Özyeğin University

Fall 2020

C.S. 554.A

Homework II

Submitted by: M. Furkan Oruc (MSc. Student at CE Department)

Student ID: S025464

Submission Date: 13/11/2020

Content

1. Nearest Mean Classifier

Nearest mean classifier operates as a non parametric classification procedure in the following steps.

In this problem, euclidian distance is used due to the fact that the entire range of pixels are provided in the same scale, 0-255 which is extracted from MNIST data. On the other hand, total of 1000 instances are provided for each, training and test purposes.

Euclidian distance, which is a commonly used distance metric can be defined as:

$$E\ (x, y) = \sqrt{\sum_{i=1}^{a} d(xi, yi)^2}$$

where x and y are two examples, a is the number of attributes and xi refers to the ith attribute value for x.

If the variances weren't assumed to be equal, it would not be possible to act with euclidian distance, we would need the Mahalanobis distance. (Alpaydın, 2014)

In the scope of this question, in order to find the euclidian distance between distinct test instances, mean values for each distinct label and each distinct pixel are calculated. The following code represents the main part of this process.

```
for k in range (0,10):
    df_pixel_sum[k] = df_train.loc[df_train["label"] == k ,:].sum()
    df_pixel_mean[k] = df_train.loc[df_train["label"] == k ,:].mean()
    k = k + 1
```

A highlighted part of the source code, where subtraction of a test instance's distinct pixel value from the mean of the related pixel and all labels can be followed as below.

```
for i in range(1, 785): # for each pixel. cover all pixels
            pixel_difference.append((df_test.iloc[r,i] - df_pixel_mean.iloc[i, m])**2)
        #For the 0th image in test data, subtract and square each pixel's value from each pixel's mean for each number
            i = i + 1
        df_pixel_difference[m] = pixel_difference # Store the calculation for each image
```

Right after calculation of distance between each number's mean and storage of that difference for each instance of the test data's each pixel values, the minimum distance to a mean is calculated as below. In this step, also index of the minimum sum value for each instance is also stored. The index value helps storing the assigned instance to the test data based on training data. Of course, while the confusion matrix was being calculated, the instances were belonging to training data.

```
 #for k in range (0,10): # For each number's mean, find the distance between 0th test data
        number_identifier.append(df_pixel_difference.iloc[:, m].sum())
        df_number_identifier [m] = number_identifier
        m = m + 1
        #k = k + 1
        number_identifier.clear()

    for s in range (0,10):         # Finding the minimum distance between 0th test element and number image means
        near_mc.append(df_number_identifier.iloc[0,s])
        df_near_mc[s] = near_mc
        near_mc.clear()
        s = s + 1
    df_near_mc_transpose = df_near_mc.transpose()

    assigned_instance = (df_near_mc_transpose.idxmin())
    df_assigned_instance[r] = assigned_instance
    #assigned_instance.clear()
```

In order to represent the assigned values of the instances, a confusion matrix is processed.

```
for f in range(len(df_test)):

    if df_assigned_instance.iloc[f,0] == df_test.iloc[f,0]:
        x = df_test.iloc[f,0]
        df_confusion.iloc[x,x] = df_confusion.iloc[x,x] + 1

    elif df_assigned_instance.iloc[f,0] != df_test.iloc[f,0]:
        y = df_test.iloc[f,0]
        x = df_assigned_instance.iloc[f,0]
        df_confusion.iloc[y,x] = df_confusion.iloc[y,x] + 1
```

In the confusion matrix, each row represents the real label of each test data. Each column represents the assigned label for each test instances. Since there are 100 instances for each label, each row makes up a total of 100 value in the matrix.

Diagonal sums represent the number of correct assignments to each label. Off-diagonal cells (sums) represent the incorrect assignments.

The total number of diagonal values represent the number of correct assignments in the entire dataset, for each case. The accuracy rates for both test dataset and training dataset is provided above. It's important to note that based on the brief of the question, both confusion matrices are based on training dataset means. On the other hand, classification is applied for both test dataset and training dataset.

$$\boldsymbol{Accuracy\ Rate\ on\ Test\ Data}: \mathbf{70.5}\%$$

$$\boldsymbol{Accuracy\ Rate\ on\ Training\ Data}: \mathbf{81.2}\%$$

Confusion Matrix for the Test Data

It can be clearly observed that the most of the incorrect assignments are observed in numbers 2,3 and 8.

The most problematic assignment has been on number 3, from which 31 of them are assigned to the number 5. It can be stated that MNIST dataset provides 5's which may be mistaken as 3 by the nearest mean classifier logic.

Index	0	1	2	3	4	5	6	7	8	9
0	86	0	1	1	1	6	4	0	0	1
1	0	95	0	0	0	5	0	0	0	0
2	1	24	59	3	2	1	0	3	6	1
3	0	3	1	59	0	31	0	3	1	2
4	0	2	0	0	69	0	2	0	0	27
5	3	4	1	9	3	67	0	4	2	7
6	5	1	7	0	14	9	63	0	1	0
7	0	10	3	0	2	1	0	73	0	11
8	2	3	3	8	3	9	2	2	58	10
9	0	1	0	2	15	0	0	4	2	76

