Project report

Machine Learning ELC

BY

Kushagr Sharma - 102203714

HANDWRITTEN DIGIT RECOGNITION USING k-NN

Department of computer science & engineering

Thapar Insitute of Engineering & Technology, Patiala

June, 2024

Table of Contents

Contents

1 Introduction

In this project, *Handwritten Digit Recognition*, I am using K-Nearest Neighbours algorithm to make classifications or predictions about the grouping of an individual data point. It will find the nearest neighbours from the training set to test the image and assign the class by majority.

1.1 About k-NN

K-Nearest Neighbors `(k-NN)` is a simple, non-parametric, instance-based and lazy learning algorithm used for classification and regression. It works by memorizing the training data and making predictions based on the 'k' nearest neighbors in the feature space, using distance metrics like Euclidean distance. For classification, it assigns the most common class among the neighbors, and for regression, it averages their values. k-NN is easy to implement and understand but can be computationally intensive for large datasets since it requires calculating distances to all training points during prediction. It is also sensitive to the scale of features, necessitating normalization. Despite its simplicity, k-NN is effective in applications like pattern recognition, recommendation systems, and anomaly detection.

A MNIST dataset is loaded, split into training and testing sets and then k-NN is used to classify the test images. The k-NN classifier is trained on the training set, where it memorizes the digit labels of the images. During prediction, the classifier computes the distance between a test image and all training images to find the nearest neighbors. It then predicts the digit label based on the majority label among these neighbors.

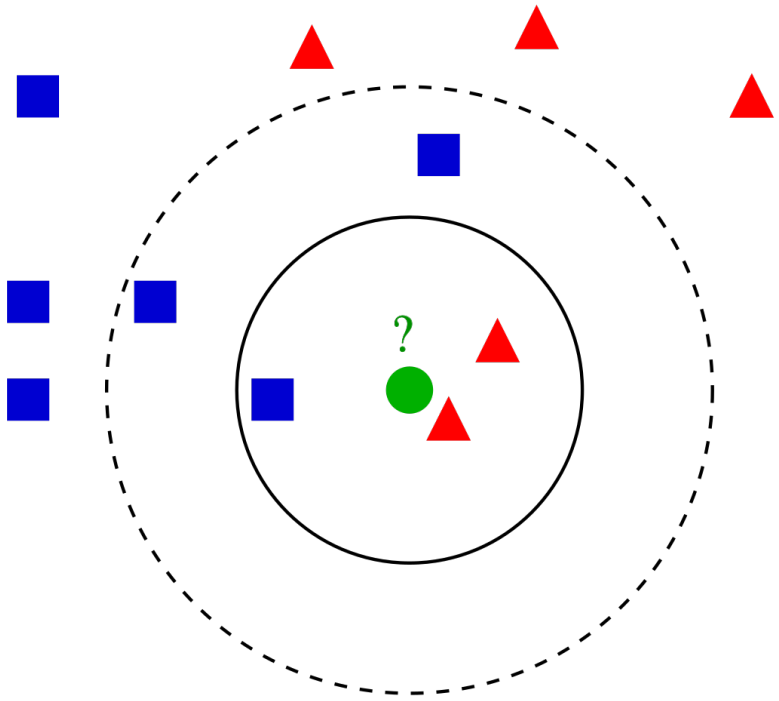

Figure 1: k-NN classification

1.2 Handwritten Digit Recognition

Handwritten digit recognition is an advanced technology aimed at automatically identifying and classifying handwritten digits, typically ranging from 0 to 9. By converting handwritten inputs into digital text, it enables more efficient storage, retrieval, and analysis of information. The handwritten digit recognition process utilizes machine learning techniques to accurately interpret and transform handwritten digits into their corresponding numeric values.

To recognize images of Handwritten digits based on classification methods for multivariate data we use **Optical Character Recognition** (OCR) technology. OCR technology is extensively used in the domain of handwritten digit recognition. This application involves converting handwritten digits into digital text, which can then be processed and analyzed by computer systems.

