MTH448 project2

Project 2: Once more with k-means

Introduction

For project 2, we will apply one new algorithm from Machine learning named K-Means Clustering algroithm to the MINST database.Hence, before we start this project, we have to know what is K-Means Clustering algroithm.

The goal in this project is to use K-Means algroithm to split a big dataset like MNIST images into different clusters. And through displaying the centroids to find do they have resemble digits or not. Two main properties will be investigated in this report with using centroids improve K-NN efficiency while maintaining reasonable accuracy and making K-NN predictions more faster. In the first portion of the report the main focus will be load the MINST dataset to process the data. Then apply K_means clustering with specific K. Through visualize the centroids to evaluate clustering for classification. In the second half of the report, report wull be conducted about a possible way of improving the speed of K-NN prediction by reducing the amount of the training data.By splitting the MINST images into training data and testing data,cluster training images by digit using K-Means algorithm.

After we know about K-Means Clustering algroithm, here have questions: Why we will use this algorithm and how we will use this algroithm?

K-Means is an unsupervised machine learning algorithm used for clustering data into K. k groups based on feature similarity. For our project, we will use K-Means to slit MINST images. Moreover,the K-Means works very well with large datasets and very easy tio implement.

Getting Started: part 1

In order to be able to use the K-Means algorithm to split the images from MINST dataset into different clusters and be able to use it to display centroids to check do they have resemble digits. At first, the code cells should import MINST.csv to load MINST big dataset.

In [65]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [66]:
```
import csv
```

In [67]:
```
#In this cell, the following code is open and read the csv file MINST_train.csv
File = open('mnist_train.csv')
```

```
Reader = csv.reader(File)
inside = []
for line in Reader:
    inside.append(line)
File.close()
Data = np.array(inside,dtype=int)
```

The code use '.Shape' method to show the MNIST databse contains 60,000 imags of had-written digits,there are 785 dimensional vectors. Moreover, we also can use '.reshape' to reshape MINST dataset. Covertes into 28 x 28 pixel images.

In [68]:
```
Data.shape
```

Out[68]:
```
(60000, 785)
```

We assign the variable name Labs for x, Dms for y.

In [69]:
```
Labs = Data[:,0]
Dms = Data[:,1:]
```

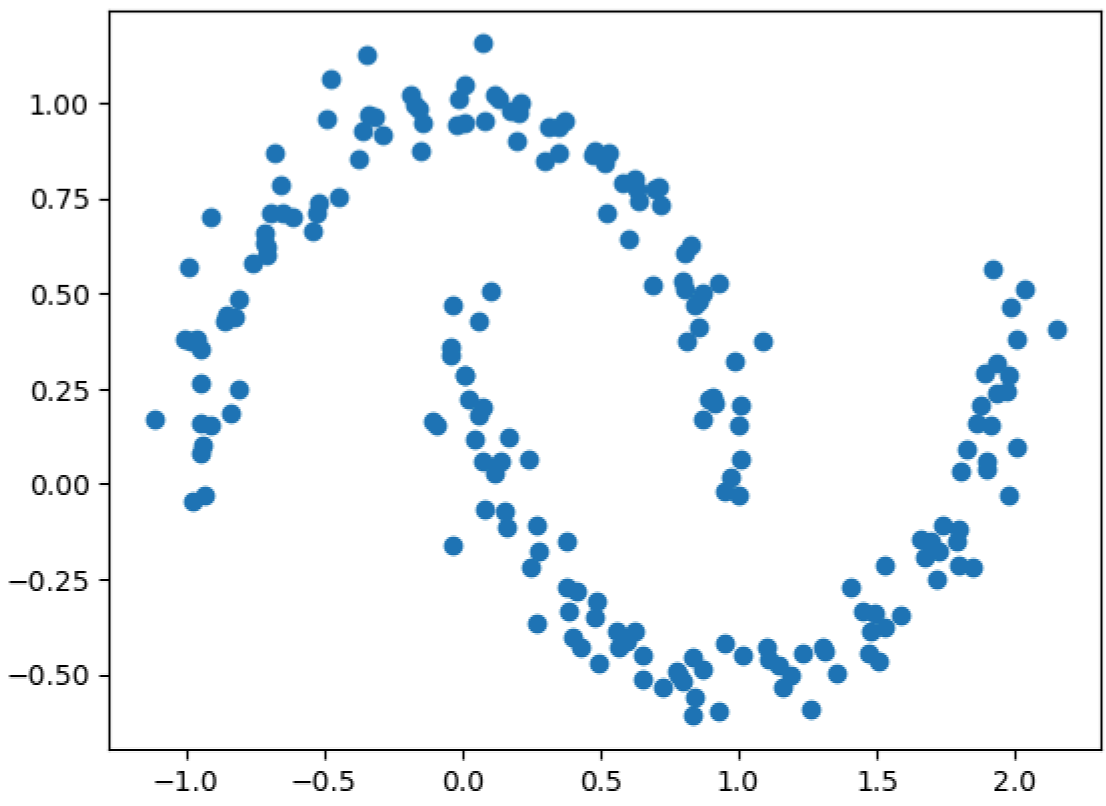
In [70]:
```
Dm = Dms[784,:].reshape(28,28)
Dm
```