Figure 1: Nearest Mean Classification Confusion Matrix for Test Dataset

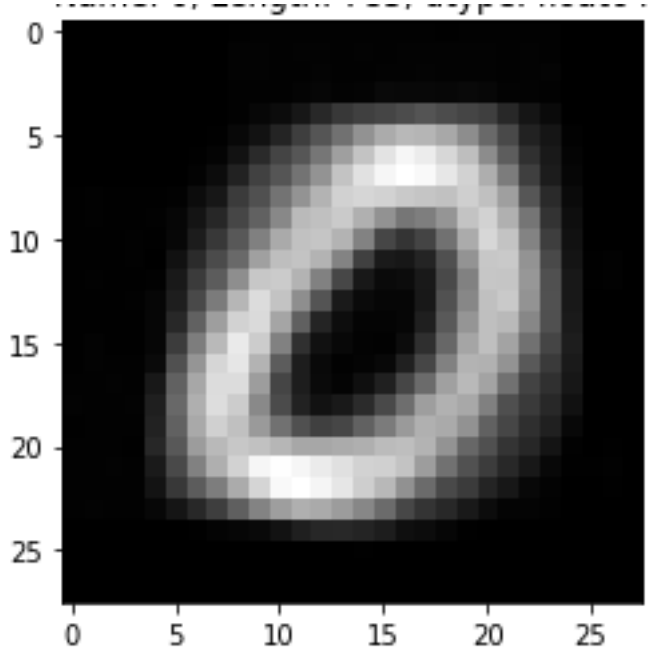
Confusion Matrix for the Training Data

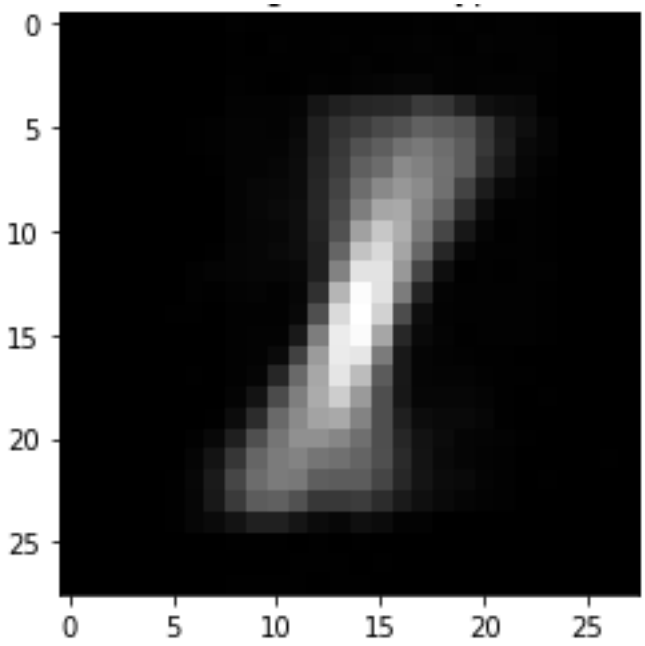
Index	0	1	2	3	4	5	6	7	8	9
0	91	0	1	0	0	3	3	1	1	0
1	0	96	0	0	0	3	0	0	1	0
2	2	14	69	3	3	1	1	4	1	2
3	1	1	1	76	0	13	0	3	2	3
4	0	2	0	0	84	0	2	0	0	12
5	4	4	1	10	5	73	2	1	0	0
6	1	5	0	0	4	6	84	0	0	0
7	0	6	1	0	2	1	0	86	1	3
8	0	8	2	5	0	8	1	0	73	3
9	1	2	0	2	7	3	1	4	0	80

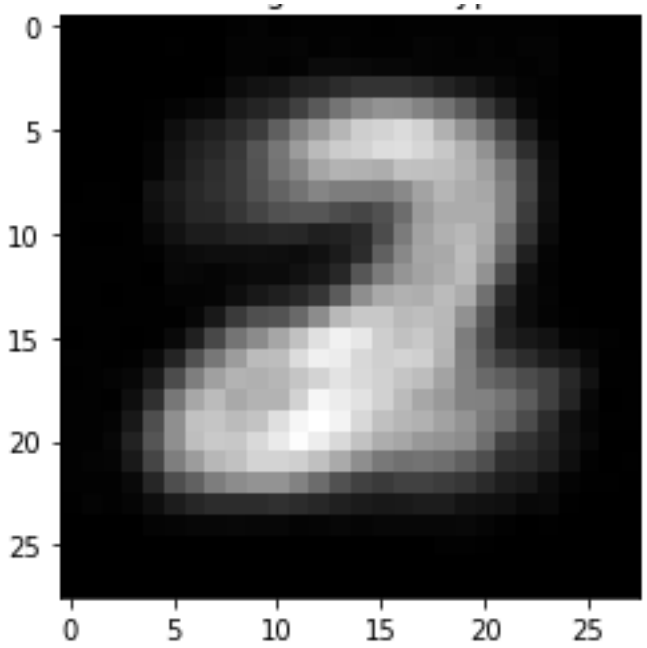
Figure 2: Nearest Mean Classification Confusion Matrix for Training Dataset

2. Images' Visualization

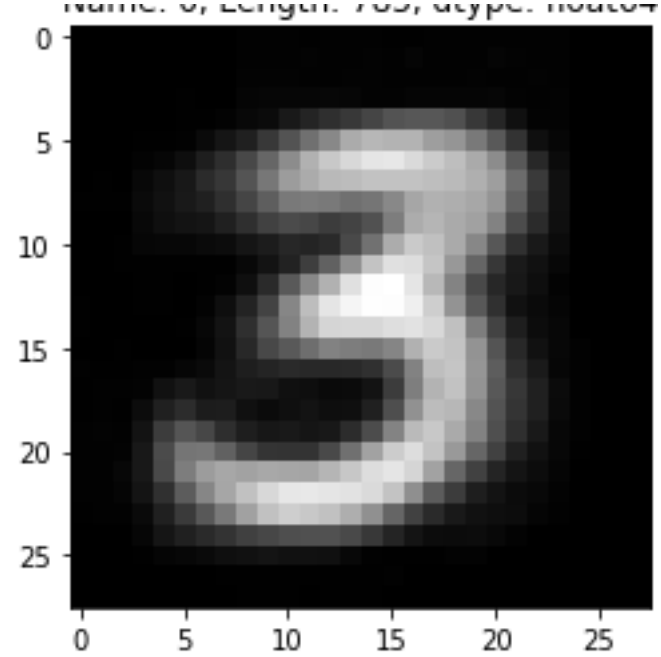
Visualization of MNIST training data's mean values for each number can be observed below.