The MNIST (Modified National Institute of Standards and Technology) dataset is a benchmark dataset widely used in the field of machine learning and OCR for handwritten digit recognition. It consists of 70,000 gray-scale images of handwritten digits, each 28x28 pixels in size, with 60,000 images for training and 10,000 images for testing. Each image is labeled with the corresponding digit it represents, ranging from 0 to 9. The dataset serves as a standard for evaluating and comparing the performance of different OCR and machine learning algorithms.

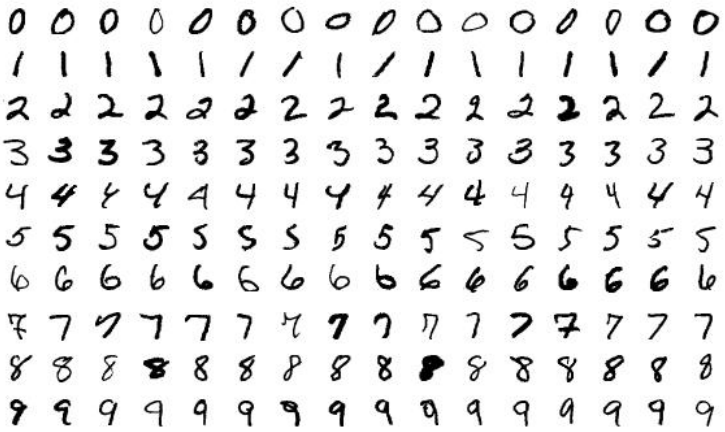

Figure 2: 0-9 Handwritten Digits

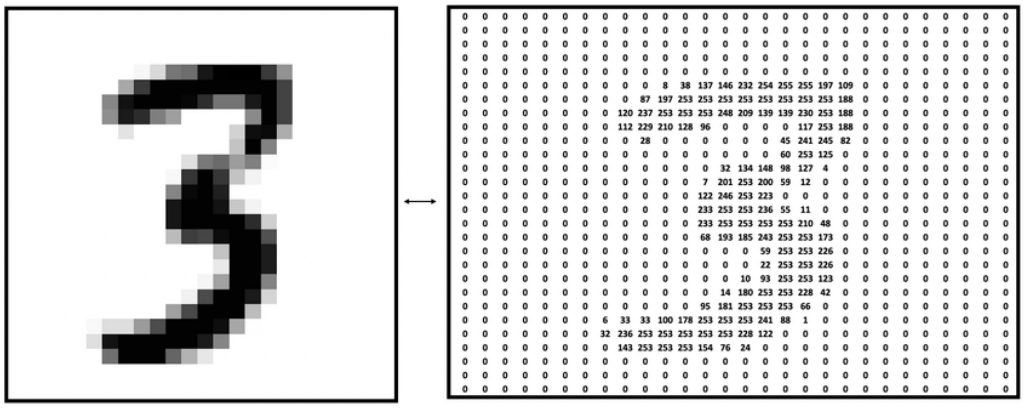

Figure 3: Representation in pixel matrix

2 Model Implementation

To implement this model using k-NN algorithm I have used *Python*(Jupyter Source File) in Visual Studio Code. We were given different k-values and train to test ratios to split the dataset. I have chosen `k = 2` and Train to Test ratio of `60:40` for this model. The source code is given below :

2.1 Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

- **Pandas** : Data manipulation and analysis library for Python, providing data structures like DataFrame for handling tabular data.
- **Numpy** : A library for numerical computing in Python, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- **MatPlotLib** : A plotting library for Python that enables the creation of static, interactive, and animated visualizations, including plots, graphs, and charts.
- **Seaborn** : A data visualization library built on top of matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- **sklearn.neighbors.KNeighborsClassifier** : A machine learning classifier provided by scikit-learn that implements the K-Nearest Neighbors algorithm for classification tasks.
- **sklearn.metrics.accuracy_score** : A function in scikit-learn for evaluating the accuracy of a classification model by comparing the predicted labels to the true labels.
- **sklearn.metrics.classification_report** : A function in scikit-learn that generates a detailed report showing the main classification metrics such as precision, recall, and F1-score.
- **sklearn.metrics.confusion_matrix** : A function in scikit-learn that computes the confusion matrix to evaluate the performance of a classification algorithm.
- **sklearn.model_selection.train_test_split** : A function in scikit-learn that splits a dataset into training and testing subsets, facilitating the evaluation of machine learning models.

