Prototyping for implementing K-nearest-neighbors on MNIST using KMeans and Hyperdimensional Computing Clustering

Fatemeh Asgarinejad (PID: A59006734)

[pseudo code can be found in the end]

Abstract

One of the approaches to deal with high computation time of implementing KNN in large datasets is prototyping. Meaning, reducing the number of training data by selecting a subset of it and treating it as training data. In this study, for choosing $\mathbf{M}$ data points from the initial training set, I present two different algorithms. In first approach, I use KMEANS to cluster training data into $\mathbf{k}$ clusters and then from each, choose $\frac{\mathbf{M}}{\mathbf{k}}$ samples, then treat the chosen data as new training sample and implement KNN for determining the class of test points. In the second approach, I implement Hyperdimensional (HD) as clustering method.

Comparison of the two proposed approaches with random sample selection shows that my Kmeans beats the random prototype selection with an average accuracy of 0.9 percent for 1000 prototypes and yields similar accuracy to random selection for 5000 and 10000 samples though consuming considerably higher time. However HD algorithm surpasses random selection with 1.8, 0.7 and 0.4 percentages for sample sizes of 1000, 5000 and 10000 respectively while consuming almost similar times except for the initial encoding which is required to be done just once.

I. Introduction to prototyping

Although KNN is the most used and well-known classification algorithm, it suffers from several drawbacks, including high storage requirements and time and energy consuming computations when dealing with large dataset. One of many approaches to deal with this issue is prototyping. Meaning selecting a subset of training data and implementing KNN using this smaller training sample. This selection can be random but in order to have a subset which better represent the training data, many subset selection have been proposed so far. In this study, I present two different sample selection methods and compare the ultimate classification results with random selection.

II. Dataset

The dataset that is used in this study is MNIST [4] which contains 60,000 training and 10,000 test images.

For the experiments, I do prototyping on the training data and for evaluate the proposed algorithms and random sampling on initial test data with 10,000 images.

III. Random prototyping

In random prototype selection, in the first phase M datapoints are randomly chosen from training data. Then, for classifying test set, k nearest neighbors of each test data $test_i$ $(i = 1\ to\ 10{,}000)$ is chosen from the randomly chosen sample. For comparison we have set M to be $1000, 5000$ and 10000 and have reported the results of 30 experiments and their comparison in figure 3.

IV. Proposed approaches: KMeaans Clustring + KNN

This approach consists of 3 main phases and is briefly pointed out in [1]. In its first step, training data is clustered into k clusters using KMeans clustering algorithm [1][2][6], then, since the goal is choosing a total of M samples of the whole training data, from each cluster

$i\ (i: 0\ to\ k-1)$, we randomly choose $\frac{M}{K}$ images.

Step 1: Kmeans starts with randomly choosing k cluster representatives, aka centroids, from the training data. Then for each training image $train_j\ \ where\ j: 1\ to\ 60000$, attributes it to one of the k cluster representatives based on the Euclidean distance (Euclidean norm). For a training point $train_j$, l is the index of the nearest centroid it is attributed to:

$$l = \underset{i=0\ to\ k-1}{argmin}\left(||train_j, centroid_i||_2\right)$$

After that, the center of each cluster is re-estimated as the mean of all the cluster components.

Then, iteration on finding closest centroid and re-estimating the clusters' centers are done until convergence. The criteria for convergence here is set to be the default 10 iterations. For a future study, we can have a better convergence criteria like a higher maximum iterations or other convergence methods.

Step 2: After clustering all the training data into k clusters, from each cluster $i\ (i: 0\ to\ k-1)$, we randomly choose $\frac{M}{K}$ training data. Thereby, having a total of M samples. I have experimented with different values for k and M. To be more precise, k is set to be $10, 50$ and 100 and M is set to be $1000, 5000$ and 10000, constructing a total of 9 different pair-wise experiments. ex. The pair ($K = 10$ and $M = 1000$) corresponds to choosing 10 clusters and $\frac{1000}{10} = 100$ random samples from each.

Step3: Having the chosen M samples from training data, we implement K nearest neighbors classification algorithm with k =10. Meaning, for each of the initial 10,000 test set, we look for its most k nearest neighbors from the chosen sample and assign the most frequent label of the neighbors.

Finally, we compare the accuracy of our two-phase classification algorithm with the actual test labels and report the results in Figure 1. Time consumption is also reported in Figure 2 in order of seconds. In Figure 1, the accuracy of different pairs of (number of clusters, samples per cluster) is reported. The reported bars are minimum, average and maximum of 10 iterations. As depicted, when choosing 10,000 samples from training, the proposed method is capable of reaching an accuracy of above 94 percent.

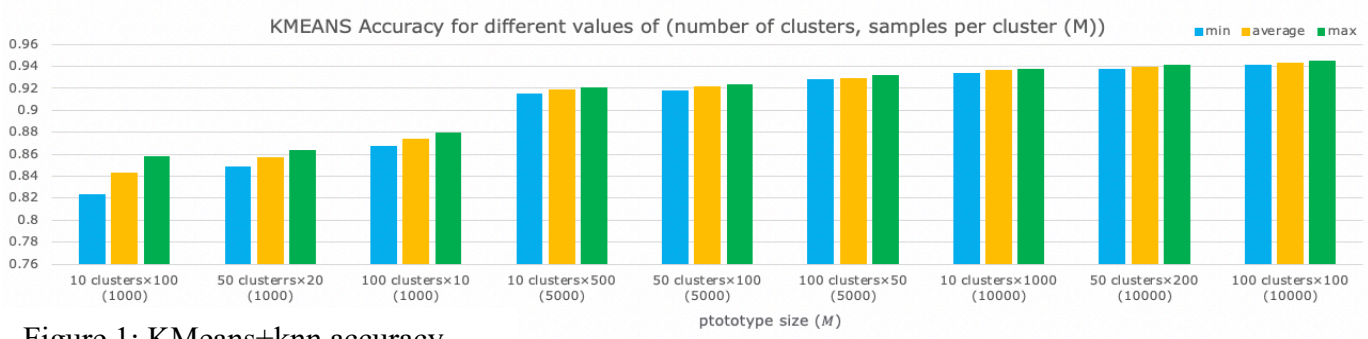

Figure 1: KMeans+knn accuracy

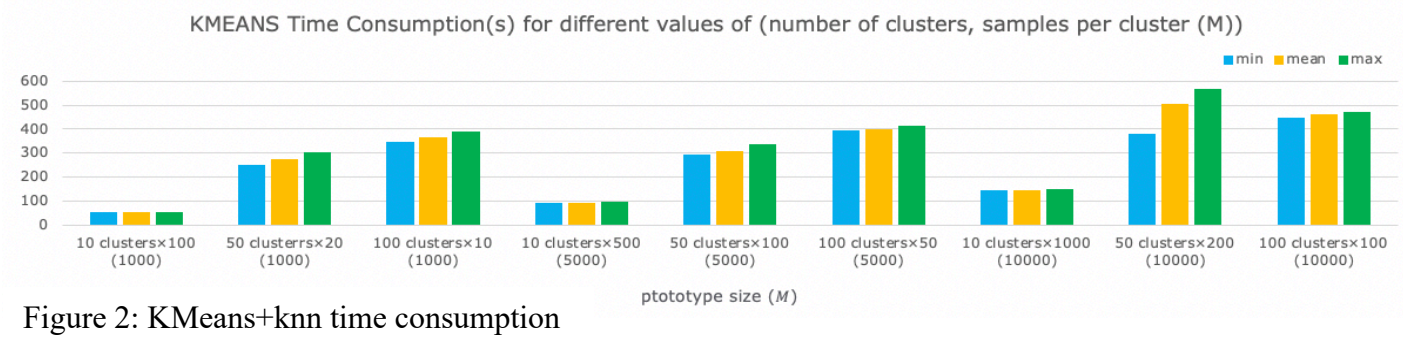

Figure 2: KMeans+knn time consumption

I. Proposed approaches: HD Clustering +KNN

Hyperdimensional (HD) computing is a set of brain-inspired methods for obtaining high-dimensional, low-precision, distributed representations of data [5][6]. Hyperdimensional Computing maps each of the data points into vectors with thousands of dimensions (usually between 1k to 10k), called hypervectors and then does all the training and querying in a high dimensional space called hyperspace.

The initial step in this algorithm is encoding in which a high dimensional vector is assigned to each training data. The encoding approach used in this study is matrix vector multiplication.

For each data point x_i, an empty high dimensional vector v_i with length $D = 4000$ is considered. Wherein, v_{ij} is the dot product of x_{ij} with $base\_matrix_j$. Each base matrix is a $D \times len(v_i)$ arbitrary square matrix.

The idea is based on the fact that if a set of points are well clustered in a low-dimensional space and can be separated by a small margin as depicted in figure 5, but cannot be linearly separated, using HD we can map them into a high dimensional vector in which they can be linearly separable.

Training phase: In this step, all of the training data with label j $where\ j \in (0, 1, 2, \dots 9)$ are considered as a single cluster. Hence, creating 10 clusters, each of which containing roughly 6000 samples of each digit. Then, element-wise summation of all elements of a $cluster\ i$ is considered as the $Centroid_{\ i}\ (i = 0, \dots, 10)$ which represents the whole cluster.

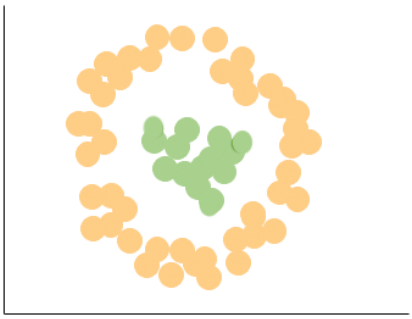

Figure 5: non-linearly separable data in 2D

Sample selection phase: Within each cluster j $(j= 0, 1, 2 \dots, 9)$, the score of each element is calculated. This value determines how similar an element is to centroid of the cluster which encompasses it. Then all elements scores are sorted. Score of each element e in a cluster j is calculated as following:

$$score(e) = \frac{< encoded(e), centroid\ j >}{||centroid\ j||}$$

Finally, $\frac{M}{10}$ samples are chosen from each cluster uniformly starting from index 0 to i in each cluster with a step size of $\frac{len(cluster\ j)}{M}$and constitute the whole sample data.

Ultimately, conventional KNN algorithm is implemented on the selected sample data as described in KMeans part.

II. Experimental results and comparison with random sampling

For all experiments of HD, random prototyping and Kmeans clustering algorithms, minimum, average and maximum of the accuracies and consumed time are reported in figures 3 and 4. For all experiments of all algorithms, standard deviation is roughly in order of 10^{-5}, hence couldn't be depicted in the following figures. For random prototyping experiments for each sample size $M\ (1000, 5000\ and\ 1000)$ is repeated 30 times. For the remaining two algorithms, results are based on ten experiments.

Kmeans beats the random prototype selection with an average accuracy of 0.9 percent for 1000 prototypes and yields similar accuracy to random selection for 5000 and 10000 samples though consuming considerably higher time. However HD algorithm surpasses random selection with 1.8, 0.7 and 0.4 percentages for sample sizes of 1000, 5000 and 10000 respectively while consuming almost similar times. However, HD requires an encoding step which takes between 125 to 328 seconds for M=1000, 5000 and 1000 but is don just once. Hence, the reported time in the following figures for HD is query time.

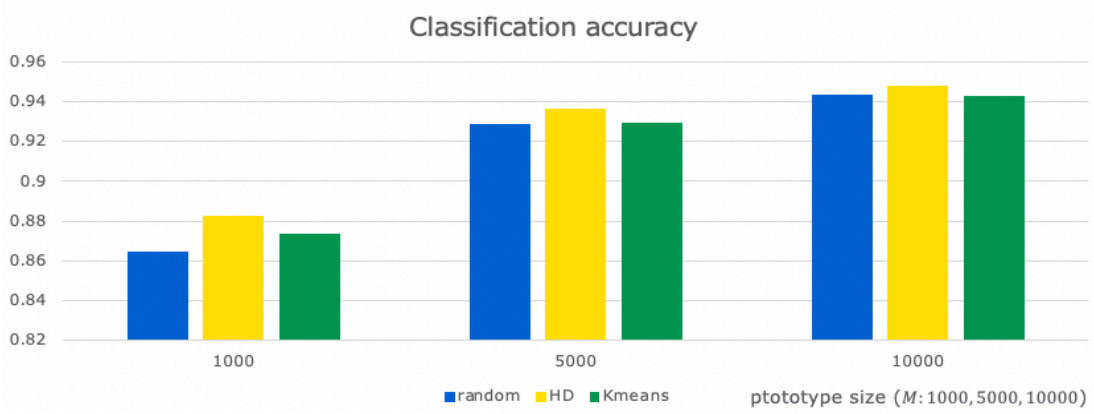

Figure 3: accuracy comparison

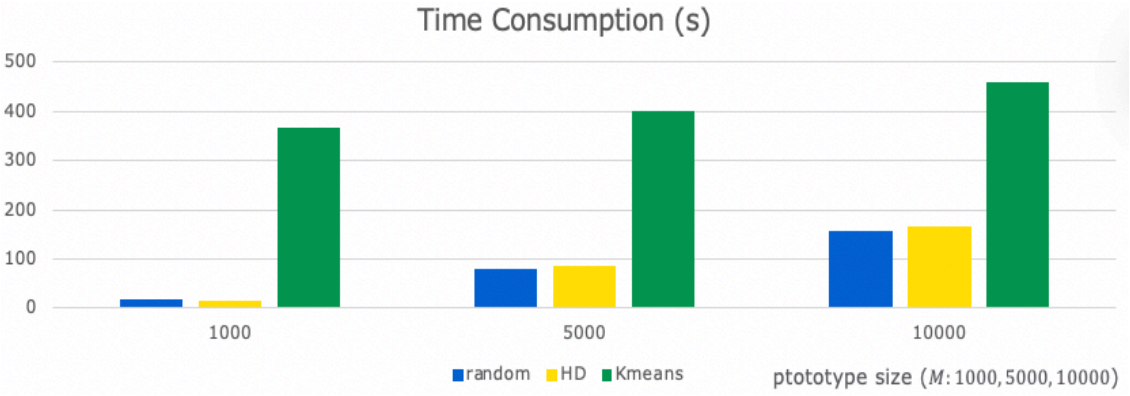

Figure 4: time consumption comparison (disregarding HD one time encoding)

III. Conclusion

Comparison of the two proposed methods with random sample selection shows that proposed HD algorithm beats the random prototype selection with an average accuracy of 1 percent for different prototype sizes (1k, 5k and 10k). Also Kmeans returns a similar accuracy for sample sizes of 5k and 10k though yielding 0.9 percent better accuracy for M=1k.

KMeans clustering algorithm groups homogeneous or similar datapoints into one cluster. Then, by selecting $\frac{M}{K}$ from each cluster, we can make sure that the subset of training data which we aim to choose consists of all different discrete samples. Thereby, the subset which is chosen with this algorithm, can be a better representor of the general training data than random as random selection might not be able to choose data from different dense regions. HD clustering also validates roughly similar frequency of each labels. The step of sample selection from HD also certifies choosing all the variants of a single digit class which eventually helps in better classification of data.

In general, with similar time consumption, HD beats random selection by 1 percent. The initial encoding time of HD is still lower than training on the whole data.

IV. Future work and further scope of improvement

It is known that the KMeans clustering algorithm is sensitive to initial starting centroids [6]. Thereby, it has triggered the study towards seeking for algorithms with refined seeding step like KMEANS++ in which centroids are gradually initialized instead of at once and as reported by literature [7], it can improve both the speed and

accuracy of KMeans. Hence, in a future study, we can substitute the augmented algorithm of KMeans (KMeans++).

Also, due to the shortness of time, I could not tune dimension Hyper-parameter of HD. It is probable that with increasing the dimension of hyper-vectors, the quality of data storing and ultimately clusters be improved. Also, in a future study, one can implement other HD methods to seek for a potential improvement.

Furthermore, as previously pointed out, for a future study, we can have a better convergence criteria for Kmeans clustering iteration like a higher maximum iterations or other convergence methods.

V. References

1. Prototype Methods slides, Henrik Christensen, Georgia Institute of Technology
2. Prototype methods and nearest neighbors, University of Auckland
3. Prototype selection for nearest neighbor classification: Taxonomy and empirical study, Garcia et al, 2012
4. MNIST Dataset: http://yann.lecun.com/exdb/mnist/index.html
5. Theorical Foundations of Hyperdimensional Computing, Thomas et al, 2020
6. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors, Kanerva, 2009
6. Refining Initial Points of K-Means Clustering, Bradley and Fayyad, 1998
7. K-means++: The Advantages of Careful Seedding, Arthur and Vassilvitskii, 2006

pseudo code! of KMeans+knn

(Initialization) Initially selecting K data points as centroids
(Iteration until convergence)
- For each training data find the nearest centroid
- Re-estimate the center for each cluster

(Sample selection) from elements of each cluster i (i=0, …k-1), choose $\frac{M}{K}$ samples randomly as the prototype data

(**Implementing KNN on the selected sample size**) storing the indices of the selected samples and implementing the conventional KNN (with k=10) on the selected data. (More can be found above) (Further explanation can be found above)

pseudo code! of HD

(**Encoding**) Assigning High dimensional vectors to each training data based on pixel values using matrix vector multiplication method (D=4000)
(**Training**) Element-wise summation of all training data with label (i) and creating a D-dimensional vector for each cluster i ($i = 0, 1, 2 \dots, 9$) and setting this value as the center of each cluster
(**Sample selection**) for elements of each cluster j (j= $0, 1, 2 \dots, 9$), calculating the score of its elements where score is the normalized dot product of each element's vector with the center of the cluster. Then selecting roughly $\frac{M}{10}$ samples from each uniformly with a step size on indices (from 0 to len(cluster i))
(**Implementing KNN on the selected sample size**) storing the indices of the selected samples and implementing the conventional KNN (with k=10) on the selected data. (Further explanation can be found above, Also, the code can be sent if needed)

code of KMeans+knn

```
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import os
import sys
import gzip #for zipping and unzipping files
import time
if sys.version_info[0] == 2:
    from urllib import urlretrieve
else:
    from urllib.request import urlretrieve

n_clusters = int(sys.argv[1])
elements = int(sys.argv[2])
iterations = int(sys.argv[3])

t = []
acc = []
def download(filename, source='http://yann.lecun.com/exdb/mnist/'):
    print("Downloading %s" % filename)
    urlretrieve(source + filename, filename)

def load_mnist_images(filename):
    if not os.path.exists(filename):
        download(filename)
    # Read the inputs in Yann LeCun's binary format.
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
    data = data.reshape(-1,784)
    return data / np.float32(256)

def load_mnist_labels(filename):
    if not os.path.exists(filename):
        download(filename)
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=8)
        #data2 = np.zeros( (len(data),10), dtype=np.float32 )
        #for i in range(len(data)):
        #    data2[i][ data[i] ] = 1.0
    return data

## Load the training set
TrainX = load_mnist_images('train-images-idx3-ubyte.gz')
TrainY = load_mnist_labels('train-labels-idx1-ubyte.gz')

## Load the testing set
TestX = load_mnist_images('t10k-images-idx3-ubyte.gz')
TestY = load_mnist_labels('t10k-labels-idx1-ubyte.gz')
```

```
print("start")
for i in range(1,iterations+1):
    print(i*10, " percent")
    s = time.time()
    X = TrainX
    kmeans = KMeans(n_clusters=n_clusters).fit(X)

    lbls = []
    for p in range(n_clusters):
        lbls.append([i for i, x in enumerate(list(kmeans.labels_)) if x == p])

    chosen = []
    for i in range(n_clusters):
        chosen.append(np.random.choice(lbls[i], elements, replace=False))

    chosen = [item for sublist in chosen for item in sublist]

    X = TrainX[chosen]
    y = TrainY[chosen]
    from sklearn.neighbors import KNeighborsClassifier
    neigh = KNeighborsClassifier(n_neighbors=10)
    neigh.fit(X, y)
    pred = neigh.predict(TestX)

    t.append(time.time()-s)
    acc.append(accuracy_score(pred, TestY))
print(min(t), np.mean(t), max(t))
print(min(acc), np.mean(acc), max(acc), np.var(acc))
print("acc", acc)
```