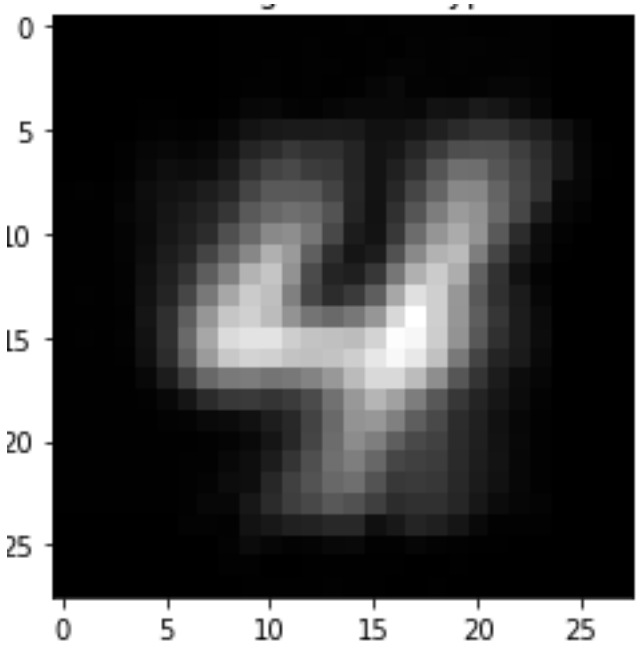


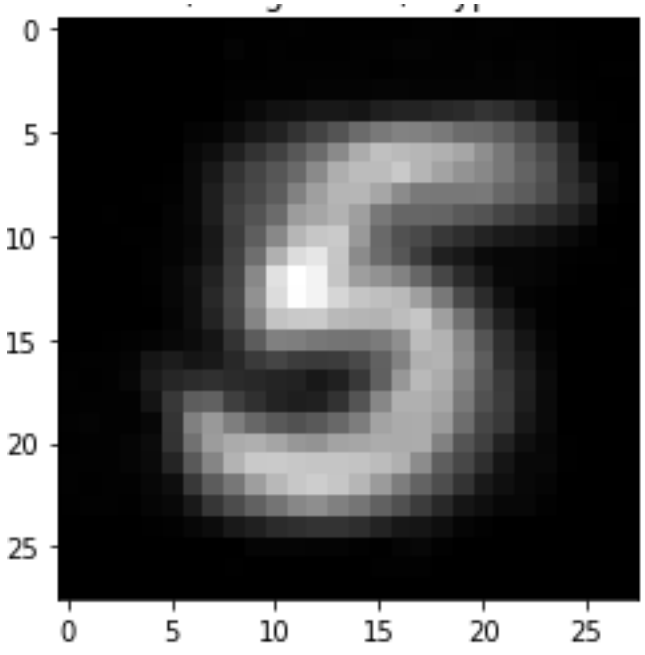


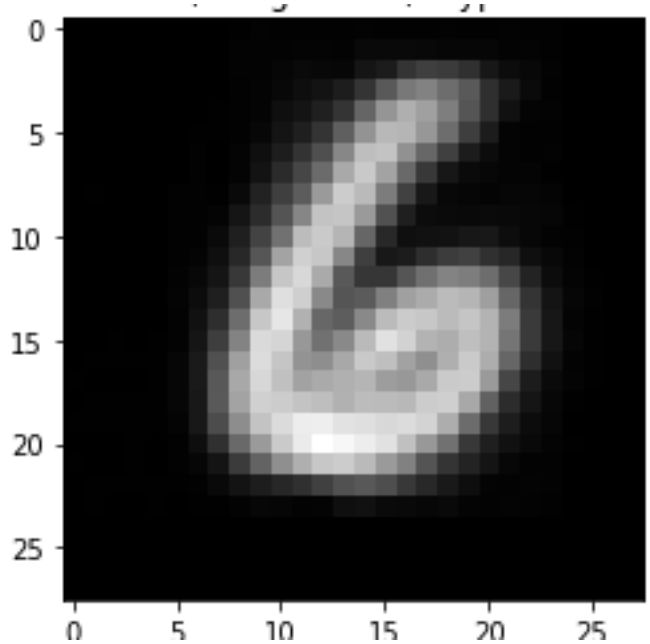


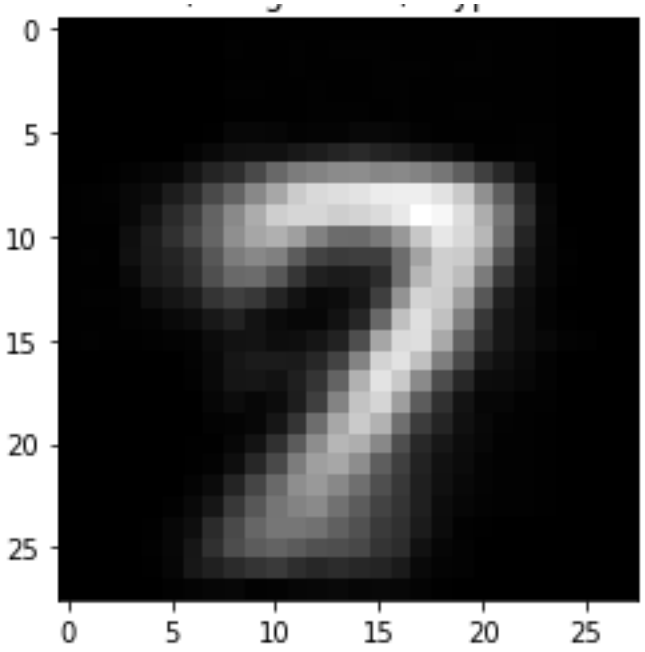
Name: 0, Length: 785, dtype: float64

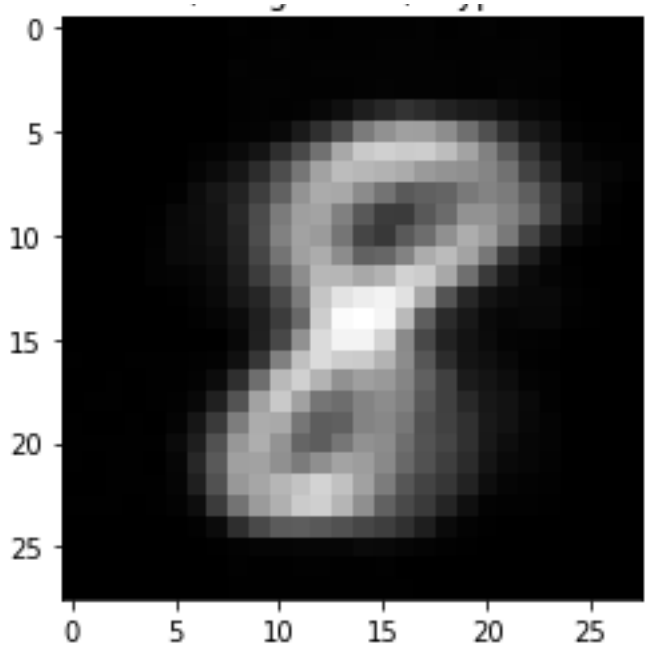


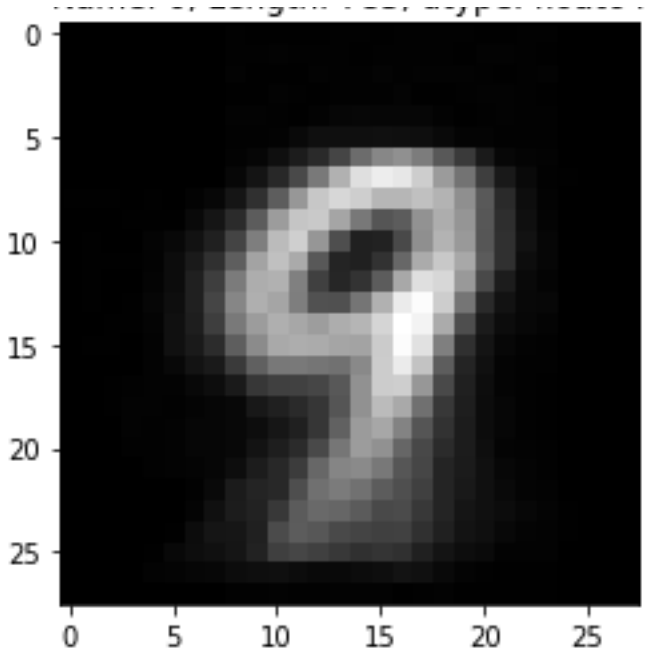

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

3. K-Nearest Neighbor Classification

In the non-parametric classification based on k-nn algorithm, an object is classified based the plurality vote of its neighbors. In other words, based on number of k's, each instance of the dataset is classified based on the number of its closest neighbors inside the "k" set.

