Examining Different Prototyping Methods on MNIST

Ryosuke Oguchi
roguchi@ucsd.edu

Abstract

The objective of this paper is to explore different methods of choosing "prototypes" to utilize nearest-neighbor classification. Moreover, the ultimate goal of this experiment is to determine a strategy that maximizes classification accuracy against a certain prototyping strategy. As a baseline model, random selection will be implemented on MNIST images to serve a point of comparison of selection methodologies. The results indicate that mathematically derived prototype selection improves classification in the lower number of prototyping points, which is key to balancing time efficiency and accuracy.

1 Idea for Prototype Selection

1.1 Random Selection

Under random selection, we will draw prototypes of size M from MNIST training data under a uniform distribution. This means that, in general,

$$P(X = x) = \frac{1}{N}$$

where $N = 60000$. However, considering the number of combinations that will be generated for the size prototypes M, that number will be

$$\approx \frac{\sqrt{2\pi N}(\frac{N}{e})^N}{\sqrt{2\pi M}(\frac{M}{e})^M \cdot \sqrt{2\pi(N-M)}(\frac{N-M}{e})^{N-M}}$$

using Stirling's approximation. The general takeaway from random selection is that as $M \to \frac{N}{2}$ the number of combinations for sampling M points increases, which increases the variability in the classification results of the 1 Nearest-Neighbors algorithm. Although there will be significant overlap of the points chosen as $M \to \frac{N}{2}$, and due to the similarity of numbers seen in MNIST, we will not see much variation in the classification output but there exists randomness in the accuracy output.

1.2 K-Means Selection

For a more mathematically driven methodology in choosing M prototypes, K-Means selection is chosen for the next experiment. The algorithm utilizes the following steps:

1. Arbitrarily choose an initial k centers $C = \{c_1, c_2, \ldots, c_k\}$.
2. For each $i \in \{1, \ldots, k\}$, set the cluster C_i to be the set of points in X that are closer to c_i than they are to c_j for all $j \neq i$.
3. For each $i \in \{1, \ldots, k\}$, set c_i to be the center of mass of all points in $C_i : c_i \frac{1}{|C_i|} \sum_{x \in C_i} x$.
4. Repeat Steps 2 and 3 until C no longer changes

As a result, this experiment will create $k = M$ centers to serve as centroids that the 1 Nearest-Neighbors algorithm will try to minimize its distance against. However, it must be recognized that general K-Means clustering is an NP-Hard problem. This means that although this prototype selection can yield higher accuracy over random selection, there is a computational tax that results in non-convergence to the global optimum over many iterations.

1.3 K-Means++ Selection

To remedy the general issues posed by the general K-Means selection, K-Means++ initialization is also implemented as a method of prototype selection. The paper by (David Arthur, 2007) introduces their rendition of the K-Means centroid selection as the following:

1a. Take one center c_1, chosen uniformly at random from X.

1b. Take a new center c_i, choosing $x \in X$ with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$

1c. Repeat Step 1b. until we have taken k centers altogether.

2-4. Proceed as with the standard k-means algorithm.

Overall, the authors of this algorithm claim that due to the carefully selected initial points of the algorithm, it leads to more separation in the clusters and thus more accurate centroids. Moreover, they claim that there is an advantage in convergence time due to the early termination of steps 2-4.

1.4 Combining Sampling and K-Means

As a thought experiment, another method of prototyping that will be used is to simply combine the random sampling with the aforementioned K-Means methods. The primary motivation for this was developed by individuals discussing the validity of utilizing K-Means++ against the NP hardness in regular K-Means. https://stackoverflow.com/questions/4706678/should-we-used-k-means-instead-of-k-means

However, combining two heuristic prototyping processes are accompanied by risks and assumptions that must be put into consideration when evaluating the accuracy of our Nearest-Neighbors algorithm that is applied to the resulting prototypes.

1. Size of random sample should be $\gg M$ so K-Means can still be applied to the number of remaining points. (Or else it will simply be random selection)

2. The random sample still well approximates the distribution of the original sample (This requires some general knowledge on how the data is distributed)

Although there is no general methodology behind choosing which points gets passed through the K-Means algorithm, simply using less points yet maintaining the overall distribution will allow the development of centroids to be quicker without compromising accuracy at a large scale.

2 Pseudocode for Implementing Prototype Selection

In general, regardless of prototype selection, 1 Nearest-Neighbors will use euclidean distance to return the label of the predicted class.

The eucliean distance is defined as follows:

$$d(x_q, x_i) = \sqrt{\sum_{k=1}^{d} (x_q^k - k_i^k)^2}$$

where k represents the feature component.

Then, the prediction operates as follows:

$$\hat{y} = y_j, j = \arg\min_i d(x_q, x_i)$$

where j is the index of the nearest neighbor.

