Case Study 3: Classifying handwritten digits

1st Alexander Ge
Computer Engineering
Washington University in St.Louis
St. Louis, Missouri
age@wustl.edu

***Abstract*—This study investigates the classification of MNIST handwritten digits using a comprehensive dataset of 60,000 images, divided into testing and training groups. We employed a series of methods to classify digits from 0 to 9 based on pixel intensities. Our approach begins with visualizing the digits to understand variations and patterns in their structures. This is followed by implementing a simple pseudo-inverse solution, derived from a small subset of the training data, to establish a preliminary classification model. To improve classification accuracy, we introduced the concept of a boundary vector, which leverages pixel prevalence to define characteristic bounds for each digit class. Further refinement involved employing a more general pseudo-inverse approach on a linear equations to ascertain the weight of each pixel in contributing to digit classification. This method provided a nuanced understanding of pixel importance and strengthened the classifier's accuracy. Ultimately, we synthesized these insights to develop a robust multi-class function capable of classifying images as digits from 0 to 9. The efficacy of this classification process is detailed through confusion matrices, illustrating the improved accuracy of the multi-stage approach. This study showcases the integration of visualization, mathematical optimization, and feature extraction in enhancing digit recognition capabilities.**

I. Introduction

The importance of this project lies in its practical application of key concepts from ESE 105, particularly least square classifiers and pseudo-inverses. By applying these mathematical tools to the task of digit classification, the study provides a foundational understanding of machine learning and artificial intelligence methodologies. Through this project, we are shown the basics, and potentially how classifers are developed into more complex algorithms and systems.

Digit classification using the MNIST dataset is a classic problem that exemplifies the challenges and strategies in pattern recognition and classification, two critical aspects of machine learning. Through methodical exploration and visualization of the dataset, followed by the implementation of least squares methods, the project demonstrates ESE principles to solve real-world problems. This project aims to develop a model that can accurately predict the digits of unknown test images while minimizing false positives and negatives, thereby establishing a baseline for more sophisticated machine learning models. The goal of the study is to create a reliable, effective classifier for the digit recognization, providing a solid groundwork for future exploration in machine learning and AI technologies.

II. Methods

A. Pseudo-Inverse

The pseudo-inverse or Moore-Penrose inverse is,

$$A^{\dagger} = (A^T A)^{-1} A^T. \tag{1}$$

where A is a matrix that is a part of a linear equation $Ax = b$ and A's columns or rows are linearly independent, $x = A^{\dagger} b$ is a solution to the linear equation, MATLAB's pinv() function executes this equation [1].

B. Confusion Matrix

A confusion matrix, and MATLAB's confusionchart() method are used to visualize classifier performance by displaying the results of the predicted digits against the actual digit labels, where the error is calculated by:

$$\frac{FalsePositives + FalseNegatives}{TotalGuesses}. \tag{2}$$

Where false positives are where the function predicted the image was in the set when it was not, false negatives are when the function predicted the image was not in the set when it was [2].

C. Visualization

MATLAB's build in imagesc() and colormap() functions can be used to visualize images, and data matrices, and distinguish between intensity levels [2].

III. Initial Visualization and Classification

In our study, we began by splitting the MNIST data set into 24000 images and labels for training, and 8000 images and labels for testing. The following introduction only uses the training set of images. Using MATLAB, we visualized the first 9 examples of each digit using imagesc() and colormap() in figures with 3 by 3 grids, demonstrating how each digit had its own unique features visually. Fig. 2 shows one of the plots for the digit 9. Then, we flattened each image of 28 by 28 pixels into a single 784 column vector for easier and more efficient manipulation with linear algebra methods.

Incorporating our understanding of pseudo-inverses, we employed a least squares approach to develop an initial classification model. We selected a subset of 500 images A, aiming to balance computational efficiency and model simplicity at this preliminary stage. Using the pseudo-inverse function from (2), on A where our b was the labels of the images in A.

We got our solution $A'b$, when we multiplied this to the other training images and rounded to the nearest number, we defined our first primitive function to define digits. This pseudo-inverse served as an initial solution to the optimization problem, converting input data into predicted digit labels. To evaluate the performance of our classifier, we utilized confusion matrices, generated via confusionchart() shown in Fig. 2. Additionally, a manual implementation was created to ensure that the confusionchart() method was performing in the way we expected as shown in Fig. 3. The matrices helped to highlight correct predictions alongside false positives and negatives, thus showing feature misclassifications and areas requiring improvement, and the confusionchart() method was used to plot confusion matrixes for the rest of the study.

