Machine Learning and Computational Statistics, Spring 2016
Homework 2: Lasso

Due: Monday, February 15, 2016, at 6pm (Submit via NYU Classes)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. You may include your code inline or submit it as a separate file. You may either scan hand-written work or, preferably, write your answers using software that typesets mathematics (e.g.

1 Preliminaries

1.1 Dataset construction

Start by creating a design matrix for regression with$m = 150$ examples, each of dimension$d = 75$. We will choose a true weight vector$\boldsymbol{\theta}$ that has only a few non-zero components:

1. Let$X \in \mathbf{R}^{m\times d}$ be the"design matrix," where the$i$'th row of$X$ is$x_i \in \mathbf{R}^d$. Construct a random design matrix$X$ using`numpy.random.rand()` function.

   Solution:

```
numData = 150
numDim = 75

# 1
X = np.random.rand(numData, numDim)
```

2. Construct a true weight vector$\boldsymbol{\theta} \in \mathbf{R}^{d\times 1}$ as follows: Set the first 10 components of$\boldsymbol{\theta}$ to 10 or -10 arbitrarily, and all the other components to zero.

   Solution:

```
#2
theta = np.zeros(numDim)
theta[0:10] = np.random.choice([-10, 10], size=10)
```

3. Let $y = (y_1, \dots, y_m)^T \in \mathbf{R}^{m\times 1}$ be the response. Construct the vector $y = X\boldsymbol{\theta} + \epsilon$, where ϵ is an $m \times 1$ random noise vector generated using `numpy.random.randn()` with mean 0 and standard deviation 0.1.

   Solution:

```python
#3
y =  X.dot(theta) + 0.1*np.random.randn(numData)
```

4. Split the dataset by taking the first 80 points for training, the next 20 points for validation, and the last 50 points for testing.

   Solution:

```python
#4
def splitData(data, numTrain,numValidate):
    data_train = data[0:numTrain]
    data_validation = data[numTrain:numTrain+numValidate]
    data_test = data[numTrain+numValidate:]
    return data_train, data_validation, data_test

numTrain = 80
numValidate = 20
numTest = 50

X_train, X_validation, X_test = splitData(X,numTrain, numValidate)
y_train, y_validation, y_test = splitData(y,numTrain, numValidate)
```

Note that we are not adding an extra feature for the bias in this problem. By construction, the true model does not have a bias term.

1.2 Experiments with Ridge Regression

By construction, we know that our dataset admits a sparse solution. Here, we want to evaluate the performance of ridge regression (i.e. ℓ_2-regularized linear regression) on this dataset.

1. Run ridge regression on this dataset. Choose the λ that minimizes the square loss on the validation set. For the chosen λ, examine the model coefficients. Report on how many components with true value 0 have been estimated to be non-zero, and vice-versa (don't worry if they are all nonzero). Now choose a small threshold (say 10^{-3} or smaller), count anything with magnitude smaller than the threshold as zero, and repeat the report. (For running ridge regression, you may either use your code from HW1, or you may use `scipy.optimize.minimize` (see the demo code provided for guidance). For debugging purposes, you are welcome, even encouraged, to compare your results to what you get from `sklearn.linear_model.Ridge`.)
   Solution:

```
# helper functions for ridge regression
def ridge(Lambda):
  # a function to generate objective functions
  # for ridge regression optimization
  def ridge_obj(theta):
    error_train = np.dot(X_train,theta) - y_train
    return ((np.linalg.norm(error_train))**2)\
           /(2*y_train.size) + Lambda*(np.linalg.norm(theta))**2
  return ridge_obj

def compute_loss(Lambda, theta):
    # a function to compute square loss on validation
    error = np.dot(X_validation,theta) - y_validation
  return ((np.linalg.norm(error_validation))**2)/(2*y_validation.size)

# ridge regression
w = np.random.rand(numDim,1)
results = []
thetaOut = []
lossOut = []
lambdaRange = 10.0**np.arange(-7,1)

for Lambda in lambdaRange:
    w_opt = minimize(ridge(Lambda), w)
    thetaOut.append(w_opt.x)
    lossOut.append(compute_loss(Lambda, w_opt.x))

idxMin = np.argmin(lossOut)
thetaOptimal = thetaOut[idxMin]
```

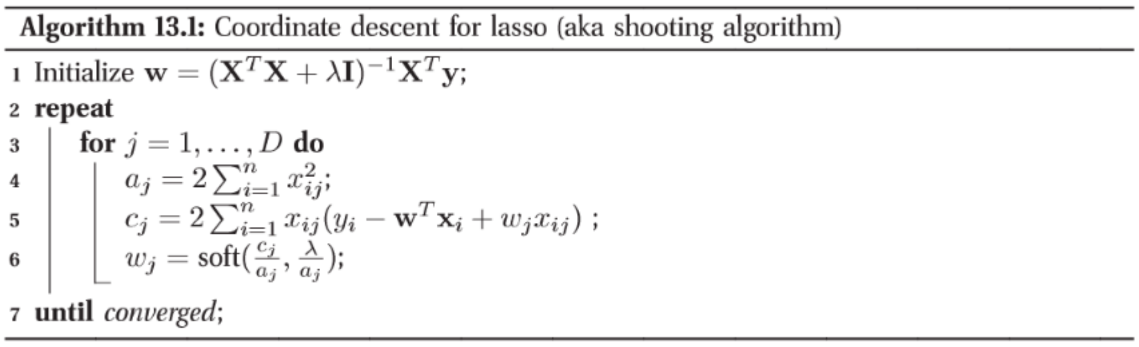
2 Coordinate Descent for Lasso (a.k.a. The Shooting algorithm)

The Lasso optimization problem can be formulated as

$$\hat{w} = \arg\min_{w \in \mathbf{R}^d} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + \lambda \|w\|_1,$$

where$h_w(x) = w^T x$, and$\|w\|_1 = \sum_{i=1}^{d} |w_i|$. Since the$\ell_1$-regularization term in the objective function is non-differentiable, it's not clear how gradient descent or SGD could be used to solve this optimization problem.