Out[70]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  24,
        108, 180, 253,  76,  19,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 252,
        252, 235, 206, 207, 117,   0,   0,  43,  22,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 170, 252,
        170,  44,   0,   0,   0,   0,  68, 246, 199,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 159, 252,
        111,   0,   0,   0,   0,  26, 203, 252, 188,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  34, 252,
        246, 146,   0,   0,  74, 205, 252, 252,  63,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  95,
        247, 253, 201,  34, 212, 253, 234,  21,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         67, 202, 252, 253, 244, 123,  17,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  93, 252, 253, 206,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        100, 224, 252, 253, 206,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  22,
        215, 252,  95, 253, 248,  63,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  13, 212,
        247,  94,   0, 255, 253, 131,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 118, 252,
        110,   0,   0, 253, 252, 183,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  85, 253, 240,
         50,   0,   0, 253, 252, 183,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 157, 245,  79,
          0,   0,   0, 253, 252,  89,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  85, 250, 230,   0,
          0,   0, 106, 253, 210,   6,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  51, 243, 244,  50,
          0,   9, 233, 255, 144,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
```

```
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  93, 252, 243,  50,
         66, 194, 252, 218,  33,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  25, 236, 253, 209,
        234, 252, 252,  84,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 188, 253, 252,
        252, 218,  56,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  63, 253, 178,
        106,  14,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]])
```

In [72]:
```
from sklearn.datasets import make_moons
X,_ = make_moons(n_samples=200,noise=0.07)
plt.scatter(X[:,0],X[:,1])
```

Out[72]: `<matplotlib.collections.PathCollection at 0x11862a790>`



Section2: Project Objectives 2.1 part1:

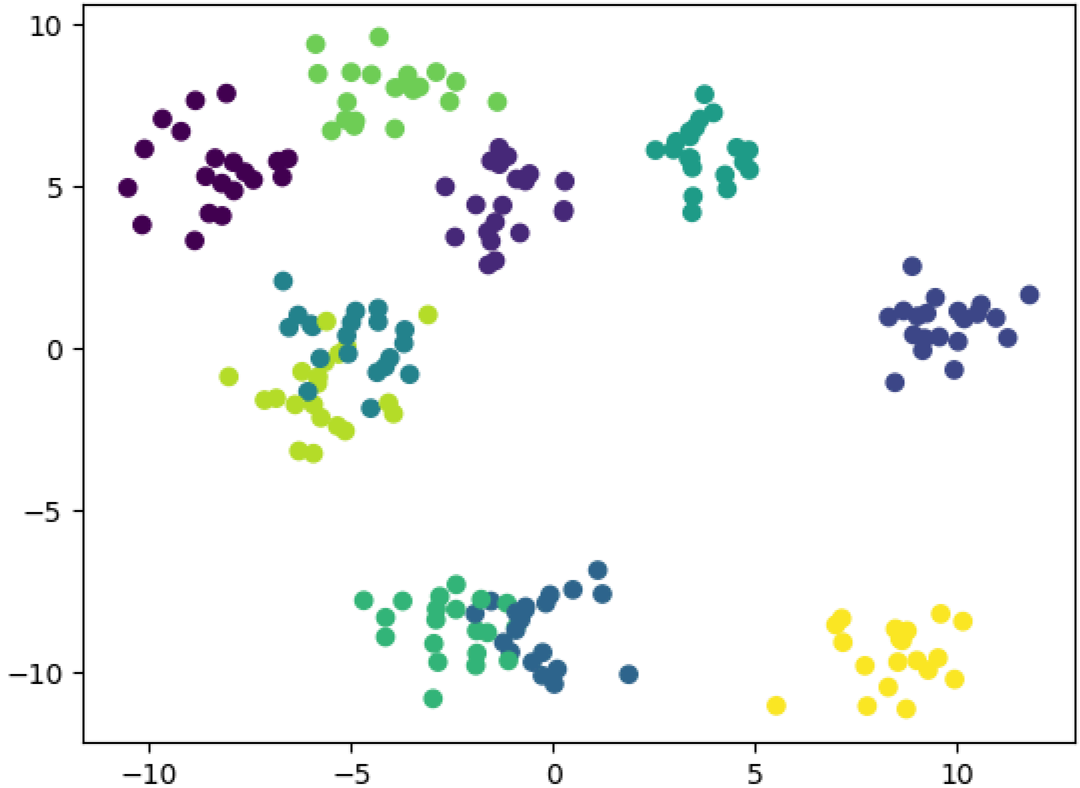
In this section of the report I will apply the k-Means algorithm to split the MINST dataset to cluster the images into 10 clusters. The 10 clusters will display the centroids to enable us to check if they resemble digits or not. By comparing the clusters to the actual labels to evaluate the clustering performance. Thus, we will know how accurately we can predict which image corrsponds to which digit.

In [73]:
```
# If we want to use K-Means algorithm, we have to import it first
from sklearn.cluster import KMeans
import pandas as pd
```

Like we discussed the K-Means clustering algroithm before, the algorithm is used to split MINST dataste into 10 groups and find each group by it's centroid. Here are some steps we need to think about: How we find the clusters in K-Means Algroithm?

- 1) We need to pick k(10 points) of the data set.And they will be initial guess of the centroids.
- 2) Randomly assign a centroid to each of the 10 clusters.
- 3) Calculate the distance of all observations to each of the 10 centroids.
- 4) Assign observations to the closets centroid.
- 5) Through evaluating the mean of each cluster to find the new location of centroids.
- 6) Repeat 3-5 until the centroids do not change position

In [76]:
```
from sklearn.datasets import make_blobs
coordsPts,labsPts = make_blobs(n_samples=200,
                          n_features=2,
                          centers = 10,
                         cluster_std=1,
                         random_state=7)
plt.scatter(coordsPts[:,0],coordsPts[:,1],c=labsPts)
plt.show()
```



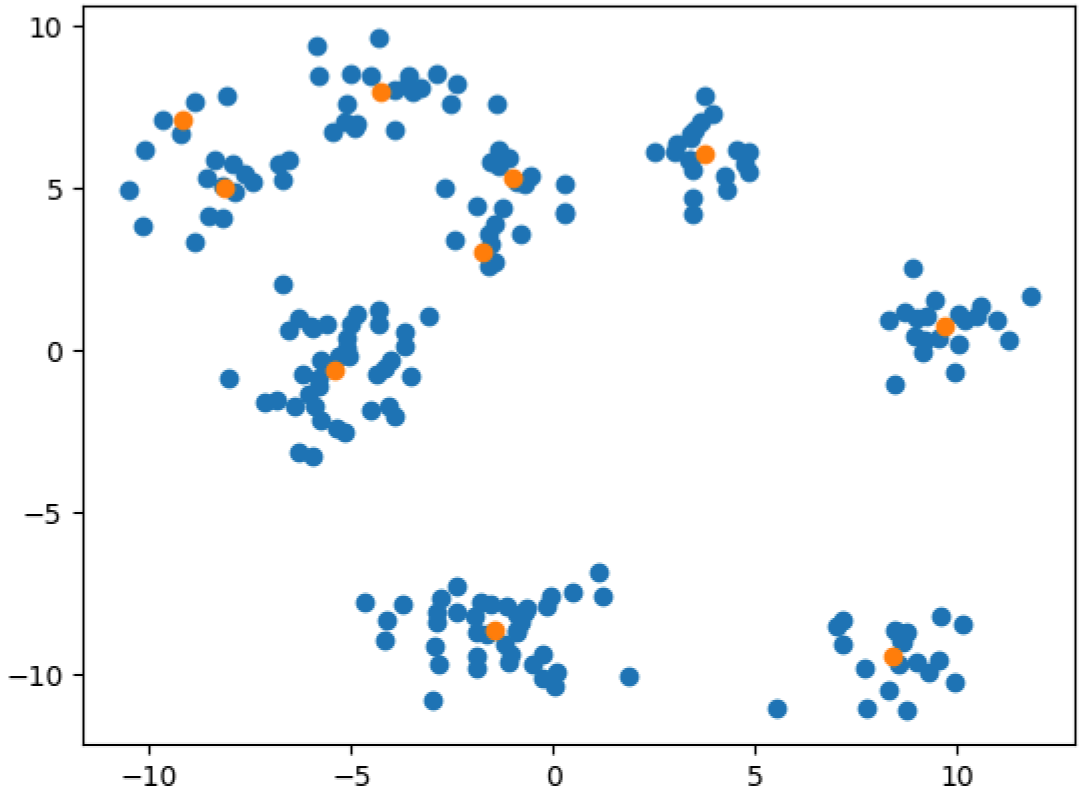
In [80]:

```
# Cutoff is the tolerance to help us stop the centroids.
def Kmeans(k,cutoff,coords=coordsPts):
    ndat,ndim = coords.shape
    choose = np.random.choice(ndat,k,replace=False)
    # Fristly, we will select 10 clusters of the training data
    Means = coords[:k,:].copy()
    # Assign a new variable to store the previous step centorid.
    MeansNew = np.zeros((k,ndim))
    # Calculate the distance from each point to each 10 centroids
    dist = np.zeros((ndat,k))
    step = 0

    # we will keep repeating until the centroids stop change too much
    while np.sqrt(np.sum((Means-MeansNew)**2)) > cutoff:
        step += 1
        # Calculate the distance of each point to each centroid
        for i in range(k):
            dist[:,i] = np.sqrt(np.sum((Means[i,:]-coords)**2,axis =1))
        # After calcluation, we can assign each point to the cluster of centor
        Closest = np.argmin(dist,axis =1)
        disst = 0
        #Create a loop to compute the total distance of points to nearest cent
        for i in range(k):
            distt = dist[:,i]
            disst += np.sum(distt[Closest == i])
        MeansNew = Means.copy()
        # After MeansNew updated, we can compute new centroids by computing the
        #each cluster
        for i in range(k):
            Means[i,:] = np.mean(coords[Closest== i],axis = 0)
    return step,Means
```

```
ns,Ms = Kmeans(10,0.00000001)
print(ns)
plt.scatter(coordsPts[:,0],coordsPts[:,1])
plt.scatter(Ms[:,0],Ms[:,1])
plt.show()
```

7



In [ ]:

In [93]:

```
Dms = Dms / 255.0
```

In [94]:

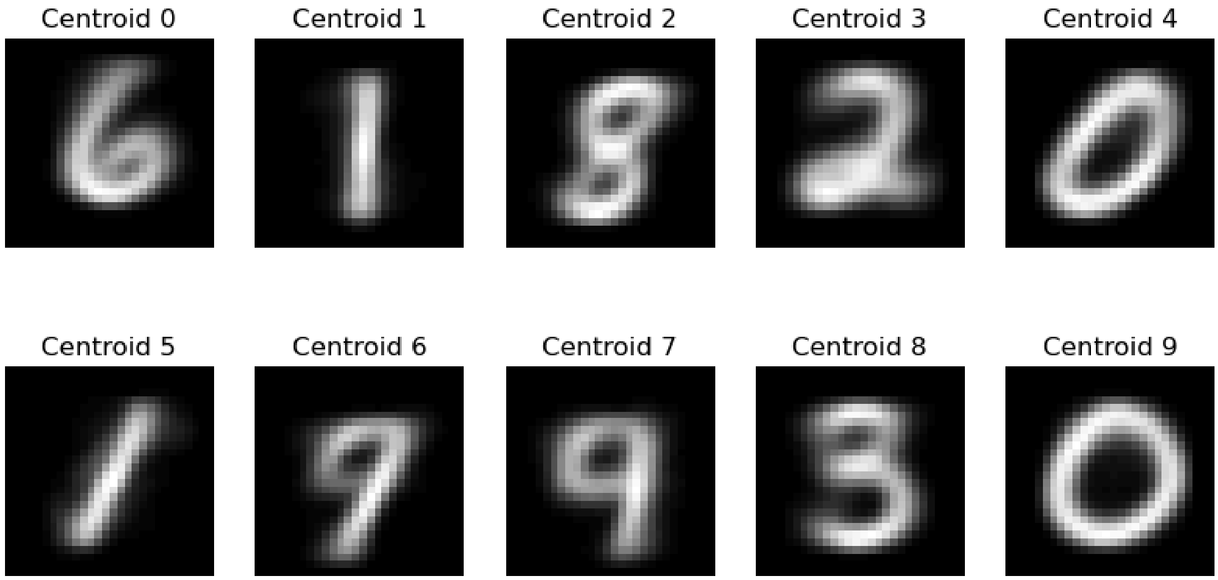
```
# Use K-Means to cluster the data into 10 clusters
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(Dms)