Each training instance votes for its class and has no other effect on other classes. Normally, the weight of vote is given to the closest instances. Though, in the k-nn classifier case, the class having the most examples is looked for. It can be also expressed as:

$$\hat{P}(C_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\hat{p}(\boldsymbol{x})} = \frac{k_i}{k}$$

In this classification's scope, k is valued as 1, 3, 5, 7, 9, 11 for each case.

There has been a quite limited number of ties. Those ties are specifically represented in the accuracy and results section.

A brief part of the source code is represented below, where the Euclidian distance is calculated between each instance of the test data and each pixel with respect to each value of the training dataset.

```
for k in range(0, 1000): #1000
    summer.clear()
    for t in range (1, 785):
        distance_knn.clear()
        for m in range (len(df_train)):

            distance_knn.append((df_test.iloc[k,t] - df_train.iloc[m, t])**2)


        df_knn[t] = distance_knn
```

Later on, each Euclidian distance is stored for each test instance and respective training instances. The sum of the pixels are calculated in order to detect the total representative distance between each instance.

```
    for n in range(0,1000):

        sum_holder = df_knn.iloc[n, :].sum()
        summer.append(sum_holder)
    df_summer[k] = summer
```

In the next step, for each instance, the values are stored and sorted. While sorting, labels for each instance is calculated.

```
for x in range (0,1000): #1000
    knn_label.clear()
    for y in range (0,11):
        knn_label.append(df_train.iloc[df_storella.iloc[y,x],0])
    df_knn_label[x] = knn_label
```

Right after the creation of the matrice where for each test data instance, the most repeated index is extracted for each k case in the loop (1, 3, 5, 7, 9, 11); mode specialty is used. By this method, the most repeated cases are determined. This helped storage of the most repeated labels for each k case distinctively.

```
for p in range (1,11,2):
    for o in range (0,1000): #1000, k = 11
        mode_holder.append(df_knn_label.iloc[:p,o].mode())

        #df_mode_holder [t] = mode_holder ###

df_mode_holder = pd.DataFrame(mode_holder)
```

The confusion matrix is created in a manner where rows' indexes represent each real label already given in the test dataset and each column index represents the assigned label, of course in different matrices for each k case.

Due to the timely cost of nonparametric classification and the brief, the classification is applied on the test data.

On the other hand, one issue which was resolved during the classification was ties. Among 6000 classification of test data, 230 of the classifications occurred with a tie.

In order to break the tie, some specific applications are utilized. Based on the wide scale practice, the closest label is assigned between the multiple members of the tie. In order to make it happen, each “tie” instance is examined and its closest top 6 labels are compared with the tie members. The algorithm suggests that if one of the maximum 5 tie labels are included in the closest 6 label set, then the closest instance among top 6 label is assigned as the tie-breaker label to the related instance.

The assumption is based on both analytical and exploratory analysis on the dataset. It is assumed that if a tie-breaker is present, it would be included between top 6 labels of that instance. In other words, one of the closest 6 label is assumed to be enough to be equal to one of the tie members. The after analysis also suggests that the accuracy on the confusion matrices is presented as 99.99%.

```
for t in range (len(nan_2_1)):
    if nan_2_1[t] > 1000 and nan_2_1[t] < 2000:
        if df_mode_holder_1.iloc[nan_2_1[t],0] == df_knn_label.iloc[0,nan_2_1[
t]-1000]:
            counter = counter + 1
            counter_list_3.append(nan_2_1[t])
    elif nan_2_1[t] > 2000 and nan_2_1[t] < 3000:
        if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[
t]-2000]:
            counter = counter + 1
            counter_list_6.append(nan_2_1[t])
    elif nan_2_1[t] > 3000 and nan_2_1[t] < 4000:
        if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[
t]-3000]:
            counter = counter + 1
            counter_list_6.append(nan_2_1[t])
    elif nan_2_1[t] > 4000 and nan_2_1[t] < 5000:
        if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[
t]-4000]:
            counter = counter + 1
            counter_list_6.append(nan_2_1[t])
    elif nan_2_1[t] > 5000 and nan_2_1[t] < 6000:
        if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[
t]-5000]:
            counter = counter + 1
            counter_list_6.append(nan_2_1[t])
for p in range (5000, 6000): # do it for each k=3,5,7,9,11 for respective 1000
-2000-3000 etc
    for z in range (1,5):
```

```
        if df_mode_holder_1.iloc[p, z] >= 0: #hold p
            if p in counter_list:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[0,p-5000]
            elif p in counter_list_2:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[1,p-5000]
            elif p in counter_list_3:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[2,p-5000]
            elif p in counter_list_4:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[3,p-5000]
            elif p in counter_list_5:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[4,p-5000]
            elif p in counter_list_6:
                df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[5,p-5000]

```

Below, each confusion matrix is provided with the respective accuracy rate. Also, The total number of diagonal values represent the number of correct assignments in the entire dataset, for each case.

The K-NN Classification is based on both test and training data instances. Following table represents the changing accuracy rate based on different k values, ranging(1,11).

Post Reasoning on the Accuracy Rate Difference Between Test & Training Datasets

It'll be observed in the next page that there is a dramatic accuracy in the training dataset, while the test data performs moderately. After an exploratory post analysis, it can be stated that, in the training case, due to the fact that the closest instance will definitely carry the label of the instance's real label, that's a strong sign. Later on, in the bigger k's such as 9 and 11, it can be stated that training data is made of instances which are ordered by their label and their pixel values (0,255) seem to be closer to each other when they are ordered line by line. This suggests that it was dramatically easier for training data instances to find neighbors which carry the same label with them, since their consecutive instances were clearly similar to each other.