Another approach to solving optimization problems is coordinate descent, in which at each step we optimize over one component of the unknown parameter vector, fixing all other components. The descent path so obtained is a sequence of steps each of which is parallel to a coordinate axis in$\mathbf{R}^d$, hence the name. It turns out that for the Lasso optimization problem, we can find a closed form solution for optimization over a single component fixing all other components. This gives us the following algorithm:



Algorithm 13.1: Coordinate descent for lasso (aka shooting algorithm)

```
1 Initialize $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$;
2 repeat
3     for $j = 1, \ldots, D$ do
4         $a_j = 2\sum_{i=1}^{n} x_{ij}^2$;
5         $c_j = 2\sum_{i=1}^{n} x_{ij}(y_i - \mathbf{w}^T\mathbf{x}_i + w_j x_{ij})$ ;
6         $w_j = \text{soft}(\frac{c_j}{a_j}, \frac{\lambda}{a_j})$;
7 until converged;
```

(Source: Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.)

The"soft thresholding" function is defined as

$$\text{soft}\,(a, \delta) = \text{sign}(a)\,(|a| - \delta)_+ \,.$$

NOTE: Algorithm 13.1 does not account for the case that $a_j = c_j = 0$, which occurs when the jth column of X is the identically 0. One can either eliminate the column (as it cannot possibly help the solution), or you can set $w_j = 0$ in that case, as that is, in fact, the coordinate minimizer in that case.

Note that Murphy is suggesting to initialize the optimization with the ridge regession solution, though this is not necessary.

The solution should have a sparsity pattern that is similar to the ground truth. Estimators that preserve the sparsity pattern (with enough training data) are said to be**"sparsistent[1]"** (sparse + consistent). Formally, an estimator$\hat{\beta}$ of parameterβ is said to be consistent if the estimator$\hat{\beta}$ converges to the true valueβ in probability as our sample size goes to infinity. Analogously, if we define the support of a vectorβ as the indices with non-zero components, i.e.$\text{Supp}(\beta) = \{j \mid \beta_j \neq 0\}$, then an estimator$\hat{\beta}$ is said to be sparsistent if as the number of samples becomes large, the support of$\hat{\beta}$ converges to the support ofβ, or $\lim\limits_{m\to\infty} P[\text{Supp}(\hat{\beta}_m) = \text{Supp}(\beta)] = 1$.

There are a few tricks that can make selecting the hyperparameterλ easier and faster. First, you can show that for any$\lambda > 2\|X^T(y - \bar{y})\|_\infty$, the estimated weight vector$\hat{w}$ is entirely zero, where$\bar{y}$ is the mean of values in the vector$y$, and$\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum absolute value of any component of the vector. Thus we need to search for an optimalλ in$[0, \lambda_{\max}]$, where$\lambda_{\max} = 2\|X^T(y - \bar{y})\|_\infty$. (Note: This expression for$\lambda_{\max}$ assumes we have an unregularized bias term in our model. That is, our decision functions are$h_{w,b}(x) = w^Tx + b$. For the experiments, you can exclude the bias term, in which case$\lambda_{\max} = 2\|X^Ty\|_\infty$.)

Second, we can make use of the fact that whenλ andλ' are close, so are the corresponding solutions$\hat{w}(\lambda)$ and$\hat{w}(\lambda')$. Start by finding$\hat{w}(\lambda_{\max})$ and initialize the optimization at$w = 0$. Next,λ is reduced (e.g. by a constant factor), and the optimization problem is solved using the previous optimal point as the starting point. This is called**warm starting** the optimization. The entire

[1]Li, Yen-Huan, et al."Sparsistency of l_1-Regularized$M$-Estimators."

technique of computing a set of solutions for a chain of nearbyλ's is called a**continuation**or **homotopy method**. In the context of finding a good regularization hyperparameter, it may be referred to as a**regularization path**approach. (Lots of names for this!)

2.1 Experiments with the Shooting Algorithm

1. Write a function that computes the Lasso solution for a givenλ using the shooting algorithm described above. This function should take a starting point for the optimization as a parameter. Run it on the dataset constructed in (1.1), and select theλ that minimizes the square error on the validation set. Report the optimal value ofλ found, and the corresponding test error. Plot the validation error vsλ. [Don't use the homotopy method in this part, as we want to measure the speed improvement of homotopy methods in part3. Also, no need to vectorize the calculations until part4, where again we'll compare the speedup. In any case, having two different implementations of the same thing is a good way to check your work.]

