Project 1 Write-up

Shaolong Li
shli@ucsd.edu

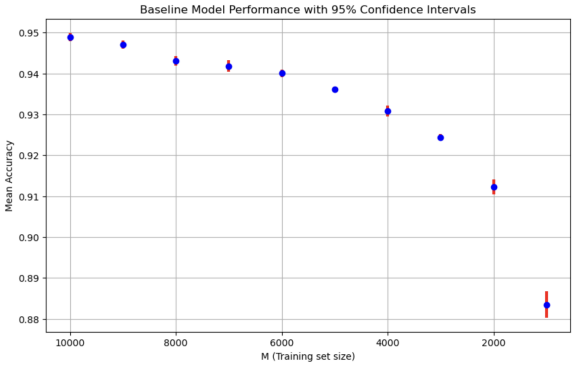
Abstract

Utilizing the entire dataset for KNN training can be time-consuming. Hence, selecting a portion of the dataset that accurately represents the entire dataset can significantly reduce training time. In this project, we have implemented various methods to identify representative prototypes from the MNIST dataset for classification tasks using the KNN algorithm. We have established one baseline method and three enhanced methods. The outcomes of the enhanced methods will be compared with the baseline.

1 Implementation Details

1.1 Baseline random selection

In the baseline model, we randomly choose M data samples from the training set to ensure randomness. We initially obtain all 60,000 indices (60,000 for training, 10,000 for testing) and then use **np.random.shuffle** to shuffle all the indices. We subsequently select the first 10,000, 9,000, 8,000, ... 1,000 data samples from the training set and then run KNN with k=1 on these training sets. The KNN model, trained with different training set sizes, was then tested using the test set of size 10,000, and the accuracy score for each training set size was calculated. We ran the baseline model 10 times and the error bar with a 95% confidence interval is plotted as follows.



The formula we used to compute confidence interval is as follows and we will use the same formula for the second model. We first compute the mean accuracy score for each M:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Then we calculate the standard error of the mean and the margin of error for 95% confidence interval:

$$SE = \frac{\sigma}{\sqrt{n}}$$

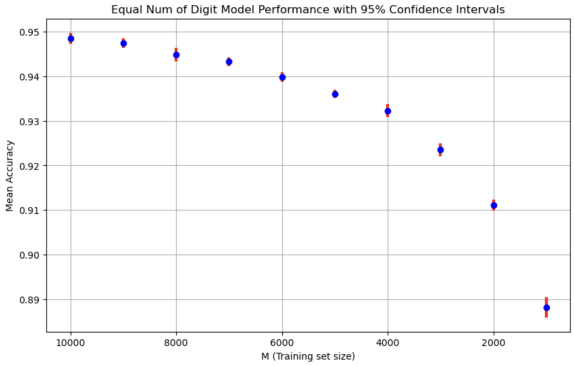
$$MOE = z \times SE$$

The confidence interval is then calculated as follows:

$$CI = \bar{x} \pm (z \times SE)$$

1.2 Equal number of digit selection

One slight improvement to enhance baseline accuracy is ensuring equal amounts of each digit in our training set. To maintain a balanced training set, we first calculated the number of data d needed for each digit corresponding to the different sizes of the training set (10,000, ... , 1,000). For example, if our training size is 10000, we then need d = 1000 for each digit. After gathering all the data for each digit from the entire 60000 training set, we used **np.random.shuffle** to ensure randomness and selected the first d data from each digit class. This step ensures the training set size exactly matches 10,000, ... , 1,000. Subsequently, we trained the KNN model and tested it on the test set. We ran the baseline model 10 times and the error bar with a 95% confidence interval is plotted as follows.

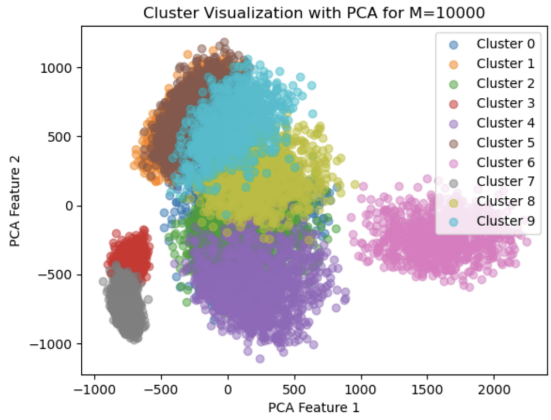


1.3 K-means Clustering Selection

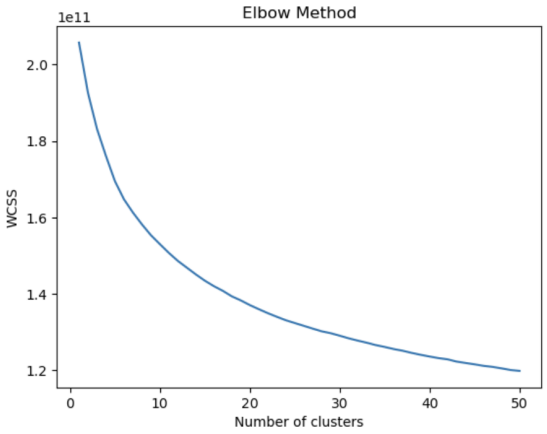
Another method to select prototypes is by using the K-means algorithm. Clustering the MNIST dataset data shows that digits with similar graphical characteristics are grouped together. However, many outliers in each cluster, distant from the cluster center, likely contribute less to KNN classification. Therefore, the model involves selecting only the points close to the cluster center. Initially, we applied K-means to cluster all 60,000 training data points, using the mean of all points in a cluster to represent its center. For each cluster, we then calculated the Euclidean distance from each point to the cluster's center and sorted these distances. Subsequently, we selected the appropriate proportion of data from each cluster so that the total number of training data summed up to 10,000, ... , 1,000. For example, if we want the training size to be 10000, we select 1/6 (10000 / 60000) of the points from each cluster, which adds up to 10000. Afterward, we trained the KNN model using this newly formed training set and tested the model on the test set.

Pseudocode

1. Run K-means alogirthm to get the cluster
2. for each cluster, get all the data in this cluster
3. num of point in this cluster = num of point in this cluster // (60000 // M)
4. center for this cluster = np.mean(all points in this cluster, axis=0)
5. sort all points in this cluster using distance
6. return sorted_x_train[:num of point in this cluster], sorted_y_train[:num of point in this cluster]



After running K-means on the training dataset, we discovered that the value of k (number of clusters) can impact the model's accuracy on the test set. Therefore, we sought to identify the optimal k value. Following some research, we decided to employ the elbow method to determine the optimal k. This method assesses the compactness of each cluster by calculating the WCSS (Within-Cluster Sum of Squares) score. Higher compactness implies a lower WCSS, indicating that the points are well-clustered. We tested 50 different k values (1 - 50) and then plotted the WCSS score for each. Although the WCSS score consistently decreased, the plot was relatively smooth and did not reveal a clear elbow point. Consequently, we experimented with all 50 k values and observed that test accuracy slightly improved as the k value increased.



1.3 Improved K-means Selection

To enhance the K-means method further, we implemented the approach detailed in the paper *A New K-Nearest Neighbors Classifier for Big Data Based on Efficient Data Pruning (2020)*. We did have some small modifications for some specific formulas in the paper to simplify the calculation. Rather than merely using K-means to cluster the training set, selecting the points closest to the center, and training KNN on the entire training set, this paper employed K-means to form clusters and then utilized a more sophisticated method to determine which cluster is most suitable for a particular test sample. Subsequently, the KNN model was trained using only the data points from this specific cluster.

To implement the algorithm from the paper, we began by setting k = 5, as there appears to be a slight elbow point at about k = 5 upon close examination of the elbow plot. We then

ran the K-means algorithm. After grouping the 60,000 training set data points into 5 clusters, we calculated the portion we needed from each cluster according to the sizes 10,000, ... , 1,000. We then calculated the Euclidean distance for the points in each cluster, sorted using the distance, and selected the portion closest to the center, similar to our previous K-means method.

When selecting the most appropriate cluster for each test sample, the algorithm initially calculates the Euclidean distance from the test sample to the center of each cluster. It then selects any cluster that is significantly close to the test sample, specifically those with a distance less than alpha times the mean of all Euclidean distances. Here, alpha is a hyperparameter ranging from 0 to 1:

$$d_i <= \alpha * avg(d)$$

If no clusters are considered significantly closer to the test sample than others, the adjusted Euclidean distance will be calculated. Here, *dij*represents the maximum Euclidean distance between the data points of the ith cluster and its center along the jth axis:

$$D_i = d_i - \sum_{j=1}^{n} \frac{(test\_sample_j - C_{ij})^2}{d_i} \times d_{ij}$$

By incorporating *dij*, the adjusted Euclidean distance essentially computes the distance from the test sample to the border of a cluster.

Once the adjusted Euclidean distances are calculated, the significantly smaller one will be selected. The cluster considered significantly smaller should satisfy the following condition:

$$D_i <= \beta * avg(D)$$

If there is still no cluster considered significantly closer than others, the density of each cluster will be calculated, and the one with the highest density will be selected.

Once the appropriate cluster is selected, we will exclusively train the KNN model using this cluster and then employ the trained model to predict the test sample. In other words, for each test sample, we first identify its most suitable cluster, use this cluster to train the KNN model, and then apply the KNN model to predict the test sample.

2 Experimental Evaluation

2.1 Compare Equal Digit Model with Baseline Model

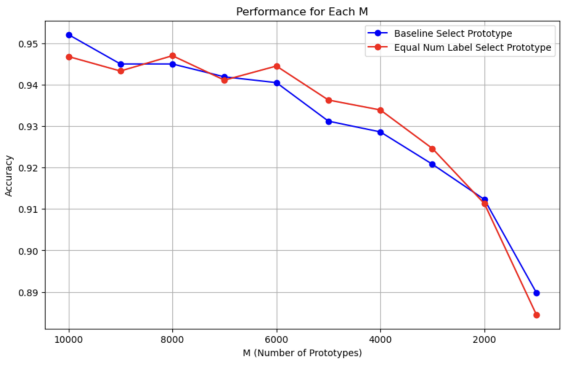
By comparing the test accuracy of the Equal Digit Model with the Baseline Model, we can observe that their test accuracies do not show much difference. Looking at the plot, both models have relatively stable test accuracies for training sizes of 10,000, ... , 6,000, with test accuracy ranging between 0.94 and 0.95. When the training size decreases to less than 4,000, the test accuracy of both models drops significantly.

Baseline Model Accuracy for Different M

	M (Training set size)	Accuracy
0	10000	0.947900
1	9000	0.947400
2	8000	0.944500
3	7000	0.943400
4	6000	0.941200
5	5000	0.934700
6	4000	0.932400
7	3000	0.918200
8	2000	0.913000
9	1000	0.888000

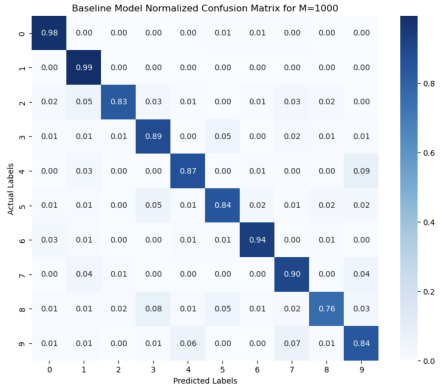
Equal Num of Digit Model Accuracy for Different M

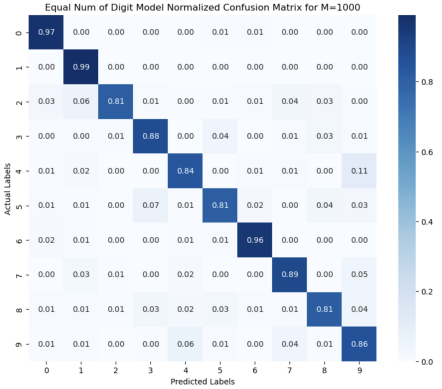
	M (Training set size)	Accuracy
0	10000	0.948500
1	9000	0.947200
2	8000	0.944100
3	7000	0.947300
4	6000	0.942200
5	5000	0.932800
6	4000	0.933200
7	3000	0.924800
8	2000	0.914000
9	1000	0.880100



Examining the confusion matrix for both models

reveals that they both perform well with a training size of 10,000 and exhibit the highest accuracy for digits 0, 1, and 6. As the training size decreases, test accuracy for other digits drops, but remains high for 0, 1, and 6. The baseline models struggle with predicting digit 8 and are likely to misclassify it as digit 3 or 5 but this does not happen to the Equal Num of Digit model. In summary, both models perform relatively the same, which is somewhat expected because the MNIST dataset is already quite balanced, so enforcing an equal number of digits for each class does not contribute significantly to improvement.



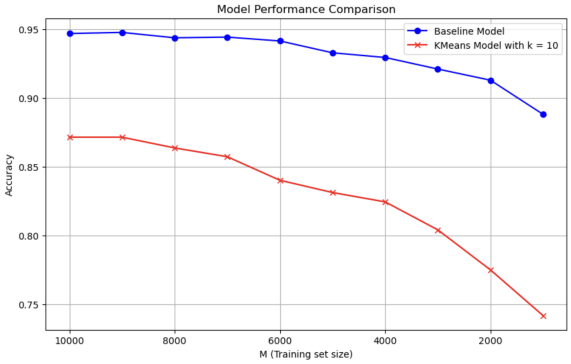


2.2 Compare K-means Model with Baseline Model

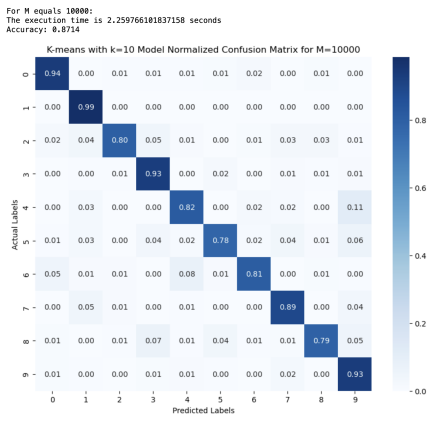
Because there are 10 digits, we first tried running the K-means model with k = 10, and the accuracy for each training set size is as follows:

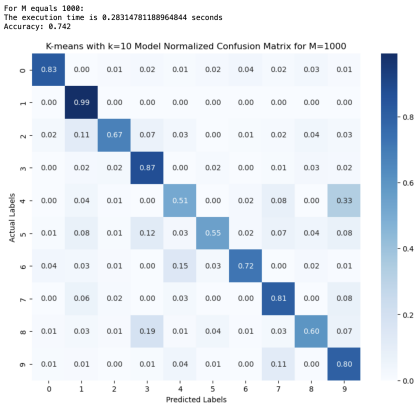
K-means with k=10 Model Accuracy for Different M

	M (Training set size)	Accuracy
0	10000	0.871500
1	9000	0.871500
2	8000	0.863700
3	7000	0.857300
4	6000	0.840200
5	5000	0.831300
6	4000	0.824500
7	3000	0.804200
8	2000	0.775100
9	1000	0.741900

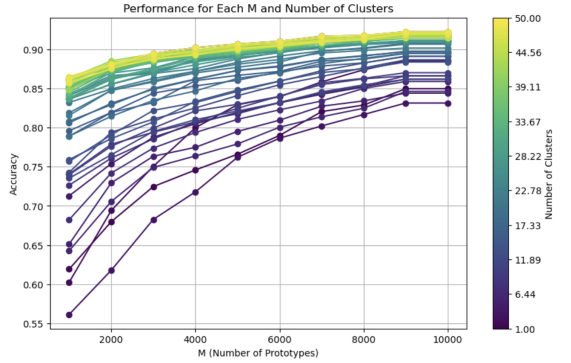


From the plot, we can see that the overall accuracy of the K-means model is significantly lower than that of the baseline model. Upon examining the confusion matrix, we observe that this model excels at classifying digits 0, 1, 3, and 9. As the training size decreases, the accuracy remains good for digits 0, 1, and 3, but it is notably low for digits 4, 5, and 8. Specifically, it consistently misclassifies digit 4 as 9. This is somewhat expected because 4 and 9 are somethwat graphically similar and as the training size decrease, training sample which contain unique feature of 4 and 9 lost which make the





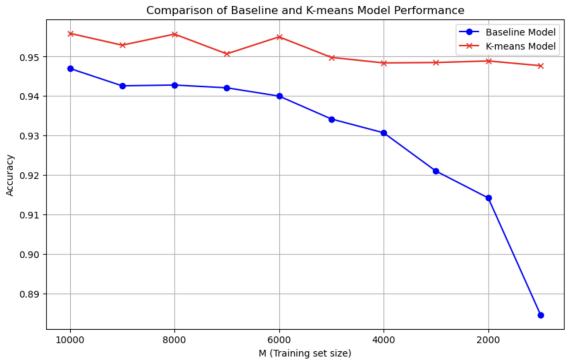
Given that the number of clusters may affect the test accuracy, we tried using the Elbow method to find the optimal cluster number K. We tested 50 K values from 1 to 50, but the plot was rather smooth and did not show a clear elbow point. We then printed out the test accuracy for all 50 K values and observed that, generally, the test accuracy increases as the K value increases. This result is logical because, as the number of clusters increases, each cluster tends to be more compact, which is also reflected in the downward trend of the curve in the elbow graph. Because each cluster is more compact and we are selecting the points that are close to the cluster centers, the points we selected are therefore more representative of certain features for a certain digit. Moreover, as K increases, we capture more variance and nuances of the data, thereby achieving higher accuracy. However, although the accuracy continues to increase as we increase the K value, the test accuracy (around 0.92 with 10000 training size and K equal to 50) is still lower than the baseline model test accuracy (around 0.94 with 10000 training size). One way to improve this model is simply to keep increasing the K value, but by examining the plot, we can see that the test accuracy improves very slowly as the K value increases.

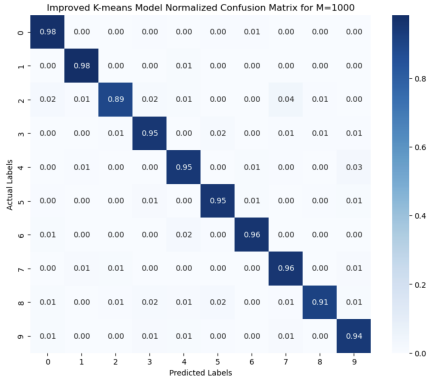


2.3 Compare Improved K-means Model with Baseline Model

The result for the improved K-means model yields the best outcome, with a test accuracy of 0.9529 when the training size is 10,000. After experimenting with different training sizes ranging from 10,000 to 1,000, we discovered that the overall test accuracy does not change significantly. Although there is a slight drop after 6,000, the overall accuracy variation remains within 0.01, with the lowest being 0.9477. By examining the confusion matrix, we can see that the model performs well on all digits except 2, which is sometimes misclassified as 7.

	M (Training set size)	K-means Accuracy
0	10000	0.9559
1	9000	0.9529
2	8000	0.9557
3	7000	0.9507
4	6000	0.9550
5	5000	0.9498
6	4000	0.9484
7	3000	0.9485
8	2000	0.9489
9	1000	0.9477





While the improved K-means model performs better than the baseline model, it does require pre-processing the input data. Given that we need to predict each test sample with its specific KNN trained with certain clusters, we must first process the entire test set and select the proper cluster for

each test sample. Then, we build a KNN classifier and fit it with each cluster. When predicting the test sample, we need to identify the specific KNN model trained using the appropriate cluster and then use it to predict each test sample. This model outperforms the one we proposed earlier because it predicts each test sample using a small subset of the training set, which only contains data points most similar to the test sample, thereby eliminating any noise that may affect the prediction. Moreover, when selecting the most suitable cluster for each test sample, the model uses multiple layers of filters to identify the best-suited cluster for the test sample. In the first layer, the model simply calculates the Euclidean distance, and if a cluster stands out, it selects this cluster. If no cluster stands out, the distance from the test sample to the border of each cluster is calculated, and the standout cluster is selected. Being closer to the centroid does not necessarily mean it's more representative because some clusters may be more spread out. Therefore, even though the centroid is slightly farther from the test sample, the cluster itself might be quite close to the test sample. By calculating the distance to the cluster border, this model successfully addresses this scenario. If still no clusters stand out after the second filter, the density of the cluster is calculated, with a higher density indicating a more compact cluster, which in turn means it's more representative of certain features. One potential way to further improve this model is by fine-tuning the hyperparameters alpha and beta. According to the paper, a higher beta value places more weight on the distance to the cluster border, while a higher alpha value emphasizes the Euclidean distance to the centroid. However, training this model is very time-consuming, so we selected the alpha and beta values of 0.5 and 0.7 as recommended by the paper.

3 Conclusion

In this project, we built three models in addition to the baseline. All three models reduced the training size and increased speed, but only the model referenced from the paper outperformed the baseline model's accuracy by an overall margin of 0.02. However, this model does require a significant amount of preprocessing, which includes finding the most suitable cluster for each test sample. Further improvements could be made either by increasing the number of clusters or by fine-tuning the hyperparameters. Additionally, alternative methods of calculating the density for the third model could be explored, such as using the WCSS (Within-Cluster Sum of Squares) score from the elbow method.

4 Reference

Saadatfar H, Khosravi S, Joloudari JH, Mosavi A, Shamshirband S. 2020. A New K-Nearest Neighbors Classifier for Big Data Based on Efficient Data Pruning. Mathematics. 8(2):286. doi:https://doi.org/10.3390/math8020286.