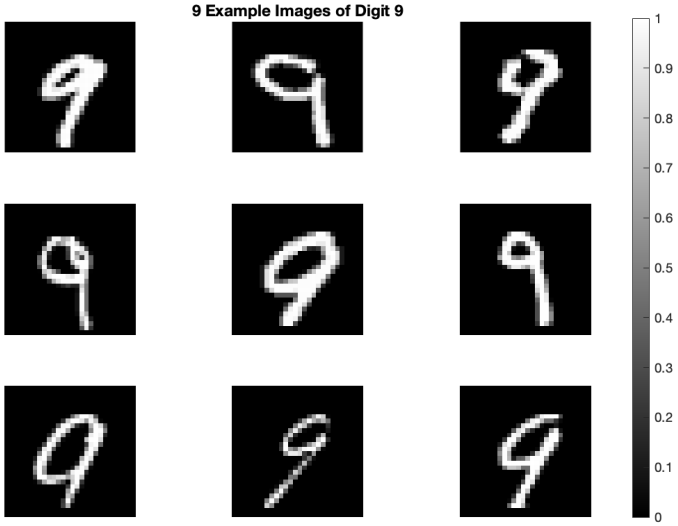

Fig. 1. A MATLAB figure of a grid of 9 images of the digit 9 from the MNIST data set.

Fig. 2. A MATLAB figure of a the confusionchart() function.

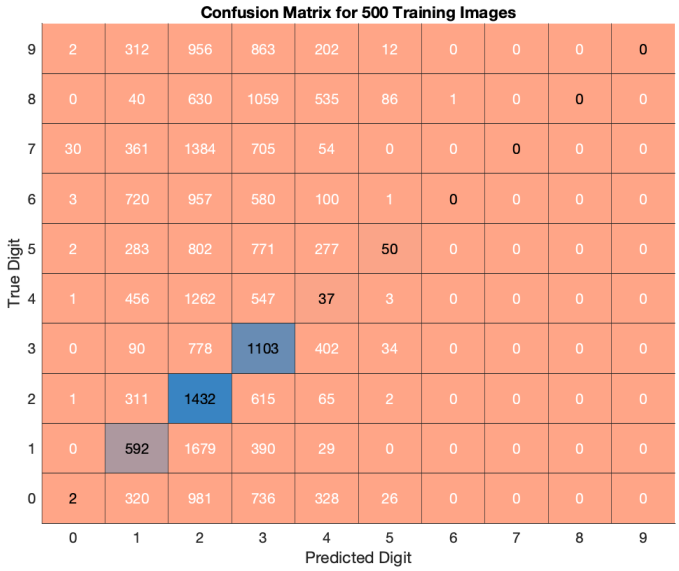

Fig. 3. A MATLAB figure of a manual confusion matrix plot using imagesc() and custom colormaps.

IV. Binary Classification

In this section we create multiple single-digit classifiers on each digit in our training set. Again, we flattened the images, then we performed these functions separately on each digit:

We would extract each image with the label of the current digit into a new vector, and also create a new binary label for the flattened test images the same size as the test labels, but the indices with the current digit would be 1 and -1 otherwise. Then we used these new vectors for the current digit to implement 3 new functions to try and predict if an image was a part of the current image set or not.

A. Simple Binary Classifier

For a simple binary classifier we determined a boolean *bounds* vector the size of an image (784x1). We loop through all of the current digits and set each index of the bounds vector to true if there exists a pixel value greater than 0 at that pixel in the set of all current digit images. We used this bounds vector to create a simple binary classifier that was 1 where bounds was true and -1 where bounds was false. To use this vector in predictions we would multiply a given image by the vector and predict that the image was either an element of the current digit set or not based on if the prediction was greater than or equal to 0 or less than 0. We calculated and displayed the error and confusion matrix for the prediction of the simple binary classifier on the test image set, and the set of current digits. We found that the classifier had a very low accuracy of 90% error (2), and needed to be refined more for the test image set, and was able to classify its own images as part of its set. Examples of this classifier are shown in Fig. 4.

B. Refined Binary Classifier

For a less primitive classifier we looked at the frequencies of each pixel across the current digit, and created a vector

that incremented each time a pixel had a value in it as it looped through all the current digit images. We then set a threshold value that we tweaked with a little to make sure that the classification of the current digit was near perfect, and such that the test image error was as low as possible. The threshold value was used to determine which values of the new refined binary classifier were -1, 0, and 1, values with no frequency stayed at 0, values under the threshold were set to -1 values above or equal to the threshold were set to 1. The predictions were made in the same way as the simple binary classifier. The average error rate for the test image set was 88.958% which was better, but this classification seemed like it was not the best choice, an example is shown in Fig. 5.