   Solution:

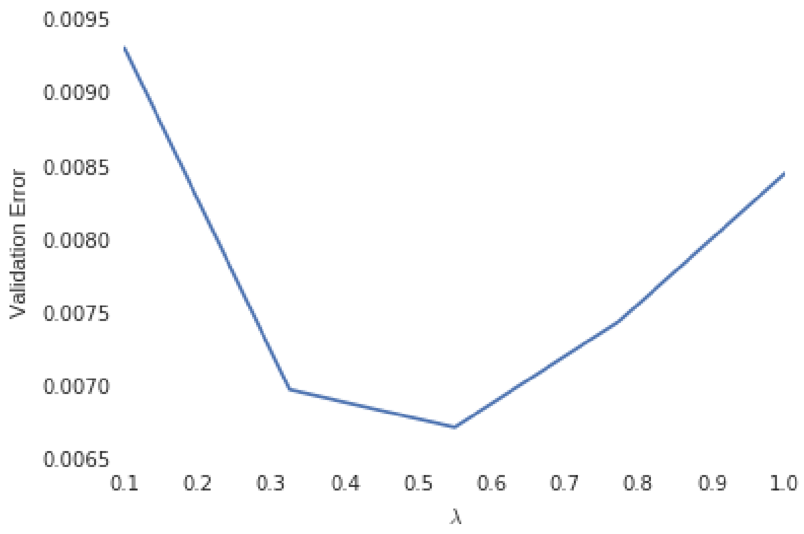


Figure 1: 2.1.1

```
def soft(a,d):
    return np.sign(a)*np.max([0, np.abs(a) - d])

def shootingAlg(L, theta, X, y,maxIter = 5000, tol = 10**-4):
# Shooting Algorithm
    n,D = X.shape
    thetaNew = theta.copy()
    change = 1
    count = 1
    while change > tol and count < maxIter:
        for j in range(D):
        a = 0
        c = 0
        for i in range(n):
            a = a + 2*X[i,j]**2
            c = c + 2*(X[i,j]*(y[i] - np.dot(X[i,:],thetaNew) + \
                thetaNew[j]*X[i,j]))
            thetaNew[j] = soft(c/a,L/a)
        change = 2*np.sqrt((thetaNew - theta).dot(thetaNew - theta))
        theta = thetaNew.copy()
        count = count + 1
    return theta
```

2. Analyze the sparsity of your solution, reporting how many components with true value zero have been estimated to be non-zero, and vice-versa.

   Solution:

   Optimial $\lambda = 0.55$. 52/65 true zero components that are calculated to be zero. 10/10 non-zero components that are calculated to be non-zero.

3. Implement the homotopy method described above. Compare the runtime for computing the full regularization path (for the same set ofλ's you tried in the first question above) using the homotopy method compared to the basic shooting algorithm.

   Solution:

   56.8 s for homotopy vs 2min 46s without.

4. The algorithm as described above is not ready for a large dataset (at least if it has been implemented in basic Python) because of the implied loop over the dataset (i.e. where we sum over the training set). By using matrix and vector operations, we can eliminate the loops. This is called"vectorization" and can lead to dramatic speedup in languages such as Python, Matlab, and R. Derive matrix expressions for computinga_j andc_j. (Hint: A matlab version of this vectorized method can be found here:`http://pmtk3.googlecode.com/svn-history/r1393/trunk/toolbox/Variable_selection/lassoExtra/LassoShooting.m`). Implement the matrix expressions and measure the speedup to compute the regularization path.

   Solution:

```
def shootingAlg_vectorized(L, theta, X, y,maxIter = 5000, tol = 10**-4):
# Shooting Algorithm using vectorization
```

```
n,D = X.shape
thetaNew = theta.copy()
change = 1
count = 1
while change > tol and count < maxIter:
        for j in range(D):
                a = 2 * sum(X[:,j]**2)
                c = 2 * (X[:,j].dot((y - np.dot(X,thetaNew) +\
                        thetaNew[j]*X[:,j])))
                thetaNew[j] = soft(c/a,L/a)
        change = 2*np.sqrt((thetaNew - theta).dot(thetaNew - theta))
        theta = thetaNew.copy()
        count = count + 1
return theta
```

2.2 Deriving the Coordinate Minimizer for Lasso

This problem is to derive the expressions for the coordinate minimizers used in the Shooting algorithm. This is often presented using subgradients (e.g.`http://davidrosenberg.github.io/ml2015/docs/2.Lab.subgradient-descent.pdf#page=15`), but here we will walk you through a bare hands approach (which is essentially equivalent).

In each step of the shooting algorithm, we would like to find thew_j minimizing

$$\begin{aligned} f(w_j) &= \sum_{i=1}^{n} \left(w^T x_i - y_i\right)^2 + \lambda \left|w\right|_1 \\ &= \sum_{i=1}^{n} \left[w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda \left|w_j\right| + \lambda \sum_{k \neq j} \left|w_k\right|, \end{aligned}$$

where we've writtenx_{ij} for the$j$th entry of the vectorx_i. This function is strictly convex inw_j, and thus it has a unique minimum. Furthermore, the only thing keeping$f$ from being differentiable is the term with$|w_j|$. Sof is differentiable everywhere except$w_j = 0$. We'll break this problem into 3 cases:$w_j > 0$,$w_j < 0$, and$w_j = 0$. In the first two cases, we can simply differentiate$f$ w.r.t.w_j to get optimality conditions. For the last case, we'll use the following more bare-hands characterization: Since$f$ is convex, 0 is a minimizer of$f$ iff

$$\lim_{\varepsilon \downarrow 0} \frac{f(\varepsilon) - f(0)}{\varepsilon} \geq 0 \quad \text{and} \quad \lim_{\varepsilon \downarrow 0} \frac{f(-\varepsilon) - f(0)}{\varepsilon} \geq 0.$$

This is a special case of the optimality conditions described in this slide`http://davidrosenberg.github.io/ml2015/docs/5.Lab.misc.pdf#page=10`, where now the"direction"v is simply taken to be the scalars1 and-1, respectively.

