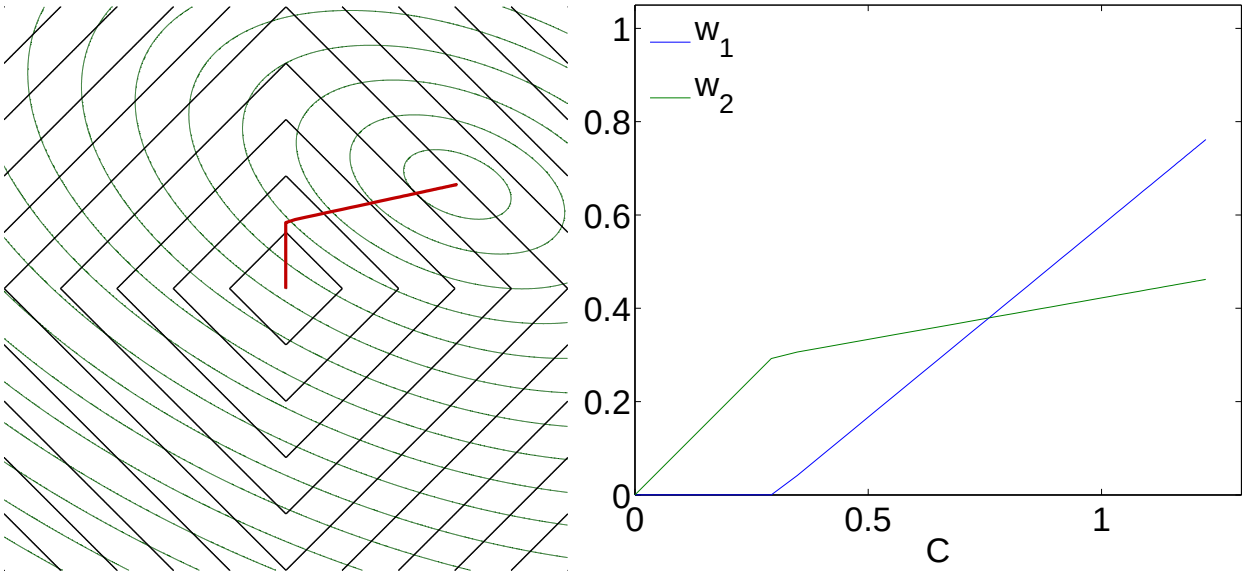
The next figures show each combination of the above risks (in green) along with the penalties (in black). The red curve shows the possible weights in each situation. To understand these

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.

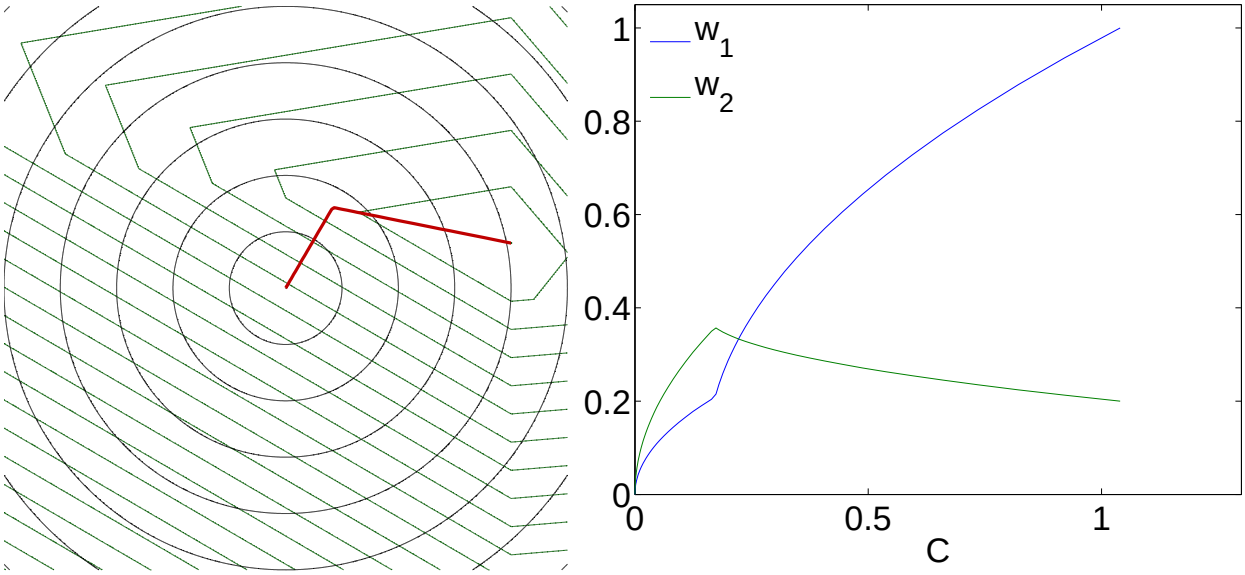
Least Squares + Ridge Penalty



Least Squares + Lasso Penalty



Least Absolute Deviation + Ridge Penalty



Least Absolute Deviation + Lasso Penalty