2.1 Random Selection

To select M prototype points to be reference points for 1 Nearest-Neighbors, utilize numpy's random.choice() function to select the indices for our data points. Against these selected points, we calculate euclidean distance between our test data and selected points and select the point that has smallest distance and return the corresponding label from our randomly selected data points.

2.2 K-Means Selection

To find M centroids for our K-Means clustering, sklearn's MiniBatchKMeans algorithm is used to process all N data points to return M centroids. Under the hood, steps for both regular K-Means and K-Means++ are iterated in the section above. Given the returned M centroids, we can view these as reference points that are used to calculate distance between the test points and centroids. From here, the method of classification remains the same between Random Selection and K-Means as 1 Nearest-Neighbors are used to return the label of the test points.

2.3 Random Selection + K-Means

To combine the two aforementioned methods, we first need to randomly select $j = \{45000, 30000, 15000\}$ points that are $\gg M$. Again, numpy's random.choice() method is used to return the indices for j points. Then, it is passed through sklearn's MiniBatchKMeans algorithm to return M centroids. From here, the method to return the labels of our points is exactly the same as all previously used methods. It is necessary to consider here that due to the initial random sampling of our j points, there is built in variation of the centroids we create from K-Means clustering; hence, there will be statistical variation in the predictions we will create, which affects accuracy.

3 Experimental Results

3.1 Random Selection

Viewing the results of random selection is necessary to establish simply confidence intervals of our accuracy due to the difference in the set of points we sample to run our 1 Nearest-Neighbors Algorithm. Said formula to calculate our confidence intervals are as follows:

$$[\bar{x} - t \cdot \frac{s}{\sqrt{n}}, \bar{x} + t \cdot \frac{s}{\sqrt{n}}]$$

Where, $\bar{x}$ is the average of our accuracy runs, t is our t-statistic controlled by the degrees of freedom of our number of runs, s is our sample standard deviation of accuracy runs, and n being the number of runs we have conducted.

Our confidence intervals for our random sampling experiment are shown in Tables 1 and 2

Prototype Size	95% LB	95% UB
10000	0.9469	0.9504
5000	0.9349	0.9390
1000	0.8745	0.8972

Table 1: Random Selection 95 Percent Confidence Intervals for Accuracy

Prototype Size	99% LB	99% UB
10000	0.9447	0.9527
5000	0.9323	0.9416
1000	0.8597	0.9120

Table 2: Random Selection 99 Percent Confidence Intervals for Accuracy

Given these results, we view these confidence intervals as our baseline goals to beat using our targeted methods of prototype selection.

3.2 K-Means Selection

For our K-Means selection, we hope to see that higher accuracy than random selection even though it is computationally difficult for the algorithm to find global convergence on our centroids. Although there may be "randomness" due to the initial reference centroids that Lloyd's algorithm utilizes, we will assume that it won't be enough to affect our accuracy. The results are illustrated in Table 3:

Fortunately, we see here that K-Means prototyping beats the 99 percent upper bounds across all prototype sizes. Moreover, we see solid margin against the $M = 5000, 1000$ cases which provides reason to select K-Means on smaller number of prototypes for good accuracy. However, it worth considering that for it to develop M centroids, we still require N points for training.

Prototype Size	Accuracy
10000	0.9592
5000	0.9542
1000	0.9372

Table 3: Accuracy of Basic K-Means Prototyping

3.3 K-Means++ Selection

The principles for K-Means++ clustering follows most of the same principles from K-Means as stated before. A key difference between the two being that the initial selection of centroids are not chosen at random as seen in Lloyd's Algorithm. However, by carefully choosing the initial centers based off of probability of distance, the convergence time of the algorithm should decrease. The results are illustrated in Table 4:

Prototype Size	Accuracy
10000	0.9555
5000	0.9552
1000	0.9368

Table 4: Accuracy of K-Means++ Prototyping

The results of our K-Means++ tell and interesting story. Against the baseline of our random selection, it appears to out perform against all prototype sizes. Where it does noticeably better on smaller prototype sizes of M. However, it only appears to outperform basic K-Means only under the $M = 1000$ case. Although the results are comparable as there is only a $< 1\%$ difference, it goes against the claims outlined in the results of Arthur and Vassilvitskii. The two originators of K-Means++ claim that "the k-means method does not perform well, because the random seeding will inevitably merge clusters together, and the algorithm will never be able to split them apart. The careful seeding method of k-means++ avoids this problem altogether, and it almost always attains the optimal results on the synthetic datasets" (David Arthur, 2007). As a further extension to these claims against our experiments, we must explore the view stated by independent users in regards to the usage of K-Means vs K-

Means++. https://stats.stackexchange.com/questions/130888/k-means-vs-k-means

Alternatively, we can view this oddity as a problem with high dimensionality. Overall, our inputs are in dimensions of $\mathbb{R}^{784}$ space which is contrary to the empirical tests that were conducted in the original paper. Moreover, the choice to have M centroids are $\gg$ than the k centroids the paper created. Hence, the deliberate choice to create calculated centroids may lead to slight over-fitting.