Test Data Accuracy Rate / K Plot

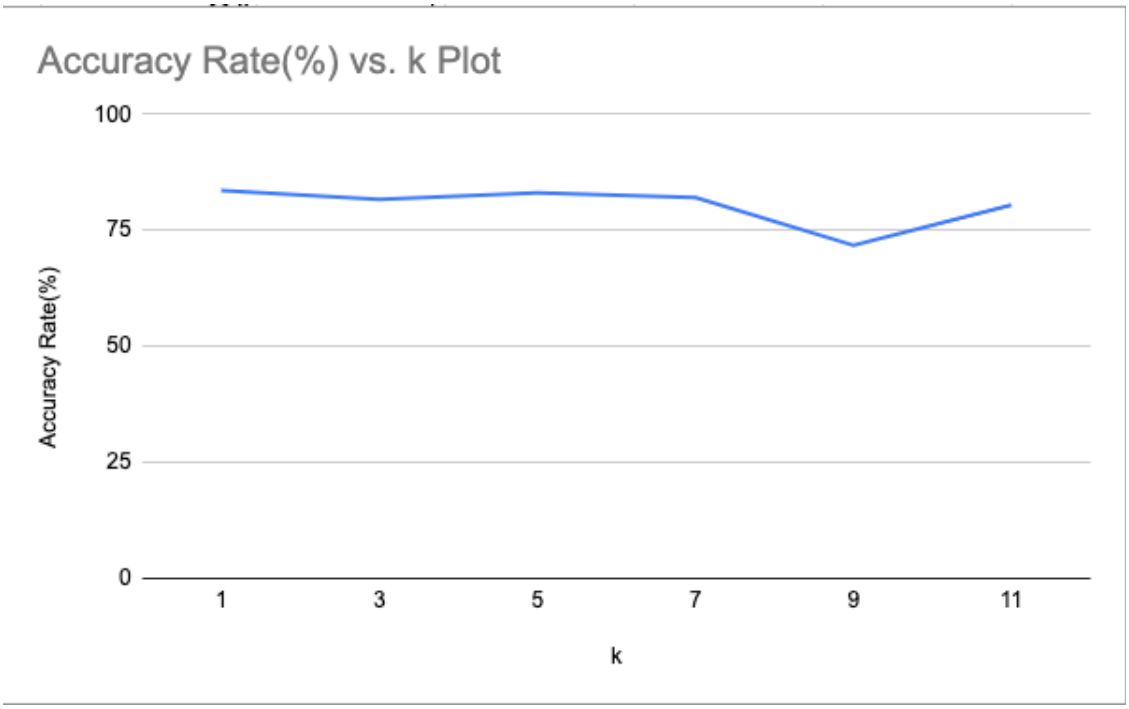


Table 1: Test Data: Plot of Changing Accuracy Rate based on different k values

Training Data Accuracy Rate / K Plot

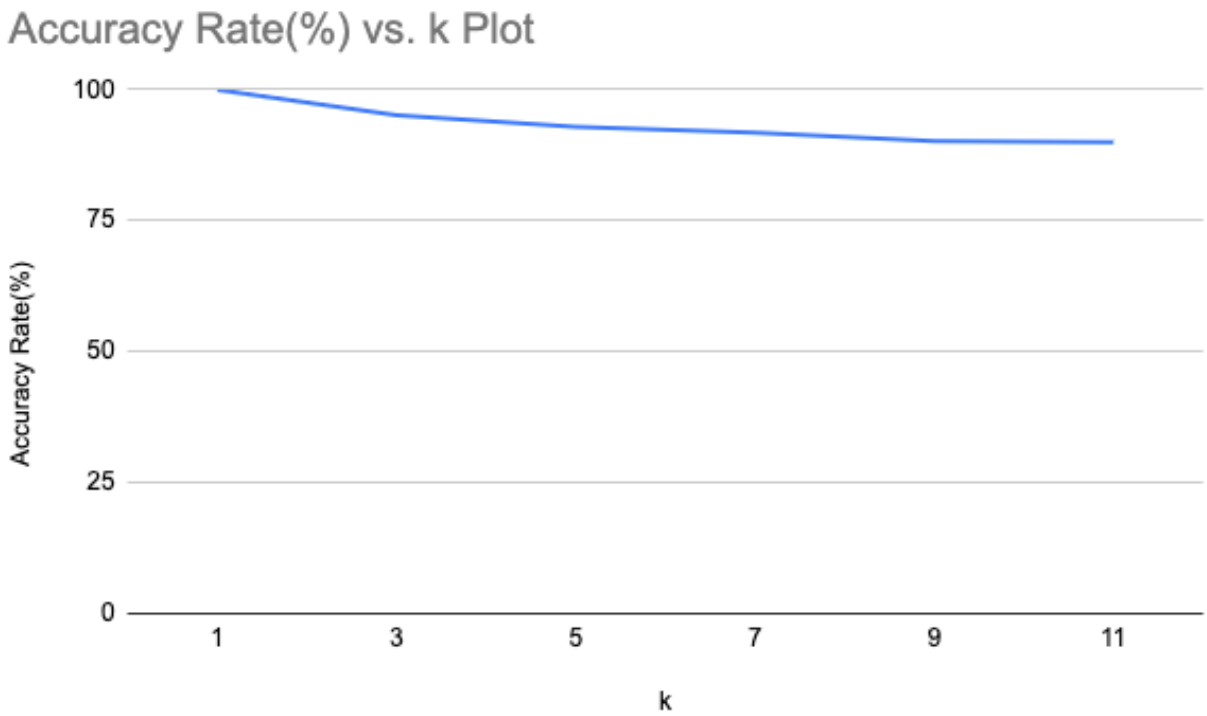


Table 2: Training Data: Plot of Changing Accuracy Rate based on different k value

Test Data Confusion Matrices

Confusion Matrix for the Test Data – Case: k=1

Index	0	1	2	3	4	5	6	7	8	9
0	94	0	0	0	1	0	3	1	0	1
1	0	100	0	0	0	0	0	0	0	0
2	1	4	80	2	1	0	2	7	3	0
3	0	0	1	76	0	14	2	3	2	2
4	0	2	0	0	69	1	2	2	0	24
5	1	3	0	6	4	74	4	2	2	4
6	4	0	0	0	1	3	92	0	0	0
7	0	4	0	0	2	1	0	90	0	3
8	2	2	6	4	1	3	4	1	73	4
9	0	0	0	1	4	1	0	6	0	88

Figure 3: K-nn classification: k = 1

$$\boldsymbol{Accuracy\ Rate\ on\ k = 1: 83.6\%}$$

Confusion Matrix for the Test Data – Case: k=3

Index	0	1	2	3	4	5	6	7	8	9
0	96	0	1	0	0	0	3	0	0	0
1	0	100	0	0	0	0	0	0	0	0
2	2	11	70	1	1	1	2	6	6	0
3	1	2	6	74	0	9	1	1	3	3
4	0	2	0	0	70	0	2	1	0	25
5	1	5	0	6	4	77	4	0	0	3
6	5	0	0	0	2	3	90	0	0	0
7	0	6	1	0	2	1	0	87	0	3
8	5	2	2	6	0	4	3	3	68	7
9	0	1	0	1	5	0	0	8	0	85

Figure 4: K-nn classification: k = 3

$$\boldsymbol{Accuracy\ Rate\ on\ k = 3: 81.7\%}$$

Confusion Matrix for the Test Data – Case: k=5

Index	0	1	2	3	4	5	6	7	8	9
0	93	0	1	0	0	1	5	0	0	0
1	0	100	0	0	0	0	0	0	0	0
2	1	13	67	1	2	1	2	7	6	0
3	0	3	1	76	0	11	1	2	3	3
4	0	2	0	0	70	0	2	1	0	25
5	1	4	0	3	2	78	5	1	0	6
6	3	0	0	0	1	3	93	0	0	0
7	0	7	1	0	1	1	0	89	0	1
8	3	2	2	5	1	2	4	2	73	6
9	0	1	0	0	2	0	0	4	1	92

