# AMATH 563 Hand-written Digits Recognition

**Huasong Zhang**
Department of Applied Mathematics
University of Washington
Seattle, WA 98195
huasongz@uw.edu

## Abstract

In this project, we were given a dataset which contains a large number of hand-written digit pictures. Each picture contains one digit and it was in a 28 by 28 pixel. Our goal in this assignment is to consider this classification task into a $A * X = b$ problem, where the A is the dataset, b is the actual digit that is shown in the picture. What we were doing is to solve x, which represents the weights on each pixel for every digit. In this homework assignment, I used different methods to solve this $A * x = b$ problem, which are least square fit, Ridge regression, and LASSO regression.

## 1 Introduction and Overview

Machine learning is playing an important role nowadays. one of the most basic and important application in machine learning is classification. This assignment is about hand-written digit recognition, so that by running the algorithm, the machine can classify what number is written. This technology is widely used in the postal office. In the old times, postal officers had to separate the mails and packages by hand. It would definitely takes a long time and it was easy to make mistakes. Nowadays, when this digit recognition techniques are applied, it's more efficient, convenient and faster. Therefore, getting a high accuracy of classification and shorter running time is the most important thing we need to pay attention to in this assignment.

## 2 Theoretical Background

### 2.1 Least Square Fit

I have used various methods to solve the $A * x = b$ problem. The most basic, which is the one we need to try first is the least square fit. Since we already have the ground truth b, our goal here is to find the x which minimizes

$$\sum (A * x - b)^2$$

.

### 2.2 Ridge Regression

After applying the most basic least square fit for this linear system, the next thing we can do is to add an extra penalty term to the regression model to regulate the model. One thing we can do is Ridge regression. The objective function for Ridge regression is:

$$minimize_x(|A * x - b|_2^2 + \lambda * |x|_2)$$

We can separate this equation into two parts. The first part is the same as the linear regression. The extra second part is the regularization term. The $\lambda$ is the parameter,and the $|x|_2$ is that we took the l2 norm of x. We are basically changing the value of $\lambda$ tp penalize the regularization term. The larger the $\lambda$, the larger the penalty term will be, which forces the magnitude of the coefficient to be small. This method will gradually make the majority of x close to 0 when making the value of $\lambda$ large, but it will not make magnitude of x exactly 0.

### 2.3 LASSO Regression

Another method which adds penalty to linear fit is LASSO regression. Instead of adding a l2 norm penalty, this method will add a l1 penalty to the regression model. The objective equation for LASSO is:

$$minimize_x(|A * x - b|_2^2 + \lambda * |x|)$$

The only difference is in the second part. The l1 norm here will promote sparsity by making some coefficients exactly 0. Therefore, the dimension for x will be reduce. This is useful when we have a great number of predictors but not all of them are necessary. By using LASSO regression, we are able to select and use the most informative ones to predict.

## 3 Algorithm Implementation and Development

The fist step to start is assignment is by loading the dataset. From this dataset, I stored the data into training set and testing set. Also, for the image data, I changed all the 28 by 28 pixel into one row vector and stacked all the image together row by row. Also, for each pixel, I divided by 255 to normalize it. Therefore, I have the matrix A which is my image data. For the labels of these images, they were originally stored as the digit which is the same as digit written in the picture; in order to get more accurate in predicting each digit, I used one hot coding to change the labels into a matrix. In this matrix, the element in a row is 1 if the index is the digit shown, while it is 0 otherwise. Therefore, I got a matrix of the labels which is the b matrix. After getting A and b matrix, I could start solving $A * x = b$ using various methods.

First of all, I implemented the least square fit by using "pinv" command. Since this image data matrix has more images comparing to the dimension of the image, I need to ues pseudoinverse. The "linalg.solve" command in "numpy" can only be used when A is a square matrix. By doing matrix multiplication with pseudoinverse of A and b, I can get x. And this multiplication was actually minimizing the l2 norm of $A * x - b$. Then, I applied Ridge regression and LASSO regression by using the command in sklearn.

After calculating, we get the x with dimension 784 by 10. In order to get the most informative pixels from analysis from the whole x matrix, I took the absolute value of x and sum across the column, so that I get 784 elements eventually. Then I ploted each method, and the ones with large number means more informative. Then I sorted and took out the largest 50 numbers and found out their indexes, and the indexes are the pixels. This is the process of analysis for rank of 50. I also tried 100 and 300 so that I can see how the rank affect the accuracy. For the three methods, the most informative pixels are different. After getting the pixels, I created another vector with length 784 with each element equal to 1. I kept the elements on the important indexes as one and changed all others to 0. Then I multiplied this vector with the x vector, so that I can get a new x vectors with the original dimension but the weights for the unimportant pixels are 0. To get the prediction, I just multiplied image data matrix with this new x matrix, and for each row, I could just picked out the largest number. And the index where the largest number at is the digit we classified. The last step is to compare the result we get with the true labels for test set, then we would get the accuracy.