C. Final Binary Classifier

In search of a better binary classifier, the study returned to a pseudo-inverse solution. We revisioned the creation of the binary classifier as a least squares problem. Where we were trying to optimize the weights of w to better reflect the labels. We created this linear system weights * flattened train images = train labels in binary (1, -1 for current digit or not). We found the pseudo-inverse of the flattened train images. Using that we found the weight by doing train labels in binary * pseudo-inverse. Using this new weight function we could make our predictions with any number of flattened images by taking the sign (positive or negative) after doing weights * flat test image(s) where each resulting column would give a positive or negative value based off of their pixels and the weights assigned to those pixels from the pseudo-inverse function. The result was an average error rate of 3.83% as shown in Fig. 6, which is much better than the other classifiers. Such that we decided to use this classifier to help build a general digit classifier not just for an individual digit.

V. General Classifier

Using the same flattened train images, for our general classifier we created a weights vector with 10 rows for each digit, where we would loop through of final binary classifier code to create the weight for the digit and store it in current digit + 1 row in weights. Then, we defined a function digitClassifier(z,weights) in MATLAB as the final general digit classifier. Where z is a 28 by 28 pixel image and weights is the weights vector we calculated before. The digitClassifier function would flatten the image, then for each digit it would store the guess of the weights * the flattened image into a vector. Where each index represented the guess for the index - 1 digit, because vector indexing starts at 1 not 0. Then the row - 1 of the highest guess value would be the predicted digit for the input image and would be returned in a value k. We ran this function on each of the unflattened test images and the result is shown in Fig. 7, where we got an average error rate of 15.3%.

VI. Results and Discussion

Our work on the classification of MNIST digits using a series of ESE techniques and MATLAB tools has demonstrated the strengths and limitations on simply classification

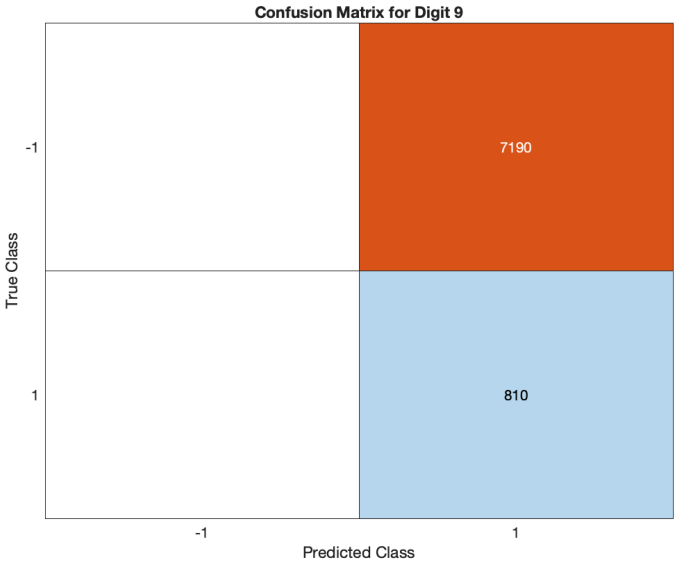

Fig. 4. A MATLAB figure of a confusion matrix for the digit 9 using the simple binary classifier on test images.

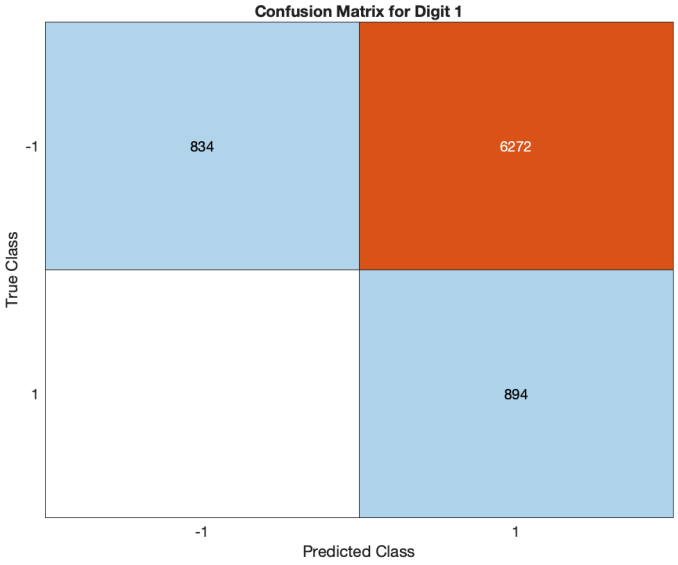

Fig. 5. A MATLAB figure of a confusion matrix for the digit 1 using the refined binary classifier on test images.

strategies, and was able to build a classifier for a general image of a digit in the MNIST set. The initial pseudo-inverse solution resulted in an accuracy that was rather low and primitive, emphasizing the need for enhanced feature extraction to improve classifier effectiveness. Confusion matrices from initial attempts highlighted frequent misclassifications, which justified the need for further refinement. Using 3 different was of binary classification, we were able to improve our accuracy in predicting the digits. Our simple binary classifier resulted in a 90% average error rate, indicating that more work needed to be done. We tried to refine the binary classifier using frequency analysis and a threshold value which very slightly improved the accuracy to an average error rate of