2.2 Reading from dataset

```
file="data.csv"
df = pd.read_csv(file)
df.head()
```

df.head() returns a specified number of rows from the top of the dataset. The head() method returns the first 5 rows if a number is not specified.

2.3 Model Building

```
df.shape
y = df['label']
x = df.drop('label', axis = 1)

plt.figure(figsize=(5,5))
row = 420
row_img = x.iloc[row].to_numpy()
plt.imshow(np.reshape(row_img, (28,28)))
print(y[row])

custom_palette = ['#ff0000', '#00ff00', '#0000ff', '#ffff00', '#ff00ff', '#11ccff',
'#800000', '#008000', '#000080', '#808000']
sb.countplot(x='label', data = df, palette = custom_palette, legend = False, hue = 'label')
```

- **df.shape** prints the number of rows and column of the dataframe
- **Lines 5-9** are for displaying the image of a digit in i_{th} row (here i = 420). The row is converted to a numpy array and is re-shaped to a 28x28 matrix.

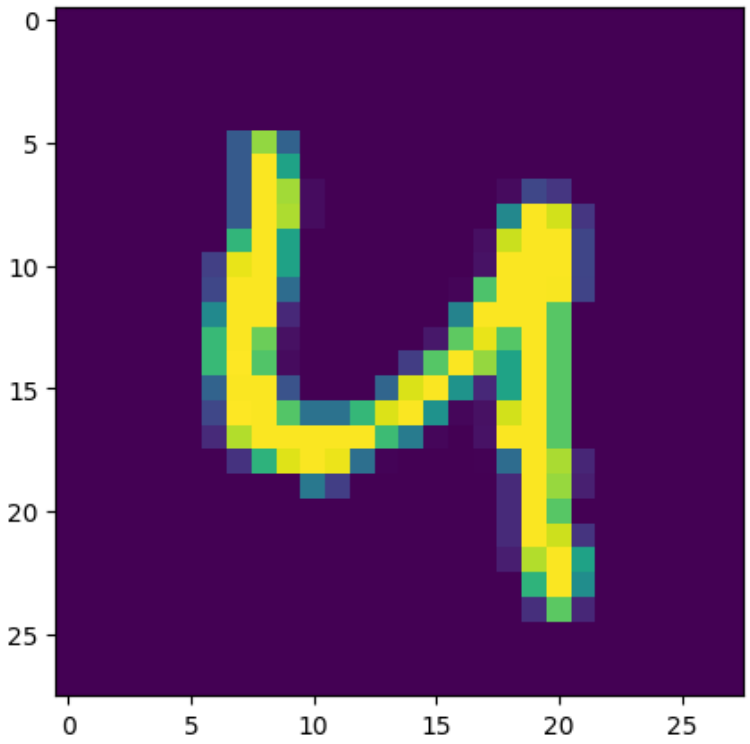

Figure 4: Digit at row 420

- **Lines 11-13** are for creating a count plot of the 'label' column using the custom pallete.

