Optimal Design Project

Cameron Sprenger

February 2024

1 Introduction

For this project, we are asked to formulate a least-squares classifier that can identify hand-written digits from 0-9 from the "mnist" data set. The data is labeled and separated into "training" and "testing" data. The training set is used to generate the model, and the testing data is used to verify that the model is correct and accurate. The model's performance will be measured as the classification accuracy of the testing data as it is entirely unique and separate from the training set.

The data sets consist of 60,000 and 10,000 images for the training and testing sets respectively. Each image has a resolution of 28x28 pixels and has been concatenated into a row vector. In other words, each row of the original image is slid to the end of the row above, turning the 28x28 image into a 784x1 vector with pixel values between 0 and 255 (8-bit). This makes handling large amounts of data much simpler as each row of the data set now represents an entire image.

The classification method that will be used is known as "1 vs the rest" because the generated classifiers are binary checkers. Each classifier is only able to determine if an images is or isn't the number that it is checking for. This is different from a multi-class classifier which takes in an image and directly outputs what number it thinks it is, but for now, that is beyond the scope of this class. The decision making structure of the classifiers can be set up in two different ways, one being analogous to a size-based coin sorter where the data is fed into the top, and binned into the first classifier that positively identifies the images as being a certain digit. The second method would be to check an image with all of the classifiers and determine which score is the most positive and classify the images as being that digit. The first method is advantageous because of the reduced computation time required to classify each of the images at the expense of accuracy. The second method is more accurate because, in theory, it can classify numbers that are ambiguous as it takes into consideration the score of each of the classifiers, not just the first one with a positive identification. For this project, I will use the second method of classification because the goal is to generate an accurate classifier with less focus on computational efficiency.

2 Background

Least squares is the method of finding the line of best fit for a data set. It calculates the distance between a data point and a line of best fit which is referred to as the residual, and minimizes the sum of squares of the residuals. The line can be of any order, but I will be using a linear least squares as it is what we've covered in class and is relatively simple. Linear least squares has the form:

$$Ax = b \tag{1}$$

In order to solve equation 1 for x, both sides must be multiplied by A^+ which is known as the pseudo-inverse. The pseudo-inverse is a matrix A^+ that solves the least squares problem $min\|Ax - b\|^2$ and $x \perp N(A)$. This means the solution, x is the shortest vector in the solution space as it is orthogonal to A. For a least squares problem, the pseudo-inverse is equal to $(A^T A)^{-1} A^T$ and the derivation can been seen below.

2.1 Least Squares Derivation

The objective of least squares is to minimize the sum of the residuals squared which can be expressed mathematically as:

$$min(J) = (mx_1 + b - y_1)^2 + (mx_2 + b - y_2)^2 + \ldots + (mx_n + b - y_n)^2 \tag{2}$$

Where y are the values of the data points, and $mx + b$ is the line of best fit evaluated at the corresponding x. The residuals can be both positive and negative, but the magnitude of the residuals is ultimately what we want to evaluate, so the most logical way to do this would be to take the absolute value of each term. The issue with this is there is no closed form solution to this, so another method has to be used. By squaring each of the terms, it effectively takes the absolute value because all of the negative residuals become positive, and unlike an absolute value, there is a closed form solution.

Using the inner product trick, the long summation that is equation 2 can be condensed and written as a single matrix inner product.

$$J = (\underbrace{\begin{bmatrix} x & -1 \end{bmatrix}}_{\hat{a_1}^T} \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}}_{x} - y_1)^2 + \ldots + (\underbrace{\begin{bmatrix} x & -1 \end{bmatrix}}_{\hat{a_n}^T} \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}}_{x} - y_n)^2 \tag{3}$$

Group unknowns m, b into x vector, and create new $\hat{a}$ variable.

$$J = \left[(\hat{a_1}^T x - y_1), \ldots, (\hat{a_n}^T x - y_n)\right] \begin{bmatrix} (\hat{a_1}^T x - y_1) \\ \vdots \\ (\hat{a_n}^T x - y_n) \end{bmatrix} \tag{4}$$

Convert the sum of square terms into an inner product of two vectors.

$$J = (Ax - Y)^T (Ax - Y) \tag{5}$$

Collect $\hat{a}$ and y terms into a single vector.

$$J = 1/2(Ax - Y)^T (Ax - Y) \tag{6}$$

Multiply by 1/2 because it makes taking the derivative in the next step cleaner. Equations don't change because it is a scalar multiple.

$$1/2(x^T A^T Ax - Y^T Ax - x^T AY + Y^T Y) \tag{7}$$

Multiply out terms.

$$J = \frac{x^T A^T Ax}{2} - Y^T Ax + \frac{Y^T Y}{2} \tag{8}$$

Group scalars together into single term.

$$\nabla_x J = 0 = A^T Ax - A^T Y \tag{9}$$

Take the derivative of J with respect to x and set to 0. Because this is convex optimization, this finds the minimum.

$$x = (A^T A)^{-1} A^T Y \tag{10}$$

Solve for x. Equation is in the form $x = A^+ Y$, so $A^+ = (A^T A)^{-1} A^T$.

In summary, the pseudo-inverse derivation above is used to solve least squared problems by minimizing the sum of the square of the residuals. This creates a line of best fit through the data points that is relatively accurate. Outliers have much more weighting on the line's fit because the residual terms are squared, thus large residuals have exponentially more weight than smaller residuals. This is a trade off we have to make because, as mentioned earlier, there is a closed form solution when using the sum of squares so it is much easier to solve.

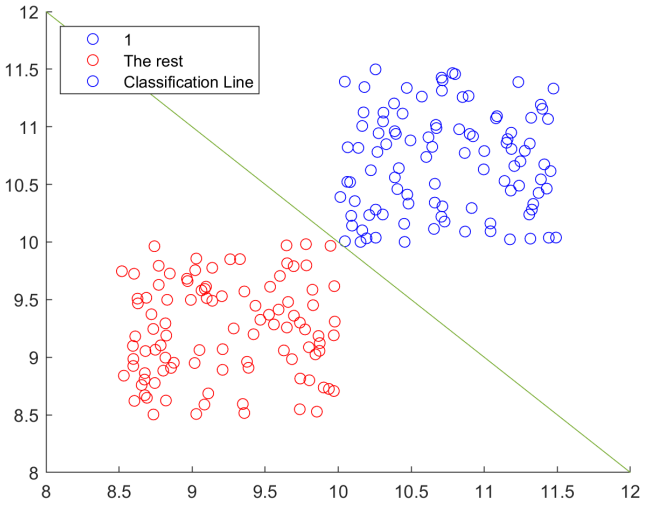


Figure 1: The clusters of images labeled positively vs the rest. Because the images have 676 pixels, instead of a plane, the features are represented in a $\mathbb{R}^{676}$ as a hyperplane.

The line of best fit in a binary classifier divides the two clusters of the data in half, where one cluster are the positively identified images, and the other cluster are the rest of the images. This line will be used to score the images by inputting the pixel values into equation 12 which outputs a score. If the score is positive, that indicates positive identification and classifies that image as the number it is checking for. The coordinates of the points are the 676 pixel values that make up the images, and since there are 676 pixel in each image, the points are in a 676 dimensional vector space.

2.2 Regularization

The image matrices mainly consist of pixel values of 0, making the matrices low rank. This proves to be an issue when solving the pseudo inverse because inverting a low rank matrix can cause errors and be impossible. A way to make the matrices full rank is to slightly manipulate the pixel values that are zero by adding $\hat{x}$ multiplied by some constant over top of the pixel values. Since $\hat{x}$ are unique and linearly independent, this makes the matrices full rank and invariable which allows for the solution to be found using the pseudo inverse. The form of regularization can be seen below in equation 11.

$$\left\| A^T\hat{x} - b \right\|^2 + \mu\|\hat{x}\|^2 \Longrightarrow \left\| \hat{A}^T\hat{x} - \hat{b} \right\|^2 \tag{11}$$

As you can see in the equation above, the total number of terms that make up total summation is now longer by the length of $\hat{x}$. This will add a total of 676 entries to the

bottom of the $\hat{A}$ matrix which we will build later. μ is the constant that is user defined and scales the effect of how much is added to the zero pixels. For my code I chose to use a μ of 1000 which is high because it yielded the highest accuracy when classifying the testing data set.

3 Formulation

3.1 Conversion from Classification to Least Squares

The binary classifier takes in the features of an image and solves the equation:

$$a^T x - b = \pm 1 \tag{12}$$

Where a and b define the separating hyperplanes, x is the vector of pixel values of an image and the solution is a number close to 1, or -1. The binary classifier takes a score that is positive as positive identification for the number it is checking for, and anything negative will be classified as not being the number the classifier is checking for. In order to train the model and generate the a and b, equation 12 has to be rearranged into the same form as equation 1. In equation 12, the two unknowns are a and b, the known is x and the solution b is a 1 or -1. Decomposing the left side of the equation into its inner product yields the following equation.

$$\underbrace{\begin{bmatrix} x^T & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{x} = \underset{b}{\pm 1} \tag{13}$$

The classification equation is now in the same form as equation 1 and can be solved using least squares. The solution to the least squares is a vector of length 677 where the first 676 entries are the values of β which represent the frequency of that pixel for a specific number, and the last entry is the intercept of the dividing hyperplane.

3.2 Matrix Formulation

The graduate section of this class is asked to solve the least squares using regularization due to the low rank of the image matrices. From earlier, the regularized least squares becomes:

$$\left\| A^T \hat{x} - b \right\|^2 + \mu \|\hat{x}\|^2 \Longrightarrow \left\| \hat{A}^T \hat{x} - \hat{b} \right\|^2 \tag{14}$$

The new $\hat{A}$ is composed of A from equation 13 which is the processed image data with a column of -1 appended to the end. These contain the equations for the regular least squares formulation of the problem. The additional regularization terms are appended below this and is made of an identity matrix multiplied by $\sqrt{\mu}$. μ is square rooted because it moves into the sum of squares term. The composition of $\hat{A}$ is below along with its dimensions.

$$\hat{A} = \underset{60676\times677}{\begin{bmatrix} \vec{x} & -\vec{1} \\ \sqrt{\mu}I & \vec{0} \end{bmatrix}} \tag{15}$$

with individual dimensions $\underset{60{,}000\times676}{\vec{x}}$, $\underset{60{,}000\times1}{-\vec{1}}$, $\underset{676\times676}{\sqrt{\mu}I}$, and $\underset{676\times1}{\vec{0}}$

There will be 10 different $\hat{b}$ vectors which represent the binary classification of the images for each classifier. For example, the $\hat{b}$ for the zeros classifier will have a 1 in the entries that align with the entries that are zero in the label data, and a -1 everywhere else. Lastly, since were are regularizing the least squares, the last 676 entries will be 0 as they don' contain any information. A shortened example of the $\hat{b}$ for zero is in the figure below.

$$\hat{b_0} = \begin{bmatrix} 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}^T$$

Figure 2: Shortened $\hat{b}$ for the zero classifier. The labels are organized 0-9

When multiplied out, the system of equations for the training of a classifier will be as follows in equation 16 where A_0 is the trained beta vector and x_0 is an image labeled as zero. The scores are set at 1 and -1 as mentioned earlier.

$$\begin{aligned} A_0x_0 &= 1 \\ A_1x_0 &= -1 \\ A_2x_0 &= -1 \\ &\vdots \\ A_9x_0 &= -1 \end{aligned} \tag{16}$$

As seen in equation 13, the solution x is comprised of a and b which describe the hyperplanes. a is a 676x1 vector contains the slope information for the 676 dimensional hyperplane and b is the y-intercept. Using a cell array, the a vector and b values can be saved in the index that corresponds to the number of each classifier.

4 Methodology

4.1 Data Preprocessing

The very first step is to preprocess the raw data into something usable that the classifiers can read in and then output a guess as to which number the images represents. Since all of the hand-written digits are centered on the image, the border of each images can be removed as no information will be lost, and the size of the images will be greatly reduced. By removing the border pixels from each of the images, the vectors that represent the images are reduced from 784 pixels (28x28) down to 676 pixels (26x26). Figure 3 shows the pixels that were removed in black and the remaining pixels in white.

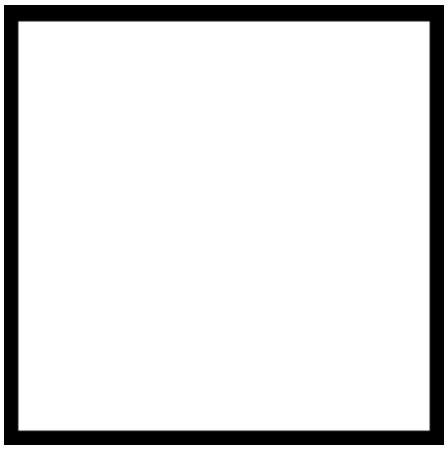
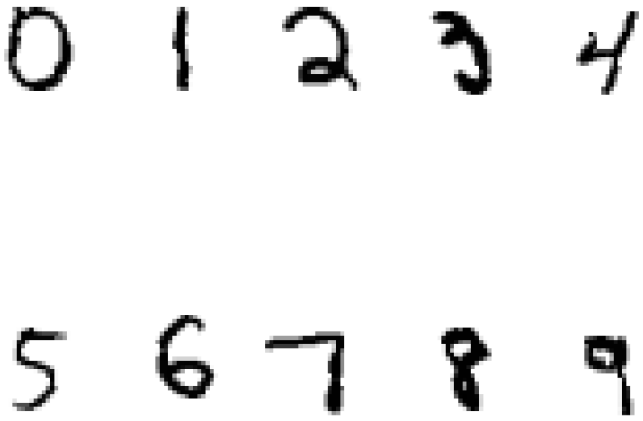


Figure 3: Border pixels that were removed. 28x28 reduced to 26x26

The pixels data in the mnist data are in integer form because values can only be integers between 0 and 255 as the data is 8-bit, so in order to normalize the pixel values between 0 and 1, the data types need to be changed from integer to double (double-precision floating-point) which allows the values to be converted to decimals with relatively high precision. After converting the data type, each pixel value can be divided by 255 and subtracted from 1 because the raw pixel values for images are inverted initially, as seen in Figure 4a. Figure 4b shows the processed images that have been inverted.

(a) Raw image data from the "mnist" data set. The grey scale colors are inverted requiring a correction of 255-(pixel value).



(b) Processed images with the border removed, pixel values normalized to 0-1, and the gray scale inverted.

Figure 4: Side-by-side comparison of raw data and processed data.

Next, the image data and the labels are ordered from 0 to 9 to group similar images

together. In Matlab, this can be done very simply by appending the label data to the first column of the image data and using the *sortrows* function. This function sorts the rows of matrices based on the elements in the first column, which are now the labels. Removing the labels leaves a sorted matrix of the image data with the images sorted from 0 to 9. I then shifted all of the data over by one place so the order was 1-9 with 0 at the end. This was done because Matlab is 1 indexed, so now each of the digits aligns with the index it is in.

4.2 Matrix Generation

As discussed earlier, the $\hat{A}$ matrix that solves the regularized least squares consists of all the image data with a -1 column appended to the end and an identity matrix multiplied by $\sqrt{\mu}$ appended to the bottom with a zero column to meet dimensionality. Next, the $\hat{b}$ vectors are created using the labels. a *for* loop iterates from 0-9 and checks if the labels match the iteration that it is on. This creates ten $\hat{b}$ vectors that contain all -1's except where the labels match the iteration, which have 1's. Lastly, zeros are appended onto the end of the vector to match the dimensionality of $\hat{A}$ which has a height of 60676.

Now that the $\hat{A}$ and $\hat{b}$ have been made, $\hat{x}$ can be found simply by using the pseudo inverse function in Matlab. This will return a vector which contains the trained a vector and b which describe the separating hyperplanes.

$$\hat{x} = pinv(\hat{A}) * \hat{b} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{17}$$

a contains the weighted intensity of the pixel values from the sorted training data, and b

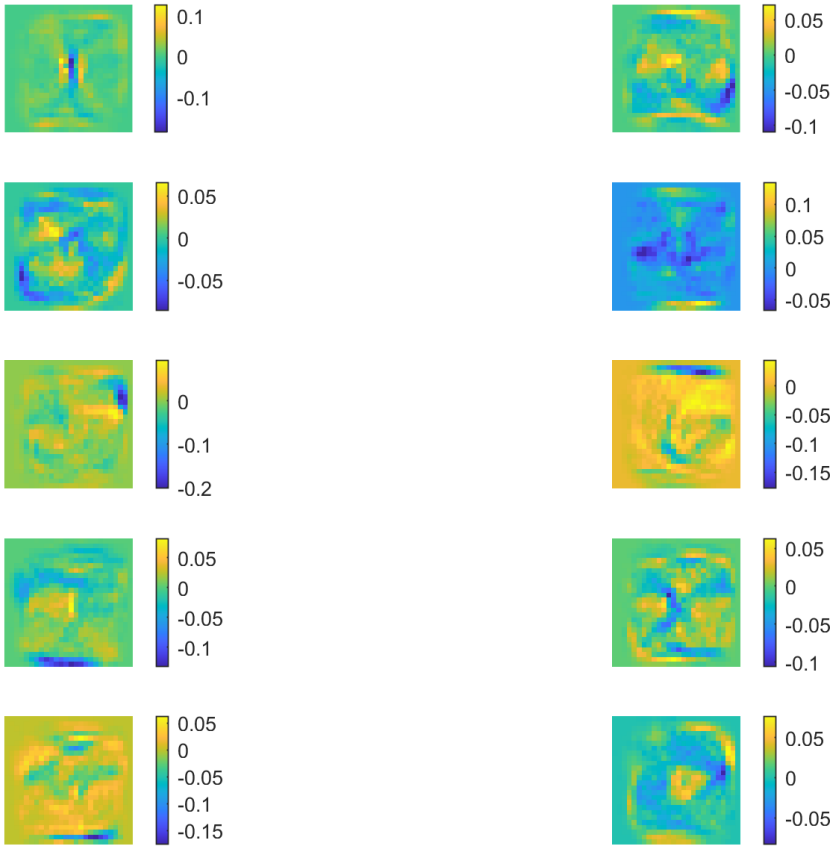


Figure 5: Plot of β for each classifier from 1-9 with zero at the end. Scaled based on *min* and *max* values of each β.

5 Testing and Results

The trained a vector and b can now be tested against the labeled testing data to measure the accuracy of the classifiers. This is a different data set from what the models were trained on because the model has never seen the test data so it will give a better representation of performance of the classifiers for any hand written number. The performance of the classifier will be the percentage of correct classifications of the images.

5.1 Individual Classifier Performance

The performance of each individual classifier is found by separating the images data into "1 vs. the rest" and analyzing the scores of the separated data. This means each classifier has an accompanying matrix with all the positively labeled images and a matrix will all of the other images. By doing so, this will allow us to see how much of the data is being classified incorrectly as seen in table 1.

Because the testing data is labeled, for a given image there are 4 outcomes. A classifier can:

Correctly identify as being a certain number
Incorrectly Identify as being a certain number
Correctly Identify as not being a certain number
incorrectly identify as not being a certain number

This can also be displayed in a confusion matrix which has the guessed outcome on the vertical axis, and the expected outcome on the horizontal axis as seen in table 1. The accuracy of the classifier is the trace of the confusion matrix which represents the total number of correct classifications, divided by the total number of entries classified.

	Expected	
Guess	1	-1
1	True Pos	False Pos
-1	False Neg	True Neg

Table 1: Possible classification outcomes.

Once the separated matrices are scored by the trained model, the two sets of scores can be plotted together to show the classification of the two groups. Below are the classifications for both data sets for each of the 10 classifiers. As seen in the histograms below, the classification line is at $x = 0$ for all of the classifiers. If we wanted to improve the accuracy of the classifiers we could optimize the position of the line to fall at the bottom of the valley between the two

modes. This would reduce the number of false negatives at the cost of slightly increased false positives. But that is beyond the scope of this project.

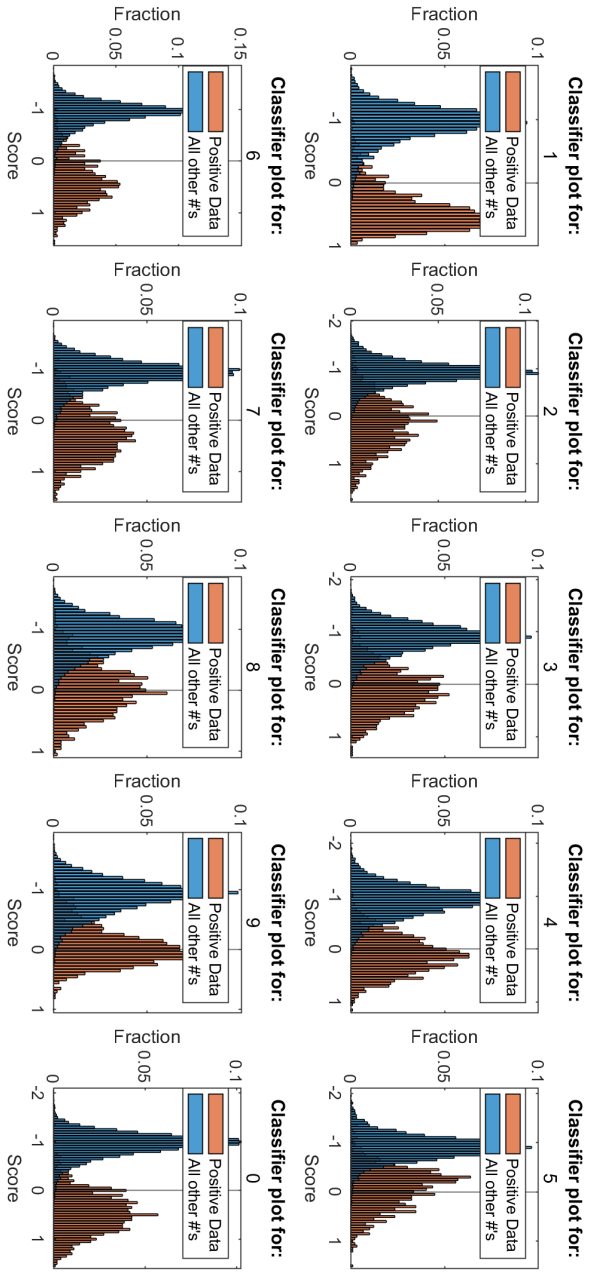


Figure 6: Classification of "1 vs the rest" for each classifier normalized to a percentage. The classifiers are very accurate when determining when a number is not what it is checking for. Some numbers like 5 and 8 which share many of the same features are harder to classify correctly.

Each histogram can also be shown in a confusion matrix which shows the exact number of classifications in each of the 4 categories listed earlier. The confusion matrix for the #5 classifier is shown in table 2. The histogram of the #5 classifier shows that there is a high rate of false negatives, and the confusion matrix confirms this because it shows that out of the 892 images of a 5, there were 532 false negatives and only 360 true positives. Even with the large amount of false negatives, the total accuracy of the classifier is 94.35% only because of the high accuracy of true negatives which make up 90% of the data.

	Expected		
Guess	1	-1	Total
1	360	532	892
-1	33	9075	9108
All	393	9607	10,000

Table 2: Performance of the # 5 classifier.

5.2 Image Classification

As mentioned in the introduction, the classification structure that I will be using checks an images with all of the classifiers and categorizes it based on the classifier with the highest score output. This should increase accuracy by reducing the number of false positives as positive scores close to zero will likely be negated. The Matlab code does this by iteratively scoring each of the images in the data set against the classifiers. The scores are recorded in a 10,000x10 matrix where the horizontal index is the number of the classifier. The *max()* function then determines the maximum values of each row and creates a final guess vector of length 10,000x1.

5.3 Image Classification Performance

The performance of the classification system can be extracted by comparing the guess vector to the label vector as the percentage of entries that are shared between them. In code, I did this by iterating through all 10,000 values and counting when the i^{th} entry of the guess was the same as the i^{th} entry of the label. Once at the end, the counter was divided by the total number of entries and multiplied by 100 for a final percentage. For the testing data set, the classifier had an accuracy of 86.63% and for the training data set, the classifier had an accuracy of 85.83%. We should expect that the training data would have the better accuracy because that is what the model is built off of, but this difference could be due to the qualify of numbers in the data sets. If there were more ambiguous digits in the training set, that would lower the total accuracy.

The confusion matrix for the classifier as a whole can be build using the guess vector and the label vector. In Matlab, I created a 10x10 zeros matrix that had a counter add 1 to the mxn entry with m index being the guess, and n index being the true label which can be seen

in figure 7. As I ran through the guess and label vector, the zeros matrix was populated with the number of instances a certain guess was made. The number of true positives were recorded on the diagonal as the guess and label were equal. The total number of correct classifications can quickly be read off as the trace of the confusion matrix. The sums of the individual columns and rows were also added together and written to a *Total* and *All* where the entries in the *All* vector are the number of expected occurrences for each number and the entries of the *Total* are the number of guessed instances for each number.

for i=1:10,000

Confusion Matrix($guess(i)$,$label(i)$)= Confusion Matrix($guess(i)$,$label(i)$)+1;

end

Figure 7: Pseudo code for generating the confusion matrix.

Below are the complete confusion matrices for the testing data set 3a and the training data set 3b. Again, the testing data classification had an accuracy of 86.63% and the training data had an accuracy of 85.83%.

	Expected										
Guess	1	2	3	4	5	6	7	8	9	0	Total
1	**1105**	64	16	21	16	11	45	53	11	0	1342
2	2	**830**	25	6	4	4	17	9	3	1	901
3	2	19	**886**	0	81	0	6	32	13	2	1041
4	2	15	5	**886**	28	17	17	24	66	1	1061
5	2	0	14	2	**641**	16	0	27	2	9	713
6	5	23	9	10	23	**890**	1	19	1	16	997
7	1	24	22	2	22	0	**896**	13	64	2	1046
8	16	35	19	10	38	5	0	**764**	6	7	900
9	0	5	10	44	18	0	41	20	**824**	1	963
0	0	17	4	1	21	15	5	13	19	**941**	1036
All	1135	1032	1010	982	892	958	1028	974	1009	980	**10,000**

(a) Testing Data

	Expected										
Guess	1	2	3	4	5	6	7	8	9	0	Total
1	**6558**	299	175	109	108	84	210	508	76	7	8134
2	37	**4759**	180	44	32	43	42	68	29	24	5258
3	11	137	**5128**	4	488	4	41	238	115	18	6184
4	7	118	27	**5219**	124	52	140	99	331	25	6142
5	35	15	118	34	**3874**	94	6	175	13	51	4415
6	8	202	73	57	199	**5510**	5	67	5	79	6205
7	15	122	136	28	65	0	**5497**	31	450	7	6351
8	64	188	125	64	223	34	14	**4437**	44	64	5257
9	5	25	127	274	140	2	251	159	**4810**	5	5798
0	2	93	42	9	168	95	59	69	76	**5643**	6256
All	6742	5958	6131	5842	5421	5918	6265	5851	5949	5923	**60,000**

(b) Training Data

Table 3: Confusion matrices for test and training data. Correct identification is shown on the diagonal in bold.

6 Conclusion

As shown in this paper, using least squares to create a binary classifier can be an effective and easy way to identify large quantities of hand-written numbers. The model only needs to be trained once on a large data set, then it is able to identify other hand-written numbers with an accuracy of 87%. If one desired, it would be simple to modify the code to train on hand written alphabetical characters and identify text.

The accuracy of the classifiers could be further optimized by modifying the position of the classification line in the histograms.