Machine Learning and Computational Statistics, Spring 2015
Homework 2: Lasso

Qingxian Lai(ql516)

1 Preliminaries

1.1 Dataset construction

Answer: the code I used to genertate the data:

```
def data_construct():
   m = 150 # number of examples
   d = 75 # number of feature

   X = np.random.rand(m, d)

   theta = [-10, -10, -10, 10, 10, -10, 10, 10, 10, -10] + [0] * 65
   theta = np.array(theta).reshape(d, 1)

   epsilon = 0.1 * np.random.randn()
   y = X.dot(theta) + epsilon

   # split dataset into train, validation and test data set
   X_tv, X_test, y_tv, y_test = train_test_split(X, y, test_size=50,
       random_state=10)
   X_train, X_vali, y_train, y_vali = train_test_split(X_tv, y_tv,
       test_size=20, random_state=11)

   return X_train, y_train, X_vali, y_vali, X_test, y_test
```

And the θ is:

```
[-10 -10 -10  10  10 -10  10  10  10 -10   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0]
```

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`.)
Answer: As is shown in Figure 1, the validation loss keep increasing when $\lambda > 1e-4$, so

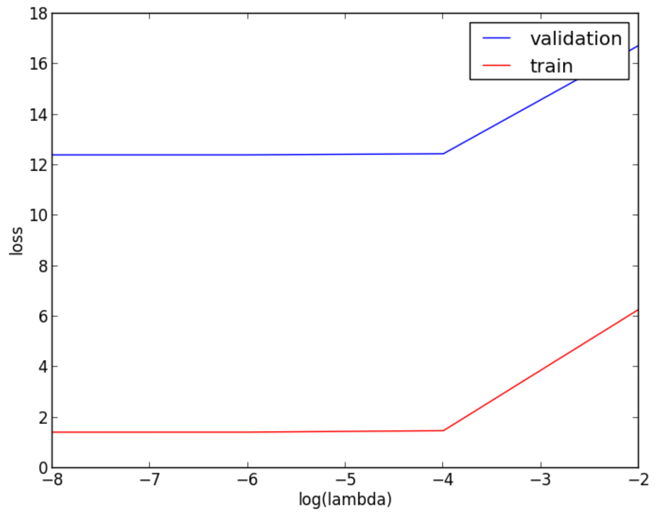


Figure 1: train loss and validation loss on different lambda

I choose $\lambda = 1e-4$,then I run the ridge regression with this λ, the result W is :

```
[-8.90973159 -8.10844847 -7.56545994  8.77586662  5.65054499 -6.67732259
  8.26641141  9.24328174  8.88410617 -8.70701354 -0.93197111  0.88908685
  0.56360085 -1.52224448  0.08329375  0.47150314 -1.42709043 -0.38352426
  0.0562735  -0.66616167 -1.33086361 -0.20521177 -1.25332038 -0.56264614
  0.57774202 -0.25611004  0.24811151  0.12800406 -2.38346156  0.33348048
  1.0484739  -2.15854848  1.4554505   1.11488761  0.39059882  0.18212963
  1.40864692  0.08552983 -0.47468652  0.74497685 -0.18371419 -0.01160113
 -1.90251595  2.07686493  0.70795689  0.59377522 -0.57368072 -0.72444929
 -0.22714665  0.56813434 -0.84015742 -0.59456302  0.76728675  1.68871346
  1.2061098   2.40856721 -0.86809609  0.78808535  0.43425638 -2.7669742
  0.04397375  0.07349903  2.12350904 -0.82458204 -1.47224954  0.17421986
 -1.1034607   0.13256707  0.25089754  1.3975324   0.6848754  -1.72133199
  0.98909634 -0.97011985  0.05567288]
```

so, all of them has been estimated as non-zero while their true values are zero.(threshold= 0.001)