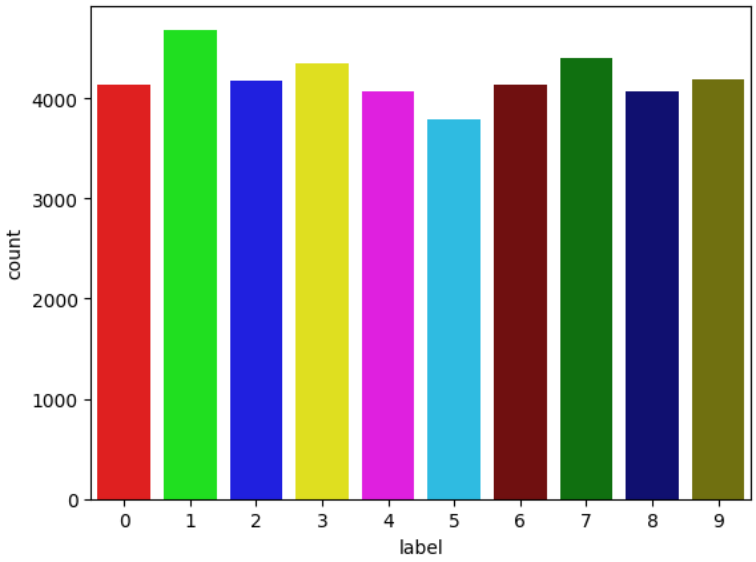

Figure 5: Digit count in dataset

2.4 k-NN Model implementation

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.40, random_state = 24)
x_train.shape, y_train.shape, x_test.shape, y_test.shape

knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train, y_train)

prediction = knn.predict(x_test)
prediction
```

Using k-NN algorithm by importing libraries from Python. The dataset is split into 60:40 ratio (train:test) and it is predicted on `k = 2` nearest neighbours.

2.5 Model Evaluation Parameters

```
print(accuracy_score(y_test, prediction))

print(classification_report(y_test, prediction))

