DS-GA 1003: Homework 2
Lasso

Due on Tuesday, Feb 16, 2016

Professor David Ronsenberg

Yuhao Zhao
Yz3085

1.Preliminaries

• 1.1 Feature Normalization.

– 1.1.1.

```
import numpy as np
import matplotlib.pyplot as plt
import time
from hw1 import *
from scipy.optimize import minimize
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import Ridge

X = np.random.rand(150,75)
```

– 1.1.2.

```
theta_true = 20*numpy.random.randint(2,size = 10)-10
theta_true = np.array([np.hstack((theta_true,np.zeros(65)))]).T
```

– 1.1.3.

```
noise = np.array([0.1*np.random.randn(150)]).T
y = np.dot(X,theta_true) + noise
```

– 1.1.4.

```
X_train_, X_test, y_train_, y_test = train_test_split(X, y, test_size =50, random_state=10)
X_train,X_validation,y_train,y_validation =
        train_test_split(X_train_, y_train_, test_size =20, random_state=11)
```

• 1.2 Experiment with Ridge Regression.

– 1.2.1.

```
temp = Ridge_lambda_search(X_train,y_train[:,0],X_validation,y_validation[:,0],
                        lambda_ = np.power(10, np.linspace(-7,6,100)))[0]
lambda_opt = np.power(10, np.linspace(-7,6,100))[np.where(temp  ==np.nanmin(temp))[0][0]]
beta_= regularized_grad_descent(X_train,y_train[:,0],lambda_reg = lambda_opt )[0][-1,:]
np.sum(theta_true[10:,:] != np.array([beta_]).T[10:])
```

According to the Ridge regression algorithm in assignment 1 (see code in appendix), the best λ is 10^{-7}. There are 65 components(every components) with true 0 but was estimated to be non-zero. None component with non-zero was estimated as zero.

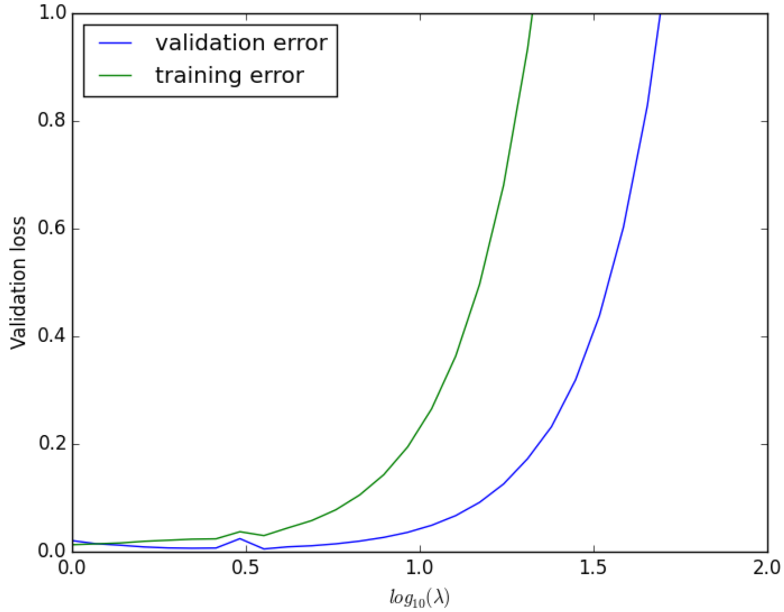
After thresholding by 10^{-3}, there are still 65 components with true 0 but was estimated to be non-zero.

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

• 2.1 Experiments with the Shooting Algorithm.