1. First let's get a trivial case out of the way. If $x_{ij} = 0$ for $i = 1, \ldots, n$, what is the coordinate minimizer w_j? In the remaining questions below, you may assume that $\sum_{i=1}^{n} x_{ij}^2 > 0$.

Solution:

Since $x_{ij} = 0$ for $i = 1, \ldots, n$**,** we can rewrite $f(w_j)$as:

$$\begin{aligned} f(w_j) &= \sum_{i=1}^{n} \left[w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda |w_j| + \lambda \sum_{k \neq j} |w_k| \\ &= \sum_{i=1}^{n} \left[\sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda |w_j| + \lambda \sum_{k \neq j} |w_k| \end{aligned}$$

Only the term $\lambda |w_j|$depends on w_j. This is minimized when $w_j = 0$.

2. Give an expression for the derivative$f(w_j)$ for$w_j \neq 0$. It will be convenient to write your expression in terms of the following definitions:

$$\begin{aligned} \text{sign}(w_j) &:= \begin{cases} 1 & w_j > 0 \\ 0 & w_j = 0 \\ -1 & w_j < 0 \end{cases} \\ a_j &:= 2 \sum_{i=1}^{n} x_{ij}^2 \\ c_j &:= 2 \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k \neq j} w_k x_{ik} \right). \end{aligned}$$

Solution:

$$\begin{aligned} w_j \neq 0 \to \frac{\partial}{\partial w_j} f(w_j) &= \sum_{i=1}^{n} \left[2x_{ij} \left(w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right) \right] + \lambda \text{sign}(w_j) \\ &= 2w_j \sum_{i=1}^{n} x_{ij}^2 - 2 \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k \neq j} w_k x_{ik} \right) + \lambda \text{sign}(w_j) \\ &= w_j a_j - c_j + \lambda \text{sign}(w_j) \end{aligned}$$

3. If$w_j > 0$ and minimizes$f$, then show that $w_j = -\frac{1}{a_j} (\lambda - c_j)$ for $a_j \neq 0$. Similarly, if$w_j < 0$ and minimizes$f$, show that$w_j = \frac{1}{a_j} (\lambda + c_j)$. Give conditions on$c_j$ that imply the minimizer$w_j > 0$ and$w_j < 0$, respectively.

Solution:

$$\begin{aligned} w_j > 0 \rightarrow \text{sign}\,(w_j) &= 1 \\ 0 &= w_j a_j - c_j + \lambda \\ -w_j a_j &= \lambda - c_j \\ w_j &= -\frac{1}{a_j}\,(\lambda - c_j) \end{aligned}$$

$$\begin{aligned} w_j > 0 \rightarrow \text{sign}\,(w_j) &= -1 \\ 0 &= w_j a_j - c_j - \lambda \\ -w_j a_j &= -\,(c_j + \lambda) \\ w_j &= \frac{1}{a_j}\,(c_j + \lambda) \end{aligned}$$

4. Derive expressions for the two one-sided derivatives at$f(0)$, and show that$c_j \in [-\lambda, \lambda]$ implies that$w_j = 0$ is a minimizer.

   Solution:

   Since $f(w_j)$ is not differentiable at $w_j = 0$, we can examine directly what happens when we move from $w_j = 0$ to $w_j = \varepsilon$, for any ε:

$$\begin{aligned} f(\varepsilon) &= \sum_{i=1}^{n} \left[\varepsilon x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda\,|\varepsilon| + \lambda \sum_{k \neq j} |w_k| \\ f(0) &= \sum_{i=1}^{n} \left[\sum_{k \neq j} w_k x_{ik} - y_i \right]^2 + \lambda \sum_{k \neq j} |w_k| \end{aligned}$$

   Generally speaking,

$$(a+b)^2 - b^2 = a^2 + 2ab,$$

   so

$$\begin{aligned} f(\varepsilon) - f(0) &= \sum_{i=1}^{n} \left[\varepsilon^2 x_{ij}^2 + 2\varepsilon x_{ij} \left(\sum_{k \neq j} w_k x_{ik} - y_i \right) \right] + \lambda\,|\varepsilon| \\ &= \frac{1}{2}\varepsilon^2 a_j - \varepsilon c_j + \lambda\,|\varepsilon| \end{aligned}$$

   and

$$\frac{f(\varepsilon) - f(0)}{\varepsilon} = \frac{1}{2}\varepsilon a_j - c_j + \lambda \text{sign}(\varepsilon)$$

   So the derivative to the right is

$$\lim_{\varepsilon \downarrow 0} \frac{f(\varepsilon) - f(0)}{\varepsilon} = -c_j + \lambda$$

and the derivative to the left is

$$\begin{aligned}
\lim_{\varepsilon\downarrow 0}\frac{f(-\varepsilon)-f(0)}{\varepsilon} &= \lim_{\varepsilon\downarrow 0}\frac{1}{\varepsilon}\left(\frac{1}{2}\varepsilon^2 a_j+\varepsilon c_j+\lambda\left|-\varepsilon\right|\right)\\
&= \lim_{\varepsilon\downarrow 0}\frac{1}{\varepsilon}\left(\frac{1}{2}\varepsilon^2 a_j+\varepsilon c_j+\lambda\varepsilon\right)\\
&= \lim_{\varepsilon\downarrow 0}\frac{1}{2}\varepsilon a_j+c_j+\lambda\\
&= c_j+\lambda
\end{aligned}$$

$w_j=0$ is a minumum iff both derivatives are nonnegative. The first limit is nonnegative iff $c_j\le\lambda$ and the second limit is nonnegative iff $c_j\ge-\lambda$. Thus $w_j=0$ is a minimum iff $c_j\in[-\lambda,\lambda]$.

5. Conclude that the minimizer is given by

$$w_j=\begin{cases}\frac{1}{a_j}\left(c_j-\lambda\right) & c_j>\lambda\\ 0 & c_j\in[-\lambda,\lambda]\\ \frac{1}{a_j}\left(c_j+\lambda\right) & c_j<-\lambda\end{cases}$$

and show that this is equivalent to the expression given in2.

Solution:

Combining results from (3) and (4), we have that :

$$w_j=\begin{cases}\frac{1}{a_j}\left(c_j-\lambda\right) & c_j>\lambda\\ 0 & c_j\in[-\lambda,\lambda]\\ \frac{1}{a_j}\left(c_j+\lambda\right) & c_j<-\lambda\end{cases}$$

This is equivalent to the expression given in 2 since

$$\begin{aligned}
soft(\frac{c_j}{a_j},\frac{\lambda}{a_j}) &= sign(\frac{c_j}{a_j})\times max(|\frac{c_j}{a_j}|-\frac{\lambda}{a_j},0)\\
&= sign(c_j)\times max(\frac{|c_j|-\lambda}{a_j},0)\\
&= \begin{cases}\frac{1}{a_j}\left(c_j-\lambda\right) & c_j>\lambda\\ 0 & c_j\in[-\lambda,\lambda]\\ \frac{1}{a_j}\left(c_j+\lambda\right) & c_j<-\lambda\end{cases}
\end{aligned}$$

3 Lasso Properties

3.1 Deriving $\lambda_{\max}$

In this problem we will derive an expression for $\lambda_{\max}$. For the first three parts, use the Lasso objective function excluding the bias term i.e, $L(w)=\|Xw-y\|_2^2+\lambda\|w\|_1$. Show that for any

$\lambda \geq 2\|X^T y\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum absolute value of any component of the vector.

1. The one-sided directional derivative of $f(x)$ at x in the direction v is defined as:

$$f'(x; v) = \lim_{h \downarrow 0} \frac{f(x + hv) - f(x)}{h}$$

Compute $L'(0; v)$. That is, compute the one-sided directional derivative of $L(w)$ at $w = 0$ in the direction v. [Hint: the result should be in terms of $X, y, \lambda,$ and v.]

Solution:

$$\begin{aligned} L(w) =& (Xw - y)^T(Xw - y) + \lambda \|w\|_1 \\ =& w^T X^T X w - 2w^T X^T y + y^T y + \lambda \|w\|_1 \end{aligned}$$

$$\begin{aligned} L(0) =& y^T y \\ L(0 + hv) =& h^2 v^T X^T X v - 2hv^T X^T y + y^T y + \lambda \|hv\|_1 \end{aligned}$$

$$\begin{aligned} L'(0, v) =& \lim_{h \downarrow 0} \frac{L(0 + hv) - L(0)}{h} \\ =& \lim_{h \downarrow 0} \frac{h(hv^T X^T X v - 2v^T X^T y + \lambda \|v\|_1)}{h} \\ =& -2v^T X^T y + \lambda \|v\|_1 \end{aligned}$$

2. Since the Lasso objective is convex, for w^* to be a minimizer of $L(w)$ we must have that the directional derivative $L'(w^*; v) \geq 0$ for all v. Starting from the condition $L'(0; v) \geq 0$, rearrange terms to get a lower bounds on λ. [Hint: this should be in terms of $X, y,$ and v.]

Solution:

Since we assume $w = 0$ is a minimizer, $L'(0, v) \geq 0$ for all v.

$$\begin{aligned} -2v^T X^T y + \lambda \|v\|_1 \geq 0 \\ \lambda \geq \frac{2v^T X^T y}{\|v\|} \end{aligned}$$

3. Since our lower bounds on λ hold for all v, we want to compute the maximum lower bound. Compute the maximum lower bound of λ by maximizing the expression over v. Show that this expression is equivalent to $\lambda_{\max} = 2\|X^T y\|_\infty$.

Solution:

claim: $\|y\|_\infty$ *is* a solution to max $f(x) = x^T y$ subject to $\|x\|_1 = 1$
proof. $f(x) = \sum_i x_i y_i \le \|y\|_\infty \sum_i x_i \le \|y\|_\infty \sum_i |x_i| \le \|y\|_\infty$. $f(x) = \|y\|_\infty$is attainable (let all entries of x be 0 except for a 1 in the position that corresponds to the max value of y).

$\lambda_{max} = max_v \frac{2v^T X^T y}{\|v\|} = 2\left\|X^T y\right\|_\infty$

4. [Optional] Show that for $L(w,b) = \|Xw + b - y\|_2^2 + \lambda\|w\|_1$, $\lambda_{\max} = 2\|X^T(y - \bar{y})\|_\infty$where $\bar{y}$ is the mean of values in the vector y.