print(confusion_matrix(y_test, prediction))
```

- The accuracy for the model with given parameters turned out to be : **0.9569642857142857**

- **Classification Report**

	Precision	Recall	F1-Score	Support
0	0.96	1.00	0.98	1630
1	0.95	1.00	0.97	1891
2	0.96	0.96	0.96	1688
3	0.93	0.97	0.95	1801
4	0.95	0.97	0.96	1648
5	0.94	0.94	0.94	11500
6	0.98	0.98	0.98	1649
7	0.95	0.97	0.96	1763
8	0.99	0.87	0.93	1562
9	0.97	0.91	0.94	1668
Accuracy			0.96	16800
Macro avg	0.96	0.96	0.96	16800
Weighted avg	0.96	0.96	0.96	16800

- **Confusion Matrix**

Actual \ **Predicted**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1625	0	1	0	0	0	3	0	0	1
1	0	1822	4	0	1	0	1	3	0	0
2	19	25	1616	4	1	0	0	21	1	1
3	3	5	23	1739	1	14	0	5	7	4
4	3	22	0	0	1594	0	6	4	0	19
5	5	5	2	62	2	1415	5	0	0	4
6	16	2	1	0	3	13	1614	0	0	0
7	2	24	9	0	8	0	0	1715	0	5
8	13	21	18	54	7	52	17	8	1363	9
9	9	5	2	11	57	11	2	49	8	1514

3 Performance analysis

Since the task was to evaluate the performance of the model over different parameters, *viz.*, k-values and train:test ratio, the sections below contain the accuracy score and the confusion matrix of the different scenarios.

3.1 k=2

3.1.1 70:30

Accuracy : 0.9588095238095238
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1239	0	0	0	0	0	1	0	0	1
1	0	1399	2	1	1	0	1	2	0	0
2	13	15	1190	3	1	1	0	13	0	1
3	2	5	15	1292	1	8	0	3	3	3
4	1	16	0	0	1205	0	3	2	0	13
5	3	3	1	44	2	1062	4	0	0	5
6	9	1	0	0	3	7	1222	0	0	0
7	2	20	8	0	7	0	0	1302	0	4
8	12	14	15	42	5	39	13	5	1008	10
9	6	2	2	8	39	8	1	39	5	1162

3.1.2 75:25

Accuracy : 0.9600952380952381
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1027	0	0	0	0	0	1	0	0	1
1	0	1161	1	0	1	0	1	2	0	0
2	11	11	999	2	1	0	1	8	1	1
3	1	4	11	1103	1	5	0	2	2	3
4	1	11	0	0	992	0	3	1	0	11
5	3	3	1	39	1	886	2	0	0	5
6	6	1	0	0	3	5	1000	0	0	0
7	1	17	6	0	6	0	0	1101	0	4
8	11	8	10	32	4	27	14	4	837	9
9	6	1	2	7	35	6	1	36	4	975

3.1.3 80:20

Accuracy : 0.96
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	816	0	0	0	0	0	1	0	0	1
1	0	948	0	0	0	0	2	0	0	0
2	10	10	811	2	1	0	1	7	1	0
3	1	3	9	878	1	3	0	1	3	3
4	1	10	0	0	798	0	1	0	0	10
5	2	3	0	29	1	705	3	0	0	3
6	7	1	0	0	3	3	800	0	0	0
7	1	15	4	0	5	0	0	881	0	4
8	9	9	7	29	3	17	12	4	649	7
9	4	2	1	4	26	5	1	27	3	778

3.1.4 90:10

Accuracy : 0.9669047619047619
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	404	0	0	0	0	0	0	0	0	0
1	0	484	0	0	0	0	0	0	0	0
2	5	3	413	1	0	0	0	2	0	0
3	0	1	6	439	1	0	0	1	1	1
4	1	8	0	0	396	0	0	1	0	3
5	1	3	0	11	0	364	0	0	1	2
6	4	0	0	0	2	3	397	0	0	0
7	0	6	2	0	1	0	0	448	0	2
8	3	5	5	12	1	8	4	0	318	3
9	3	0	0	2	6	2	1	11	0	398

3.1.5 95:5

Accuracy : 0.9671428571428572
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	213	0	0	0	0	0	0	0	0	0
1	0	242	0	0	0	0	0	0	0	0
2	2	3	199	1	1	0	0	2	0	0
3	0	1	1	211	0	0	0	1	0	0
4	1	4	0	0	199	0	0	0	0	1
5	0	3	0	5	0	172	0	0	0	2
6	3	0	0	0	0	1	199	0	0	0
7	0	3	1	0	0	0	0	209	0	0
8	1	5	3	6	0	4	1	0	171	0
9	1	0	0	1	3	1	0	7	0	216

3.2 k=4

3.2.1 60:40

Accuracy : 0.9657142857142857
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1623	0	0	0	0	0	4	1	0	2
1	0	1882	3	0	1	0	2	3	0	0
2	19	23	1605	4	0	2	1	28	4	2
3	2	6	9	1746	0	13	0	8	9	8
4	3	18	0	0	1593	0	5	4	0	25
5	4	3	3	30	1	1440	10	2	0	7
6	11	1	1	0	2	6	1628	0	0	0
7	1	25	8	0	4	0	0	1713	0	12
8	9	18	10	38	7	33	14	7	1414	12
9	4	3	3	10	27	7	1	30	3	1580

3.2.2 70:30

Accuracy : 0.9647619047619047
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1238	0	0	0	0	0	1	1	0	1
1	0	1399	1	1	1	0	1	3	0	0
2	14	15	1174	2	0	1	2	24	3	2
3	2	5	7	1287	0	10	0	7	7	7
4	1	14	0	0	1202	0	3	1	0	19
5	3	4	1	27	1	1077	5	1	0	5
6	8	1	0	0	2	5	1226	0	0	0
7	1	22	5	0	4	0	0	1303	0	8
8	8	12	8	30	4	23	13	4	1048	13
9	3	1	3	6	24	4	1	25	3	1202

3.2.3 75:25

Accuracy : 0.9655238095238096
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1026	0	0	0	0	0	1	1	0	1
1	0	1160	2	0	1	0	1	2	0	0
2	10	14	989	0	0	0	2	15	3	2
3	1	4	4	1099	0	6	0	4	7	7
4	1	11	0	0	987	0	3	0	0	17
5	2	3	1	24	1	899	4	1	0	5
6	6	1	0	0	2	3	1003	0	0	0
7	0	18	3	0	3	0	0	1104	0	7
8	6	8	6	19	5	22	11	3	867	9
9	3	1	2	7	23	4	1	25	3	1004

3.2.4 80:20

Accuracy : 0.9658333333333333
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	815	0	0	0	0	0	1	1	0	1
1	0	949	0	0	0	0	1	0	0	0
2	9	11	805	0	0	0	2	13	2	1
3	1	4	4	877	0	4	0	2	5	5
4	1	10	0	0	794	0	1	0	0	14
5	2	2	0	16	1	717	6	0	0	2
6	4	1	0	0	2	2	805	0	0	0
7	0	15	3	0	3	0	0	883	0	6
8	5	9	6	16	3	19	9	3	668	8
9	2	1	2	5	15	4	1	19	2	800

3.2.5 90:10

Accuracy : 0.97
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	402	0	0	0	0	0	1	0	0	1
1	0	483	0	0	0	0	1	0	0	0
2	2	3	410	0	1	0	0	7	1	0
3	0	2	2	441	1	1	0	1	1	1
4	0	8	0	0	391	0	1	0	0	9
5	1	2	0	8	1	369	0	0	0	1
6	3	0	0	0	2	2	399	0	0	0
7	0	5	3	0	0	0	0	449	0	2
8	2	3	3	9	1	9	3	0	325	4
9	1	0	0	3	3	2	1	8	0	405

3.2.6 95:5

Accuracy : 0.9704761904761905
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	212	0	0	0	0	0	0	0	0	1
1	0	242	0	0	0	0	0	0	0	0
2	1	3	199	0	1	0	0	4	0	0
3	0	2	0	210	0	1	0	1	0	0
4	0	4	0	0	196	0	1	0	0	4
5	0	2	0	3	1	175	0	0	0	1
6	2	0	0	0	0	1	200	0	0	0
7	0	3	2	0	0	0	0	207	0	1
8	1	3	2	4	0	3	1	0	176	1
9	0	0	0	2	1	1	0	4	0	221

3.3 k=5

3.3.1 60:40

Accuracy : 0.965654761904762
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1622	0	0	0	0	0	5	1	0	2
1	0	1879	4	1	1	0	2	4	0	0
2	12	23	1601	7	2	1	3	32	5	2
3	1	7	7	1737	0	21	1	8	9	10
4	3	17	0	0	1577	0	4	2	0	45
5	5	2	2	25	1	1440	15	1	1	8
6	9	1	1	0	2	6	1630	0	0	0
7	1	23	7	1	3	0	0	1709	0	19
8	10	16	8	29	6	34	10	7	1431	11
9	4	5	3	9	18	7	1	22	2	1597

3.3.2 70:30

Accuracy : 0.9665873015873016
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1237	0	0	0	0	0	2	1	0	1
1	0	1398	2	1	1	0	1	3	0	0
2	12	14	1173	2	1	1	2	27	3	2
3	0	5	7	1282	0	18	1	5	6	8
4	1	14	0	0	1195	0	3	0	0	27
5	1	2	1	21	1	1079	10	1	1	7
6	5	1	0	0	2	5	1229	0	0	0
7	1	20	5	1	2	0	0	1299	0	15
8	8	12	5	22	4	21	9	5	1065	12
9	2	2	3	5	14	3	1	16	4	1222

3.3.3 75:25

Accuracy : 0.9680952380952381
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1026	0	0	0	0	0	1	1	0	1
1	0	1160	2	0	1	0	1	2	0	0
2	8	12	989	1	1	0	2	17	3	2
3	0	4	4	1095	0	12	0	4	6	7
4	1	11	0	0	982	0	3	1	0	21
5	2	3	1	18	1	899	8	1	0	7
6	3	1	0	0	2	3	1006	0	0	0
7	0	18	3	0	1	0	0	1102	0	11
8	6	10	4	14	4	19	7	3	879	10
9	2	2	1	6	12	3	1	16	3	1027

3.3.4 80:20

Accuracy : 0.9677380952380953
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	815	0	0	0	0	0	1	1	0	1
1	0	947	1	0	1	0	1	0	0	0
2	6	11	805	1	0	0	2	15	2	1
3	0	4	4	871	0	9	0	2	6	6
4	1	10	0	0	790	0	2	0	0	17
5	2	2	0	11	1	718	8	1	0	3
6	3	1	0	0	2	2	806	0	0	0
7	0	16	2	0	1	0	0	879	0	12
8	4	9	4	13	4	13	6	3	681	9
9	2	1	1	4	6	3	1	13	3	817

3.3.5 90:10

Accuracy : 0.9707142857142858
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	403	0	0	0	0	0	0	0	0	1
1	0	483	0	0	0	0	1	0	0	0
2	2	2	409	0	1	0	1	8	1	0
3	0	2	2	440	1	2	0	1	1	1
4	1	8	0	0	390	0	0	0	0	10
5	1	2	0	5	1	371	1	0	0	1
6	1	0	0	0	1	2	402	0	0	0
7	0	6	2	0	0	0	0	444	0	7
8	2	4	2	7	1	7	2	0	330	4
9	1	0	0	3	3	2	1	8	0	405

3.3.6 95:5

Accuracy : 0.9723809523809523
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	212	0	0	0	0	0	0	0	0	1
1	0	242	0	0	0	0	0	0	0	0
2	1	1	198	0	2	0	1	5	0	0
3	0	2	0	210	0	1	0	1	0	0
4	1	4	0	0	196	0	0	0	0	4
5	0	2	0	2	1	176	0	0	0	1
6	1	0	0	0	0	1	201	0	0	0
7	0	3	1	0	0	0	0	206	0	3
8	1	3	2	3	0	2	0	0	179	1
9	0	0	0	1	1	1	0	4	0	222

3.4 k=6

3.4.1 60:40

Accuracy : 0.9639880952380953
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1621	0	0	0	0	0	6	1	0	2
1	0	1880	4	1	0	0	3	3	0	0
2	19	27	1592	9	2	3	3	28	3	2
3	2	7	8	1744	0	13	1	8	9	9
4	3	18	0	0	1586	0	5	3	0	33
5	4	3	2	32	1	1432	13	1	2	10
6	11	1	1	0	2	7	1627	0	0	0
7	1	26	5	0	6	0	0	1708	0	17
8	10	23	6	41	8	25	11	7	1418	13
9	4	5	2	12	18	5	1	31	3	1587

3.4.2 70:30

Accuracy : 0.9664285714285714
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1237	0	0	0	0	0	2	1	0	1
1	0	1400	2	1	0	0	1	2	0	0
2	15	17	1170	3	1	1	3	23	2	2
3	1	5	7	1287	0	13	1	5	6	7
4	1	14	0	0	1199	0	3	0	0	23
5	3	3	1	23	0	1079	6	1	1	7
6	5	1	0	0	2	6	1228	0	0	0
7	1	21	6	0	1	0	0	1302	0	12
8	7	17	4	29	4	16	8	5	1062	11
9	3	2	2	9	14	3	1	22	3	1213

3.4.3 75:25

Accuracy : 0.9680952380952381
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1025	0	0	0	0	0	2	1	0	1
1	0	1163	1	0	0	0	1	1	0	0
2	10	13	988	1	1	0	3	15	2	2
3	0	4	5	1099	0	9	0	3	5	7
4	1	12	0	0	985	0	3	0	0	18
5	1	3	1	19	0	901	8	1	0	6
6	3	1	0	0	2	3	1006	0	0	0
7	0	16	4	0	1	0	0	1107	0	7
8	6	15	5	20	3	16	6	4	871	10
9	2	2	1	9	12	3	1	20	3	1020

3.4.4 80:20

Accuracy : 0.9683333333333334
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	814	0	0	0	0	0	2	1	0	1