Figure 5: K-nn classification: k = 5

$$\boldsymbol{Accuracy\ Rate\ on\ k = 5: 83.1\%}$$

Confusion Matrix for the Test Data – Case: k=7

Index	0	1	2	3	4	5	6	7	8	9
0	93	0	1	0	0	1	5	0	0	0
1	0	100	0	0	0	0	0	0	0	0
2	1	18	63	1	2	0	2	7	6	0
3	0	3	2	77	0	9	1	2	3	3
4	0	2	0	0	71	0	2	1	0	24
5	1	4	0	4	1	74	5	1	2	8
6	3	0	0	0	1	3	93	0	0	0
7	0	9	1	0	2	0	0	84	0	4
8	3	3	1	4	1	3	2	3	74	6
9	0	2	0	0	2	0	0	3	1	92

Figure 6: K-nn classification: k = 7

$$\boldsymbol{Accuracy\ Rate\ on\ k = 7: 82.1\%}$$

Confusion Matrix for the Test Data – Case: k=9

Index	0	1	2	3	4	5	6	7	8	9
0	93	0	0	0	1	2	3	1	0	0
1	0	100	0	0	0	0	0	0	0	0
2	1	19	61	1	2	0	2	7	6	1
3	0	5	2	78	0	6	1	2	3	3
4	0	2	0	0	71	0	2	1	0	24
5	1	4	0	4	1	76	4	2	0	8
6	3	0	0	0	2	3	92	0	0	0
7	0	9	0	1	0	1	0	84	0	5
8	4	3	1	4	2	3	1	3	70	9
9	0	1	0	0	3	0	0	2	1	93

Figure 7: K-nn classification: k = 9

$$\boldsymbol{Accuracy\ Rate\ on\ k = 9: 71.8\%}$$

Confusion Matrix for the Test Data – Case: k=11

Index	0	1	2	3	4	5	6	7	8	9
0	93	0	1	0	1	2	2	1	0	0
1	0	100	0	0	0	0	0	0	0	0
2	1	21	58	1	3	0	3	6	6	1
3	0	6	1	76	0	8	1	2	3	3
4	0	3	0	0	68	0	2	1	0	26
5	1	4	0	3	2	75	5	2	0	8
6	3	0	0	0	4	3	90	0	0	0
7	0	9	0	1	3	0	0	83	0	4
8	3	3	2	4	2	4	2	3	69	8
9	0	1	0	0	2	0	0	3	1	93

Figure 8: K-nn classification: k = 11

$$\boldsymbol{Accuracy\ Rate\ on\ k = 11: 80.5\%}$$

Training Data Confusion Matrices

Confusion Matrix for the Training Data – Case: k=1

Index	0	1	2	3	4	5	6	7	8	9
0	100	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0
2	0	0	100	0	0	0	0	0	0	0
3	0	0	0	100	0	0	0	0	0	0
4	0	0	0	0	100	0	0	0	0	0
5	0	0	0	0	0	100	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0
7	0	0	0	0	0	0	0	100	0	0
8	0	0	0	0	0	0	0	0	100	0
9	0	0	0	0	0	0	0	0	0	100

$$\boldsymbol{Accuracy\ Rate\ on\ k = 1: 100\%}$$

Confusion Matrix for the Training Data – Case: k=3

Index	0	1	2	3	4	5	6	7	8	9
0	99	0	0	0	0	0	1	0	0	0
1	0	99	0	0	0	0	0	1	0	0
2	0	3	93	0	0	0	1	1	1	1
3	0	0	1	94	0	2	1	0	0	2
4	0	2	0	0	92	0	1	0	0	5
5	0	1	0	2	0	93	2	0	0	2
6	1	1	0	0	0	0	98	0	0	0
7	0	4	0	0	1	0	0	93	0	2
8	0	3	0	1	0	1	1	0	93	1
9	0	0	0	1	1	0	1	0	0	97

$$\boldsymbol{Accuracy\ Rate\ on\ k = 3: 95.1\%}$$

Confusion Matrix for the Training Data – Case: k=5

Index	0	1	2	3	4	5	6	7	8	9
0	97	0	0	0	0	1	2	0	0	0
1	0	98	0	1	0	0	0	1	0	0
2	0	5	88	0	0	0	3	2	1	1
3	1	1	0	92	0	2	1	1	0	2
4	0	3	0	0	90	1	1	0	0	5
5	0	2	1	2	1	89	2	0	1	2
6	1	2	0	0	0	1	96	0	0	0
7	0	6	0	0	0	0	0	91	0	3
8	0	3	0	2	0	2	0	0	92	1
9	1	0	0	1	2	0	0	0	0	96

$$\boldsymbol{Accuracy\ Rate\ on\ k = 5: 92.9\%}$$

Confusion Matrix for the Training Data – Case: k=7

Index	0	1	2	3	4	5	6	7	8	9
0	98	0	0	0	0	1	1	0	0	0
1	0	99	0	0	0	0	0	1	0	0
2	0	8	80	0	1	1	2	6	1	1
3	1	1	0	92	0	2	1	1	0	2
4	0	4	0	0	91	0	1	0	0	4
5	0	2	1	3	1	88	2	0	1	2
6	1	2	0	0	0	1	96	0	0	0
7	0	6	0	0	0	0	0	91	0	3
8	0	4	0	3	0	3	1	0	88	1
9	1	0	0	2	1	0	0	1	0	95

$$\boldsymbol{Accuracy\ Rate\ on\ k = 7: 91.8\%}$$

Confusion Matrix for the Training Data – Case: k=9

Index	0	1	2	3	4	5	6	7	8	9
0	98	0	0	0	0	1	1	0	0	0
1	0	98	0	1	0	0	0	1	0	0
2	1	11	78	0	1	1	1	5	1	1
3	1	3	1	87	0	2	1	2	1	2
4	0	4	0	0	91	0	2	0	0	3
5	0	2	1	2	2	88	2	0	1	2
6	2	2	0	0	0	1	95	0	0	0
7	0	6	0	0	1	0	0	89	0	4
8	0	4	0	4	0	2	1	0	87	2
9	1	0	0	1	1	0	1	4	1	91

$$\boldsymbol{Accuracy\ Rate\ on\ k = 9: 90.2\%}$$

Confusion Matrix for the Training Data – Case: k=11

Index	0	1	2	3	4	5	6	7	8	9
0	97	0	0	0	0	2	1	0	0	0
1	0	98	0	1	0	0	0	1	0	0
2	1	12	78	0	2	1	0	4	1	1
3	1	3	1	87	0	2	1	2	1	2
4	0	4	0	0	91	0	2	0	0	3
5	0	2	1	4	1	87	1	0	2	2
6	2	3	0	0	0	1	94	0	0	0
7	0	5	0	0	1	0	0	89	0	5
8	0	4	0	3	0	2	1	0	88	2
9	2	0	0	1	3	0	1	2	0	91

$$\boldsymbol{Accuracy\ Rate\ on\ k = 11: 90.0\%}$$

4. Source Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  9 18:32:52 2020

@author: User
"""

