Regression and Digit Recognition

Ron Wilson

April 17, 2020

Abstract

Algorithms of machine learning are implemented on the `MNIST` [1] data set in order to interpret the results of the $AX = B$ problem associated with this data set.

1 Introduction and Overview

The `MNIST` data set includes training and testing sets. The training set contains 60000 images (each image is 28 pixels by 28 pixels) of handwritten digits, each digit corresponding to one number from 0 to 9, and a vector of labels representing the digit contained in each of the images. The testing set also has images and their labels, but only contains 10000 images. In the context of solving an $AX = B$ problem, the A matrix is the images, while the B matrix is their labels. The goal of this project is to find the X matrix for the training set and use it to try and accurately match up the digits of the testing set with their labels.

2 Theoretical Background

In MATLAB, you can solve $AX = B$ by using the backslash command ($X = A\backslash B$). However, we have a non-square A matrix in this problem. Since A has dimension 60000-by-784, we have an over-determined system. This means that there are an infinite number of solutions. Thus, there is no unique solution to this problem. Because of this, I will be using three different solution methods: Pseudo-Inverse method, Lasso regression, and Ridge regression.

2.1 Pseudo-Inverse Method

The pseudo-inverse of a matrix is used to solve a linear system of equations in which there is no unique solution, which fits for our over-determined system. You can compute the pseudo-inverse of a matrix in MATLAB with the command `pinv(A)` (see Appendix A). The result can then be used to solve the following equation for X:

$$AX = B \implies pinv(A) * AX = pinv(A) * B \implies X = pinv(A) * B \tag{1}$$

The solution for X from the Pseudo-Inverse method is the minimum-norm least squares solution to the system. Thus, the X found in equation 1 is the same as the solution of the following:

$$\underset{X}{\text{minimize}}\, |AX - B|^2 \tag{2}$$

2.2 Lasso Regression

Equation 2 gives a basic regression model of the system of equations. However, we can also add a penalty term to the regression in order to regulate the model. One of these potential models is the following:

$$\underset{X}{\text{minimize}}(|AX - B|_2^2 + \lambda\,|X|) \tag{3}$$

Instead of using the minimum-norm for $AX = B$, here we are taking the L2 norm. Also, the regulation term $\lambda\,|X|$ adds an L1 penalty to the regression. This model is called Lasso regression. We can easily solve equation 3 for X in MATLAB with the `lasso(A,B,λ)` command (see Appendix A).

2.3 Ridge Regression

Alternatively, you could use an L2 penalty term instead of an L1 term as in a Ridge regression model:

$$\underset{X}{\text{minimize}}(|AX - B|_2^2 + \lambda\,|X|_2) \tag{4}$$

In MATLAB, you can solve equation 4 with the command `ridge(B,A,λ)` (see Appendix A).

It is important to note that the different penalty terms of the three methods (L1, L2, and 0) will affect the magnitude of the regression coefficients and thus produce much different solutions for X.

3 Algorithm Implementation and Development

In this section I will detail the algorithm used on the data set. For the MATLAB code corresponding to this algorithm, see Appendix B.

1. First, we need to load the data sets. To do this, I used the functions `loadMNISTImages` and `loadMNISTLabels` (see Appendix A) for both the training set and testing set. See [2] for these functions.
   - Note that the images are scaled by $\frac{1}{255}$ to normalize the data within the `loadMNISTImages` function.
2. Next, the labels are returned as values from 0 to 9 representing the digit in each image. However, we want B to be a `[number of MNIST images]`-by-10 matrix where each column represents one digit from 0 to 9. To do this, I one-hot encoded the label vector into the desired B matrix. In this process, the 1 digit was encoded to be the following vector:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

   I then encoded the remaining digits in a similar way, where the 1 is in the position representing the digit (note that the 0 digit has the 1 in the 10th position).
3. Now I have the matrices A and B representing the images and their labels, respectively. I can now implement the three different $AX = B$ solvers on the training set. For the Pseudo-Inverse method, the computation follows equation 1 exactly. For the Lasso & Ridge regression methods, I used the MATLAB commands `lasso` and `ridge` within a loop, where I am looping through the 10 columns of B in order to generate coefficients of X that are the correct dimensions for the problem.
   - Recall that the equations for Lasso & Ridge regression (equations 3 & 4) used the coefficient λ on the penalty term. I chose $\lambda = 0.1$, by trying multiple λ values during cross-validation of Lasso regression. This value produced the most accurate results, so I used the same λ for Ridge regression as well.
4. Next, I plotted the absolute values of X from each of the three different methods to observe which pixels are the most important for correctly labeling the digits. In these plots, the pixels with the larger magnitude are the most informative. See figures 1, 2, and 3 for the Pseudo-Inverse method, Lasso regression, and Ridge regression plots, respectively.

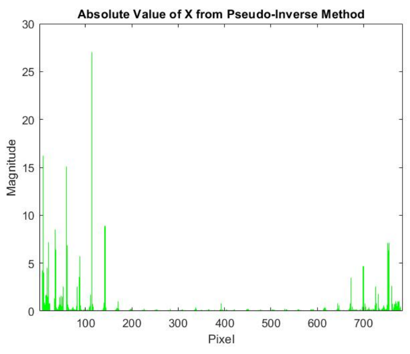

Figure 1: Pseudo-Inverse pixels

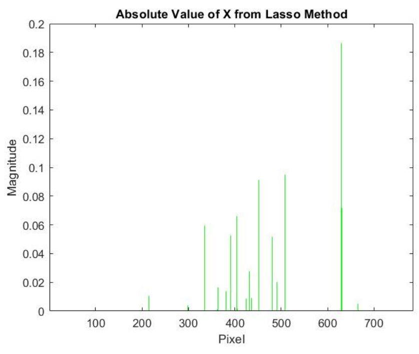

Figure 2: Lasso regression pixels

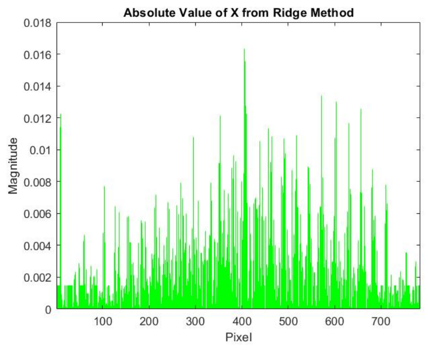

Figure 3: Ridge regression pixels

5. I decided to take the 50 most important pixels (pixels with the largest magnitude) to try and correctly label the digits in the testing set. To do this, I sorted the absolute values of X in descending order, such that the first 50 indices correspond to the 50 largest values. I then created a matrix of zeros, except at the indices corresponding to the largest values, which had ones in their place. I used element-wise matrix multiplication of this matrix with the X matrix to generate a sparse matrix X that contains only the most informative pixels, and zeros for everything else.

6. With the sparse matrix X for each of the three methods, I computed $A * X$ and checked if that product was equal to B, where A and B are the images and labels of the testing set. The check produced a matrix of logical values; 0 if the digit was labeled incorrectly and 1 if it was labeled correctly. I then used a counter to count how many images were labeled correctly, and divided this counter by the total number of images to get the accuracy of the most important pixels on the testing set and plot the accuracy's for each of the three methods.

7. Finally, I repeated steps 5-6, but for each digit individually. So I took the 50 most important pixels for each digit and tested on each individual column to get an accuracy for each of the 10 digits in the three methods, and plotted a bar graph showing how accurate the method was on each digit.

4 Computational Results

Figures 1, 2, & 3 show drastically different magnitudes of the absolute values of X. This is because each method uses a different penalty term. Thus, the most important pixels in correctly labeling the testing set are different for each method. There are a handful of spikes in the plots of the Pseudo-Inverse method and Lasso regression. However, the Ridge regression plot has values much more close together in magnitude. In order to accurately predict the correct digits in the testing set, I chose to use the 50 largest values for each method. 50 was chosen because it is in between the number of spikes of the first two figures (which were

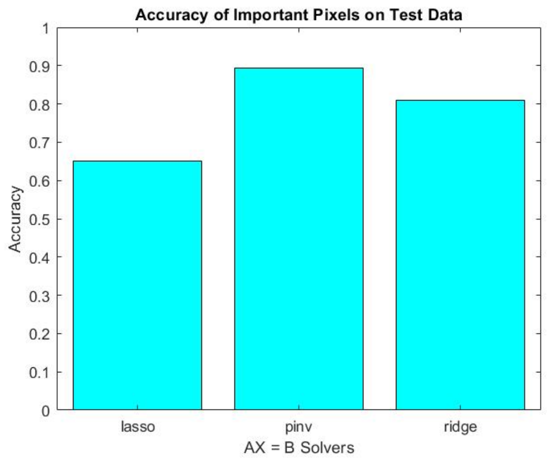

Figure 4: Accuracy's of the three different $AX = B$ solvers on the test data

about 25 values) and the much larger number of spikes in the third figure.

Using only the 50 most important pixels for each of the three methods, I plotted a bar graph of the accuracy's of each method on correctly labeling the testing set. Figure 4 shows these accuracy's. For the Pseudo-Inverse method, the accuracy was 89.45%. Lasso regression had an accuracy of 65.17%, while Ridge regression was 80.98% accurate. It is a little surprising that Lasso regression produced a much less accurate result than the other methods and that Pseudo-Inverse method was more accurate than either regression methods. To better understand what is happening here, we can look at the accuracy's of the three methods on each of the ten digits.

In figure 5, you can see how accurate the labeling was for each digit using the Pseudo-Inverse method. Here, we can see why this method produced such a good accuracy previously. The accuracy for every digit is around 90%. Now let's compare this with the accuracy's on each digit for the two regression methods.

Figure 6 shows the accuracy of individual digits from Lasso regression, while the accuracy of each digit from Ridge regression can be seen in figure 7. Lasso regression was very inconsistent on the individual digits, which explains the low accuracy from figure 4. It was highly accurate for the 2nd, 4th, 5th, 7th, 8th, & 9th

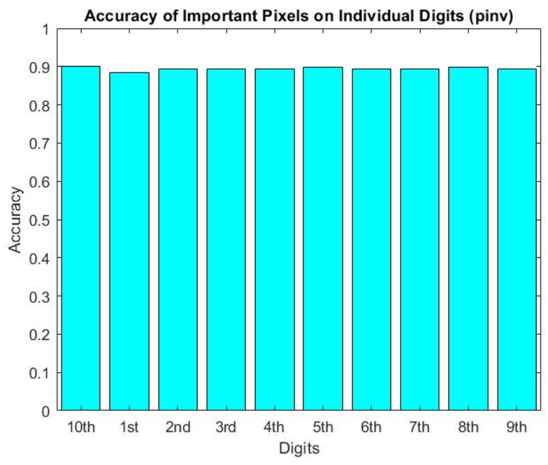

Figure 5: Pinv accuracy on individual digits

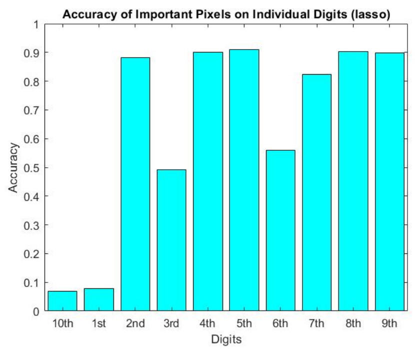

Figure 6: Lasso accuracy on individual digits

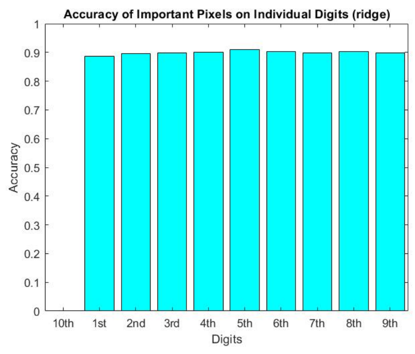

Figure 7: Ridge accuracy on individual digits

digits, but digits 3 & 6 were not very accurate and digits 1 & 10 were very inaccurate. Meanwhile, for Ridge regression, every digit except the 10th was very accurate. The 10th digit was 0% accurate, which is very strange, especially considering the high accuracy of the other digits.

5 Summary and Conclusions

The fact that the Pseudo-Inverse method produced highly accurate results while the other two methods were a little more inconsistent suggests that the added penalty term in the regression models has a big affect on the overall accuracy's. My choice of λ is clearly important here. Additional cross-validation of these methods, trying many more choices for λ, could greatly improve the accuracy of the regression models.

For the purposes of this project, solving $AX = B$ for a mapping from the image space to the label space can be fairly easy. However, using this mapping with algorithms of machine learning to accurately label digits in the testing set is more difficult. There are further elements that could be explored to improve the accuracy. Another thing to consider would be additional $AX = B$ solvers from the three that I chose. But of the three I chose, the Pseudo-Inverse method was the most accurate.

References

[1] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *The MNIST Database of Handwritten Digits*. URL: `http://yann.lecun.com/exdb/mnist/`.

[2] David Stutz. *Introduction to Neural Networks*. Seminar Report, Human Language Technology and Pattern Recognition Group, RWTH Aachen University, 2014. URL: `https://github.com/davidstutz/matlab-mnist-two-layer-perceptron`.

Appendix A MATLAB Functions

The following is a list of the important MATLAB functions that were used:

- `images = loadMNISTImages(filename)` returns a 28^2`-by-[number of MNIST images]` matrix containing the raw `MNIST` images. See [2] for the function.
- `labels = loadMNISTLabels(filename)` returns a `[number of MNIST images]-by-1` matrix containing the labels for the `MNIST` images. See [2] for the function.
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension.
- `B = pinv(A)` returns the Moore-Penrose Pseudoinverse of matrix `A`.
- `B = lasso(X,y,Name,Value)` returns fitted least-squares regression coefficients for linear models of the predictor data `X` and the response `y` with additional options specified by one or more name-value pair arguments.
- `B = ridge(y,X,k)` returns coefficient estimates for ridge regression models of the predictor data `X` and the response `y`. Each column of `B` corresponds to a particular ridge parameter `k`.
- `bar(x,y,color,Name,Value)` creates a bar graph with one bar for each element in `y` with bars drawn at the locations specified by `x`, sets the color for all the bars, and specifies properties of the bar graph using one or more name-value pair arguments.
- `B = sort(A,direction)` sorts the elements of `A`, treating the columns of `A` as vectors and sorting each column in the order specified by `direction`.

Appendix B MATLAB Code

The MATLAB code used to generate the results of this write-up (`Regression_and_Digit_Recognition.m`) can be found at the following GitHub repository: `https://github.com/ronwilson016/Data-Science-Projects/tree/master/Machine%20Learning/Regression%20and%20Digit%20Recognition/MATLAB%20Code`