# Get the cluster centroids
centroids = kmeans.cluster_centers_
```

```
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

In [95]:

```
# Reshape the centroids to 28x28 images and display them
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(centroids[i].reshape(28, 28), cmap='gray')
    ax.set_title(f'Centroid {i}')
    ax.axis('off')
plt.show()
```



After we display the centroid, we can see these 10 centroids are resemble digits. We can use these 10 centroids to compute cluster purity. For each cluster, we can find the most common true label and measure accuracy.

If we want to calculate cluster purity, we can create a confusion matrix. Because confusion matrix can helps evaluare how well the clusters align with the true classes.Before we create a confusion matrix, we have to understand what is confusion matrix and how we use it.

- Confusion matrxi shows how data points are distributed accross clusters and true classes. As we all know, each row represents a true class and each column represents a predicted cluster.For confusion[w,s] means that the number of data points from true class 'w' that are assigned to cluster 's'.
- This will help us understand which clusters correspond to which true classes.

In [100…

```
from sklearn.metrics import accuracy_score
from scipy.stats import mode

# Assign each cluster to the most frequent digit in that cluster
cluster_labels = np.zeros_like(kmeans.labels_)
for i in range(10):
    mask = (kmeans.labels_ == i)
    cluster_labels[mask] = mode(Labs[mask])[0]  # Assign the most frequent labe

# Calculate accuracy
accuracy = accuracy_score(Labs, cluster_labels)
print(f"Clustering Accuracy: {accuracy * 100:.2f}%")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[100], line 8
      6 for i in range(10):
      7     mask = (kmeans.labels_ == i)
----> 8     cluster_labels[mask] = mode(Labs[mask])[0]  # Assign the most freq
uent label
     10 # Calculate accuracy
     11 accuracy = accuracy_score(Labs, cluster_labels)

IndexError: boolean index did not match indexed array along dimension 0; dimen
sion is 60000 but corresponding boolean dimension is 700
```

Section 2 2.2 part 2a

In this section of the report, the goal is to Investigate how K-means clustering can be recude the amount of training data. Improving the speed of predication accuracy. We displayed how many dimensional vector in MNIST data set, the data set consists of 60,000 images of handwritten digits (0-9), 784 dimensional vectors. And also each of size 28 x 28 pixels. Thus we can split the dataset into a training set and test set. It also should ensure the proportion of each digit is maintained in training and test sets.

For the first step we have to use the MNIST dataset import the csv file from local device. It allows us easy access to dataset MNIST. We also need to import " train_test_split", because this function spilts a dataset into training and testing set. It helps splits data so that the model can learn from training data. Hence, we import them to help us to load the MNIST dataset and split into traing data and test data.

In [121…

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load data from CSV file, we already read the MINST data set before
# So here we just replace the variable name
print("Step 1: Loading data from minst_train.csv file:")
mnist_train = Dms

# Separate features (X) and labels (y), we will keep using the same variable na
# Assuming the first column is the label and the rest are pixel values

# Split into training and test sets, create the correspnding testing variable n
X_train, X_test, y_train, y_test = train_test_split(Labs, Dms, test_size=10000
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Step 1: Loading data from minst_train.csv file:
Training data shape: (49999, 784)
Test data shape: (10000, 784)
```

After dividing the MINST images into training and test data, we need to reduce the training data using K-Means algorithm. For each number (0-9), we can use the K-Means algorithm to divide these images into several different clusters, for example 100 clusters. In addition, the centroids of these clusters can be used as new training data. Why do we use the centroids of these clusters? Because when we want to reduce the amount of training data and increase the accuracy. We can directly use the centroids of 100 clusters to make the training data size smaller.

In [124…

