Project 2 - Least Squares Image Classification

Kenzie MacNicol & David Young

Fall Semester 2023

Introduction

The least square approximation is used when a system of equations are over defined; i.e, there are more equations than there are unknowns. Due to this surplus of information, our course of action will then be to find a set of values that will hopefully minimize the error (or distance) between our result and the given data points. Mathematically, we want to minimize the equation:

$$min\|A\theta - y\|^2 \tag{1}$$

In a geometric sense, if we are given a 2-D plane and a point in a 3-D space, then we want to find the shortest distance between the plane and the point in 3-D space [1]. In other words, we want to find the line of best fit between the plane and the point in 3-D space. In this project, we will be using the least squares method to classify images by solving equation (1) for θ. The basic idea of how the classifier works is that once we obtain the trained coefficients θ, the larger positive coefficients suggest that when this particular pixel has a non-zero intensity, then the image is more likely to be classified as a 1; in other words, the image probably shows the number zero. To expound, the classifier classifies an image to hold a zero or a non-zero number depending on the sum of the training coefficients multiplied with the intensity of every feature in image i. If the overall sum is positive, then the image is most likely to hold the number zero, while if the overall sum is a negative number, then the image most likely holds a non-zero number. In a mathematical sense [1],

$$f(\widetilde{x}) = sign(\theta_1 \cdot f_1(x) + \cdots + \theta_M \cdot f_M(x)) \tag{2}$$

where sign() turns the input into a -1 if the input is negative number, or returns a 1 if the input is a positive number.

Also, visualizing the geometric metaphor for our project problem requires imagining 5000-dimensional space which many may find challenging to do.

Procedures and Results

#1)

Part 1 of the project was trivial. We used the function provided to us to import the files we needed into the Matlab workspace. The only challenge was adding the necessary files to the compiler path accessible by the live script.

#2)

We found the non-zero intensity pixels iteratively by first looking at the pixel [1, 1] on image 1. If this pixel had a non-zero intensity, then we incremented the counter that keeps track of the number of non-zero intensity pixels at location [1, 1]. This process of checking the pixel at location [1, 1] for a non-zero intensity was repeated for all 5000 images, and each time an image had a non-zero intensity pixel at location [1, 1]; the counter that keeps track of the number of non-zero intensity pixels at that location was incremented.

After looking at pixel location [1, 1] of all 5000 images, we checked if at least 600 images had a non-zero intensity pixel at location [1, 1]. In other words, if the counter was equal to or greater than 600, we put a 1

in the features matrix at index $i = 1$ and $j = 1$, which corresponds to the location of the pixel on the images. We repeated this process for all pixel locations. In the end, we had a features matrix that contained a 1 in the index location that corresponded to the location of pixels on the images that had a non-zero intensity in at least 600 images.

We then created a vector wherein each row was a feature represented as an index pair [row column]. This vector and all other objects referenced in this document are present in the Matlab code submitted with this pdf.

#3)

The A matrix was constructed with rows corresponding to the i-th image, and the columns corresponding to the j-th feature, where the entries of A_ij was the intensity of the pixel of image i for the j-th feature.

The y vector was created by converting the contents of the vector of labels, imported from the given training data, to 1s and -1s, where 1s represented 0s in the labels vector, and -1s representing non-zero values in the label vector.

With the A matrix and y vector computed, we calculated the vector θ, which was the vector of trained coefficients, by using the backslash command in Matlab:

$$\theta = A \backslash Y. \tag{3}$$

In order to plot the coefficients, the vector of trained coefficients was first converted to a matrix by placing the value from the vector of trained coefficients at the same index where a feature is significant (ie. equal to 1) according to the features matrix. The for loop that was used to convert the trained coefficients into a matrix iterated across all the features for image 1 first, then through all of the features of image 2, and so on. Every time, there was a 1 in the features matrix, the next value in the trained coefficients vector was placed in that matrix location. To clarify this process, part of the Matlab code is depicted below:

```
coefficentsAsMatrix = zeros(20,20);
featureIndex = 1; % set feature index to 1
for i = 1:20 % for every row...
    for j = 1:20 % for every column
        if featureMatrix(i,j) == 1 % if the row/column index is a feature...
            coefficentsAsMatrix(i,j) = trainedCoefficents(featureIndex); % set to the coefficient for that pixel.
            featureIndex = featureIndex + 1;
        end
    end
end
```

The resulting graph depicts a 2-D plane of 20x20 pixels, representing an image, with bars rising above and below the plane. These bars represent the values of the trained coefficients. This graph is seen in figure 1.

#4)

To test our trained coefficients, we imported 5000 images from the Test image data file along with their labels. Our goal was to create a new matrix called testMatrixA from the imported images, multiply testMatrixA by our trained coefficients, obtain a y vector which we will convert to 1s and -1s as done in part 3), and compare this y vector to the true test image labels.

Matrix testMatrixA was obtained in the same way that matrixA was obtained in part 2), resulting with testMatrixA containing the intensities A_ij, where i is the i-th image, and j is the j-th feature. The new y vector was obtained by first multiplying testMatrixA by the trained coefficients matrix, and then running the sign function on vector's entries. This new y vector was then compared entry-wise with the label vector from the test image data. The result of the comparison was stored in a Boolean vector - the result vector - where 1 meant that our trained coefficients model was able to correctly identify the test image, and where 0 meant that our model was not able to correctly classify the test image. We computed the error rate of our

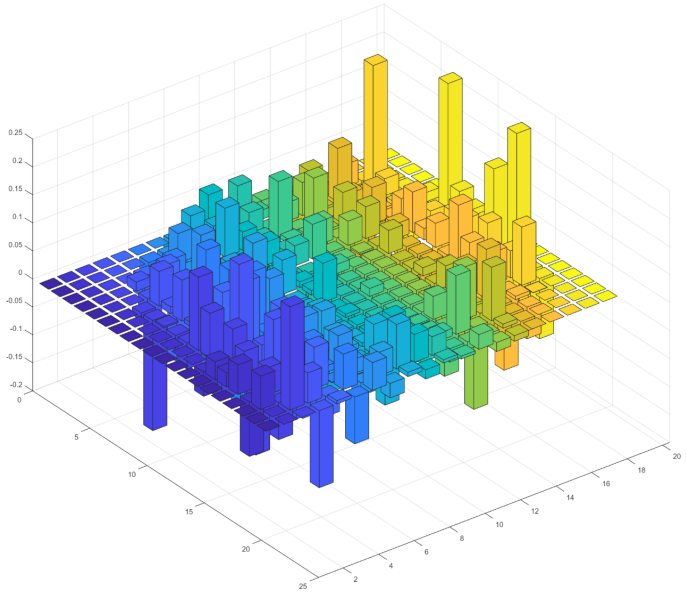

Figure 1: This is a 3-D bar graph for part #3) with the magnitudes of the trained coefficients.

model, where we divided the number of zeros, or the number of incorrect classifications by the number of images we imported; which turned out to be 0.0194.

Two more types of errors that we calculated were the false positives and false negatives. A false positive is where our model classifies the image to be a 1 whereas the actual label was a -1. In the case of a false negative, our model would classify an image to be -1 whereas the label would be 1. The false positive error was calculated by dividing the number of times a false positive occurred by the number of -1s in the label vector. Our calculated false positive error was 0.0115. Similarly, the false negative error was calculated by dividing the number of times a false negative occurred by the number of 1s in the label vector. Our calculated false negative error was 0.0939.

#5)

In this section, we will denote the new A matrix created for this section as matrix A5. As before the A5 matrix was created with rows equal to the number of images imported, and columns equal to the number of features found in step 2. In this case, A5 will be a matrix of 100 rows by 319 columns. As before, the entries of Matrix A5 contained the intensities of the i-th image and of the j-th feature. And as before, the y labels obtained from the imported data, was converted to 1s and -1s (refer to section to section 4 and 3). The new trained coefficient vector was calculated using equation 3. The new trained coefficient vector was then converted into a matrix as done in section #3). The resulting graph of the 3-D bar graph can be seen in figure 2.