#hw2 with mnist dataset, image classifier

import pandas as pd

df_train = pd.read_csv("train1k.csv", sep =",")
df_test = pd.read_csv("test1k.csv", sep =",")


df_train.iloc[0, 0]


#Creating a dataframe in order to store the means of each pixels for each numbers in th
e training data


pixel_sum = []
df_pixel_sum = pd.DataFrame(pixel_sum)
pixel_mean = []
df_pixel_mean = pd.DataFrame(pixel_mean)

label_hold = []
k = 0
label_holder = df_train["label"]

for k in range (0,10):
    df_pixel_sum[k] = df_train.loc[df_train["label"] == k ,:].sum()
    df_pixel_mean[k] = df_train.loc[df_train["label"] == k ,:].mean()
    k = k + 1

t=0
indexer = []
for t in range(785):
    indexer.append(t)

#Set mean pixel's index to numbers for easy access

df_pixel_mean.set_index([indexer], inplace=True)

#Nearest Mean Classifier

#Right Now: Only for 0 Image & 1st Row of Test Data

i = 1
pixel_difference = []
number_identifier = []
near_mc = []
#assigned_instance
df_assigned_instance = pd.DataFrame()
df_pixel_difference = pd.DataFrame(pixel_difference)
df_number_identifier = pd.DataFrame(number_identifier)
```

```
df_near_mc = pd.DataFrame(near_mc)
m = 0
k = 0
r = 0
s = 0
l = 0

for r in range (0,1000):
    for m in range (0, 10): # for each mean of each number
        pixel_difference.clear()
        for i in range(1, 785): # for each pixel. cover all pixels
            pixel_difference.append((df_test.iloc[r,i] - df_pixel_mean.iloc[i, m])**2)

        #For the 0th image in test data, subtract and square each pixel's value from each pixel's mean for each number
            i = i + 1
        df_pixel_difference[m] = pixel_difference # Store the calculation for each image

    #for k in range (0,10): # For each number's mean, find the distance between 0th test data
        number_identifier.append(df_pixel_difference.iloc[:, m].sum())
        df_number_identifier [m] = number_identifier
        m = m + 1
        #k = k + 1
        number_identifier.clear()

    for s in range (0,10):         # Finding the minimum distance between 0th test element and number image means
        near_mc.append(df_number_identifier.iloc[0,s])
        df_near_mc[s] = near_mc
        near_mc.clear()
        s = s + 1
    df_near_mc_transpose = df_near_mc.transpose()

    assigned_instance = (df_near_mc_transpose.idxmin())
    df_assigned_instance[r] = assigned_instance
    #assigned_instance.clear()

df_assigned_instance = df_assigned_instance.transpose()


# Self Test for the Accuracy

q = 0
rank = 0

for q in range (0,1000):
          if df_assigned_instance.iloc[q,0] == df_test.iloc[q,0]:
              rank = rank + 1
      print(rank)


      # Confusion matrix as 0s

      confusion = [0] * 10
      df_confusion = pd.DataFrame(confusion)
      f = 0
      for f in range (1,10):
          df_confusion[f] = df_confusion[0]

```

```
    # Create the Confusion Matrix

    f = 0
    x = 0
    y = 0

    for f in range(len(df_test)):

        if df_assigned_instance.iloc[f,0] == df_test.iloc[f,0]:
            x = df_test.iloc[f,0]
            df_confusion.iloc[x,x] = df_confusion.iloc[x,x] + 1

        elif df_assigned_instance.iloc[f,0] != df_test.iloc[f,0]:
            y = df_test.iloc[f,0]
            x = df_assigned_instance.iloc[f,0]
            df_confusion.iloc[y,x] = df_confusion.iloc[y,x] + 1




    # knn time
    knn = []
    df_knn = pd.DataFrame(knn)

    dataframe_holder_list = {}
    k = 0
    m = 0
    n = 0
    y = 0
    distance_knn = []
    sum_holder = 0
    min_holder = 0
    minner = []
    summer = []
    trial = []
    df_summer = pd.DataFrame(summer)
    df_minner = pd.DataFrame(minner)
    df_trial = pd.DataFrame()



    for k in range(0, 1000): #1000
        summer.clear()
        for t in range (1, 785):
            distance_knn.clear()
            for m in range (len(df_train)):

                distance_knn.append((df_test.iloc[k,t] - df_train.iloc[m, t])**2)

            df_knn[t] = distance_knn

        for n in range(0,1000):

            sum_holder = df_knn.iloc[n, :].sum()
            summer.append(sum_holder)
        df_summer[k] = summer

        #for y in range(0,1000):

            #min_holder = df_summer.iloc[:,k].idxmin()
            #minner.append(min_holder)
```

```
            #df_minner[k] = minner


        df_trial = df_summer.copy()
        #df_trial = df_trial.sort_values(by = [0], ascending = True)
        #df_trial = df_trial.set_index(0)
        storella = []
        df_storella = pd.DataFrame(storella)

        #storella.append(df_trial.index[0:11])

        storella.clear()
        t = 0
        for t in range (0,1000): #1000
            df_trial = df_trial.sort_values(by = [t], ascending = True)
            storella.append(df_trial.index[0:11])


            #####
        df_storella = pd.DataFrame(storella)
        df_storella = df_storella.transpose()

        x = 0
        y = 0
        u = 0
        o = 0
        knn_label = []
        df_knn_label = pd.DataFrame(knn_label)
        mode_holder = []
        df_mode_holder = pd.DataFrame(mode_holder)


        for x in range (0,1000): #1000
            knn_label.clear()
            for y in range (0,11):
                knn_label.append(df_train.iloc[df_storella.iloc[y,x],0])
            df_knn_label[x] = knn_label


        #mode

        mode_holder.clear()
        p = 1
        t = 0
        o = 0

        mode_holder_1 = []
        df_mode_holder_1 = pd.DataFrame(mode_holder_1)

        for p in range (1,13,2):
            for o in range (0,1000): #1000, k = 11
                mode_holder.append(df_knn_label.iloc[:p,o].mode())

                #df_mode_holder [t] = mode_holder ###

        df_mode_holder_1 = pd.DataFrame(mode_holder)