```
# import the KMeans from sklean.cluster
from sklearn.cluster import KMeans

# We can create a function to reduce training data using k-means clustering, ar
def reduce_training_data(X_train, y_train, n_clusters=100):
    print("\nStep 2: Reducing training data using k-means clustering...")
    Smaller_X_train = []
    Smaller_y_train = []

    for digit in range(10):
        # The digit from data set is 0-9, so select all training images for the
        X_digit = X_train[y_train == digit]

        # Apply k-means clustering algroithm
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        kmeans.fit(X_digit)
        # After we got the cluster centroids, we can use the cluster centroids
        Smaller_X_train.extend(kmeans.cluster_centers_)
        Smaller_y_train.extend([digit] * n_clusters)

    return np.array(Smaller_X_train), np.array(Smaller_y_train)

# Reduce the training data and we pick 100 clusters.
n_clusters = 100  # Number of clusters per digit
Smaller_X_train,Smaller_y_train = reduce_training_data(X_train, y_train, n_clus
print("Smaller training data shape:", Smaller_X_train.shape)
```

```
Step 2: Reducing training data using k-means clustering...
```

```
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/xichenzhang/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kme
ans.py:1412: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Smaller training data shape: (1000, 784)
```

Why we use K-NN algroithm here?

At first, k-NN compares every test image to all training images, which makes it slow for large datasets like MNIST. For our project, the original data set is bery big and high dimensional data 28×28 = 784 features.

When we get the new traing data by using k-means clustering, the data shape will be (1000,784). We will apply K-NN algroithm on new traing data, we can call new traing data "Smaller training data". After we got the "smaller training data", we can train this data and evaluate by using K-NN algorithm.

Since we need to apply the K-NN algorithm, we should import the K-NN classifer from Scikit-Learn, which is used to classify based on nearest neighbors. Additionally, when calculating the prediction accuracy of the training data, we need to import accuracy_score, Accuracy Score = (Correct Predictions) / (Total Predictions) Import Python's time module to measure the amount of time a process takes in order to compare which training set ran to compare which training set runs faster.

In [126…

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import time

# Train k-NN on the 'smaller training data'
print("\nStep 3: Training k-NN on smaller training data:")
knn_reduced = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
knn_reduced.fit(Smaller_X_train, Smaller_y_train)
reduced_training_time = time.time() - start_time

# Evaluate k-NN on the test set
print("Evaluating K-NN on smaller training data:")
start_time = time.time()
y_pred_reduced = knn_reduced.predict(X_test)
reduced_prediction_time = time.time() - start_time
accuracy_reduced = accuracy_score(y_test, y_pred_reduced)

print(f"Accuracy (Smaller Training Data): {accuracy_reduced:.4f}")
print(f"Prediction Time (Smaller Training Data): {reduced_prediction_time:.4f}
```

```
Step 3: Training k-NN on smaller training data:
Evaluating K-NN on smaller training data:
Accuracy (Smaller Training Data): 0.9485
Prediction Time (Smaller Training Data): 0.5261 seconds
```

From the running time, we can realized when we use the K-NN algroithm for smaller data set, the running time be improved and the predication time is more faster.

In [127…

```
# After we train the smaller data set, we also need to train k-NN on the origi
# to check the difference between accuracy and prediction time.
print("\nStep 4: Training K-NN on original training data:")
KNN_original = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
KNN_original.fit(X_train, y_train)
original_training_time = time.time() - start_time

# Evaluate k-NN on the test set
print("Evaluating K-NN on original training data:")
# we change the variable name to make sure more clearly.
Start_time = time.time()
y_pred_original = KNN_original.predict(X_test)
original_prediction_time = time.time() - start_time



accuracy_original = accuracy_score(y_test, y_pred_original)
print(f"Accuracy (Original Training Data): {accuracy_original:.4f}")
print(f"Prediction Time (Original Training Data): {original_prediction_time:.4
```

```
Step 4: Training K-NN on original training data:
Evaluating k-NN on original training data:
Accuracy (Original Training Data): 0.9709
Prediction Time (Original Training Data): 14.9678 seconds
```

As aboved shows that the accuracy for original training data is more accurate, however, the prediction time is getting slowely. To be more specific, compare the 'smaller training data', the orignal training data must have more data. Hence, the running time getting slowly.