When we compare graphs 1 and 2, we see that there are more heavily weighted positive and negative coefficients in the fully trained model, which suggests that the fully trained model is more certain in their classification. To clarify, the larger a positive the trained coefficient at a particular pixel is, the more likely the model will classify an image to be a positive number if an image has a non-zero intensity at that pixel with a large positive coefficient because the overall sum of the pixel intensities of an image multiplied by the coefficients will tend toward a positive number as explained in the introduction. On the other hand, when we have a large negative trained coefficient at a certain pixel, it is more likely that the model will classify an image to be a non-zero number when there is a non-zero intensity at that pixel.

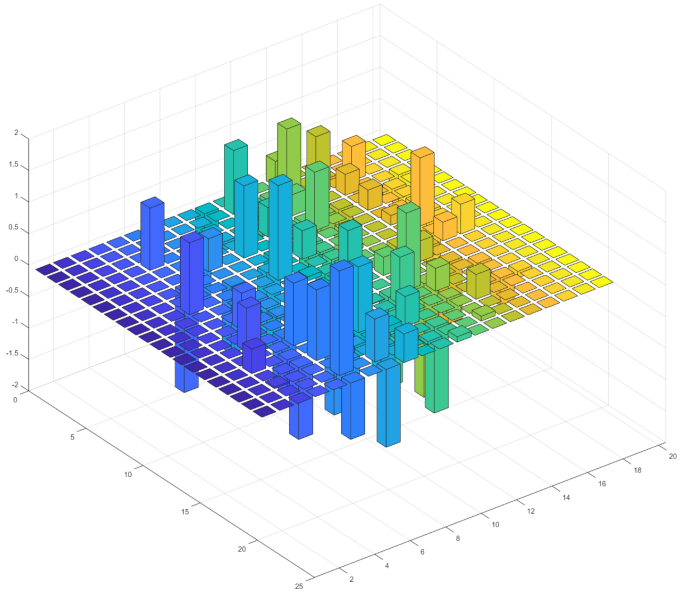

Figure 2: This is the 3-D bar graph for part #5) with the magnitudes of the trained coefficients.

The error rate, false positive rate, and the false negative rate in this section was calculated in the same way as the error rate was calculated in section #4). The error rate in section #5) was 0.2272, the false positive rate was 0.2064, and teh false negative rate was 0.4326. When we compare all of the error rates from section #4), the fully trained model, to section #5), the partially trained model, we see that all of the error rates in the fully trained model was less than the error rates in the partially trained model which makes sense as the partially trained model only used 100 images to calibrate, whereas the fully trained model used 5000 images to calibrate its model.

#6)

For this part of the project, we were to experiment with changing the feature set. This means, instead of using the 319 (pixel) features we previously found to be significant, we will create a randomly generated matrix representing M (randomly generated) features. First, we generated a matrix R with M rows and a number of column equal to the number previously determined features (319 columns), M0. The entries in this matrix were randomly -1 or 1. A new feature vector for a given image, i, was calculated as this random sign matrix R times the old vector of features from part #2 for the given image, i. Then, if this value was negative, it was instead set to zero; otherwise, it is left as-is. We chose to combine all these new feature vectors together into a new feature matrix wherein every row represented an image and each column represented a feature.

With this new feature matrix, we followed a procedure very similar to part #3 to solve the least-squares problem that resulted in the vector of trained coefficients for the newly generated features.

#7)

While challenging to code, parts #6 and #7 are conceptually the same as parts #3 and #4. We create a matrix A that captures a set of features for each image then we solve the least-squares problem for a set of trained coefficients for those features. We can then capture a new matrix of features from new images, then multiply them by these trained coefficients. Taking the sign of the product will give the algorithm's determination if the image should be classified as -1 or 1 representing either a nonzero digit or a zero digit.