– **2.1.1.**

```
def LassoShooting(X, y, lambda_ = 0.1,epsilon = 0.0001,n_iter = 1000,
                                beta_init = np.zeros((X.shape[1],1)) ):
        """
        input: Training data X, traing data label, regulization, converging criterion,
        warm starting value for homotopy method.

        output: optimized beta and training error
        """
        num_instances, num_features = X.shape
        beta = beta_init
        t = 0
        converged = False
        Loss = np.linalg.norm(np.dot(X,beta) - y)**2 *(1.0/(2*num_instances))
        while (not converged and t <=n_iter ):
                beta_start = beta
                for j in range(num_features):
                        aj = 0
                        cj = 0
                        for i in range(num_instances):
                                aj += 2*X[i,j]**2
                                cj += 2*X[i,j] *(y[i][0] - np.inner(beta_start[:,0],X[i,:])
                                                                +beta_start[j][0]*X[i,j])

                        beta[j] = np.sign(aj)* (lambda x: np.sign(x[0])* ((abs(x[0]) - x[1])
                                        if (abs(x[0]) - x[1])>0 else 0))([cj/aj,lambda_/aj])
                t+=1
                converged = abs(Loss - np.linalg.norm(np.dot(X,beta) - y)**2
                                        *(1.0/(2*num_instances))) < epsilon
                Loss = np.linalg.norm(np.dot(X,beta) - y)**2*(1.0/(2*num_instances))
        return (beta,Loss)

try_list =  np.power(10,np.linspace(0,2,30))
loss_hist = np.zeros(try_list.shape[0])

time1 = time.time()
for i,reg in enumerate(try_list):
        beta = LassoShooting(X_train,y_train,lambda_ = reg)[0]
        loss_hist[i] = np.linalg.norm(np.dot(X_validation,beta) - y_validation)**2
                                *(1.0/(2*num_instances))
time_hist_1 = time.time()-time1
```



The optimal λ is 3.56224 and the corresponding test error is 0.11532799428214691

– 2.1.2.

```
>>> np.sum(theta_true[10:,:] != beta_opt[10:,:])
>>> 2
>>>beta_opt[:10,:]
>>>array([[ -9.1941852 ],
array([[-9.92480386],
       [-9.7845817 ],
       [ 9.69160619],
       [-9.74515336],
       [-9.73290275],
       [ 9.6222794 ],
       [-9.81244892],
       [-9.81588341],
       [ 9.63381792],
       [ 9.78238686]])
```

There are 2 components with true value zero that was estimated non-zero. None component with non-zero was estimated as zero.

– 2.1.3.

```
loss_hist_3 = np.zeros(try_list.shape[0])
pre = np.zeros((75,1))
time2 = time.time()
for i,reg in enumerate(try_list):
        beta = LassoShooting(X_train,y_train,lambda_ = reg,beta_init =pre )[0]
```

```
        pre,loss_hist[i] = beta, np.linalg.norm(np.dot(X_validation,beta) -
                                 y_validation)**2*(1.0/(2*num_instances))
time2  = time.time() -time2
```

The basic shooting algorithm takes 14.76 seconds and the homotopy method takes 17.2901 seconds.

– **2.1.4.**

```
def LassoShooting_mat(X, y, lambda_ = 0.1,epsilon = 0.0001,n_iter = 1000,
                        beta_init = np.zeros((X.shape[1],1)) ):
        """
        input: Training data, training labels, regulization, converging criterion,
        warm starting value for homotopy method.

        output: optimized beta and training error
        """
        num_instances, num_features = X.shape
        beta = beta_init
        t = 0
        converged = False
        XX2 = np.dot(X.T,X)*2;
        Xy2 = np.dot(X.T,y)*2
        Loss = np.linalg.norm(np.dot(X,beta) - y)**2*(1.0/(2*num_instances))
        while (not converged and t <=n_iter ):
                beta_start = beta
                for j in range(num_features):
                        aj = XX2[j,j]
                        cj = (Xy2[j] - np.dot(XX2[j,:],beta_start) + XX2[j,j]*beta_start[j])[0]
                        beta[j] = np.sign(aj)* (lambda x: np.sign(x[0])* ((abs(x[0]) - x[1])
                                        if (abs(x[0]) - x[1])>0 else 0))([cj/aj,lambda_/aj])
                t+=1
                converged = abs(Loss - np.linalg.norm(np.dot(X,beta) - y)**2
                                        *(1.0/(2*num_instances))) < epsilon
                Loss = np.linalg.norm(np.dot(X,beta) - y)**2*(1.0/(2*num_instances))
        return (beta,Loss)

pre = np.zeros((75,1))
time3 = time.time()
for i,reg in enumerate(try_list):
        beta = LassoShooting_mat(X_train,y_train,lambda_ = reg,beta_init =pre )[0]
        pre,loss_hist[i] = beta, np.linalg.norm(np.dot(X_validation,beta) -
                        y_validation)**2*(1.0/(2*num_instances))
time_hist_3 = time.time()-time3
```

Since $a_j = 2\sum_{i=1}^n x_{ij}^2 = 2 \times x_{:j}^T \cdot x_{:j}$, which is 2 times j,j th element of X^TX matrix

$c_j = 2\sum_{i=1}^n x_{ij}(y_i - w^Tx_i + w_jx_{ij}) = 2(\sum_{i=1}^n x_{ij}y_i - \sum_{i=1}^n w^Tx_ix_{ij} + \sum_{i=1}^n w_jx_{ij}x_{ij})$
$\sum_{i=1}^n x_{ij}y_i$ is the jth column of X dot product with y, which is the jth element of $2X^Ty$
$\sum_{i=1}^n w^Tx_ix_{ij}$ is the jth row of X^TX dot product with w
$\sum_{i=1}^n w_jx_{ij}x_{ij}$ is j,j th element of X^TX matrix times the jth element of w
Therefore, $c_j = 2([X^Ty]_j + [X^TX]_{j:} \cdot w + w_j \times [X^TX]_{j,j})$

The running time for this matrix based expression is 0.476298 seconds which is much better than the previous ones.

• 2.2 Derive the Coordinate Minimizer for Lasso.

– 2.1.1.

If $x_{i,j} = 0$ for $i = 1, ..., n$ $f(w_j) = \sum_i (y_i^2) + \lambda|w_j| + \lambda \sum_{k \neq j} |w_k|$
By taking derivative w.r.t w_j, the coordinate minimizer is $w_j = 0$

– 2.2.2.

For $w_j \neq 0$,

$$\frac{\partial f}{\partial w_j} = \sum_i 2(w_j x_{i,j} + \sum_{k \neq j} w_k x_{i,k} - y_i)x_{i,j} + \lambda sign(w_j)$$

$$= \sum_i 2w_j x_{i,j}^2 - 2\sum x_{i,j}(y_i - \sum_{j \neq k} w_k x_{i,k}) + \lambda sign(w_j) = a_j w_j - c_j + \lambda sign(w_j)$$

– 2.1.3.

For $w_j > 0, sign(w_j) = 1$, to solve $\frac{\partial f}{\partial w_j} = 0$, we have:

$$a_j w_j - c_j + \lambda = 0$$

Then

$$w_j = \frac{1}{a_j}(-\lambda + c_j) = -\frac{1}{a_j}(\lambda - c_j)$$

$w_j < 0, sign(w_j) = -1$, similarly,

$$w_j = \frac{1}{a_j}(\lambda + c_j)$$

– 2.1.4.

By definition, for $w_j = 0$ to be a minimizer, we have to show its two-sided derivatives are non-negative at f(0).
$\lim\limits_{\epsilon \downarrow 0} \frac{f(\epsilon) - f(0)}{\epsilon} = \lim\limits_{\epsilon \downarrow 0} \frac{\sum_i (\epsilon x_{i,j} + \sum_{k \neq j} w_k x_{i,k} - y_i)^2 + \lambda|\epsilon| + \lambda \sum_{k \neq j} |w_k| - \sum_i (\sum_{k \neq j} w_k x_{i,k} - y_i)^2 - \lambda \sum_{k \neq j} |w_k|}{\epsilon}$
$= \lim\limits_{\epsilon \downarrow 0} \frac{\sum_i (\epsilon^2 x_{i,j}^2 + 2\epsilon x_{i,j} \sum_{j \neq k} w_k x_{i,k} - y_i) + \lambda\epsilon}{\epsilon} = \lim\limits_{\epsilon \downarrow 0} \frac{\sum_i \epsilon^2 x_{i,j}^2}{\epsilon} - c_j + \lambda = \lambda - c_j \geq 0$
Then, we have $c_j \leq \lambda$
Similarly, on the other side, $\lim\limits_{\epsilon \downarrow 0} \frac{f(-\epsilon) - f(0)}{\epsilon} = \lim\limits_{\epsilon \downarrow 0} \frac{\sum_i (\epsilon^2 x_{i,j}^2 - 2\epsilon x_{i,j} \sum_{j \neq k} w_k x_{i,k} - y_i) + \lambda\epsilon}{\epsilon} = c_j + \lambda \geq 0$
Then we have $c_j \geq -\lambda$
Therefore, $c_j \in [-\lambda, \lambda]$ implies $w_j = 0$ is a minimizer.

– 2.1.5.

From 2.1.3 we know that for $w_j > 0$, we have the solution $w_j = \frac{1}{a_j}(c_j - \lambda)$, since $a_j \geq 0$, we have $c_j > \lambda$
For $w_j < 0$, we have the solution $w_j = \frac{1}{a_j}(c_j + \lambda)$, since $a_j \geq 0$, we have $c_j < -\lambda$
From 2.1.4, we know that for $c_j \in [-\lambda, \lambda]$, the solution is $w_j = 0$
The minimizer is indeed

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

On the other hand, $soft(\frac{c_j}{a_j}, \frac{\lambda}{a_j}) = 0$ for $c_j \in [-\lambda, \lambda], = \frac{1}{a_j}(c_j - \lambda)$ for $c_j > \lambda$ and $= \frac{1}{a_j}(c_j + \lambda)$ for $c_j < -\lambda$. This expression is equivalent to the expression given in 2.

3 Lasso Properties

- **3.1 Deriving λ_{max}.**

  - **3.1.1.**

Since $L(w) = (Xw - y)^T(Xw - y) + \lambda|w|_1$

$$L'(0,v) = \lim_{h\downarrow 0} \frac{L(vh) - L(0)}{h} = \frac{(hXv - y)^T(hXv - y) + \lambda h|v|_1 - (-y)^T(-y)}{h}$$

$$= \lim_{h\downarrow 0} \frac{h^2v^TX^TXv - 2hv^TX^Ty + y^Ty - y^Ty + \lambda h|v|_1}{h} = -2v^TX^Ty + \lambda|v|_1$$

  - **3.1.2.**

In order for w^* to be a minimizer, L(w) we must have non-negative directional derivative in any direction v.
$L'(0,v) \geq 0$, then $-2v^TX^Ty + \lambda|v|_1 \geq 0$.
$\lambda \geq \frac{2v^TX^Ty}{|v|_1}$

  - **3.1.3.**

Since the lower bounds on λ should hold for all v, $\lambda > g(v) = \frac{2v^TX^Ty}{|v|_1}$ for all v. Since v is a directional vector, we constrain $||v||_{L2} = 1$
$|v|_1^2 = (|v_1| + |v_2| + ... + |v_n|)^2 \geq (|v_1 + v_2 + ... + v_n|)^2 \geq v_1^2 + v_2^2 + ... + v_n^2 = 1$ and the equal sign occurs when $v_i = 1, v_j = 0 \forall j \neq i$
The maximum value for $f(v)$ therefore occurs when $v_i = 1$ where i is the index that X^Ty has the largest absolute value at the ith index.
Therefore, $max(g(v)) = 2||X^Ty||_\infty$. This is equivalent to say $\lambda \geq 2||X^Ty||_\infty$
$\lambda_{max} = 2||X^Ty||_\infty$

  - **3.1.4.**

For model with bias, $L(w) = (Xw + b - y)^T(Xw + b - y) + \lambda||w||_1$
$L'(0,v) = \lim_{h\downarrow 0} \frac{L(vh) - L(0)}{h} = \frac{(hXv+b-y)^T(hXv+b-y)+\lambda h|v|_1-(b-y)^T(b-y)}{h}$
$= \lim_{h\downarrow 0} \frac{h^2v^TX^TXv+2hv^TX^T(b-y)+\lambda h|v|_1}{h} = 2v^TX^T(b-y) + \lambda|v|_1 \geq 0$
Therefore, $\lambda \geq \frac{2v^TX^T(b-y)}{|v|_1}$

- **3.2 Feature Correlation.**

  - **3.2.1.**

Since $X_{.i}, X_{.j}$ are exactly the same, we can regard $\hat{\theta}_i, \hat{\theta}_j$ as one parameter C. we want to solve
Min$|\hat{\theta}_i| + |\hat{\theta}_j|$, s.t $\hat{\theta}_i + \hat{\theta}_j = C$
We notice that to minimize the function, $\hat{\theta}_i, \hat{\theta}_j$ should have the same sign.
Now we solve C, C should be the j-th coefficient that was solved by removing the i-th column in X:
From Question 2, we know that:

$a_j = 2\sum_m x_{m,j}^2, c_j = 2\sum_m x_{mj}(y_m - \sum_{k\neq j\neq i} w_k x_{mk})$
Therefore, depends on the value of c_j,

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

If we know that in the optimal solution, $\hat{\theta}_i = a, \hat{\theta}_j = b$, then $\hat{\theta}_i + \hat{\theta}_j = a + b = C$

– **3.2.2.**

Since Ridge regression has a solution, we just have to take derivative of the loss function.
$L = (X\theta - y)^T(X\theta - y) + \theta^T\theta$
$\frac{\partial L}{\partial \theta} = 2X^TX\theta - 2X^Ty + 2\lambda\theta = 0$

$\frac{\partial L}{\partial \theta_i} = (X_{.i}\cdot X_{.1}, X_{.i}\cdot X_{.2}, ..., X_{.i}\cdot X_{.n})\begin{bmatrix}\theta_1\\ \theta_2\\ ...\\ \theta_n\end{bmatrix} - X_{.i}\cdot y + 2\lambda\theta_i = 0$

$\frac{\partial L}{\partial \theta_j} = (X_{.j}\cdot X_{.1}, X_{.j}\cdot X_{.2}, ..., X_{.j}\cdot X_{.n})\begin{bmatrix}\theta_1\\ \theta_2\\ ...\\ \theta_n\end{bmatrix} - X_{.j}\cdot y + 2\lambda\theta_j = 0$

Since $X_{.i} = X_{.j}$, the above two equation are the same. Therefore $\hat{\theta}_i = \hat{\theta}_j$

– **3.2.3.**

When $X_{.i}, X_{.j}$ are highly correlated, ridge will have a equal or very similar solution to θ_i, θ_j, but lasso will have one non-zero θ, and another zero θ. Because in the optimizations, lasso only arbitrarily select one feature and optimize along that direction. Our target is to minimize $|\theta_i| + |\theta_j|$ subject to $\theta_i + \theta_j = C$, optimization along one direction means zero at another direction.

4 The Ellipsoids in the l_1/l_2 regularization picture

• 4.1.

Since $\hat{w}^T = y^TX(X^TX)^{-1}$

$$\hat{R}(\hat{w}) = \frac{1}{n}(\hat{w}^TX^TX\hat{w} + y^Ty - \hat{w}^Tx^Ty) = \frac{1}{n}(y^TX\hat{w} + y^Ty - 2\hat{w}^TX^Ty)$$

Since $y^TX\hat{w} = \hat{w}^TX^Ty$

$$\hat{R}(\hat{w}) = \frac{1}{n}(y^Ty - y^TX\hat{w})$$

• 4.2.

$n(\hat{R}(w) - \hat{R}(\hat{w})) = (Xw - y)^T(Xw - y) - y^Ty + y^TX\hat{w}$
$= w^Tx^TXw + y^Ty - 2w^TX^Ty + y^TX\hat{w} - y^Ty$
$= w^TX^TXw - 2w^TX^Ty + y^TX\hat{w}$
$= w^TX^TXw - w^TX^Ty - y^TXw + \hat{w}^TX^Ty$
$= w^T(X^TXw - X^Ty) - (y^TXw - \hat{w}^TX^Ty)$
$= w^T(X^TXw - X^Ty) - \hat{w}^T(X^TXw - X^Ty) = (w^T - \hat{w}^T)(X^TXw - X^TX\hat{w}) = (w - \hat{w})^TX^TX(w - \hat{w})$
Therefore, $\hat{R}(w) = \frac{1}{n}(w - \hat{w})^TX^TX(w - \hat{w}) + \hat{R}(\hat{w})$

- **4.3.**

Since from 4.2 $\hat{R}(w) = \frac{1}{n}(w-\hat{w})^T X^T X(w-\hat{w}) + \hat{R}(\hat{w})$
$X^T X$ is positive semi-definite, for any $(w-\hat{w}) \neq 0$, $(w-\hat{w})^T X^T X(w-\hat{w}) > 0$
In order to minimize the empirical risk, we need to set $w - \hat{w} = 0$, then the risk is minimized by $w^* = \hat{w}$

- **4.4.**

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

Then,

$$(w-\hat{w})^T X^T X(w-\hat{w}) = nc$$

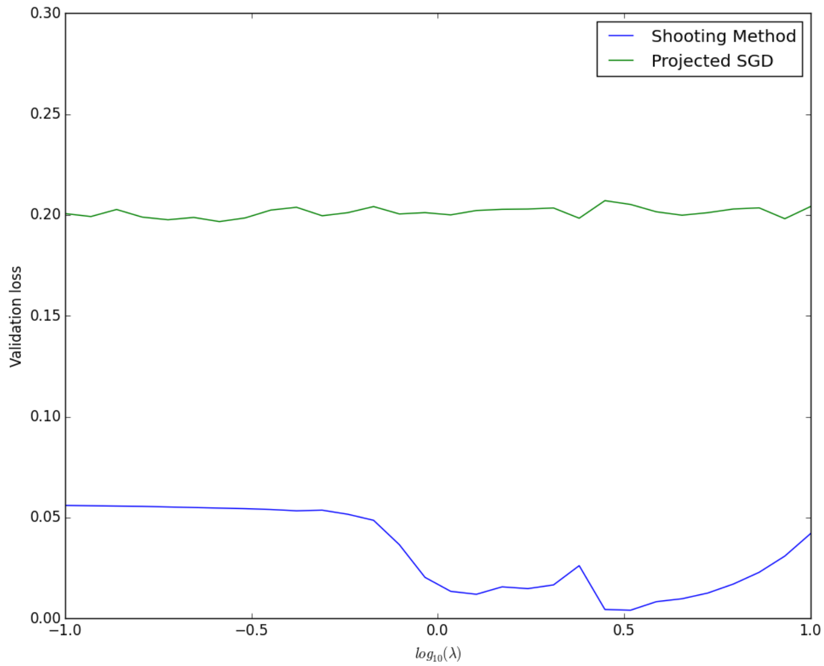
This set of w is an ellipse, and the center is $w = \hat{w}$

5 Projected SGD via Variable Splitting

- **5.1.**

$\nabla_{\hat{\theta}^+} L(\hat{\theta}^+, \hat{\theta}^+) = 2((\hat{\theta}^+ - \hat{\theta}^-)^T x_i - y_i)x_i + \lambda I_{n,1}$
$\nabla_{\hat{\theta}^+} L(\hat{\theta}^+, \hat{\theta}^+) = -2((\hat{\theta}^+ - \hat{\theta}^-)^T x_i - y_i)x_i + \lambda I_{n,1}$

```
def projected_SGD(X, y, lambda_ = 0.1,alpha = 0.01, num_iter = 1000,
                theta_init = np.zeros((X.shape[1],1))):
        num_instances, num_features = X.shape[0], X.shape[1]
        theta_p = theta_init
        theta_n = theta_init
        theta = theta_p - theta_n

        for n in list(xrange(num_iter)):
                generator = np.random.permutation(list(xrange(num_instances)))
                for i in list(xrange(0,num_instances)):

                num = generator[i]
                theta_p = theta_p - alpha*(np.inner(X[num,:],theta.T) - y[num])*
                                        np.array([X[num,:]]).T +lambda_
                theta_n = theta_n +   alpha*(np.inner(X[num,:],theta.T) - y[num])*
                                        np.array([X[num,:]]).T +lambda_

                theta_p = (theta_p>0)*theta_p
                tehta_n = (theta_n<0) *theta_n
                theta = theta_p - theta_n

        loss_hist = np.linalg.norm(np.dot(X,theta)-y)**2 *(1.0/(2*num_instances))

return (theta,loss_hist)
```



From the plot we can see that, the Shooting Method have less validation loss for all different regularization parameters.

- **5.2.**

According to the result, the best λ that performed best is 0.2592. And the solution is not sparse at all. There are none zero solutions in the $\hat{w}$

Appendix

```
def Ridge_lambda_search(X_train , y_train ,X_test, y_test, lambda_):
        """
        checking the convergency for different regulization lambda
        returns the last element of loss_hist returned by regularized_grad_descent
        """

        res_training =  np.zeros(len(lambda_))
        res_testing = np.zeros(len(lambda_))

        for i in list(xrange(len(lambda_))):
                temp = regularized_grad_descent(X_train,y_train,lambda_reg=lambda_[i])
                theta = temp[0][-1,:]
                res_testing[i] = compute_square_loss(X_test,y_test,theta)
                res_training[i] = compute_square_loss(X_train,y_train,theta)

        return (res_testing,res_training)

def regularized_grad_descent(X, y, alpha=0.1, lambda_reg=1, num_iter=1000):

        (num_instances, num_features) = X.shape
        theta = np.ones(num_features) #Initialize theta
        theta_hist = np.zeros((num_iter+1, num_features))  #Initialize theta_hist
        loss_hist = np.zeros(num_iter+1) #Initialize loss_hist

        theta_hist[0,:] = theta
        loss_hist[0] = compute_square_loss(X,y,theta) + lambda_reg*np.linalg.norm(theta)**2

        for i in list(xrange(1,num_iter+1)):
                theta = theta - compute_regularized_square_loss_gradient(X,y,
                                        theta,lambda_reg)*alpha
                theta_hist[i,:] = theta
                loss_hist[i] = compute_square_loss(X,y,theta) +
                                        lambda_reg*np.linalg.norm(theta)**2

        return (theta_hist,loss_hist)

def compute_regularized_square_loss_gradient(X, y, theta, lambda_reg):

        return compute_square_loss_gradient(X,y,theta) + 2*lambda_reg*theta
```