2 Lasso

The Lasso optimization problem can be formulated as

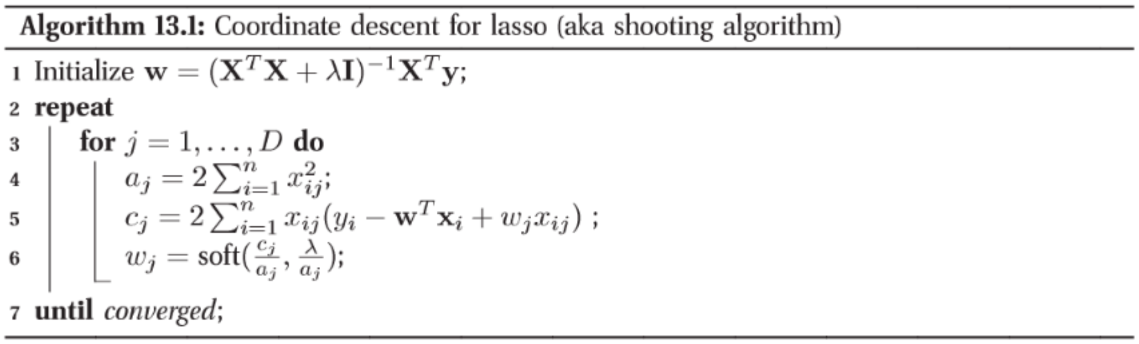
$$\hat{w} = \underset{w \in \mathbf{R}^d}{\arg\min} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + \lambda \|w\|_1,$$

where $h_w(x) = w^T x$.

Since the ℓ_1-regularization term in the objective function is non-differentiable, vanilla gradient descent cannot solve this optimization problem. Next, we will learn two techniques to solve the optimization problem.

2.1 Shooting algorithm

One standard way to solve an optimization problem is coordinate descent, in which at each step, we optimize over one component of the unknown 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)

```
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 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

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

+ 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. 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 $\lim_{m\to\infty} P[\text{Supp}(\hat{\beta}_m) = \text{Supp}(\beta)] = 1$.

There are a few tricks that can make selecting the hyperparameter λ 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 optimal λ in $[0, \lambda_{\max}]$, where $\lambda_{\max} = 2\|X^T(y-\bar{y})\|_\infty$.

Second, we can make use of the fact that when λ and λ' 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 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 technique of computing a set of solutions for a chain of nearby λ'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!)

Hence, to optimize for λ, we start with $\lambda = \lambda_{\max}$, and solve the resulting optimization problem. This process is repeated till the error on the test sample keeps reducing.

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.
   Answer: The code of shoorting methods:

```
def shooting_method(X_train, y_train, X_vali, y_vali,w_init,lam):
   n,d = X_train.shape
   loss = [lasso_loss(X_train,y_train,w_init,lambda_reg=lam)]
   W = w_init # W is a numpy array
   vali_loss = []
   iter = 0
   while True:
      a = np.zeros(d)
      c = np.zeros(d)
      for j in range(d):
         a[j] = 2*(np.linalg.norm(X_train[:,j])**2)
         c[j]=0
         for i in range(n):
            tmp =
                X_train[i,j]*(y_train[i]-W.T.dot(X_train[i])+W[j]*X_train[i,j])
            c[j] = c[j] + tmp
         c[j] = 2*c[j]
         test = c[j]/a[j]
         W[j] = soft(c[j]/a[j],lam/a[j])