To do analysis for each digit, it was very similar. The only difference is the part of getting the most informative pixels. So this times, we need to pick out the most important pixels column by column in x, instead of combining them together and find the largest 50. After finding out the indexes, as what I did before, I created a vector containing only 1's and change the unimportant elements to 0 and multiply it with the specific column that I did analysis on. After finishing modifying all columns of x, I got a new x matrix. Then I multiplied image data matrix with this new x and took out the largest index in each row from the results, which were my predictions. Then I compared my predictions with the true test label, and the accuracy was calculated.

## 4 Computational Results

The following accuracy result is calculated using the full rank of x. For LASSO and Ridge regression method, I took $\lambda = 0.05$. From the result we can see that the basic least square fit did a good job in classification.

```
The accuracy by using pinv is 0.8534
The accuracy by using LASSO is 0.7699
The accuracy by using Ridge is 0.8116
```

The Figure1 Figure2 and Figure3 shows the analysis on the entire x matrix. The ones with higher bar means more informative. As we can see from three bar plots that for each method, the most important pixels are different across different methods.

In order to analysis the effects of rank on accuracy, I took 50, 100, and 300 as rank, and tested on the testing set to see the difference. As we can see from the following results that when the rank is low, the LASSO has the highest accuracy since this method forces some coefficients to 0 already; while the other two will never make coefficients equal to 0. As the rank number goes up, the accuracy goes up as well. At last, the accuracy for three methods will be very close.

```
Accuracy with rank =  50 for method pinv is 0.099
Accuracy with rank =  50 for method LASSO is 0.5467
Accuracy with rank =  50 for method Ridge is 0.0989
Accuracy with rank =  100 for method pinv is 0.1021
Accuracy with rank =  100 for method LASSO is 0.6461
Accuracy with rank =  100 for method Ridge is 0.1027
Accuracy with rank =  300 for method pinv is 0.4569
Accuracy with rank =  300 for method LASSO is 0.7699
Accuracy with rank =  300 for method Ridge is 0.4671
```

