Homework Set 1, CPSC 8420

LastName, FirstName

Due 10/07/2024, 11:59PM EST

Lasso

As we discussed in class, one of the most efficient algorithm to find the optimal solution to Lasso is *coordinate minimization* (also called alternating minimization) that every time we fix the rest while optimizing a certain coordinate until convergence.

$$\frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_1, \tag{1}$$

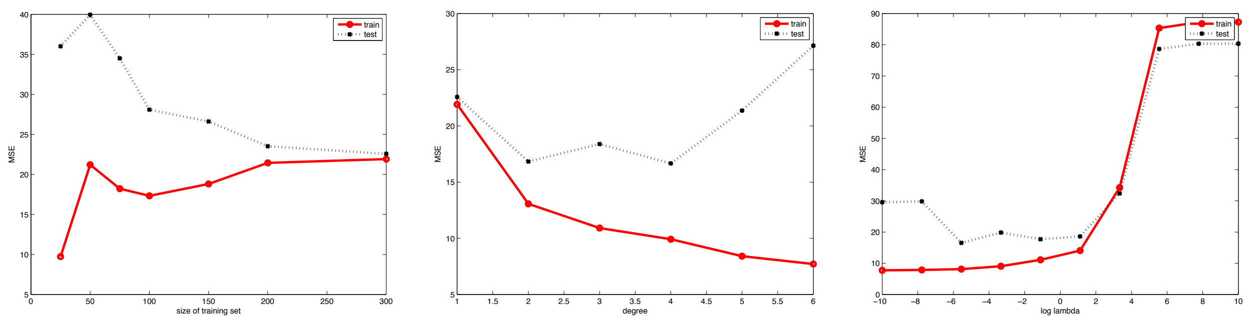
Now, please randomly generate $A \in \mathbb{R}^{20\times 10}, y \in \mathbb{R}^{20}$, and change λ from 0.01 to 1000 with *logspace*. You are expected to implement the codes with coordinate minimization method and the function should accept any A, b, λ and return x^*. Compare the output with standard Lasso solver built in Matlab or python. And finally please plot the changes of x with λ varies.

Least Squares Extension

Assume $\mathbf{A}, \mathbf{X}, \mathbf{C}, \mathbf{Y}, \mathbf{U} \in \mathbb{R}^{n\times n}$. Given the fact that $vec(\mathbf{AUC}) = (\mathbf{C}^T \otimes \mathbf{A})vec(\mathbf{U})$, where $\otimes$ denotes *Kronecker product*, please use *least squares* we learned in class to solve:

$$\min_{\mathbf{X}} \|\mathbf{AX} + \mathbf{XC} - \mathbf{Y}\|_F^2. \tag{2}$$

To verify the correctness of your solution: 1) randomly generate $\mathbf{A}, \mathbf{X}^*, \mathbf{C}$; 2) set $\mathbf{Y} = \mathbf{AX}^* + \mathbf{X}^*\mathbf{C}$; 3) by making use of least squares, you can obtain the optimal $vec(\mathbf{X})$ given $\mathbf{A}, \mathbf{C}, \mathbf{Y}$ and 4) compare with $vec(\mathbf{X}^*)$.



(a) (b) (c)

Figure 1: MSE vs (a) training set size, (b) polynomial degree, (c) size of ridge penalty. Solid Red = training, dotted black = test.

Ridge Regression

Please first conduct experiments to show that $\|(A^TA + \lambda\mathbf{I})^{-1}A^Ty\|_2$ is a monotone (decreasing) function with respect to λ (you may randomly generate A, y and plot the objective changes with λ increases). After you are pretty confident the argument is true, please prove it mathematically.

Linear Regression and its Extension

In the Boston housing dataset, there are 506 records. We will use first 13 features as inputs, x, and the 14th feature, median house price, as the output y. All features are continuous, except feature 4, which is binary. However, we will treat this like any other continuous variable.

1. Load the housing.data file. We will use the first 300 cases for training and the remaining 206 cases for testing. However, the records seem to be sorted in some kind of order. To eliminate this, we will shuffle the data before splitting into a training/test set. So we can all compare results, let use the following convention:

```
data = load('housing.data');
x = data(:, 1:13);
y = data(:,14);
[n,d] = size(x);
seed = 2; rand('state',seed); randn('state', seed);
perm = randperm(n); % remove any possible ordering fx
x = x(perm,:); y = y(perm);
Ntrain = 300;
```

```
Xtrain = x(1:Ntrain,:); ytrain = y(1:Ntrain);
Xtest = x(Ntrain+1:end,:); ytest = y(Ntrain+1:end);
```

2. Now extract the first n records of the training data, for $n \in \{25, 50, 75, 100, 150, 200, 300\}$. For each such training subset, standardize it (you may use *zscore* function in Matlab), and fit a linear regression model using least squares. (Remember to include an offset term.) Then standardize the whole test set in the same way. Compute the mean squared error on the training subset and on the whole test set. Plot MSE versus training set size. You should get a plot like Figure 1(a). Turn in your plot and code. Explain why the test error decreases as n increases, and why the train error increases as n increases. Why do the curves eventually meet? As a debugging aid, here are the regression weights I get when I train on the first 25 cases (the first term is the offset, w0): $[26.11, -0.58, 3.02, \ldots, -0.21, -0.27, -1.16]$.

3. We will now replace the original features with an expanded set of features based on higher order terms. (We will ignore interaction terms.) For example, a quadratic expansion gives:

$$\begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{pmatrix} \rightarrow \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} & x_{11}^2 & x_{12}^2 & \ldots & x_{1d}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} & x_{n1}^2 & x_{n2}^2 & \ldots & x_{nd}^2 \end{pmatrix} \tag{3}$$

   The provided function degexpand(X,deg,addOnes) will replace each row of X with all powers up to degree deg. Use this function to train (by least squares) models with degrees 1 to 6. Use all the the training data. Plot the MSE on the training and test sets vs degree. You should get a plot like Figure 1(b). Turn in your plot and code. Explain why the test error decreases and then increases with degree, and why the train error decreases with degree.

4. Now we will use ridge regression to regularize the degree 6 polynomial. Fit models using ridge regression with the following values for λ:

$$lambdas = [0 \ logspace(-10, 10, 10)]$$

   Use all the training data. Plot the MSE on the training and test sets vs $log_{10}(\lambda)$. You should get a plot like Figure 1(c). Turn in your plot and code. Explain why the test error goes down and then up with increasing λ, and why the train error goes up with increasing λ.