      vali_loss.append(compute_square_loss(X_vali,y_vali,W))
```

```
        iter = iter +1
        if iter>300: break
    W = W.reshape(d,1)
    vali_loss = compute_square_loss(X_vali,y_vali,W)[0,0]
    lass_loss = lasso_loss(X_train,y_train,W,lam)
    return lass_loss,vali_loss,W
```

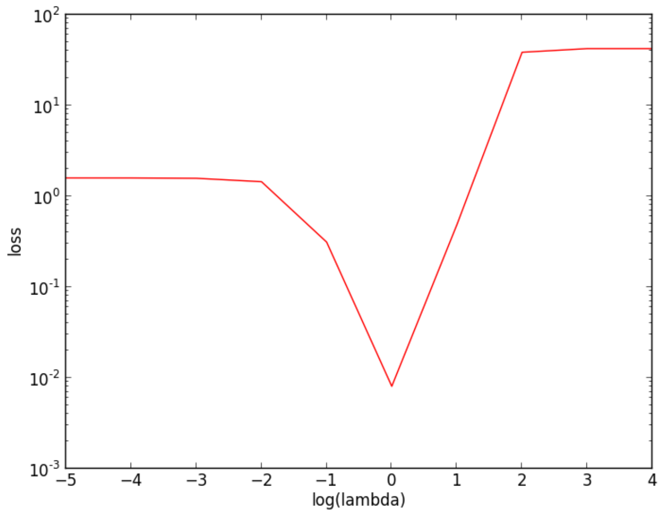


Figure 2: validation loss on different lambda

As is shown in Figure 2, I choose $\lambda = 10^0 = 1$ as the optimal value of λ. At this time, the test error is: 0.0237

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.
   Answer: The result w is :

```
[[ -9.94657212e+00  -9.91238811e+00  -9.94712964e+00   9.79733187e+00
    9.90573534e+00  -9.96978766e+00   9.83506738e+00   9.95941398e+00
    9.88937163e+00  -9.90671506e+00  -0.00000000e+00  -0.00000000e+00
    0.00000000e+00   0.00000000e+00   4.63300555e-03   6.44457873e-03
    0.00000000e+00  -0.00000000e+00   0.00000000e+00   2.39311242e-02
    7.11317798e-03   0.00000000e+00   3.99079153e-03   0.00000000e+00
    0.00000000e+00   0.00000000e+00   1.38271428e-02   8.14784803e-03
    2.02603278e-02   3.63187762e-02   0.00000000e+00   3.27821538e-03
    0.00000000e+00   1.46236747e-03   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   -0.00000000e+00   2.32313262e-02   0.00000000e+00   0.00000000e+00
    0.00000000e+00   5.70615783e-02   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00  -0.00000000e+00   0.00000000e+00
    0.00000000e+00   1.22274907e-02   0.00000000e+00   8.79283746e-04
    0.00000000e+00   5.75594872e-03   9.24983473e-03   0.00000000e+00
    0.00000000e+00   0.00000000e+00   0.00000000e+00   1.48268085e-02
    0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   -0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   -0.00000000e+00   0.00000000e+00   0.00000000e+00]]
```

28 components are estimated as non-zero while their true value is zero.

3. Implement the homotopy method described above. Compare the runtime for computing the full regularization path (for the same set of λ's chosen above) using the homotopy method compared to starting with the same intial point every time.
Answer: In this question, I choose the same λ set, which contains 9 value from λ_{max} to 0. Then use both homotopy and non-homotopy method to compute the regularization path and time them. The result is:

   (a) Homotopy method : 22.355 seconds
   (b) Non-homotopy method : 31.194 seconds

   The homotopy method is much faster.

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 here: `https://code.google.com/p/pmtk3/source/browse/trunk/toolbox/Variable_selection/lassoExtra/LassoShooting.m?r=1393`).
Answer:

$$a_j = 2 * \sum_{i=1}^{n} X_{i,j}^2 = 2 * X_{,j}^T X_{,j}$$

$$\begin{aligned} c_j &= 2 * \sum_{i=1}^{n} X_{i,j}(y_i - w^T X_i + w_j X_{i,j}) \\ &= 2 * [X_{,j}^T y - X_{,j}^T X w + w_j X_{,j}^T X_{,j}] \\ &= 2 * X_{,j}^T [y - Xw + w_j X_{,j}] \end{aligned}$$

5. Implement the matrix expressions and measure the speedup to compute the regularization path.
Answer: Computing the regularization path with the same λ set, which contains 9 value from λ_{max} to 0, and both using the homotopy method:

   (a) before vectorization : 25.705 seconds
   (b) after vectorization : 1.283 seconds

So, there are 24.422 seconds speedup

6. (Optional) Derive the expression used in the algorithm for the coordinate minimizer w_j. (Hint: Most if it is worked out in KPM's book as well as our slides on subgradients. The missing piece is writing the derivative of the empirical loss in terms of a_j and c_j.)

7. (Optional) Derive the above expression for $\lambda_{\max}$.

2.2 (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

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

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^+ 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.)
Answer: the code of implementing projected SGD:

```python
def stochastic_grad_descent(X, y, X_vali,y_vali, alpha=0.05,
    lambda_reg=50, num_iter=100):

    num_instances, num_features = X.shape[0], X.shape[1]
    rand = np.arange(num_instances)

    theta_p = np.random.rand(num_features)
    theta_n = np.random.rand(num_features)

    np.random.shuffle(rand)
    t = 1
    losses = []

    for i in range(num_iter):
        for j in range(num_instances):
            alpha = 0.01

            x = X[rand[j]]
            yy = y[rand[j]]

            tmp1 = projected_SGD_p(x,yy,theta_p,theta_n,lambda_reg)
            theta_p_tmp = theta_p - alpha*tmp1
            a = [theta_p_tmp>=0,theta_p_tmp<0]
            choice = [theta_p_tmp,0]
            theta_p_tmp = np.select(a,choice)

            tmp2 = projected_SGD_n(x,yy,theta_p,theta_n,lambda_reg)
            theta_n_tmp = theta_n - alpha*tmp2
            a = [theta_n_tmp>=0,theta_n_tmp<0]
            choice = [theta_n_tmp,0]
            theta_n_tmp = np.select(a,choice)

            theta_p = theta_p_tmp.copy()
            theta_n = theta_n_tmp.copy()

        loss = compute_square_loss(X_vali,y_vali,(theta_p-theta_n))[0,0]
        losses.append(loss)

    return losses,theta_p-theta_n
```

Using both projected SGD and Shooting methods to calculate the validation loss on different λ, I got the following graph: Figure 3

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?
Answer: The Figure 3 shows that the best λ is $1e-2$, using this λ, I got the optimal w as follow:

```
[ -9.80144302e+00  -9.84241880e+00  -9.86249877e+00   9.90336727e+00
   9.83591782e+00  -9.87848557e+00   9.99172972e+00   9.89074051e+00
   9.97662735e+00  -9.91477181e+00   1.86279738e-03   1.63508122e-03
   2.85561042e-04  -1.21503371e-03   2.13502337e-03   2.30491328e-03
   1.40232245e-03   1.63895441e-03   1.72565687e-03   1.18428843e-03
   2.08154525e-03   1.71861173e-03   4.21785567e-03   2.04483973e-03
   7.12051596e-04   1.99518024e-03   2.43254958e-03   1.60823558e-03
   1.40702931e-03   1.93722707e-03   2.77658725e-03   6.43422168e-04
   9.08680824e-03   1.90338800e-03  -2.68653628e-04   1.81259881e-03
   1.66225266e-03   2.19658572e-03   2.63723304e-03   2.03196838e-03
   2.08765200e-03   1.30398015e-03   1.88867987e-03   2.84497630e-04
   2.43494959e-03   6.84720083e-04   1.15000890e-04   1.85422072e-03
   1.56783547e-03   1.82386276e-03   1.23223988e-03   2.11266925e-04
```

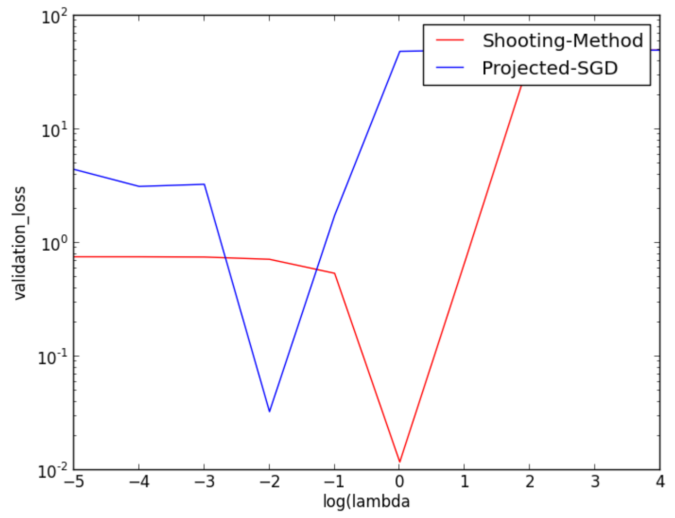


Figure 3: validation loss comparison between projected SGD and Shooting Method on different lambda

