MNIST Dataset & K-Means Algorithm

Elijah Booth

MAP4191: Mathematical Modeling of data

Dr. Teng Zhang

December 2, 2023

I. Introduction

This project applies the K-Means algorithm to the MNIST dataset in an attempt to optimally and accurately predict it's images in Jupyter. The MNIST dataset is a classic dataset for students and researchers alike to develop algorithms for classification. It comes with 70,000 handwritten digits (zero through nine), where 60,000 are for training and 10,000 are for testing. In addition, each image is a 28 x 28 image which will require some pre-processing.

This data will be passed through a K-Means clustering algorithm, it uses centroids which group data together based on their Euclidean distance. The motivation for selecting K-Means came from its easily interpretable strategy called the elbow method for finding the optimal K values. Essentially, one can graph the accuracy given the error rate at different values of K, and as K increases, find the bend in the graph which looks like an elbow; this is the optimal K-value. The K-values used in this project were 10, 20, 40, 80, and 160.

Applied alongside the K-Means algorithm will be kfolds, a method for cross-validation that splits a dataset into a defined number of partitions. One of the partitions is then used as the test set while all other partitions are identified as the training set. This method of using one partition as testing and all others as training is then completed for all combinations of partitions.

II. Code Overview

A thorough walkthrough of the Jupyter code can be found via the comments in the code. The comments carefully guide the reader through the importation of the data, its preprocessing, and the implementation of the algorithms and predictions. Here will be explained some highlights of the code and a general overview. It should be noted that the file is meant to be open and ran in Jupyter; issues can arise when running from a strictly python environment.

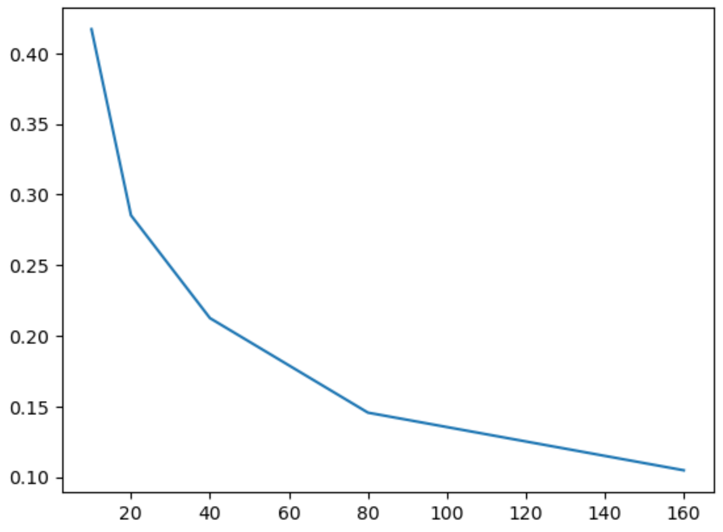
The first step, preprocessing, was rather simple; the 28 by 28 pixel image array had to be flattened so it could be processed, and the array it sat within was normalized. This was done in the first Jupyter cell.

The next Jupyter cell is that which creates five predictive models for each of the five kfolds for the first K value: K = 10. It does this by enumerating through each of the kfold sets, fitting and transforming based off those unique sets, then looping through the clusters and the labels for the data found within those clusters. Then by creating a reference dictionary with the key-value pair found most often in that cluster we are able to test and predict. This prediction is performed for each model and the average error of those models are saved for later to graph the average error and find the optimal K; it should be mentioned that the code to test the average prediction error is used later to find the accuracy of each model. This Jupyter cell to build and create predictive models is then repeated for each of the K-values.

III. Results

The K-Means algorithm is known to get more and more accurate as the number of clusters increases, but this comes with the risk of overfitting, and the cost of the computation can become quite high.

This is the graph of our average prediction error rate (Y-axis) given the cluster size (X-axis) as cluster size increases:



Given in numerical form, this is the prediction accuracy of each K-Means five models with the runtime of each model (%%time):

10-Clusters:

```
3min 41s

0.5794285714285714
0.5845714285714285
0.5781428571428572
0.5822142857142857
0.5900714285714286
```

20-Clusters:

```
5min 3s

0.7340714285714286
0.7071428571428572
0.7069285714285715
0.7053571428571429
0.72
```

40-Clusters:

```
8min 54s

0.7806428571428572
0.7718571428571429
0.7716428571428572
0.7974285714285714
0.816
```

80-Clusters:

```
15min

0.8596428571428572
0.8575
0.8446428571428571
0.8457857142857143
0.8646428571428572
```

160-Clusters:

```
24min

0.9008571428571429
0.8867857142857143
0.8938571428571429
0.8913571428571428
0.9035714285714286
```

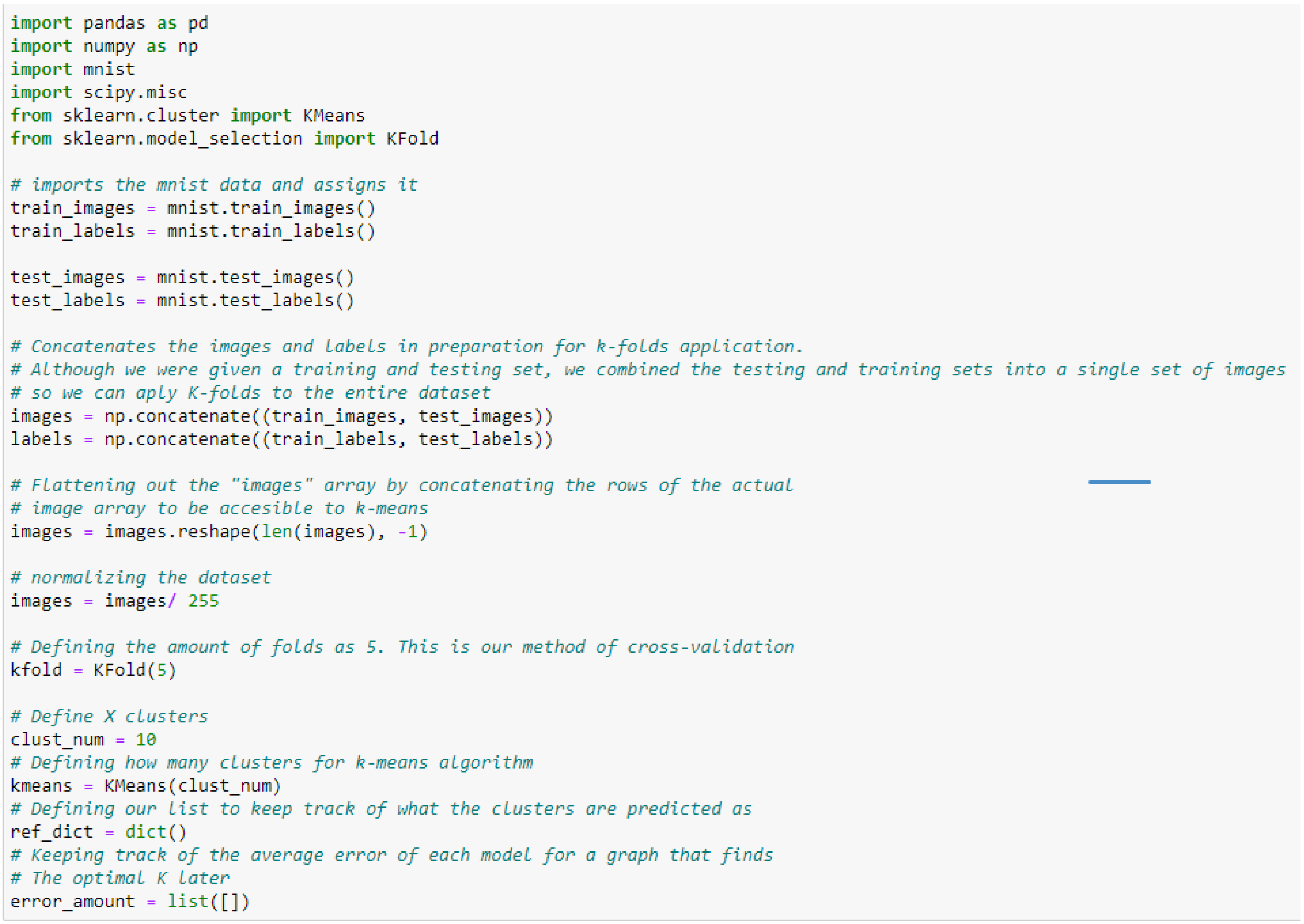
From the range of clusters, 10 to 160, the accuracy varies from 57% to 90%, and although one is given higher accuracy for more clusters, it is not necessarily good for computational time. This corresponds to the runtime that varied from 3mins 41s to 24min. As such, and as is closer to the elbow of the graph, K = 80 clusters has an acceptable error rate of 15% and run time of 15 minutes. This would be the selected algorithm to predict more Test MNIST Datasets.

IV. Conclusion

The implementation of the K-Means algorithm on the MNIST dataset for classification resulted in a predictive model with an accuracy of 85% and a runtime of 15min. With a more powerful machine or efficient algorithm implementation, the more expensive predictions could become more feasible and since the elbow point has subjective interpretation it could be placed further down the curve. This is the result of the graph not having a "sharp" elbow or a more dramatic change in slope and its high interpretability. Nevertheless, K-Means is a model that, if allowed, can achieve high accuracy at a rate of at least 90%.

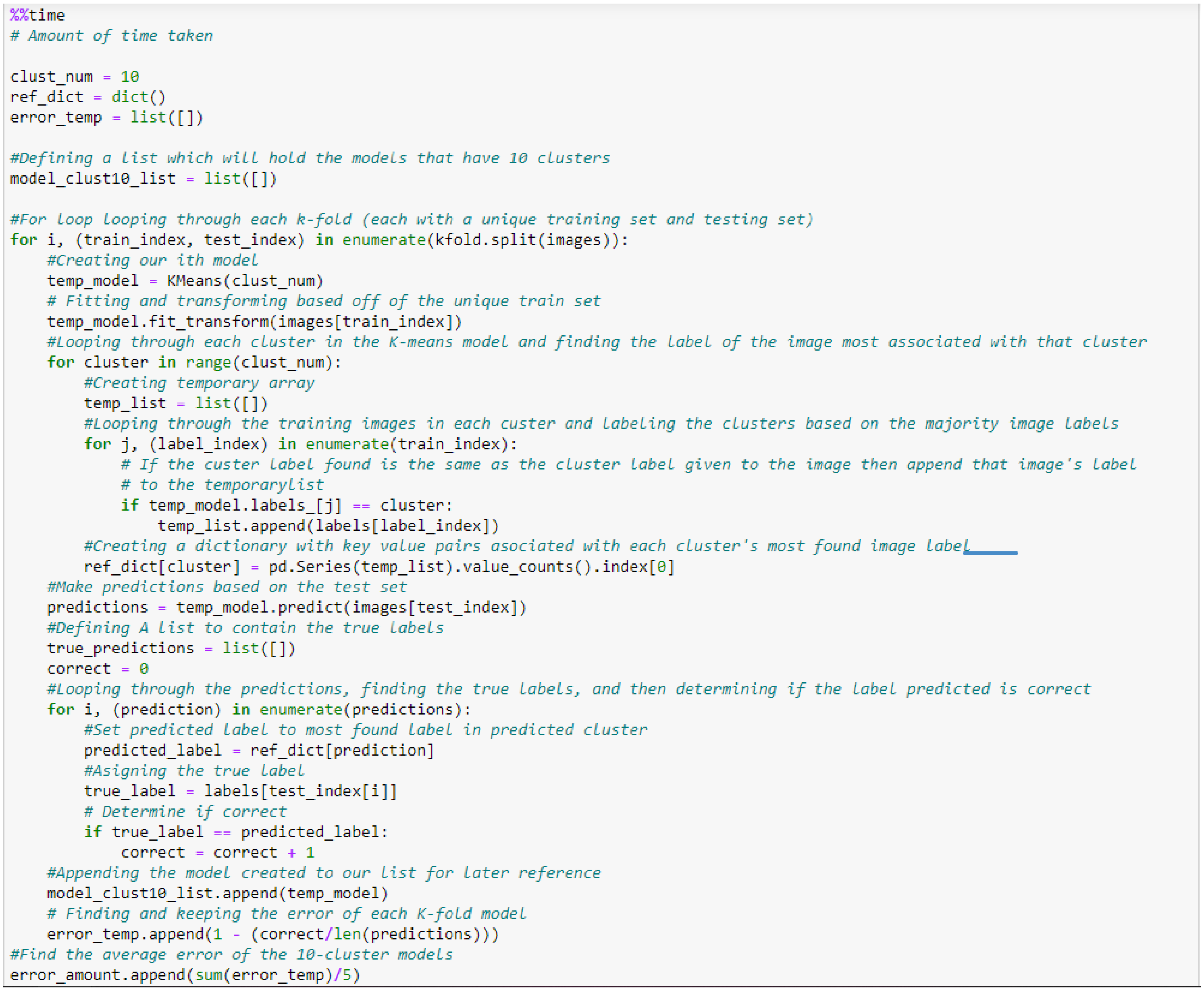
Appendix A: Commented Jupyter Code

This is the first Jupyter cell Containing the Initial preparation:



```
import pandas as pd
import numpy as np
import mnist
import scipy.misc
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold

# imports the mnist data and assigns it
train_images = mnist.train_images()
train_labels = mnist.train_labels()

test_images = mnist.test_images()
test_labels = mnist.test_labels()

# Concatenates the images and labels in preparation for k-folds application.
# Although we were given a training and testing set, we combined the testing and training sets into a single set of images
# so we can aply K-folds to the entire dataset
images = np.concatenate((train_images, test_images))
labels = np.concatenate((train_labels, test_labels))

# Flattening out the "images" array by concatenating the rows of the actual
# image array to be accesible to k-means
images = images.reshape(len(images), -1)

# normalizing the dataset
images = images/ 255

# Defining the amount of folds as 5. This is our method of cross-validation
kfold = KFold(5)

# Define X clusters
clust_num = 10
# Defining how many clusters for k-means algorithm
kmeans = KMeans(clust_num)
# Defining our list to keep track of what the clusters are predicted as
ref_dict = dict()
# Keeping track of the average error of each model for a graph that finds
# The optimal K later
error_amount = list([])
```

This is the Repeated Jupyter cell Containing the K-Means algorithm and prediction. The one displayed is the K = 10 cluster code:



```
%%time
# Amount of time taken

clust_num = 10
ref_dict = dict()
error_temp = list([])

#Defining a list which will hold the models that have 10 clusters
model_clust10_list = list([])

#For loop looping through each k-fold (each with a unique training set and testing set)
for i, (train_index, test_index) in enumerate(kfold.split(images)):
    #Creating our ith model
    temp_model = KMeans(clust_num)
    # Fitting and transforming based off of the unique train set
    temp_model.fit_transform(images[train_index])
    #Looping through each cluster in the K-means model and finding the label of the image most associated with that cluster
    for cluster in range(clust_num):
        #Creating temporary array
        temp_list = list([])
        #Looping through the training images in each custer and labeling the clusters based on the majority image labels
        for j, (label_index) in enumerate(train_index):
            # If the custer label found is the same as the cluster label given to the image then append that image's label
            # to the temporarylist
            if temp_model.labels_[j] == cluster:
                temp_list.append(labels[label_index])
        #Creating a dictionary with key value pairs asociated with each cluster's most found image label
        ref_dict[cluster] = pd.Series(temp_list).value_counts().index[0]
    #Make predictions based on the test set
    predictions = temp_model.predict(images[test_index])
    #Defining A list to contain the true labels
    true_predictions = list([])
    correct = 0
    #Looping through the predictions, finding the true labels, and then determining if the label predicted is correct
    for i, (prediction) in enumerate(predictions):
        #Set predicted label to most found label in predicted cluster
        predicted_label = ref_dict[prediction]
        #Asigning the true label
        true_label = labels[test_index[i]]
        # Determine if correct
        if true_label == predicted_label:
            correct = correct + 1
    #Appending the model created to our list for later reference
    model_clust10_list.append(temp_model)
    # Finding and keeping the error of each K-fold model
    error_temp.append(1 - (correct/len(predictions)))
#Find the average error of the 10-cluster models
error_amount.append(sum(error_temp)/5)
```