        # raw confusion matrix

        knn_holder_list = []
```

```
235.
236.        knn_confusion = [0] * 10 # number??
237.        df_knn_confusion = pd.DataFrame(knn_confusion)
238.        f = 0
239.        for f in range (1,10): # range??
240.            df_knn_confusion[f] = df_knn_confusion[0]
241.
242.
243.
244.        #k - nn confusion matrix
245.
246.        d = 0
247.        s = 0
248.        key = 0
249.        x = 0
250.        u = 0
251.        #for b in range (1,7):
252.
253.        for s in range (1000, 2000):
254.
255.            if df_test.iloc[d, 0] == df_mode_holder_1.iloc[s,0]:
256.                x = df_test.iloc[d,0]
257.                df_knn_confusion.iloc[x,x] = df_knn_confusion.iloc[x,x] + 1
258.
259.            elif (df_test.iloc[d, 0]) != (df_mode_holder_1.iloc[s,0]):
260.                x = df_test.iloc[d,0]
261.                u = int(df_mode_holder_1.iloc[s,0])
262.                df_knn_confusion.iloc[x,u] = df_knn_confusion.iloc[x,u] + 1
263.            d = d + 1
264.            s = s + 1
265.        conf_holder_df = df_knn_confusion.copy()
266.        knn_holder_list.append(conf_holder_df)
267.        key = key + 1
268.
269.        r = 0
270.        z = 0
271.        m = 0
272.        e = 0
273.        nan_hunter = []
274.        for r in range (1000,2000):
275.            for z in range (1,5):
276.                if df_mode_holder_1.iloc[r,z] >= 0:
277.                    nan_hunter.append(r)
278.
279.
280.
281.        len(set(nan_hunter))
282.
283.        """
284.
285.
286.        #mode
287.
288.        mode_holder.clear()
289.        p = 1
290.        t = 0
291.        o = 0
292.
293.        mode_holder_1 = []
294.        df_mode_holder_1 = pd.DataFrame(mode_holder_1)
295.
```

```
        for p in range (1,13,2):
            for o in range (0,1000): #1000, k = 11
                mode_holder.append(df_knn_label.iloc[:p,o].mode())

                #df_mode_holder [t] = mode_holder ###

        df_mode_holder_1 = pd.DataFrame(mode_holder)


        # raw confusion matrix

        knn_holder_list = []

        knn_confusion = [0] * 10 # number??
        df_knn_confusion = pd.DataFrame(knn_confusion)
        f = 0
        for f in range (1,10): # range??
            df_knn_confusion[f] = df_knn_confusion[0]



        #k - nn confusion matrix

        d = 0
        s = 0
        key = 0
        x = 0
        u = 0
        z=0
        #for b in range (1,7):
        for s in range (5000, 6000):

            if df_test.iloc[d, 0] == df_mode_holder_1.iloc[s,0]:
                x = df_test.iloc[d,0]
                df_knn_confusion.iloc[x,x] = df_knn_confusion.iloc[x,x] + 1

            elif (df_test.iloc[d, 0]) != (df_mode_holder_1.iloc[s,0]):
                x = df_test.iloc[d,0]
                u = int(df_mode_holder_1.iloc[s,0])
                df_knn_confusion.iloc[x,u] = df_knn_confusion.iloc[x,u] + 1
            d = d + 1
            s = s + 1
        conf_holder_df = df_knn_confusion.copy()
        knn_holder_list.append(conf_holder_df)
        key = key + 1

        r = 0
        z = 0
        m = 0
        e = 0
        g = 0
        nan_hunter = []
        nan_2 = []
        saglama = 0
        saglama_2 = []
        for r in range (5000,6000):
            for z in range (1,5):
                if df_mode_holder_1.iloc[r,z] >= 0:
                    for g in range (0,5):
                        if df_mode_holder_1.iloc[r,g] == df_knn_label.iloc[0,r-
    5000]: #Check!
```

```
                            saglama = saglama + 1
                            break
                        else:
                            nan_2.append(r)
                    nan_hunter.append(r)

        len(set(nan_hunter))
        nan_2_1 = set(nan_hunter)


        # Equal Vote Salvation: Problem Solved by manipulating modes
        # Do again now.
        p = 0
        pi = 0
        z = 0
        for p in range (5000, 6000): # do it for each k=3,5,7,9,11 for respective 1000-2000-3000 etc
            for z in range (1,5):
                if df_mode_holder_1.iloc[p, z] >= 0: #hold p
                    if p in counter_list:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[0,p-5000]
                    elif p in counter_list_2:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[1,p-5000]
                    elif p in counter_list_3:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[2,p-5000]
                    elif p in counter_list_4:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[3,p-5000]
                    elif p in counter_list_5:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[4,p-5000]
                    elif p in counter_list_6:
                        df_mode_holder_1.iloc[p,0] = df_knn_label.iloc[5,p-5000]









        # nan 2_1 check
        nan_2_1 = list(nan_2_1)
        t = 0
        counter = 0
        counter_list = []
        counter_list_2 = []
        counter_list_3 = []
        counter_list_4 = []
        counter_list_5 = []
        counter_list_6 = []


        for t in range (len(nan_2_1)):
            if nan_2_1[t] > 1000 and nan_2_1[t] < 2000:
                if df_mode_holder_1.iloc[nan_2_1[t],0] == df_knn_label.iloc[0,nan_2_1[t]-1000]:
                    counter = counter + 1
                    counter_list_3.append(nan_2_1[t])
            elif nan_2_1[t] > 2000 and nan_2_1[t] < 3000:
                if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[t]-2000]:
```

```
                    counter = counter + 1
                    counter_list_6.append(nan_2_1[t])
            elif nan_2_1[t] > 3000 and nan_2_1[t] < 4000:
                if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[t]
    -3000]:
                    counter = counter + 1
                    counter_list_6.append(nan_2_1[t])
            elif nan_2_1[t] > 4000 and nan_2_1[t] < 5000:
                if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[t]
    -4000]:
                    counter = counter + 1
                    counter_list_6.append(nan_2_1[t])
            elif nan_2_1[t] > 5000 and nan_2_1[t] < 6000:
                if df_mode_holder_1.iloc[nan_2_1[t],4] == df_knn_label.iloc[5,nan_2_1[t]
    -5000]:
                    counter = counter + 1
                    counter_list_6.append(nan_2_1[t])

        len(set(counter_list_2))

        in_counter_list = set(counter_list)
        in_counter_list_2 = set(counter_list_2)
        in_counter_list_3 = set(counter_list_3)

        insecondnotfirst = in_counter_list_2 - in_counter_list
        insecondnotthird = in_counter_list_2 - in_counter_list_3
        infirstnotsecond = in_counter_list - in_counter_list_2
        infirstnotsthird = in_counter_list - in_counter_list_3
        inthirdnotfirst = in_counter_list_3 - in_counter_list

        result_list =
```

Image Visualization

```
from matplotlib import pyplot
import numpy as np
import csv
import matplotlib.pyplot as plt
k = 0
for k in (0,9):

    label = df_pixel_mean_transpose[k]

    pixels_mean = df_pixel_mean_transpose[1:]

    pixels_mean = np.array(pixels_mean, dtype='uint8')

    deneme = pixels_mean[:,k].reshape(28, 28)   #CHANGE k based on numbers

    plt.title('Label is {label}'.format(label=label))
    plt.imshow(deneme, cmap='gray')
    plt.show()
    k = k + 1


deneme = []
```

5. References

1. Alpaydın, 2014.