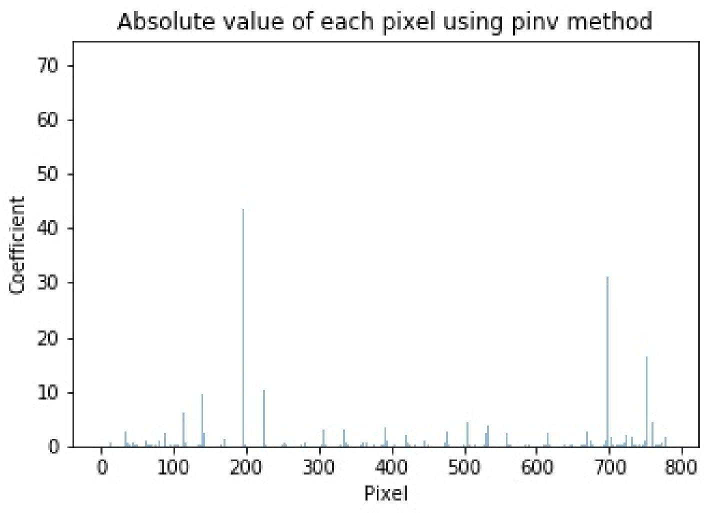

Figure 1: Coefficients for pinv

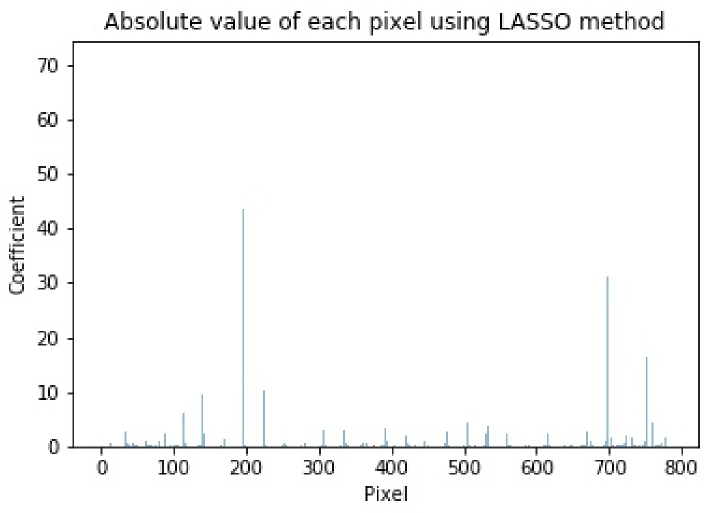

Figure 2: Coefficients for LASSO

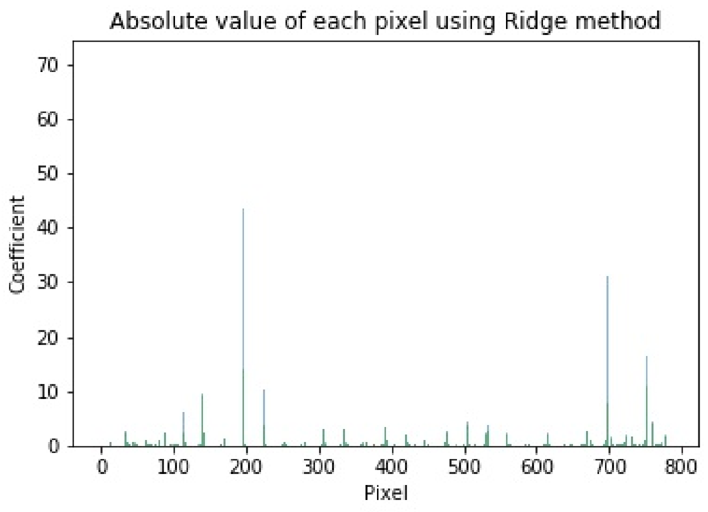

Figure 3: Coefficients for Ridge

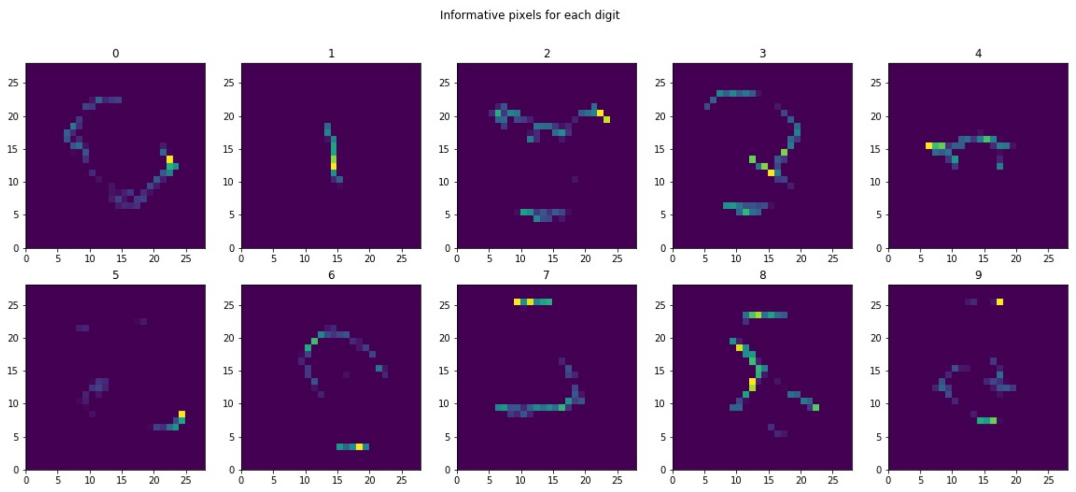

Figure 4: Analysis on informative pixels for each digit

However, what is interesting here is when I did analysis of reducing dimension of x for each digit, LASSO did not have the highest accuracy. At the same level of $rank = 300$, pinv and Ridge regression achieves higher accuracy comparing taking the same pixel index. This makes sense because each digit have different weights on each pixel. Therefore, when we do specific analysis for each digit, a higher accuracy is achieved. The result is shown as below.

```
Accuracy with rank = 300 for method pinv is 0.6962
Accuracy with rank = 300 for method LASSO is 0.7014
Accuracy with rank = 300 for method Ridge is 0.7022
```

## 5 Summary and Conclusions

In my testings, when I used the full rank of x, pinv achieves the highest accuracy; however, I think LASSO and Ridge may perform better than they have right now if I change value of $\lambda$. In the future, I can further investigate this assignment by doing cross validation to find the best $\lambda$ for LASSO and Ridge. Hopfully they can perform better than what I get right now. For a low rank, LASSO performs better because it already forces some of the pixels equal to 0. Therefore, it may not affect the result if we take a low rank. At the same level of rank, reducing the dimension based on each digit will create higher accuracy than reducing dimension of x as a whole. Also, I used the method of taking the absolute value for each element in x and add them up across the column. Other methods can also be used to get the vector x and they may have different results.