After we apply K-NN algroithm by using the centroid triaining data, we can still check another condiction. training K-NN algroithm on randomly selected subet to check the accuracy of random subset. Furthermore, we also can check do they have difference under two different condition.

In [128…

```
# After compare two different training data,
# we also can train K-NN on a randomly selected subset of the original training
print("\nStep 5: Training K-NN on randomly selected subset:")
# We will pick the same size as smaller training data
n_samples = Smaller_X_train.shape[0]
random_index = np.random.choice(X_train.shape[0], n_samples, replace=False)
# At first, assign the variable to the "labs" and "Dms"
X_train_Random = X_train[random_index]
y_train_Random = y_train[random_index]

KNN_random = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
KNN_random.fit(X_train_Random, y_train_Random)
random_training_time = time.time() - start_time

# Applu K-NN algorithm on the test set
print("Evaluating k-NN on randomly selected subset:")
start_time = time.time()
y_pred_random = KNN_random.predict(X_test)
random_prediction_time = time.time() - start_time
accuracy_random = accuracy_score(y_test, y_pred_random)

# After this, we can try to print out the time
print(f"Accuracy (Random Subset): {accuracy_random:.4f}")
print(f"Prediction Time (Random Subset): {random_prediction_time:.4f} seconds"
```

```
Step 5: Training K-NN on randomly selected subset:
Evaluating k-NN on randomly selected subset:
Accuracy (Random Subset): 0.8781
Prediction Time (Random Subset): 0.5765 seconds
```

As the result show, when we selected subset randomly, the prediction time fpr random subest is getting faster. However, the prediction accuracy is gettering lower.

In order to show the difference between two condictions, we can try to print out each running time for different condidtions.

In [130…

```
# Try to print out all the accuracy and prediction time to show the differnt be

print("\nStep 6: Comparison of Results:")
print(f"Accuracy (Smaller Training Data): {accuracy_reduced:.4f}")
print(f"Accuracy (Original Training Data): {accuracy_original:.4f}")
```

```
print(f"Accuracy (Random Subset): {accuracy_random:.4f}")
print(f"\nPrediction Time (Smaller Training Data): {reduced_prediction_time:.4
print(f"Prediction Time (Original Training Data): {original_prediction_time:.4
print(f"Prediction Time (Random Subset): {random_prediction_time:.4f} seconds"
```

```
Step 6: Comparison of Results:
Accuracy (Smaller Training Data): 0.9485
Accuracy (Original Training Data): 0.9709
Accuracy (Random Subset): 0.8781

Prediction Time (Smaller Training Data): 0.5261 seconds
Prediction Time (Original Training Data): 14.9678 seconds
Prediction Time (Random Subset): 0.5765 seconds
```

Conclusion

The problem explored in this report is to use the K-Means algorithm to group MNIST images into 10 clusters. By displaying the centroids, we can explore whether the centroids are similar to the numbers. The primary goal of Part I 2.1 is to understand what the K-Means algorithm is and how and why we should use it.

For Section 2.2, we can find that the K-NN algorithm has high accuracy on the MNIST dataset in Step 6, with an accuracy of about 95%-97%. Smaller k-values can lead to overfitting, on the contrary, larger k-values can improve the generalization. Therefore, when we test the smaller training data and the original training data, it will show a significant difference.

- We also explored how the limitations of the K-NN algorithm can be addressed using k-means clustering, where we use k-means clustering to group the training data for each digit into clusters and use the centroids of the clusters as a smaller training set. This approach significantly reduces the size of the training data (e.g., from 60,000 centroids to 1,000), making the K-NN algorithm faster and more memory-efficient.

- When we compared to a randomly selected subset of the training data, the K-means algorithm approach performed better in terms of accuracy. Because the centroids represent meaningful patterns in the data, whereas random sampling may miss important information.

- For the prediction time, The prediction speed of k-NN improved significantly when using the reduced dataset.
- For future extension of this project: we can Explore advanced variants(like different training data) of k-NN (nearest neighbors) to handle larger datasets more effectively.

In [ ]: