Programming Project 1
Prototype selection for nearest neighbor

Rajasvi Vinayak Sharma
(PID: A59012988)
University of California - San Diego
rvsharma@ucsd.edu

January 24th 2022

1 Problem

Nearest neighbor classification algorithm is slow for large datasets since every time it makes a prediction it compares every single point in testing set with all the training data points. Therefore in order to speed up the Nearest neighbor classification one of the strategy is to carefully select subset of "prototypes" from training set which can give good classification performance. In this project I explore strategies for prototype-selection algorithm which are used by 1-NN model over MNIST data, bearing in mind that the ultimate goal is good classification performance on test data.

2 Proposed Approach

I experimented with primarily 2 prototype selection algorithms in this project. First one is utilizing K-Means clustering where I use K-Means [Steinley(2006)] on training data per class to find centroids of clusters (representative of each class) which are used as prototype for training set. For instance, if I generate 5 clusters per class label, I will have 10 centroids (MNIST data has 10 classes; 5 centroids * 10 classes) and in total 50 points which are used as a prototype set to classify test data using 1-NN model.
Secondly, I have used an hybrid absorption based approach where I incrementally increase the prototype set by comparing any new point from training set with existing the prototype set (initialized with K-Means centroid prototype set). I find the label of closest point in the existing prototype set and if the class label from closest prototype is not equal to true label of the new point, I add it to the prototype set. This approach is based on the logic that the newly added point (which was misclassified by existing prototype set) represents points not covered by existing prototype set. As a starting point for the prototype set I use K-Means centroids generated from 1st approach which fastens the process. This approach is a variation of Fast Condensing Nearest Neighbors [Angiulli(2005)] with additional early stopping mechanism as per the required prototype set size M.

3 Pseudocode

Algorithm 1. Refers to K-Means clustering based prototype selection algorithm.
Algorithm 2. Refers to Absorption based approach utilising K-Means prototypes as starting seed.

4 Experimental results

Baseline: Random samples of different sizes (M = 100,200,1000,2000,5000,10000) are selected from the original dataset containing 60,0000 points and treated as prototype for 1-NN model. For each M random samples the 1-NN model is trained tested upon 10 times to obtain average accuracy scores to be used as a baseline score for other models.

K-Mean Clustering Centroids method: In this method, we generated prototypes of sizes M (M = 100,200,1000,2000,5000,10000),where each class had $M/NumOfClassess$ representative prototype points. I ran the model 10 times for each M and fond the average accuracy for each case on testing set using Kmeans-prototype as training for 1-NN model experimentation. It was clearly visible that K-Means centroid based prototypes performed significantly better than Randomly sampled

Algorithm 1 K-Means Clustering (Per Class Label) Centroids Prototype method

Input: Training dataset and size of required prototype set.
Output: Prototype Set (True Label, Selected Data Points)

procedure KMEANSCENTROIDSPROTOTYPE($TrainSet$, clusters = M)
    $C \leftarrow M/NumOfClasses$ ▷ Get number of clusters per class label
    $P \leftarrow \emptyset$ ▷ Empty prototype set
    for $i \leftarrow 0$ to 9 **do**
        $Model \leftarrow sklearn.kmeans(clusters = c).fit(TrainSet)$
        $Centroids_i \leftarrow Model.getClusterCenters$
        $P.insert(i, Centroids_i)$ ▷ Insert centroid with tag i as true label
    end for
    return P

Algorithm 2 Absorption based Nearest Neighbor Prototype method (FCNN Variation)

Input: Training dataset T and size of required prototype set.
Output: Prototype Set (True Label, Selected Data Points)

procedure ABSORPTION_NN($TrainSet$, clusters = M)
    $P \leftarrow \emptyset$
    $\Delta P \leftarrow KMeansCentroidsPrototype(TrainSet, M * 0.5)$ ▷ seed k-means prototype: 50%
    while $\Delta P \neq \emptyset$ **do**
        $P \leftarrow P \cup \Delta P$
        $\Delta P = \emptyset$
        for each $q \in (T - P)$ **do**
            $p = closestNearestNeigbor[q]$ such that $p \in P$ ▷ closest point to q from store P
            if $label[p] \neq label[q]$ **then**
                $\Delta P = \Delta P \cup q$
            end if
            if $size[P] + size[\Delta P] \geq M$ **then**
                $break$;
            end if
        end for
    end while
    return P

prototypes of equal size achieving an average accuracy of 96.9% for $M = 10000$. Even for as low $M = 100$ it reached 92% accuracy which is more than baseline which barely reaches 68.4% avg. accuracy.

Absorption based Nearest Neighbor method: In this method, I started with $Store$ (to be prototype set), with seed values as K-Mean cluster centroids for half the required size of total prototype points. Now for the remaining $M/2$ points, I used Algortithm 2 where I calculated distance of every point q in store from Training set to find the closest point p to the point q in store. Now if p doesn't have the same label as q then I absorb the p point into the $Store$.This process is repeated iteratively until no point is left which can be included in $Store$ prototype set or the size of $Store$ becomes equal to M(= 100,200,1000,2000,5000,10000). I ran the experiment for 10 epochs each for every value of required size M. For instance, if $M == 2000$, K-Mean centroid comprised of 1000 points and remaining 1000 were populated in store using absorption nearest neighbor method. This hybrid method is able to achieve accuracy score close to K-Mean clustering centroids method.

Distance metrics used for nearest neighbor method was Euclidean Distance or L2-norm distance which has following equation:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{1}$$

For each strategy the experiment is run for 10 epochs and average accuracy is calculated on test set. The confidence interval is calculated by the following formula:

$$Accuracy_{lower} = \overline{Accuracy} - z * \frac{\sigma}{\sqrt{n}} \tag{2}$$

$$Accuracy_{upper} = \overline{Accuracy} + z * \frac{\sigma}{\sqrt{n}} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

Overall average accuracy comparison details can be seen in graph 1 and the table 1. It can be clearly seen that proposed methods i.e. K-Means clustering centroids and Absorption nearest neighbor methods perform substantially better.

5 Critical Evaluation

From experiments, it can be seen that for larger datasets like MNIST which has around 60k datapoint, its better to use prototypes for 1-NN models instead of using entire training set while making predictions. Using prototypes is better performance wise as well as space wise since it's giving better accuracy even for same size randomly sampled training set. Furthermore, in both proposed methods it can be easily seen how they are better than random model which barely achieve 94% avg. accuracy for $M = 10000$ whereas both models reach around 96% average accuracy. Furthermore, while experimenting with hybrid model like proposed absorption nearest neighbor model, I got to understand how combining techniques can be used to further reduce training time as well as prototype size. It has been developed by modifying other methods I came across like Fast Condensing Nearest neighbor (FCNN) where cluster size was expanded without constraints.
In future, I wish to explore other interesting prototype selection techniques like variations of Condensing Nearest Neighbors [Hart(1968)], Learning Vector Quantization (LVQ)[Kohonen(1995)], and Gaussian Mixtures based prototype models.

References

[Angiulli(2005)] Fabrizio Angiulli. 2005. Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, pages 25–32.

[Hart(1968)] Peter Hart. 1968. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516.

[Kohonen(1995)] Teuvo Kohonen. 1995. Learning vector quantization. In *Self-organizing maps*, pages 175–189. Springer.

[Steinley(2006)] Douglas Steinley. 2006. K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34.

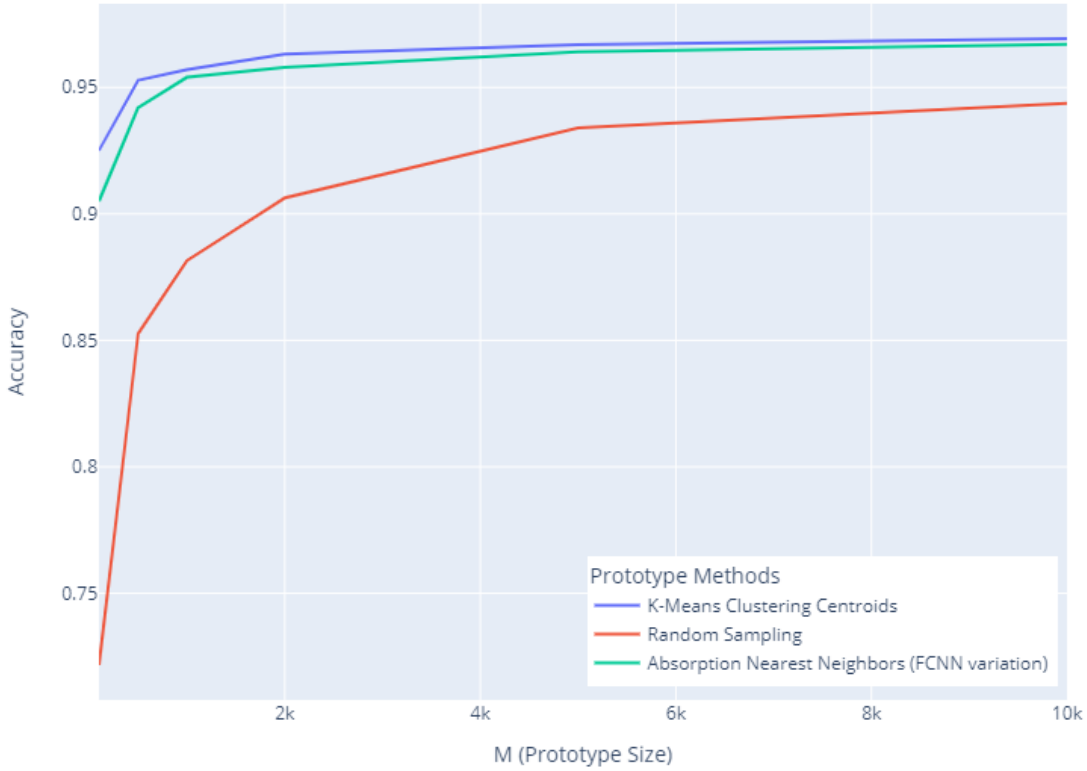

Figure 1: Average Accuracy comparison plot

Methods	**Prototype size (M)**	**1-NN model accuracy**
1. Baseline model(random sampling)	100	0.684
2. Baseline model(random sampling)	500	0.849
3. Baseline model(random sampling)	1000	0.880
4. Baseline model(random sampling)	2000	0.910
5. Baseline model(random sampling)	5000	0.937
6. Baseline model(random sampling)	10000	0.946
1. K-Means Clustering Centroids	100	0.925
2. K-Means Clustering Centroids	500	0.952
3. K-Means Clustering Centroids	1000	0.957
4. K-Means Clustering Centroids	2000	0.963
5. K-Means Clustering Centroids	5000	0.966
6. K-Means Clustering Centroids	10000	0.969
1. Absorption Nearest Neighbor	100	0.905
2. Absorption Nearest Neighbor	500	0.942
3. Absorption Nearest Neighbor	1000	0.954
4. Absorption Nearest Neighbor	2000	0.958
5. Absorption Nearest Neighbor	5000	0.964
6. Absorption Nearest Neighbor	10000	0.967

Table 1: Comparison between average accuracy values for different proposed methods.

1 Mini Project 1: Prototype selection for nearest neighbor

by Rajasvi Vinayak Sharma (PID: A59012988)

[28]:
```
import numpy as np
import requests, gzip, os, hashlib
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
import plotly.express as px
import pandas as pd
import random
```

[29]:
```
# Download MNIST
path='data'
def fetch(url):
    fp = os.path.join(path, hashlib.md5(url.encode('utf-8')).hexdigest())
    if os.path.isfile(fp):
        with open(fp, "rb") as f:
            data = f.read()
    else:
        with open(fp, "wb") as f:
            data = requests.get(url).content
            f.write(data)
    return np.frombuffer(gzip.decompress(data), dtype=np.uint8).copy()

X = fetch("http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz")[0x10:].
 ↪reshape((-1, 28*28))
Y = fetch("http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz")[8:]
X_test = fetch("http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.
 ↪gz")[0x10:].reshape((-1, 28*28))
y_test = fetch("http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz")[8:]
```

[30]:
```
print(f"Training set size {X.shape}")
print(f"Test set size {X_test.shape}")
```

```
Training set size (60000, 784)
Test set size (10000, 784)
```

[27]:
```
# Prototype selection sizes
M=[100,500,1000,2000,5000,10000]
```

[5]:
```
# Baseline: Random Sampling
rand_scores=[]
for _ in M:
    idxs = random.choices(range(len(X)),k=_)
    X_train=X[idxs]
    y_train=Y[idxs]
    knn_model = KNeighborsClassifier(n_neighbors=1)
    knn_model.fit(X_train,y_train)
```

```
    rand_scores.append(knn_model.score(X_test,y_test))

df= pd.DataFrame.from_dict({'rand_scores':rand_scores})
px.line(df,x=M,y='rand_scores')
```

[6]:
```
# K-Means Clustering Centroid based Prototype Selection Method
kmeans_scores=[]
numClasses = len(np.unique(Y))

cnt=0
for sizee in M:
    kmeansCenters = []
    print(f"Creating prototypes based on cluster size {sizee//numClasses} per␣
 ↪label")
    for i in range(numClasses):
        # print(f"Creating prototypes for label {i}")
        x_i = X[np.where(Y==i)]
        kmeans_i = KMeans(n_clusters=sizee//numClasses,␣
 ↪init='k-means++',random_state=0).fit(x_i)
        kmeansCenters.append(kmeans_i.cluster_centers_)

    # print(f"Created K-means center based prototype for each label.")
    prototype_x = np.array(kmeansCenters)
    prototype_y = np.repeat(np.array(range(numClasses)),sizee//numClasses)
    knn_model = KNeighborsClassifier(n_neighbors=1)
    knn_model.fit(np.concatenate(prototype_x),prototype_y)
    kmeans_scores.append(knn_model.score(X_test,y_test))

    print(f" Prototype based 1-NN Score:{kmeans_scores[cnt]}")
    cnt+=1
```

```
Creating prototypes based on cluster size 10 per label
 Prototype based 1-NN Score:0.9251
Creating prototypes based on cluster size 50 per label
 Prototype based 1-NN Score:0.9528
Creating prototypes based on cluster size 100 per label
 Prototype based 1-NN Score:0.957
Creating prototypes based on cluster size 200 per label
 Prototype based 1-NN Score:0.9631
Creating prototypes based on cluster size 500 per label
 Prototype based 1-NN Score:0.9669
Creating prototypes based on cluster size 1000 per label
 Prototype based 1-NN Score:0.9693
```

[ ]:
```
# Absorption Nearest Neighbor/modified FCNN based Prototype Selection Method
absNN_scores=[]
numClasses = len(np.unique(Y))
```

```
cnt=0
for sizee in [100,500,1000,2000,5000,10000]:
    kMeansClusters = sizee//2
    kmeansCenters = []
    # Finding K-Means Cluster Centroids
    print(f"Creating prototypes based on cluster size {kMeansClusters//
 ↪numClasses} per label")
    for i in range(numClasses):
        x_i = X[np.where(Y==i)]
        kmeans_i = KMeans(n_clusters=kMeansClusters//numClasses,␣
 ↪init='k-means++',random_state=0).fit(x_i)
        kmeansCenters.append(kmeans_i.cluster_centers_)

    # Absorption based CNN - FCNN Variation
    prototype_x = np.array(kmeansCenters)
    prototype_y = np.repeat(np.array(range(numClasses)),kMeansClusters//
 ↪numClasses)

    store=list(zip(np.concatenate(prototype_x),prototype_y))
    prev_grabbag=[ (X[i],Y[i]) for i in range(len(X))]

    for epoch in range(10):
        print(f"\nEpoch: {epoch}")
        print(f"*Size of Grabbag: {len(prev_grabbag)}")

        grabbag=[]
        cnt=1
        diff=0

        for x,y in prev_grabbag:
            closestLabel = -1
            closestDist = float("inf")

            xt = np.array([i[0] for i in store])
            closestLabel = store[np.argmin(np.linalg.norm(x-xt,axis=1))][1]

            if closestLabel==y:
                grabbag.append((x,y))
            else:
                store.append((x,y))
                diff+=1

            # Keep populating store unless it reaches required size
            if len(store)>=sizee:
                diff=0
                break
```

```
            if cnt%1000==0:
                print(f"**Completed Samples: {cnt}")
                print(f"**Size of store: {len(store)}")

            cnt+=1

        if diff==0:
            print(f"*Stopping at epoch: {epoch}")
            break

        prev_grabbag=grabbag

    xProto = [i[0] for i in store]
    yProto = [i[1] for i in store]

    knn_model = KNeighborsClassifier(n_neighbors=1)
    knn_model.fit(xProto,yProto)
    absNN_scores.append(knn_model.score(X_test,y_test))
```

[23]:
```
# Accuracy comparision plots for M={100,500,1000,2000,5000,10000}
df= pd.DataFrame.from_dict({'M (Prototype Size)':M,'Random Sampling':
 ↪rand_scores,'K-Means Clustering Centroids':kmeans_scores,"Absorption Nearest␣
 ↪Neighbors (FCNN variation)":absNN_scores})
df=pd.melt(df,id_vars=['M (Prototype Size)'],value_vars=['K-Means Clustering␣
 ↪Centroids','Random Sampling','Absorption Nearest Neighbors (FCNN variation)'])
df=df.rename(columns={"value":'Accuracy',"variable":"Prototype Methods"})
fig=px.line(df,x='M (Prototype Size)',y='Accuracy',color='Prototype␣
 ↪Methods',width=800,height=600)
fig.update_layout(legend=dict(
    yanchor="bottom",
    y=0.02,
    xanchor="right",
    x=0.99
))
fig.show()
```