Machine Learning and Computational Statistics
Homework 2: Lasso Regression

Due: Monday, February 13, 2017, at 10pm (Submit via Gradescope)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document.

1 Introduction

In this homework you will investigate regression with ℓ_1 regularization, both implementation techniques and theoretical properties. On the methods side, you'll work on coordinate descent (the "shooting algorithm"), homotopy methods, and [optionally] projected SGD. On the theory side you'll deriver the explicit solution to the coordinate minimizers used in coordinate descent, you'll derive the largest ℓ_1 regularization parameter you'll ever need to try, you'll investigate what happens with ridge and lasso regression when you have two copies of the same feature, and you'll [optionally] work out the details of the classic picture that "explains" why ℓ_1 regularization leads to sparsity.

For the experiments, we constructed a small dataset with known properties. The file `generate_data.py` contains the code used to generate the data. Section 1.1 describes the steps followed to construct the dataset.

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.

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.

3. Let $y = (y_1, \ldots, y_m)^T \in \mathbf{R}^{m\times 1}$ be the response. Construct the vector $y = X\boldsymbol{\theta} + \epsilon$, where ϵ is an $m \times 1$ random noise vector where each entry is sampled i.i.d. from a normal distribution with mean 0 and standard deviation 0.1. You can use `numpy.random.randn()` to generate ϵ in Python.

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.

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.

2 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:

```
D = X_training.shape[1]
w = numpy.random.rand(D,1)
def ridge(Lambda):
    def ridge_obj(theta):
        return ((numpy.linalg.norm(numpy.dot(X_training,theta) - y_training))**2)
    /(2*80) + Lambda*(numpy.linalg.norm(theta))**2
    return ridge_obj
def compute_loss(Lambda, theta):
    return ((numpy.linalg.norm(numpy.dot(X_validation,theta) - y_validation))**2)
    /(2*20)
# plot
L = []
Loss = []
for i in range(-6,6):
    Lambda = 10**i;
    w_opt = minimize(ridge(Lambda), w)
    L.append(Lambda)
    Loss.append(compute_loss(Lambda, w_opt.x))
fig = plt.figure(figsize=[10,6])
ax = fig.add_subplot(111)
plt.plot(range(-6,6), Loss,label='loss')
plt.legend(loc='upper left')
plt.xlim(-6.5,6)
plt.ylim(-50,800)
plt.show()
fig.savefig("1_2_1.png")

# Sparsity for optimal lambda
w = numpy.random.rand(D,1)
Lambda = 10**-6;
w_opt = minimize(ridge(Lambda), w)
```

```
sum(np.abs(w_opt.x) == 0)
sum(np.abs(w_opt.x) < 10**(-3))
sum(np.abs(w_opt.x) < 10**(-2))
sum(np.abs(w_opt.x) < 10**(-1))
```

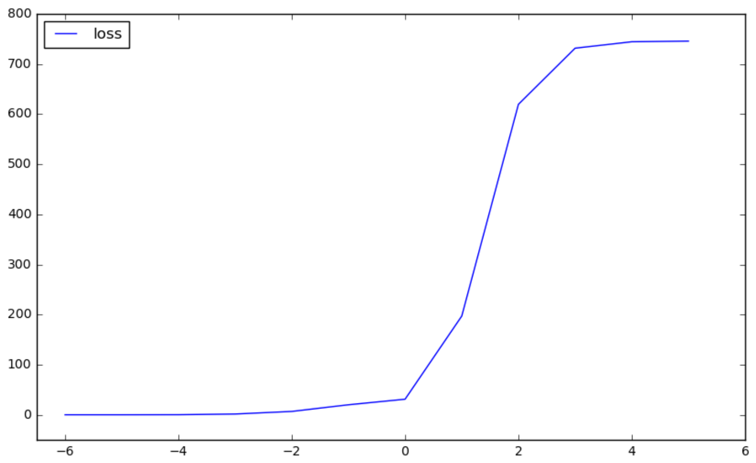


Figure 1: Validation Loss versus logλ

$\lambda = 10^{-6}$ minimizes the square loss on the validation set in this case. For the chosen λ, with true value 0 as the threshold, all coefficients are nonzero. With threshold 10^{-3}, we have the same result. With threshold 10^{-2}, there are 6 zero coefficients, and with threshold 10^{-1}, the sparsity there are 40 zero coefficients.

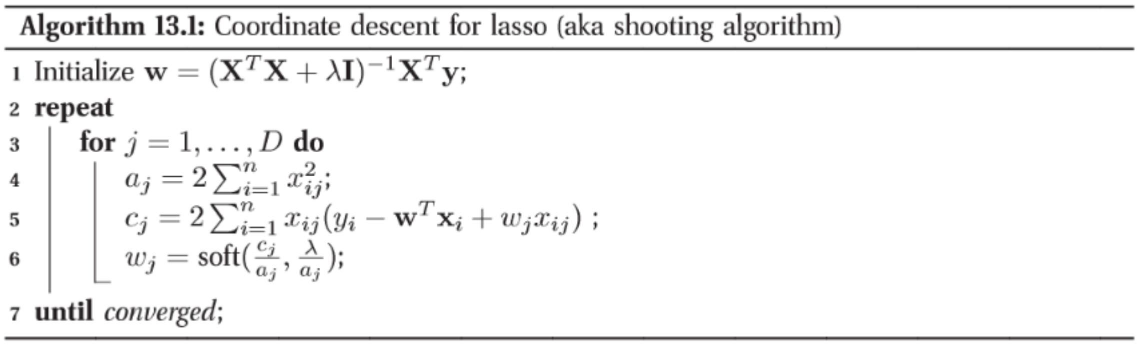
3 Coordinate Descent for Lasso (a.k.a. The Shooting algorithm)

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 ℓ_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. (In fact, as we'll see in the next homework on SVMs, we can use "subgradient" methods when the objective function is not differentiable.)

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, known as the **shooting algorithm**:



```
Algorithm 13.1: Coordinate descent for lasso (aka shooting algorithm)
Initialize $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$;
repeat
    for $j = 1, \ldots, D$ do
        $a_j = 2\sum_{i=1}^n x_{ij}^2$;
        $c_j = 2\sum_{i=1}^n x_{ij}(y_i - \mathbf{w}^T\mathbf{x}_i + w_j x_{ij})$ ;
        $w_j = \text{soft}(\frac{c_j}{a_j}, \frac{\lambda}{a_j})$;
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 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 since it is, as you can easily verify, the coordinate minimizer. Note also that Murphy is suggesting to initialize the optimization with the ridge regession solution. Although theoretically this is not necessary (with exact computations and enough time, coordinate descent will converge for lasso from any starting point), in practice it's helpful to start as close to the solution as we're able.

There are a few tricks that can make selecting the hyperparameter λ easier and faster. First, you can show that for any $\lambda \geq 2\|X^T(y - \bar{y})\|_\infty$, 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 over the absolute values of the components of a vector. Thus we need to search for an optimal λ 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 $h_{w,b}(x) = w^T x + b$. For the experiments, you can exclude the bias term, in which case $\lambda_{\max} = 2\|X^T y\|_\infty$.)

The second trick is to use the fact that when λ and λ' are close, the corresponding solutions $\hat{w}(\lambda)$ and $\hat{w}(\lambda')$ are also close. Start with $\lambda = \lambda_{\max}$, for which we know $\hat{w}(\lambda_{\max}) = 0$. You can run the optimization anyway, 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 technique of computing a set of solutions for a chain of nearby λ's is called a **continuation** or **homotopy method**. The resulting set of parameter values $\hat{w}(\lambda)$ as λ ranges over $[0, \lambda_{\max}\,]$ is known as a **regularization path**.

3.1 Experiments with the Shooting Algorithm

1. Write a function that computes the Lasso solution for a given λ 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 the λ that minimizes the square error on the validation set. Report the optimal value of λ found, and the corresponding test error. Plot the validation error vs λ. [Don't use the homotopy method in this part, as we want to measure the speed improvement of homotopy methods in question 3. Also, no need to vectorize the calculations until question 4, 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.]

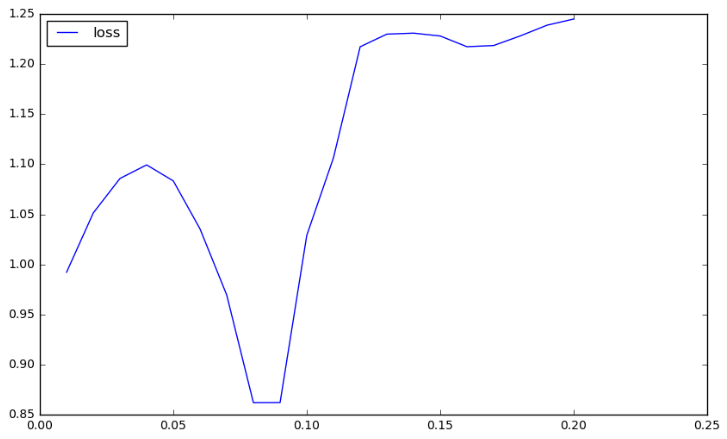
Solution:

