# K-Means Based Prototype Selection for Nearest Neighbor Classifier

**Sijie Wang**
siw019@ucsd.edu

## Abstract

Speeding up nearest neighbor classification can be accomplished by replacing the training set by a carefully chosen subset of ”prototypes”. A K-Means based algorithm can be used to choose prototypes from the training set. The prototypes chosen was used for 1-NN based classification and the algorithm was implemented and tested on the MNIST data set.

## 1 High Level Description

KMeans algorithm can partition the dataset into K pre-defined distinct non-overlapping clusters where each data point belongs to only one group. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid is at the the minimum. The less variation we have within clusters, the more similar the data points are within the same cluster. Thus, using KMeans algorithm can improve the performance.

## 2 Pseudocode

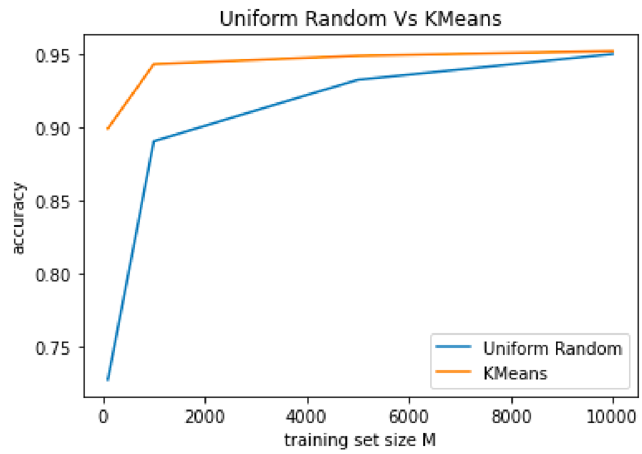
**Algorithm 1** KMeans Algorithm

Specifiy the number n of clusters to assign
Random initialize n centroids
Repeat:
Assign each data point to the closest centroid
Then, compute and update the mean of each cluster
End until the centroid is fixed
Use above algorithm to fit the model with samplings, predict the labels by the model and compute the accuracy

## 3 Experimental Results

Cases are considered M = 100, 1000, 5000, 10000 1-NN classification was performed and 10 clusters was selected for KMeans algorithm.

| M | Uniform Random Accuracy | KMeans Accuracy |
|---|---|---|
| 100 | 0.7159 | 0.8992 |
| 1000 | 0.8865 | 0.9432 |
| 5000 | 0.9343 | 0.9489 |
| 10000 | 0.9478 | 0.9521 |




## 4 Critical Evaluation

Based on above table and graph, clearly that accuracy after implementing KMeans is higher than Uniform Random. KMeans algorithm can capture the properties of the dataset accurately.
Further improvement: For further improving accuracy, we can try different number of clusters. In this project, I only choose 10 clusters, we may find the best accuracy in a different number of cluster.

In [1]:
```
import operator
import struct
import numpy as np
import datetime
import matplotlib.pyplot as plt
```

In [2]:
```
train_images_idx3_ubyte_file = './data/train-images-idx3-ubyte'
train_labels_idx1_ubyte_file = './data/train-labels-idx1-ubyte'
test_images_idx3_ubyte_file = './data/t10k-images-idx3-ubyte'
test_labels_idx1_ubyte_file = './data/t10k-labels-idx1-ubyte'
```

In [3]:
```
def load_idx3_ubyte(idx3_ubyte_file):
    bin_data = open(idx3_ubyte_file, 'rb').read()
    offset = 0
    fmt_header = '>iiii'
    count_number, num_images, num_rows, num_cols = struct.unpack_from(fmt_h

    image_size = num_rows * num_cols
    offset += struct.calcsize(fmt_header)
    fmt_image = '>' + str(image_size) + 'B'
    images = np.empty((num_images, num_rows, num_cols))
    for i in range(num_images):
        images[i] = np.array(struct.unpack_from(fmt_image, bin_data, offset
        offset += struct.calcsize(fmt_image)
    return images
```

In [4]:
```
def load_idx1_ubyte(idx1_ubyte_file):
    bin_data = open(idx1_ubyte_file, 'rb').read()

    offset = 0
    fmt_header = '>ii'
    count_number, num_images = struct.unpack_from(fmt_header, bin_data, off

    offset += struct.calcsize(fmt_header)
    fmt_image = '>B'
    labels = np.empty(num_images)
    for i in range(num_images):
        labels[i] = struct.unpack_from(fmt_image, bin_data, offset)[0]
        offset += struct.calcsize(fmt_image)
    return labels
```

In [5]:
```
X_train = load_idx3_ubyte(train_images_idx3_ubyte_file)
Y_train = load_idx1_ubyte(train_labels_idx1_ubyte_file)
X_test = load_idx3_ubyte(test_images_idx3_ubyte_file)
Y_test = load_idx1_ubyte(test_labels_idx1_ubyte_file)
```

In [6]:
```
print(len(X_train))
print(len(Y_train))
print(len(X_test))
print(len(Y_test))
```

```
60000
60000
10000
10000
```

In [7]:
```
from sklearn import metrics
```

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [9]:
```
def get_rand_accuracy(X_train, Y_train, M):
    random_indices = np.random.choice(X_train.shape[0], replace=False, size
    X_random = X_train[random_indices]
    Y_random = Y_train[random_indices]
    knn_rand = KNeighborsClassifier(n_neighbors = 1)
    X_random = X_random.reshape(len(X_random),-1)
    Y_random = Y_random.reshape(len(Y_random),-1)
    knn_rand.fit(X_random, Y_random)
    X_test = X_test.reshape(len(X_test),-1)
    y_randpred = knn_rand.predict(X_test)
    return metrics.accuracy_score(Y_test, y_randpred)
```

In [10]:
```
M = [100, 1000, 5000, 10000]
acc_list = []
for m in M:
    acc_list.append(get_rand_accuracy(X_train, Y_train, m))
print(acc_list)
```

```
<ipython-input-9-feaff7c0179b>:8: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y t
o (n_samples, ), for example using ravel().
  knn_rand.fit(X_random, Y_random)
<ipython-input-9-feaff7c0179b>:8: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y t
o (n_samples, ), for example using ravel().
  knn_rand.fit(X_random, Y_random)
<ipython-input-9-feaff7c0179b>:8: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y t
o (n_samples, ), for example using ravel().
  knn_rand.fit(X_random, Y_random)
<ipython-input-9-feaff7c0179b>:8: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y t
o (n_samples, ), for example using ravel().
  knn_rand.fit(X_random, Y_random)

[0.7159, 0.8865, 0.9343, 0.9478]
```

In [11]:
```
import numpy as np
from os.path import join
import struct as st
import matplotlib.pyplot as plt
import copy
```

In [12]:
```
class KMeans:
    def __init__(self, X_train, Y_train, n, M):
        self.x_train = np.array(X_train)
        self.y_train = np.array(Y_train)
        self.n = n
        self.M = M


    def create_centroids(self):
        np.random.seed(np.random.randint(0, M))
        self.centroids = []
        for i in range(self.n):
            self.centroids.append(self.fit_data[np.random.choice(range(len(

    def update_centroids(self):
        for i in range(self.n):
            cluster = self.clusters['data'][i]
            if cluster != []:
                self.centroids[i] = np.mean(np.vstack((self.centroids[i], c
            else:
                self.centroids[i] = self.fit_data[np.random.choice(range(le

    def fit(self, fit_data, fit_labels):
        self.fit_data = fit_data
        self.fit_labels = fit_labels
        self.pred_labels = [None for _ in range(self.fit_data.shape[0])]
        self.create_centroids()

        prev_centroids = [np.zeros(shape=(fit_data.shape[1],)) for _ in ran
        prev_centroids = copy.deepcopy(self.centroids)

        self.clusters = {'data': {i: [] for i in range(self.n)}}
        self.clusters['labels'] = {i: [] for i in range(self.n)}

        for i, cen in enumerate(self.centroids):
            dist = np.linalg.norm(cen)
            if dist < min_dist:
                min_dist = dist
                self.pred_labels[j] = i

        if self.pred_labels[j] is not None:
            self.clusters['data'][self.pred_labels[j]].append(sample)
            self.clusters['labels'][self.pred_labels[j]].append(self.fit_la

        for id, matrix in list(self.clusters['data'].items()):
            self.clusters['data'][id] = np.array(matrix)

        self.update_centroids()

        #calculate the accuracy
        for cluster, labels in list(self.clusters['labels'].items()):
            if isinstance(labels[0], (np.ndarray)):
                labels = [l[0] for l in labels]
            counter = 0
            max_label = max(set(labels), key=labels.count)
            self.clusters_labels.append(max_label)
```

```
            for label in labels:
                if label == max_label:
                    counter += 1
            acc = counter/len(list(labels))
            self.clusters_accuracy.append(acc)
            self.accuracy = sum(self.clusters_accuracy)/self.n
```

In [13]:

```
M = [100, 1000, 5000, 10000]
KM_acc_list = []
for m in M:
    random_indices = np.random.choice(X_train.shape[0], replace=False, size
    X_random = X_train[random_indices]
    Y_random = Y_train[random_indices]
    kmeans = KMeans(X_random, Y_random, n = 10, M = m)
    kmeans.fit(X_random,Y_random)
    KM_acc_list.append(kmeans.accuracy)
print(KM_acc_list)
```

```
[0.8992, 0.9432, 0.9489, 0.9521]
```

In [14]:

```
x = M
y1 = acc_list
y2 = KM_acc_list
plt.plot(x, y1, label = "Uniform Random")
plt.plot(x, y2, label = "KMeans")
plt.xlabel("training set size M")
plt.ylabel("accuracy")
plt.title('Uniform Random Vs KMeans')
plt.legend()
plt.show()
```