3.4 Basic K-Means + Sampling

As stated before, we have to reconcile with the fact that there is a level of variation that is associated with the random selection of the initial points K-Means will be operated on. The confidence intervals are as in Table 5-7:

Prototype Size	95% LB	95% UB
10000	0.9538	0.9605
5000	0.9482	0.9576
1000	0.9238	0.9344

Table 5: Random Selection for 45k Points + K-Means 95 Percent Confidence Intervals for Accuracy

Prototype Size	95% LB	95% UB
10000	0.9385	0.9486
5000	0.9421	0.9509
1000	0.9262	0.9339

Table 6: Random Selection for 30k Points + K-Means 95 Percent Confidence Intervals for Accuracy

Prototype Size	95% LB	95% UB
10000	0.9319	0.9403
5000	0.9415	0.9435
1000	0.9125	0.9302

Table 7: Random Selection for 15k Points + K-Means 95 Percent Confidence Intervals for Accuracy

In general, we see that even for our smallest sampling, implementing K-Means over this subset is still competitive the 99 percent confidence interval for smaller prototype sizes. We presume that for smaller subsets and larger number of prototypes, the centroids being created are over-fitting to the samples we selected. However, it appears that it is not beating K-Means nor K-Means++ clustering. This logically upholds as we expect the outputs from K-Means to be well representative of different types of data as $n \to \infty$.

3.5 K-Means++ + Sampling

The need to create confidence intervals for our K-Means++ experiments follows the same reasoning as before. Tables 8-10 are the 95 percent confidence intervals for our experiments:

Prototype Size	95% LB	95% UB
10000	0.9499	0.9622
5000	0.9504	0.9556
1000	0.9312	0.9367

Table 8: Random Selection for 45k Points + K Means++ 95 Percent Confidence Intervals for Accuracy

Prototype Size	95% LB	95% UB
10000	0.9427	0.9533
5000	0.9432	0.9526
1000	0.9221	0.9303

Table 9: Random Selection for 30k Points + K Means++ 95 Percent Confidence Intervals for Accuracy

Prototype Size	95% LB	95% UB
10000	0.9350	0.9495
5000	0.9380	0.9403
1000	0.9124	0.9280

Table 10: Random Selection for 15k Points + K Means++ 95 Percent Confidence Intervals for Accuracy

We see here that there is a mixed bag of results for this experiment. At worst, we see that it beats our baseline confidence intervals for even 99 percent. However, as seen before, it does not seem to outperform the 1 Nearest Neighbors that are applied on our full training data. Again, the explanation for this is because K-Means performs better as $n \to \infty$.

4 Critical Evaluation

4.1 General Results

In general, we see that using K-Means and K-Means++ clustering to generate centroids for 1-Nearest Neighbors to be applied to beats random selection. This is more evident when using smaller number of Prototypes for selection. However, the general drawback of creating M centroids from N

points will be computationally taxing, especially in K-Means++ as it requires every initial centroid to be calculated. Therefore, the usage of random selection was introduced alongside K-Means selection to reduce the training time of the K-Means algorithm. However, because of the untargeted nature of deciding what points to train on, our Nearest Neighbors classification suffered in accuracy.

4.2 Further Improvement and Potential Extensions

To maintain the idea of time efficiency while not trading away accuracy, a general proposal is to implement method to remove points that do not help in training accuracy for K-Means centroids. We can possibly do this by considering which points are "farthest" from our initial M centroids, and then retrain K-Means with points that best represent the original dataset. With this method, we can remove the noise that may impact the K-Means centroid generalizations. Although "bad" data points are not being considered, we may be arbitrarily be creating over-fitting to our training data.

Alternatively, we may need to revisit how we view distance in our case. In terms of MNIST, each data point is considered as a gradient scale of black-white. Hence, based off the centering or rotation of each number, general euclidean for 1 Nearest-Neighbors may classify certain images as "far" when translations exist. So by utilizing tangent distance may better represent the distance better. This still requires domain expertise on what the data looks like before outright committing to tangent distance over euclidean.

Overall, considering K-Means clustering as a form of prototyping to reduce classification and improve accuracy is a basic, yet strong form of prototype selection. Any large scale improvements will require hyper-parameter tuning or an adoption of alternative clustering algorithms.

References

Sergei Vassilvitski David Arthur. 2007. k-means++: The advantages of careful seedings.