   Solution:

   Assuming $w = 0$ is the minimizer, we try to minimize the following:

$$\begin{aligned} L(b) &= \|b - y\|_2^2 \\ &= (b\mathbf{1} - y)^T(b\mathbf{1} - y) \\ &= b^2\mathbf{1}^T\mathbf{1} - 2by^T\mathbf{1} + y^T y \\ \\ L' &= 2b\mathbf{1}^T\mathbf{1} - 2y^T\mathbf{1} \\ \\ b &= \frac{y^T\mathbf{1}}{\mathbf{1}^T\mathbf{1}} \\ &= \bar{y} \end{aligned}$$

   From here, we can use the same procedure as parts 1 - 3 with $L(w) = \|Xw - (y - \bar{y})\|_2^2 + \lambda\|w\|_1$.

   Alternative Solution for 3.1 Using Subgradients:

   Let $g(\theta)$ be a subgradient of the ridge regression loss function (S_θ is the subgradient of $\|\cdot\|_1$at θ).

$$g(\theta) = 2X^T X\theta - 2X^T y + \lambda S_\theta$$

   Then θ is a minimizer of our loss iff 0 is a subgradient at $\theta = 0$. This means $g(0) = 0$ must be possible. $g(0) = 0$ if an only if $S_0 = \frac{2X^T y}{\lambda}$. S_0(the subgradient of the l_1 norm at 0) is a vector where each entry is in $[-1, 1]$[2]. For this to hold, $\lambda \ge \left\|2X^T y\right\|_\infty$.

[2]See proposition 3.5, http://www.cims.nyu.edu/˜cfgranda/pages/OBDA_spring16/material/convex_optimization.pdf

3.2 [Optional] Feature Correlation

In this problem, we will examine and compare the behavior of the Lasso and ridge regression in the case of an exactly repeated feature. That is, consider the design matrix$X \in \mathbf{R}^{m\times d}$, where $X_{\cdot i} = X_{\cdot j}$ for some i and j, where $X_{\cdot i}$ is thei^{th} column of$X$. We will see that ridge regression divides the weight equally among identical features, while Lasso divides the weight arbitrarily. In an optional part to this problem, we will consider what changes when$X_{\cdot i}$ and$X_{\cdot j}$ are highly correlated (e.g. exactly the same except for some small random noise) rather than exactly the same.

1. [Optional] Derive the relation between $\hat{\theta}_i$ and $\hat{\theta}_j$, the i^{th} and the j^{th} components of the optimal weight vector obtained by solving the Lasso optimization problem.
   [Hint: Assume that in the optimal solution,$\hat{\theta}_i =a$ and$\hat{\theta}_j =b$. First show that a and b must have the same sign. Then, using this result, rewrite the optimization problem to derive a relation between$a$ and$b$.]

   Solution:

   Without a loss of generality, assume the first two colums of X are our repeated features. Partition X and θ as follows:

   $$X = \begin{bmatrix} x_1 & x_2 & X' \end{bmatrix}, \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta' \end{bmatrix}$$

   We can write the objective function as:

   $$\begin{aligned} L(\theta) &= \|X\theta - y\|_2^2 + \lambda \|\theta\|_1 \\ &= \|x_1\theta_1 + x_2\theta_2 + X'\theta' - y\|_2^2 + \lambda|\theta_1| + \lambda|\theta_2| + \lambda \|\theta'\|_1 \end{aligned}$$

   Let $\hat{\theta}$ be a minimizer of $L(\theta)$. We can construct another weight vector, θ^*, such that $L(\hat{\theta}) = L(\theta^*)$ i.e., θ^* is another solution to our problem and the weights on the repeated features are set arbitrarily . Let $\hat{\theta}$ and θ^* have the following form:

   $$\hat{\theta} = \begin{bmatrix} a \\ b \\ \theta' \end{bmatrix}, \theta^* = \begin{bmatrix} c \\ d \\ \theta' \end{bmatrix}$$

   First note that a and b, the weights on the repeated feature, must have the same sign. Suppose the signs are different i.e. $a < 0 < b$. Then we can construct θ^* such that $L(\theta^*) < L(\hat{\theta})$. This is a contradiction since $\hat{\theta}$ minimizes $L(\theta)$. For example, let

   $$\theta^* = \begin{bmatrix} a+b \\ 0 \\ \theta' \end{bmatrix}$$

   The error term of the objective function will be the same using either weight vector but the regularization term will be smaller for θ^* since $|a+b| < |a|+|b|$. Since the signs of θ_1 and $\theta_2$are the same, we can write the objective function as

$$L(\theta) = \|(\theta_1+\theta_2)x_1 + X'\theta' - y\|_2^2 + \lambda|\theta_1+\theta_2| + \lambda\|\theta'\|_1$$

From this, we can see that θ^* is also a solution as long as $c+d = a+b$ and c, d have the same sign.

2. [Optional] Derive the relation between$\hat{\theta}_i$ and$\hat{\theta}_j$, thei^{th} and thej^{th} components of the optimal weight vector obtained by solving the ridge regression optimization problem.
Solution:

Following the notation from above, we can write the ridge regression objective function as

$$\begin{aligned} L(\theta) &= \|X\theta - y\|_2^2 + \lambda\|\theta\|_2 \\ &= \|(\theta_1+\theta_2)x_1 + X'\theta' - y\|_2^2 + \lambda(\theta_1^2+\theta_2^2) + \lambda\|\theta'\|_2^2 \end{aligned}$$

For each value of $\theta_1+\theta_2 = \mathit{constant}$, we must minimize $\theta_1^2+\theta_2^2$. This is achieved when $\theta_1 = \theta_2$.