```
def soft(a, delta):
    var_1 = abs(a) - delta
    var_2 = np.max([var_1, 0.])
    var_3 = np.sign(a)*var_2
    return var_3

def compute_square_loss(X, y, theta):
    loss = 0
    m = X.shape[0]
    vector = np.dot(X, theta) - y
    loss = float(np.dot(vector, vector))/(2*m)
    return loss

def Lasso_Shooting(X, y, theta_0, lambda_reg = 1., num_iter = 1000):
    theta = np.zeros(theta_0.shape)
    theta[0: len(theta)] = theta_0
    d = X.shape[1]
    times = 0
    loss = compute_square_loss(X, y, theta)
    loss_change = 1.
    while (loss_change>1e-6) and (times<num_iter):
        loss_old = loss
        for j in range(d):
            X_d = X[:, j]
            a = np.dot(X_d.T, X_d)*2
            vector = y - np.dot(X, theta.T) + X_d*theta[j]
            c = np.dot(X_d.T, vector)*2
            theta[j] = soft(c/a, lambda_reg/a)
        times +=1
        loss = compute_square_loss(X, y, theta)
        loss_change = abs(loss - loss_old)
    return theta
```

Figure showing Validation Loss versus λ:



$\lambda = 0.08$ would be the optimal value, and the corresponding validation loss is 0.8619681632728108.

2. Analyze the sparsity[1] of your solution, reporting how many components with true value zero have been estimated to be non-zero, and vice-versa.

   Solution: Lasso performs better in terms of sparsity. There are 3 zero components that are estimated as exactly zero. With threshold 10^{-3}, there are 4. With threshold 10^{-2}, there are 6 zero coefficients. The first 10 components are all close to 10 or -10.

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

   Solution: The homotopy method should speed up the regularization path a lot.

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 computing a_j and c_j. (Hint: A matlab version of this vectorized method can be found in the assignment zip file.) Implement the matrix expressions and measure the speedup in computing the regularization path.

[1]One might hope that the solution will 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"** (sparse + consistent). Formally, an estimator $\hat{\beta}$ of parameter β is said to be consistent if the estimator $\hat{\beta}$ converges to the true value β in probability as our sample size goes to infinity. Analogously, if we define the support of a vector β 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 of β, or $\lim_{m\to\infty} P[\text{Supp}(\hat{\beta}_m) = \text{Supp}(\beta)] = 1$.

Solution:

```python
def Lasso_Shooting_matrix(X, y, theta_0, lambda_reg = 1., num_iter = 1000):
    theta = np.zeros(theta_0.shape)
    theta[0: len(theta)] = theta_0
    d = X.shape[1]
    times = 0
    loss = compute_square_loss(X, y, theta)
    loss_change = 1.
    XX2 = np.dot(X.T, X)*2
    Xy2 = np.dot(X.T, y)*2
    while (loss_change>1e-6) and (times<num_iter):
        loss_old = loss
        for j in range(d):
            a = XX2[j, j]
            c =Xy2[j] - np.dot(XX2[j,:], theta.T) + a*theta[j]
            theta[j] = soft(c/a, lambda_reg/a)
        times +=1
        loss = compute_square_loss(X, y, theta)
        loss_change = abs(loss - loss_old)
    return theta
```

3.2 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 derived using subgradients (slide 15), but here we will take a bare hands approach (which is essentially equivalent).

In each step of the shooting algorithm, we would like to find the w_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 written x_{ij} for the jth entry of the vector x_i. This function is strictly convex in w_j, and thus it has a unique minimum. The only thing keeping f from being differentiable is the term with $|w_j|$. So f 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 fact that since $f : \mathbf{R} \to \mathbf{R}$ 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 slide 12 here, where now the "direction" v is simply taken to be the scalars 1 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 $i = 1, \dots, n$, 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:

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

3. If $w_j > 0$ and minimizes f, show that $w_j = \frac{1}{a_j}\left(c_j - \lambda\right)$. Similarly, if $w_j < 0$ and minimizes f, show that $w_j = \frac{1}{a_j}\left(c_j + \lambda\right)$. Give conditions on c_j that imply that a minimizer w_j is positive and conditions for which a minimizer w_j is negative.
Solution: Setting $f'(w_j) = 0$, we get

$$w_j = \frac{1}{a_j}\left[c_j - \lambda \text{sign}(w_j)\right].$$

Thus if $c_j > \lambda$, then $w_j = \frac{1}{a_j}\left[c_j - \lambda\right]$ satisfies this condition and is a minimizer (and is positive). If $c_j < -\lambda$ then $w_j = \frac{1}{a_j}\left[c_j + \lambda\right]$ is a minimizer (and is negative).

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

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

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

5. Putting together the preceding results, we conclude the following:

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

Show that this is equivalent to the expression given in 3.
Solution:

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

4 Lasso Properties

4.1 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:
For any $w \in \mathbf{R}^d$ we have

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

So for any $h > 0$ and $v \in \mathbf{R}^d$ we have

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

Thus the one-sided directional derivative of L at 0 is

$$\begin{aligned} L'(0,v) &= \lim_{h\downarrow 0}\frac{L(0+hv)-L(0)}{h} \\ &= -2v^TX^Ty+\lambda\left\|v\right\|_1 \end{aligned}$$

2. Since the Lasso objective is convex, w^* is minimizer of $L(w)$ if and only if 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:

$$\begin{aligned} L'(0,v) &\geq 0 \\ \iff -2v^TX^Ty+\lambda\left\|v\right\|_1 &\geq 0 \\ \iff \lambda &\geq \frac{2v^TX^Ty}{\left\|v\right\|_1} \end{aligned}$$

3. In the previous problem, we get a different lower bound on λ for each choice of v. Compute the maximum lower bound of λ by maximizing the expression over v. Show that this expression is equivalent to $\lambda_{\max}=2\|X^Ty\|_\infty$. .
   Solution:

   Claim: $\|y\|_\infty=\max_{x:\|x\|_1=1}x^Ty$.
   In words, this is true because we can put all the weight in x on the component corresponding to the entry of y with the largest absolute value.
   Proof. $x^Ty=\sum_i x_iy_i\leq\|y\|_\infty\sum_i x_i\leq\|y\|_\infty\sum_i|x_i|\leq\|y\|_\infty$. Let $i^*=\arg\max_i|y_i|$, and let $x_{i^*}=\text{sign}(y_{i^*})$ and $x_j=0$ for $j\neq i$. Then $\|x\|_1=1$, and $x^Ty=y_{i^*}=\|y\|_\infty$. **QED**

   Applying the claim, note that

$$\begin{aligned} \lambda &\geq \frac{2v^TX^Ty}{\left\|v\right\|_1}\text{ for all } v\neq 0 \\ \iff \lambda &\geq \sup_{v\neq 0}\frac{2v^TX^Ty}{\left\|v\right\|_1} \\ &= 2\max_{v:\|v\|_1=1}v^T\left(X^Ty\right) \\ &= 2\|X^Ty\|_\infty \end{aligned}$$

   Putting this together with the previous problem gives our conclusion.

4. [Optional] Show that for $L(w,b)=\|Xw+b\mathbf{1}-y\|_2^2+\lambda\left\|w\right\|_1$, $\lambda_{\max}=2\|X^T(y-\bar{y})\|_\infty$ where $\bar{y}$ is the mean of values in the vector y, and $\mathbf{1}\in\mathbf{R}^n$ is a column vector of 1's .
   Solution:

For any $w \in \mathbf{R}^d, b \in \mathbf{R}$ we have

$$\begin{aligned} L(w,b) =& (Xw + b\mathbf{1} - y)^T (Xw + b\mathbf{1} - y) + \lambda \|w\|_1 \\ =& w^T X^T X w - 2w^T X^T (b\mathbf{1} - y) + (b\mathbf{1} - y)^T (b\mathbf{1} - y) + \lambda \|w\|_1 \end{aligned}$$

So for any $h > 0$, $v_w \in \mathbf{R}^d$ and $v_b \in \mathbf{R}$, we have

$$\begin{aligned} L(0, \bar{y}) &= (\bar{y}\mathbf{1} - y)^T (\bar{y}\mathbf{1} - y) \\ L(0 + hv_w, \bar{y} + hv_b) &= h^2 v_w^T X^T X v_w - 2hv_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) \\ &\quad + (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y)^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + \lambda h \|v_w\|_1 \\ &= h^2 v_w^T X^T X v_w - 2hv_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + h^2 v_b^2 \mathbf{1}^T\mathbf{1} \\ &\quad + 2hv_b\mathbf{1}^T (\bar{y}\mathbf{1} - y) + (\bar{y}\mathbf{1} - y)^T (\bar{y}\mathbf{1} - y) + \lambda h \|v_w\|_1 \\ \frac{L(0 + hv_w, \bar{y} + hv_b) - L(0, \bar{y})}{h} &= hv_w^T X^T X v_w - 2v_w^T X^T (\bar{y}\mathbf{1} + hv_b\mathbf{1} - y) + hv_b^2 \mathbf{1}^T\mathbf{1} \\ &\quad + 2v_b\mathbf{1}^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 \end{aligned}$$

Thus the one-sided directional derivative of L at $0, \bar{y}$ is

$$\begin{aligned} L'(0, \bar{y}; v_w, v_h) &= \lim_{h \downarrow 0} \frac{L(0 + hv_w, \bar{y} + hv_b) - L(0, \bar{y})}{h} \\ &= -2v_w^T X^T (\bar{y}\mathbf{1} - y) + 2v_b \underbrace{\mathbf{1}^T (\bar{y}\mathbf{1} - y)}_{=0} + \lambda \|v_w\|_1 \\ &= -2v_w^T X^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 \end{aligned}$$

So $(0, \bar{y})$ is a minimum iff

$$\begin{aligned} L'(0, \bar{y}; v_w, v_h) &\geq 0 \text{ for all } (v_w, v_h) \in \mathbf{R}^{d+1} \\ \iff -2v_w^T X^T (\bar{y}\mathbf{1} - y) + \lambda \|v_w\|_1 &\geq 0 \text{ for all } v_w \neq 0 \text{ (always true for } v_w = 0) \\ \iff \lambda &\geq \frac{2v_w^T X^T (\bar{y}\mathbf{1} - y)}{\|v_w\|_1} \text{ for all } v_w \neq 0 \\ \iff \lambda &\geq \sup_{v_w \in \mathbf{R}^d} \frac{2v_w^T X^T (\bar{y}\mathbf{1} - y)}{\|v_w\|_1} \\ &= 2 \max_{v_w : \|v_w\|_1 = 1} v_w^T X^T (\bar{y}\mathbf{1} - y) \\ &= 2\|X^T (\bar{y}\mathbf{1} - y)\|_\infty, \end{aligned}$$

where the last step is from the Lemma in the previous problem.

(From Peter Li) Alternative Solution for 3.1 Using Subgradients:

Let $g(\theta)$ be a subgradient of the lasso objective 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 and only if $S_0 = \frac{2X^Ty}{\lambda}$. 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, $\lambda \geq \left\|2X^Ty\right\|_\infty$.

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

$$X = \begin{pmatrix} x_1 & x_2 & X_r \end{pmatrix}, \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_r \end{pmatrix}$$

We can write the Lasso objective function as:

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

With repeated features, there will be multiple minimizers of $L(\theta)$. Suppose that

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

is a minimizer of $L(\theta)$. Give conditions on c and d such that $\left(c, d, r^T\right)^T$ is also a minimizer of $L(\theta)$. [Hint: First show that a and b must have the same sign, or at least one of them is zero. Then, using this result, rewrite the optimization problem to derive a relation between a and b.]
Solution:

Suppose $a < 0 < b$. Then we could construct $\hat{\theta}'$ with smaller objective value. In particular, take

$$\hat{\theta}' = \begin{pmatrix} a+b \\ 0 \\ r \end{pmatrix}.$$

Then the ℓ_1 penalty term is smaller, since $|a+b| < |a| + |b|$ if a and b have different signs. Meanwhile, the loss piece stays the same. So $L(\hat{\theta}') < L(\hat{\theta})$, which contradicts that $\hat{\theta}$ minimizes $L(\theta)$.

[2]See proposition 3.5 in Carlos Fernandez-Granda's Convex Optimization notes.

Now we can rewrite the objective function as

$$L(\theta) = \|(x_1(\theta_1 + \theta_2) + X_r\theta_r - y\|_2^2 + \lambda|\theta_1 + \theta_2| + \lambda \|\theta_r\|_1 ,$$

from which we can see that we can take so long as $c + d = a + b$ and c and d have the same sign, then $\left(c, d, r^T\right)^T$ also minimizes the Lasso objective.

2. Using the same notation as the previous problem, suppose

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

minimizes the ridge regression objective function. What is the relationship between a and b, and why?
Solution:

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

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

Note that the empirical risk piece of the objective function is invariant to changes to θ_1 and θ_2, so long as their sum remains the same. However, the regularization piece $\lambda(\theta_1^2 + \theta_2^2)$ will change. The regularization piece is minimized when $\theta_1 = \theta_2$, which can be shown with basic calculus.

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 theoretically.

5 [Optional] The Ellipsoids in the ℓ_1/ℓ_2 regularization picture

Recall the famous picture purporting to explain why ℓ_1 regularization leads to sparsity, while ℓ_2 regularization does not. Here's the instance from Hastie et al's *The Elements of Statistical Learning:*

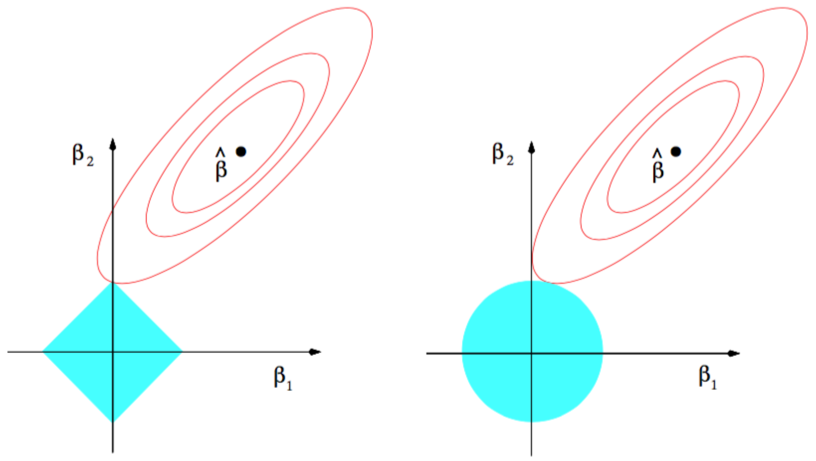


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the 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 $f(x) = w^T x$ 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. [Optional] 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:

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

2. [Optional] Show that for any w we have

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

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 $\hat{R}_n(w) = \frac{1}{n}(Xw - y)^T (Xw - y)$. After expanding this 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:

$$x^T Mx - 2b^T x = \left(x - M^{-1}b\right)^T M(x - M^{-1}b) - b^T M^{-1} b$$

Solutions:

$$(Xw - y)^T (Xw - y) = w^T X^T X w - 2y^T X w + y^T y$$

Let's complete the square with$M = X^T X$ and$b = X^T y$. Then

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

Putting it together,

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

3. [Optional] Using the expression derived for $\hat{R}_n(w)$ in 2, give a very short proof that $\hat{w} = \left(X^T X\right)^{-1} X^T y$ is the empirical risk minimizer. That is:

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

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

Solution:

Since X^TX is positive semidefinite, $(w-\hat{w})\,X^TX\,(w-\hat{w}) \geq 0$ for all w. In particular, it achieves its minimum value of 0 when $w=\hat{w}$. To spell it out in more detail, note that

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

4. [Optional] 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 $c>0$. If X is full rank, then X^TX is positive definite, and this set is an ellipse – what is its center?
Solution:

We're talking about the set of all w's that satisfy the following (equivalent) equations:

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

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

6 [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 vector θ as a difference of two non-negative vectors θ^+ and θ^-, the ℓ_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

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

where $h_{\theta^+,\theta^-}(x) = (\theta^+-\theta^-)^Tx$. 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^+ and θ^- to zero.

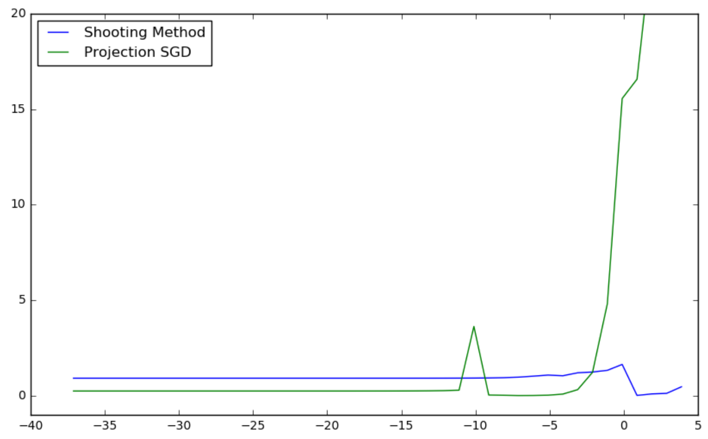
1. [Optional] Implement projected SGD to solve the above optimization problem for the same λ's as used with the shooting algorithm. Since the two optimization algorithms should find essentially the same solutions, you can check the algorithms against each other. Report the differences in validation loss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

Solution:

```
def find_positive(x):
    result = max(x, 0)
    return result

def positive_project(x):
    result = map(find_positive, x)
    return np.array(result)

def projection_SGD_split(X, y, theta_positive_0, theta_negative_0, lambda_reg = 1.0,
     alpha = 0.1, num_iter = 1000):
    m, n = X.shape
    theta_positive = np.zeros(n)
    theta_negative = np.zeros(n)
    theta_positive[0:n] = theta_positive_0
    theta_negative[0:n] = theta_negative_0
    times = 0
    theta = theta_positive - theta_negative
    loss = compute_square_loss(X, y, theta)
    loss_change = 1.
    while (loss_change>1e-6) and (times<num_iter):
        loss_old = loss
        for i in range(m):
            X_sample = X[i, :]
            y_sample = y[i]
            var_1 = np.dot(X_sample, theta.T)
            var_2 = var_1 - y_sample
            var_3 = 2*var_2*X_sample
            grad_positive = var_3 + lambda_reg
            grad_negative = (-1.)*var_3 + lambda_reg
            theta_positive = theta_positive - alpha*grad_positive
            theta_negative = theta_negative - alpha*grad_negative
            theta_positive = positive_project(theta_positive)
            theta_negative = positive_project(theta_negative)
            theta = theta_positive - theta_negative
        loss = compute_square_loss(X, y, theta)
        loss_change = abs(loss - loss_old)
        times +=1
    return theta
```



2. [Optional] Choose the λ 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?

Solution:
$\lambda = 0.0073$ minimizes the square loss on the validation set in this case. For the chosen λ, with true value 0 as the threshold, the sparsity is 0. With threshold 10^{-3}, the sparsity is 39. With threshold 10^{-2}, the sparsity is 70. The sparsity is better than the shooting method.