We performed this process in step #7 for a variety of values of M. We expected this process to generally be worse than the training method in part #3, but for the error rate to come down with many many random features as in M=10000. We thought that the many random equations could be expected to have some inexplicably perfect and useful equations mixed in that work well for the classifier. Our results are summarized below:

1. M = 20:
   (a) error rate: 0.0698
   (b) false positive rate: 0.0095
   (c) false negative rate: 0.6652
2. M = 50:
   (a) error rate: 0.0432
   (b) false positive rate: 0.0064
   (c) false negative rate: 0.4065
3. M = 1000:
   (a) error rate: 0.0084
   (b) false positive rate: 0.0029
   (c) false negative rate: 0.0630
4. M = 5000:
   (a) error rate: 0.4446
   (b) false positive rate: 0.4454
   (c) false negative rate: 0.4370
5. M = 10000:
   (a) error rate: 0.3490
   (b) false positive rate: 0.3432
   (c) false negative rate: 0.4065

The data did not align with our hypothesis. The error rate was lowest in the range of M = (100,3000) with fairly little deviation around a 5% error rate. The error rate grew apparently exponentially after M=4400. To detect these overall trends, we ran the process for values of M from 1 to 5001 in steps of 50. The results are plotted below in figure 3. Overall, the error rate was fairly low - lower often than the more intuitive features from part #3. The false positive error rate tended to be lower while the false negative error rate tended to be higher using this method of randomized features. This would make it less suitable for applications like disease diagnosis if this trend is true.

1 References

[1] Clark, Andrew, Module Three Lecture Notes. Washington University in St. Louis, Missouri, 28 September 2023. Accessed 10 November 2023.

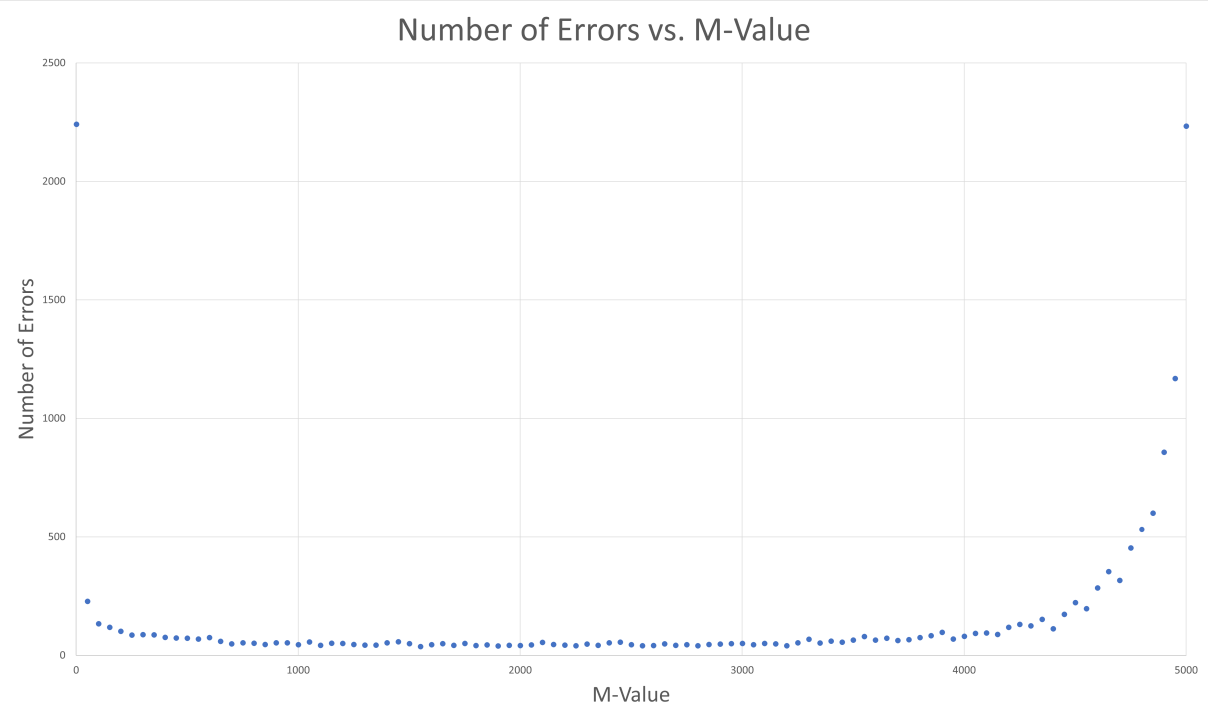

Figure 3: This plot shows the number of classification errors versus various values of M.