3. [Optional] What do you think would happen with Lasso and ridge when$X_{\cdot i}$ and$X_{\cdot j}$ are highly correlated, but not exactly the same. You may investigate this experimentally or geometrically.

4 The Ellipsoids in theℓ_1/ℓ_2 regularization picture

Recall the famous picture purporting to explain whyℓ_1 regularization leads to sparsity, whileℓ_2 regularization does not. Here's the instance from Hastie et al's*The Elements of Statistical Learning:*

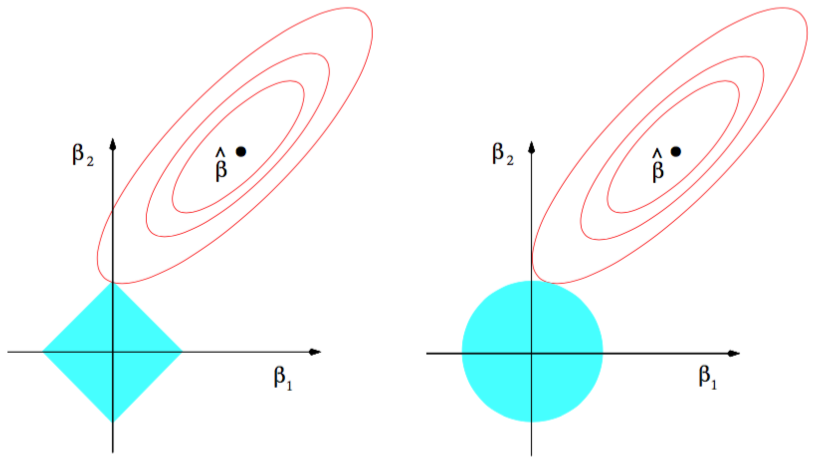


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

(While Hastie et al. use β for the parameters, we'll continue to use w.)

In this problem we'll show that the level sets of the empirical risk are indeed ellipsoids centered at the empirical risk minimizer $\hat{w}$.

Consider linear prediction functions of the form $x \mapsto w^T x$. Then the empirical risk for any w, the empirical risk for $f(x) = w^T x$ under the square loss is

$$\begin{aligned}\hat{R}_n(w) &= \frac{1}{n}\sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 \\ &= \frac{1}{n}\left(Xw - y\right)^T\left(Xw - y\right).\end{aligned}$$

1. Let $\hat{w} = \left(X^T X\right)^{-1} X^T y$. Show that $\hat{w}$ has empirical risk given by

$$\hat{R}_n(\hat{w}) = \frac{1}{n}\left(-y^T X \hat{w} + y^T y\right)$$

Solution:

$$\begin{aligned}
\frac{1}{n}(X\hat{w}-y)^T(X\hat{w}-y) &= \frac{1}{n}\left(X\left(X^TX\right)^{-1}X^Ty-y\right)^T\left(X\left(X^TX\right)^{-1}X^Ty-y\right) \\
&= \frac{1}{n}y^TX\left(X^TX\right)^{-1}X^TX\left(X^TX\right)^{-1}X^Ty \\
&\quad -\frac{2}{n}y^TX\left(X^TX\right)^{-1}X^Ty+\frac{1}{n}y^Ty \\
&= -\frac{1}{n}y^TX\left(X^TX\right)^{-1}X^Ty+\frac{1}{n}y^Ty \\
&= -\frac{1}{n}y^TX\hat{w}+\frac{1}{n}y^Ty
\end{aligned}$$

2. Show that for any$w$ we have

$$\hat{R}_n(w)=\frac{1}{n}\left(w-\hat{w}\right)^TX^TX\left(w-\hat{w}\right)+\hat{R}_n(\hat{w}).$$

Note that the RHS (i.e.“right hand side”) has one term that’s quadratic in$w$ and one term that’s independent of$w$. In particular, the RHS does not have any term that’s linear in$w$. On the LHS (i.e.“left hand side”), we have$\hat{R}_n(w)=\frac{1}{n}\left(Xw-y\right)^T\left(Xw-y\right)$. After expanding the out, you’ll have terms that are quadratic, linear, and constant in$w$. Completing the square is the tool for rearranging an expression to get rid of the linear terms. The following“completing the square” identity is easy to verify just by multiplying out the expressions on the RHS:

$$x^TMx-2b^Tx \quad = \quad \left(x-M^{-1}b\right)^TM(x-M^{-1}b)-b^TM^{-1}b$$

Solutions:

$$\left(Xw-y\right)^T\left(Xw-y\right) \quad = \quad w^TX^TXw-2y^TXw+y^Ty$$

Let’s complete the square with$M=X^TX$ and$b=X^Ty$. Then

$$\begin{aligned}
\left(Xw-y\right)^T\left(Xw-y\right) &= \left(w-\left(X^TX\right)^{-1}X^Ty\right)X^TX\left(w-\left(X^TX\right)^{-1}X^Ty\right) \\
&\quad -y^TX\left(X^TX\right)^{-1}X^Ty+y^Ty \\
&= \left(w-\hat{w}\right)X^TX\left(w-\hat{w}\right)-y^TX\hat{w}+y^Ty
\end{aligned}$$

Putting it together,

$$\begin{aligned}
\hat{R}_n(w) &= \frac{1}{n}\left(Xw-y\right)^T\left(Xw-y\right) \\
&= \frac{1}{n}\left(w-\hat{w}\right)X^TX\left(w-\hat{w}\right)-\frac{1}{n}y^TX\hat{w}+\frac{1}{n}y^Ty \\
&= \frac{1}{n}\left(w-\hat{w}\right)X^TX\left(w-\hat{w}\right)+\hat{R}_n(\hat{w})
\end{aligned}$$

3. Using the expression derived for$\hat{R}_n(w)$ in2, give a very short proof that$\hat{w} = \left(X^TX\right)^{-1} X^Ty$ is the empirical risk minimizer. That is:

$$\hat{w} = \arg\min_w \hat{R}_n(w).$$

Hint: Note thatX^TX is positive semidefinite and, by definition, a symmetric matrix M is positive semidefinite iff for all$x \in \mathbf{R}^d$,$x^TMx \geq 0$.

Solution:

$$\begin{aligned}\arg\min_w \hat{R}_n(w) &= \arg\min_w \frac{1}{n}\left((w-\hat{w})\,X^TX\,(w-\hat{w})\right) + \hat{R}_n(\hat{w}) \\ &= \arg\min_w \frac{1}{n}\left((w-\hat{w})\,X^TX\,(w-\hat{w})\right) \\ &= \arg\min_w \left((w-\hat{w})\,X^TX\,(w-\hat{w})\right)\end{aligned}$$

- Because the matrix X^TX is positive semidefinite, $\left((w-\hat{w})\,X^TX\,(w-\hat{w})\right) \geq 0$ for all w
- $\left((w-\hat{w})\,X^TX\,(w-\hat{w})\right) = 0$ when $w = \hat{w}$
- Therefore $\hat{w}$ is a minimizer of $\hat{R}_n(w)$

4. Give an expression for the set of$w$ for which the empirical risk exceeds the minimum empirical risk$\hat{R}_n(\hat{w})$ by an amount$c > 0$. This set is an ellipse – what is its center?

Solution:

$$\begin{aligned}\hat{R}_n(w) - \hat{R}_n(\hat{w}) = c \\ c &= \frac{1}{n}\left((w-\hat{w})\,X^TX\,(w-\hat{w})\right) \\ cn &= (w-\hat{w})\,X^TX\,(w-\hat{w})\end{aligned}$$

Since X^TX is positive definite, this is an equation of an ellipse centered at $\hat{w}$.

5 [Optional] Projected SGD via Variable Splitting

In this question, we consider another general technique that can be used on the Lasso problem. We first use the variable splitting method to transform the Lasso problem to a smooth problem with linear inequality constraints, and then we can apply a variant of SGD.

Representing the unknown vectorθ as a difference of two non-negative vectors θ^+ and θ^-, theℓ_1-norm of θ is given by$\sum_{i=1}^{d} \theta_i^+ + \sum_{i=1}^{d} \theta_i^-$. Thus, the optimization problem can be written as

$$(\hat{\theta}^+, \hat{\theta}^-) = \underset{\theta^+, \theta^- \in \mathbf{R}^d}{\arg\min} \sum_{i=1}^{m} (h_{\theta^+,\theta^-}(x_i) - y_i)^2 + \lambda \sum_{i=1}^{d} \theta_i^+ + \lambda \sum_{i=1}^{d} \theta_i^-$$
$$\text{such that } \theta^+ \geq 0 \text{ and } \theta^- \geq 0,$$

where$h_{\theta^+,\theta^-}(x) = (\theta^+ - \theta^-)^T x$. The original parameter θ can then be estimated as $\hat{\theta} = (\hat{\theta}^+ - \hat{\theta}^-)$.

This is a convex optimization problem with a differentiable objective and linear inequality constraints. We can approach this problem using projected stochastic gradient descent, as discussed in lecture. Here, after taking our stochastic gradient step, we project the result back into the feasible set by setting any negative components of θ^+ andθ^- to zero.

1. [Optional] Implement projected SGD to solve the above optimization problem for the same λ's as used with the shooting algorithm. Since the two optimization algorithms should find essentially the same solutions, you can check the algorithms against each other. Report the differences in validation loss for eachλ between the two optimization methods. (You can make a table or plot the differences.)

   Solutions:

```
def lasso_variable_splitting(Lambda, w_init, X, y):
   w_plus = numpy.maximum(w_init, numpy.zeros(D))
   w_minus = numpy.maximum(-w_init, numpy.zeros(D))
   max_epochs = 1000
      for epoch in range(1,max_epochs+1):
         for n in range(N_train):
            step = 0.01/N_train
            grad_w_plus = 2*N_train*numpy.dot((numpy.dot(X[n,:],(w_plus - w_minus))\
                     - y[n]).T,X[n,:]) + Lambda*numpy.ones(D).T

            grad_w_minus = -2*N_train*numpy.dot((numpy.dot(X[n,:],(w_plus - w_minus)) \
                      - y[n]).T,X[n,:]) + Lambda*numpy.ones(D).T

            w_plus = w_plus - step * grad_w_plus
            w_minus = w_minus - step * grad_w_minus
            w_plus = numpy.maximum(w_plus, numpy.zeros(D))
            w_minus = numpy.maximum(w_minus, numpy.zeros(D))
   return w_plus - w_minus
```

2. [Optional] Choose theλ that gives the best performance on the validation set. Describe the solution $\hat{w}$ in term of its sparsity. How does the sparsity compare to the solution from the shooting algorithm?