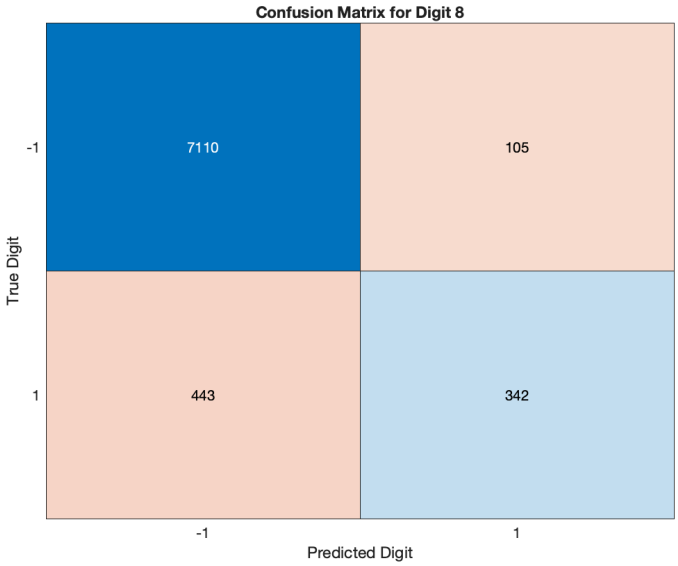

Fig. 6. A MATLAB figure of a confusion matrix for the digit 9 using the final binary classifier on test images.

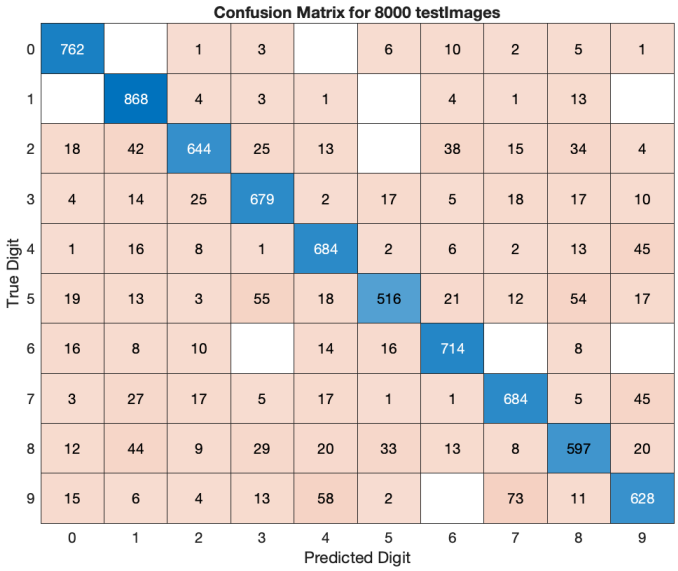

Fig. 7. A MATLAB figure of a confusion matrix for digitClassifier function on test images.

88.958%. Our final binary classifier which used a pseudo-inverse on the binary labels to determine the weight of each pixel in determine the final digit drastically improved results, reducing the error rate to an average of 3.83% across all digits. Using the final binary classifier we created a function that acts like a general classifier, and we were able to get an average error rate of 15.3% across the testing set which demonstrates strong potential in classifying a random given image. Our study was limited to a dataset, which might not represent all different handwritings well, and our models are very basic in computation, only using linear transformations on the input data set, where more complicated functions could potentially yield better classification. This study also did not do any feature recognization on the training images, other than flattening the image for easier manipulation.

VII. Conclusion

This study finds that ESE tools such as pseudo-inverses can provide a solid basis for developing classifiers that identify patterns in the data mathematically. This could further be expanded into multi function approaches that help advance the basics into further more in depth machine learning strategies. This study proved that basic theoretical concepts could be applied to real world situations to develop functions to classify digits. The very little complexity of the algorithm could be a shortcoming, this is shown in the decent but seemingly improvable error rate for the general classifier. In the future further works could implement more detailed and in-depth functions, and better optimize the parameters, and feature identification. In conclusion, this study is an introductory exploration into classification of MNIST using a methodological framework based on core ESE principles, projecting a foundational path for future machine learning applications.

References

[1] S. Boyd and L. Vandenberghe, “Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares,” Introduction to applied linear algebra – vectors, matrices, and least squares, https://web.stanford.edu/boyd/vmls/.

[2] MATLAB documentation, ”MATLAB R2023a,” The MathWorks Inc., Accessed October 2023. Available: https://www.mathworks.com/products/matlab.html