## Appendix Codes

```
from mnist import MNIST
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure


# load data
def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
    return X_train, labels_train, X_test, labels_test
```

```
X_train, Y_tr, X_test, Y_te = load_dataset()
Y_train = np.zeros((60000,10))
Y_train[np.arange(60000), Y_tr] = 1
Y_test = np.zeros((10000,10))
Y_test[np.arange(10000), Y_te] = 1
m = X_train.shape[1]
n = Y_train.shape[1]
method = ['pinv','LASSO','Ridge']


# use various methods to solve this problem
# pinv
x1 = np.dot(np.linalg.pinv(X_train), Y_train)

# LASSO
clf = linear_model.Lasso(alpha=0.005)
clf.fit(X_train, Y_train)
x2 = clf.coef_
x2 = np.transpose(x2)

# Ridge
clf1 = Ridge(alpha=0.005)
clf1.fit(X_train, Y_train)
x3 = clf1.coef_
x3 = np.transpose(x3)


# Analysis on the overall coefficients
sum_x1 = np.sum(abs(x1),axis = 1)
plt.bar(np.arange(m), sum_x1, alpha=0.5)
plt.xlabel('Pixel')
plt.ylabel('Coefficient')
plt.title('Absolute value of each pixel using pinv method')
plt.savefig('coeff_pinv.jpeg')
sum_x2 = np.sum(abs(x2),axis = 1)
plt.bar(np.arange(m), sum_x2)
plt.xlabel('Pixel')
plt.ylabel('Coefficient')
plt.title('Absolute value of each pixel using LASSO method')
plt.savefig('coeff_LASSO.jpeg')
sum_x3 = np.sum(abs(x3),axis = 1)
plt.bar(np.arange(m), sum_x3, alpha=0.5)
plt.xlabel('Pixel')
plt.ylabel('Coefficient')
plt.title('Absolute value of each pixel using Ridge method')
plt.savefig('coeff_Ridge.jpeg')


# find out the accuracy
for i in range(3):
    count = 0
    X = globals()["x" + str(i+1)]
    globals()['Y'+str(i+1)] = np.dot(X_test,X)
    globals()['Y'+str(i+1)] = np.argmax(globals()['Y'+str(i+1)], axis=1)
    for j in range(len(globals()['Y'+str(i+1)])):
        if globals()['Y'+str(i+1)][j] == Y_te[j]:
            count += 1
    accuracy = count/len(globals()['Y'+str(i+1)])
    print('The accuracy by using ',method[i],'is ',accuracy)
```

```
# find out the most informative pixels sum_xi_d for i = 1:5
R = np.array([50,100,300])
for j in range(3):
    r = R[j] #fix r
    a = np.ones((10,784))
    for i in range(3):
        x = np.zeros(784,)
        xx = globals()['sum_x'+str(i+1)]
        index = xx.argsort()[-r:][::-1]
        x[index] = 1
        x_index = np.transpose(a*x)
        x_reduced = globals()['x'+str(i+1)]*x_index
        # test accuracy
        pre = np.dot(X_test, x_reduced)
        l = len(pre)
        count = 0
        for h in range(l):
            max_value = max(pre[h,])
            max_index = np.where(pre[h,] == max_value)
            if max_index[0][0] == Y_te[h]:
                count += 1
        accuracy = count/l
        print('Accuracy with rank = ',R[j],'for method',method[i],'is', accuracy)


r = 300
# Analysis on the coefficient for each digit
for j in range(3):
    x = globals()['x'+str(j+1)]
    for i in range(10):
        x_initial = np.zeros(784,)
        x_digit = x[:,i]
        index = x_digit.argsort()[-r:][::-1]
        x_initial[index] = 1
        x[:,i] = x_digit*x_initial
    globals()['x_digit_reduced_'+str(j+1)] = x

# calculate the accuracy
for i in range(3):
    pre = np.dot(X_test, globals()['x_digit_reduced_'+str(i+1)])
    count = 0
    l = len(pre)
    for h in range(l):
        max_value = max(pre[h,])
        max_index = np.where(pre[h,] == max_value)
        if max_index[0][0] == Y_te[h]:
            count += 1
    accuracy = count/l
    print('Accuracy with rank = 300','for method',method[i],'is', accuracy)


fig, ax = plt.subplots(2,5,figsize=(20, 8))
for i in range(10):
    ax = plt.subplot(2,5,i+1)
    size = np.reshape(x_digit_reduced_2[:,i],(28,28))
    plt.pcolor(size)
    ax.set_title(i)
fig.suptitle('Informative pixels for each digit')
fig.savefig('digit.jpeg')
```