MNIST Digit Classification Using an Adjusted K-Means Implementation

Rex Paster
McKelvey School of Engineering
Washington University
Saint Louis, USA
p.rex@wustl.edu

Connor Strong
McKelvey School of Engineering
Washington University
Saint Louis, USA
c.j.strong@wustl.edu

Abstract – Living in the 21st century, digit and character recognition function as a bridge between humans and machines across countless industries. Processes as intuitive as depositing a check or sending a letter rely heavily on the correct interpretation of handwriting by a machine. With such wide and imperative uses, this study attempted to recreate an arbitrary implementation of digit classification, using an adjusted implementation of the k-means clustering algorithm to cluster a set of MNIST digits. Additionally, we set out to develop a process to detect images which contained noise or other variations that made it harder to correctly read. Using cluster representatives trained from a set of 1500 training images, we were able to correctly classify a set of 200 unique test images with 11 planted outliers; yielding an accuracy of 81.5% of images accurately clustered—with 8/11 planted outliers being correctly read.

I. Introduction

Given its wide range of uses, there have been countless implementations of Optical Character Recognition (OCR) algorithms. Like any computer learning algorithm, OCR can be approached in two main ways. The first option is a supervised learning method—where a computer is given presorted data and must draw a pattern between each preassigned categorization. The other method—unsupervised learning—works to sort unlabeled datasets based on their similarities [2]. Though both methods come with their own strengths and downfalls, k-means is a natively unsupervised approach, and as such, our implementation of k-means follows an unsupervised learning approach. In this project, we apply k-means clustering to handwritten digits exploring the classification of MNIST digits, along with the detection of likely miscategorized image-vectors.

II. Methods

K-means, the backbone of our classification program, is a widely studied and commonly implemented algorithm, due to its simple and versatile approach to grouping similar vectors. The textbook implementation of a k-means algorithm resembles the following roadmap:

1. Initialize Centroids

   In order for k-means to start grouping vectors, it first requires a beginning set of vectors known as centroids. These Centroids act as representatives for each grouping. In its most basic form, k-means uses a random selection of vectors from the data set as these initial centroids.

2. Assign Datapoints to Centroids

   In this step, the similarity of every vector to each centroid is measured; commonly calculated using Euclidian Distance. Then, each vector is assigned to the centroid which is most similar (i.e. closest) to the vector. The final k groups of vectors are known as clusters.

3. Recompute Centroids based on assigned Datapoints

   Once the vectors have been clustered (i.e. sorted into k groups), the centroids are recalculated. This is done by setting the new centroid of each cluster to the elemental mean of the respective vectors within them.

4. Repeat Steps 2 & 3 Until Convergence

   With each repetition of steps 2 and 3, the clusters become more distinct and in turn, more accurate. This process continues until the clusters stop improving (i.e. the sum of distances between the each vector and its respective centroid stays constant between iterations) [1].

In this implementation of k-means each vector contains 784 elements, each holding integers ranging from values of 0 to 255. These values signify the brightness of each pixel in a 28x28 greyscale image—along with a 785th element containing the correct classification of the MNIST image-vector. Using the k-means algorithm, we attempted to classify 1500 image-vectors of handwritten digits (0 to 9) into 40 groups, resulting in 40 cluster representatives which could then be used on another dataset to sort multiple image-vectors of handwritten digits.

In order to test how successful our implementation of k-means was in classifying the image-vectors, we took the cluster-representatives generated from the set of training data, and used them to sort a separate testing set of 200 image-vectors. Notably, 11 of these image-vectors had been ingrained with a randomized 13x13 noise element, placed over the center of the image, which acted as an obstacle to our sorting approach.

While designing and implementing the k-means algorithm, we considered what changes we could make in order to produce cluster representatives better suited to accurately sort the MNIST data. Thus, we employed an iterative approach to designing an adjusted k-means algorithm, where after each modification, we tested its impact on the accuracy of test set classifications.

The aforementioned method of iterative testing led us to implementing multiple beneficial adjustments to our OCR process. The first of these tweaks was the implementation of k-means++; an adjusted implementation of centroid initialization, designed to improve clustering while minimizing iterations required for convergence.

The k-means++ method of centralization, process, and logic resembles the following roadmap:

1. Initialize 1st Centroid

   The first centroid representative is chosen at random from the vector dataset ***x***.

2. Calculate Vector Distances to their nearest centroid

   For each vector in the dataset, the minimum distance from the vector's nearest centroid $D(x_i)$ is calculated (1).

$$D(x_i) = min_j \left\| x_i - c_j \right\|^2 \quad (1)$$

   *Where **j** denotes the number of centroids that have been selected so far.*

3. Choosing the Next Centroid

   The next centroid is then chosen by using a weighted probability, where the likelihood of a vector being chosen as the next centroid is proportional to the vector's distance to its nearest centroid (2).

$$P(x_i) = \frac{D(x_i)}{\sum_{v=1}^{n} D(x_v)} \quad (2)$$

   *Where **n** denotes the number vectors in the dataset **x**.*

4. Repeat Steps 2 & 3

   Steps 2 & 3 are repeated until ***j*** is equal to ***k***; where ***k*** denotes the target number of initial centroids to generate [3][4].

Another tweak we implemented was the decision to pre-process all image-vector data to normalize variations of brightness caused by different writing styles. To do this, we measured the largest element (i.e. its largest brightness value) within each image-vector, and divided every element by its respective largest brightness value. As a result, every element in each image vector within both the training and testing set had a value between 0 – 1. This newly defined smaller range in which every element now fit between reduced the impact of fluctuating element magnitudes within each vector when calculating Euclidean distance, resulting in the comparisons of Euclidean distance being more accurate for the MNIST set.

With the aforementioned tweaks implemented, we shifted focus—testing different k values to determine how many clusters should be used for the dataset. After multiple iterations of different max cluster counts, we found that increasing the number of clusters, in the scope of up to 40 clusters, would increase the accuracy of our classifications. With these findings in mind, we chose to use the maximum permissible number of clusters: 40.

One of the final challenges we were tasked with was developing a method to determine what datapoints had been ingrained with this noise element. When developing this algorithm, we focused on creating a process that could find image-vectors which didn't fit particularly well into any singular cluster. While we tested multiple methods, our final process for finding outliers worked as follows: We first took the Euclidean distance from each image-vector to their two closest centroids. We then computed the absolute difference between each of the two distances for each cluster. With these absolute distances, we could find the *n* borderline datapoints (i.e. vectors where their 2 closest centroids were most similar in distance).

Finally, we developed a method based off of the standard deviation, which could be used to predict the number of outliers within a dataset. For each vector we measured it's Euclidian distance from the nearest centroid. This method would then mark a vector as an outlier if this distance was greater than ***t*** standard deviations of the distance from the clustered vectors to their respective centroid. However, because we were given a set number of planted outliers, we did not extensively explore a good value for ***t*** on this dataset; however, we can provide a base value of t = 1.8 as guidance for further exploration.

III. Results and Discussion

Through the process described above, we were able to design an implementation of the k-means algorithm which produced centroids that could classify the testing data with an accuracy of 81.5%. This accuracy shows promise; however, we were interested in the source of the misclassifications of 18.5% of the image-vectors. To investigate this error, we began by analyzing the specific numbers which were most commonly miscategorized.

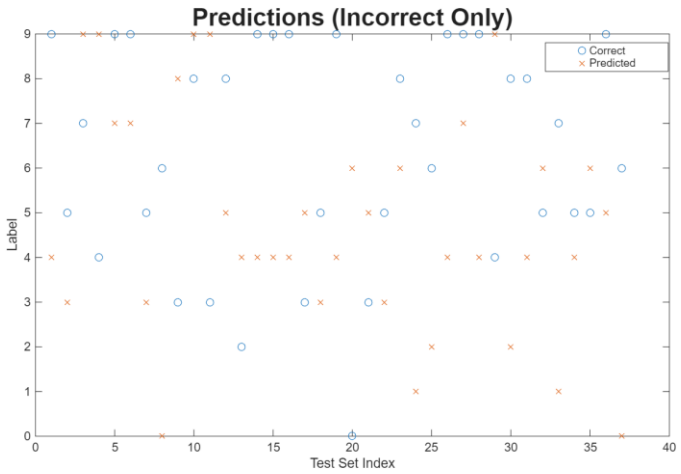

Figure 1: Misclassified Test Set Image-Vectors and Their Correct Classifications. [Similar Chart of All Test Set Vectors can be found in Appendix A]

As seen in Figure 1, there were various discrepancies in the categorization of the MINST data; digits 0, 1, 2, 3, 4, 6, 7, and 8 having between 1-4 inaccurate predictions, while digit 5 accounted for 7 inaccurate predictions, and digit 9 accounted for the most with 11 inaccurate predictions.

Upon further analysis it is apparent why digits 5 and 9 contributed to the most inaccuracies. Beginning with the digit 5, every incorrect prediction was either of a 3 or an 8, and observing the general shape of these digits, it is clear why this may occur, as digit 5 has certain distinct qualities, most notably the curve of its bottom half, which both the digit 3 and digit 8 also share. This trend was further validated when inspecting the inaccuracies from digit 9, as most inaccuracies of this digit came from being incorrectly grouped into a centroid representing digit 4, a digit which shares the qualities of both a bottom tail and an upper curve/slant with digit 9. With this conclusion in mind, a future adjustment to further increase the accuracy of our algorithm could be to create an additional layer of filtering which analyzes the elements of specific indices for vectors where incorrect centroid assignment is most common.

Another issue we found stems from the suitability of Euclidian distance comparisons for this dataset in general. Due to Euclidian distance's high reliance on magnitude, along with its susceptibility to the shifting of a figure in an image, Euclidian distance is not a good fit for comparing our data. Attempting to combat this, we also experimented with implementing the cosine distance formula as a method to calculate image similarity; however, this proved counteractive to the classification success rate. We hypothesize that a separate method to determine the similarity between vectors altogether can also provide utility in increasing the accuracy of the OCR.

We also found that by implementing k-means ++, the average number of iterations till convergence decreased by an average 20%, reflecting an increase in computational efficiency.

In addition to the 81.5% accuracy of classifications, we also were able to successfully classify 8/11 of the planted outliers using our k-means implementation. This is an remarkable result, showing a mere 8.8% discrepancy between the algorithm's ability to classify tampered data in comparison to its ability to classify standard MNIST digits. Notably, upon closer analysis we found that in some iterations of the algorithm, all outliers would be classified correctly. However, in such cases, our overall classification accuracy would experience a decrease, e.g. 77%. This suggests that the 13 by 13 noise matrix inscribed upon designated outliers was not enough to cause the algorithm to significantly struggle in classifying the tampered numbers.

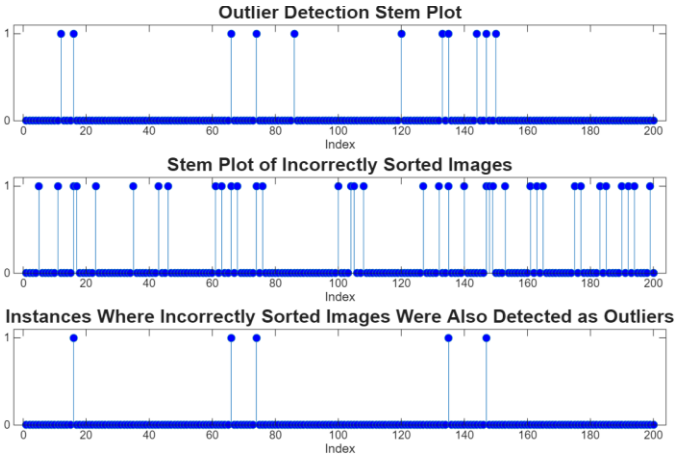

Figure 2: Stem Plots of The Flagged Outliers, The Misclassified Image-Vectors, and the Overlap between Flagged Outliers and Misclassified Image-Vectors

Prior to this understanding, there was concern due to the outlier detection algorithm failing to identify ten of the eleven planted outliers. However, with the aforementioned in mind, we shifted our focus. As shown in figure 2, we found that while the outlier recognition algorithm was failing to identify most of our planted outliers, it was successfully flagging datapoints which had been miscategorized. When tasked to find eleven outliers, 5/11 indexes which the algorithm returned correctly corresponded to miscategorized image-vectors. Given that only three outliers were missorted, it was both exciting, and in hindsight, expected, that the outlier finding algorithm could be expanded to find images that were misclassified at a success rate better than the rate in which it could find planted outliers.

The successful flagging of likely misclassified images showed promise, however, upon scaling it to find all 37 of the misclassified images, it only successfully flagged 13 of the 37 misclassified images. This suggests that there may be a better approach to finding misclassified images. Given that the outlier flagging algorithm looks for images that don't fit well into a singular cluster, we hypothesize that implementing a larger number of clusters or a better comparison metric for MNIST digits would enable greater success of outlier detection, as outliers would be less likely to sufficiently fit into a singular cluster.

IV. Conclusion

At the tail end of this study, we were able to actualize our goal of designing a k-means implementation to classify MNIST digits with an accuracy of 81.5%. Additionally, we found partial success in an outlier detection algorithm, successfully tagging 13/37 misclassified image-vectors. Based on this data, an

unsupervised training model such as k-means can be implemented to successfully classify hand-written digits. Although we are extremely satisfied with these results, we are aware that this case study was limited by multiple factors including a time constraint of 2 weeks, a limitation on the maximum permissible cluster count, and a lack of knowledge on the implementation of other methods of comparing vectors outside of Euclidian and Cosine distance. If additional research was to be done on this subject, we would suggest exploring a more specialized comparison metric for image-vectors of MNIST digits, and/or an exploration into a higher maximum cluster count to account for stylistic differences between similar digits. We hypothesize that with such improvements, the algorithm would likely produce better results not only in the classification accuracy, but also of the success of our outlier detection algorithm in locating misclassified image-vectors.

References

[1] S. P. Boyd and Lieven Vandenberghe, *Introduction to Applied Linear Algebra : Vectors, Matrices, and Least Squares*. Cambridge: Cambridge University Press, 2018.

[2] J. Delua, "Supervised vs. Unsupervised learning: What's the difference? | IBM," *www.ibm.com*, Mar. 12, 2021. https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning

[3] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," 2006. Available: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[4] Wikipedia Contributors, "k-means++," *Wikipedia*, May 02, 2020. https://en.wikipedia.org/wiki/K-means [Accessed 9/27/2025]

Statement on the use of AI

Artificial Intelligence was utilized in multiple aspects of the development of this case study. Throughout the development of our MATLAB implementation of a k-means algorithm, AI was used for Error Analysis, Optimization (regarding speed and memory usage), Comment Grammar Checking, and Formatting to produce an easily implementable and resource efficient script. However, all code, apart from few single-line statements, was written by Rex Paster or Connor Strong.

In the process of writing this document, AI was also utilized for Analysis of Writing to determine weakly written or developed areas, along with assistance in assigning specific ideas to their corresponding sections. However, as with the MATLAB Implementation, no writing in this document is a direct or adjusted product of AI, rather this case study was written entirely by Rex Paster and Connor Strong.

Additional Awknowlagements

We would like to thank Ilan Kilman for the original suggestion of investigating a k-means++ implementation.

Appendix A:
Full Prediction vs. Correct Assignment Figure