```
6.24289185e-04   1.54809995e-03   1.55589054e-03   1.87363919e-03
5.62515594e-04   2.58427441e-03   1.07257014e-03   2.58607074e-03
2.64301269e-03   2.86901161e-03   1.27868496e-03   1.53572575e-03
2.69986862e-03   1.78290408e-03   7.55802845e-04   1.75117521e-03
1.40284618e-03   1.98130677e-03   2.82148854e-03   7.08500862e-03
2.28939220e-02   1.19728937e-03   2.43059979e-03]
```

With a threshold 0.001, only 22 components of w were estimated as non-zero while their true value is zero. Although they are very close to zero, none of these components are exact zero.

2.3 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. 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.]

Answer: let X_i, X_j denote the identical columns in X, and W_i, W_j denote the corresponding weight W. Without loss of generality, we set them as the last two columns in X and the last two components in W:

$$X = [X', X_i, X_j]$$
$$W = [W', W_i, W_j]^T$$

And its optimal solution is $\widehat{W} = [\widehat{W'}, a, b]^T$, Then the Emprical risk function is:

$$J(W) = \|X'W' + X_iW_i + X_jW_j\|^2 + \lambda\|W'\|_1 + \lambda|W_i| + \lambda|W_j|$$

Let its partial derivatives on W_i, W_j equal to zero, we have:

$$\frac{\partial J(W)}{\partial W_i} = 2 * (X'W' + X_iW_i + X_jW_j)^T X_i + \lambda * sign(W_i) = 0$$
$$\frac{\partial J(W)}{\partial W_j} = 2 * (X'W' + X_iW_i + X_jW_j)^T X_j + \lambda * sign(W_j) = 0$$

Then, we have:

$$\lambda * sign(a) = -2 * (X'W' + X_i * a + X_j * b)^T X_i$$
$$\lambda * sign(b) = -2 * (X'W' + X_i * a + X_j * b)^T X_j$$

Since "$X_i = X_j$, the right sides of the above two equations are equal, so their left sides are equal, which means a and b has the same sign.
With this result, we can rewrite the emprical rish function as follow:

$$J(W) = \|X'W' + X_i(W_i + W_j)\|^2 + \lambda\|W'_1\| + \lambda|W_i + W_j|$$

Let $v = (W_i + W_j)$, we can get its optimal solution: $\widehat{W} = [\widehat{W'}, (a + b)]^T$, since this is a convex problem, the $\widehat{W}$ is the only optmal solution. So the relationship between a, b is that their sum is constant.

2. 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 ridge regression optimization problem.
Answer: set the partial derivatives equal zero, we have:

$$\frac{\partial J(\theta)}{\partial \theta_i} = 2 * (X\theta - y)^T X_{,i} + 2 * \lambda\theta_i = 0$$
$$\frac{\partial J(\theta)}{\partial \theta_j} = 2 * (X\theta - y)^T X_{,j} + 2 * \lambda\theta_j = 0$$

Then we have:

$$\theta_i = -\frac{1}{\lambda}(X\theta - y)^T X_{,i}$$
$$\theta_j = -\frac{1}{\lambda}(X\theta - y)^T X_{,j}$$

Since $X_{,i} = X_{,j}$, we know $\theta_i = \theta_j$.

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.

3 Feedback (not graded)

1. Approximately how long did it take to complete this assignment?
2. Any other feedback?