1	0	948	1	0	0	0	1	0	0	0
2	9	11	802	1	0	0	3	14	2	1
3	0	4	4	876	0	6	0	2	5	5
4	1	10	0	0	794	0	1	0	0	14
5	1	2	0	13	0	721	6	1	0	2
6	3	1	0	0	2	2	806	0	0	0
7	0	16	2	0	1	0	0	883	0	8
8	5	13	5	16	3	11	6	4	675	8
9	2	1	1	5	7	3	1	14	2	815

3.4.5 90:10

Accuracy : 0.9707142857142858
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	403	0	0	0	0	0	0	0	0	1
1	0	483	0	0	0	0	1	0	0	0
2	2	2	409	0	0	0	1	9	1	0
3	0	2	2	440	0	2	0	1	2	1
4	0	8	0	0	391	0	1	0	0	9
5	1	2	0	7	1	369	1	0	0	1
6	1	0	0	0	1	2	402	0	0	0
7	0	6	2	0	0	0	0	446	0	5
8	2	5	2	10	1	3	2	1	331	2
9	1	0	0	3	4	2	1	9	0	403

3.4.6 95:5

Accuracy : 0.97
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	212	0	0	0	0	0	0	0	0	1
1	0	242	0	0	0	0	0	0	0	0
2	1	2	198	0	0	0	1	6	0	0
3	0	2	0	210	0	1	0	1	0	0
4	0	4	0	0	196	0	1	0	0	4
5	0	2	0	3	1	175	0	0	0	1
6	1	0	0	0	0	1	201	0	0	0
7	0	3	1	0	0	0	0	206	0	3
8	1	4	1	5	1	1	1	0	177	0
9	0	0	0	2	2	1	0	4	0	220

3.5 k=7

3.5.1 60:40

Accuracy : 0.9638095238095238
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1621	0	1	0	0	0	5	0	0	3
1	0	1879	4	1	0	0	3	4	0	0
2	15	28	1590	9	2	2	3	32	5	2
3	2	7	8	1738	0	17	1	9	9	10
4	3	20	0	0	1571	0	6	2	0	46
5	5	4	1	26	1	1433	16	1	1	12
6	9	1	1	0	2	7	1629	0	0	0
7	1	26	5	0	5	0	0	1704	0	22
8	10	20	5	30	6	23	10	6	1434	18
9	4	5	2	11	16	5	1	28	3	1593

3.5.2 70:30

Accuracy : 0.9657142857142857
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1237	0	0	0	0	0	2	0	0	2
1	0	1399	2	1	0	0	1	3	0	0
2	13	15	1169	3	2	1	3	26	3	2
3	1	5	6	1282	0	18	1	5	6	8
4	1	17	0	0	1187	0	4	1	0	30
5	3	3	1	17	0	1079	10	1	1	9
6	4	1	0	0	2	6	1229	0	0	0
7	1	21	4	0	1	0	0	1300	0	16
8	6	14	3	25	3	19	10	5	1063	15
9	3	3	2	7	9	3	1	19	2	1223

3.5.3 75:25

Accuracy : 0.9674285714285714
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1024	0	0	0	0	1	2	0	0	2
1	0	1160	3	0	0	0	1	2	0	0
2	9	13	988	1	1	0	3	15	3	2
3	0	4	4	1092	0	14	0	4	7	7
4	1	13	0	0	974	0	4	0	0	27
5	2	3	1	17	0	899	10	1	0	7
6	1	1	0	0	2	3	1008	0	0	0
7	0	17	3	0	0	0	0	1103	0	12
8	6	12	3	16	1	16	8	4	879	11
9	3	2	1	7	6	2	1	18	2	1031

3.5.4 80:20

Accuracy : 0.9666666666666667
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	813	0	0	0	0	1	2	0	0	2
1	0	948	1	0	0	0	1	0	0	0
2	8	11	800	1	1	0	4	14	3	1
3	0	4	4	871	0	8	0	3	7	5
4	1	11	0	0	785	0	2	0	0	21
5	2	2	0	14	1	713	9	1	0	4
6	1	1	0	0	2	2	808	0	0	0
7	0	15	2	0	0	0	0	881	0	12
8	5	11	2	15	2	11	8	4	680	8
9	2	1	2	4	3	2	1	13	2	821

3.5.5 90:10

Accuracy : 0.9702380952380952
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	402	0	0	0	0	0	1	0	0	1
1	0	483	0	0	0	0	1	0	0	0
2	2	2	406	0	1	0	1	11	1	0
3	0	2	1	440	0	2	0	1	3	1
4	1	8	0	0	389	0	0	0	0	11
5	1	2	0	7	0	368	1	1	0	2
6	0	0	0	0	1	2	403	0	0	0
7	0	5	2	0	0	0	0	445	0	7
8	2	4	0	8	1	4	2	1	334	3
9	1	0	0	2	3	3	1	8	0	405

3.5.6 95:5

Accuracy : 0.97

Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	212	0	0	0	0	0	0	0	0	1
1	0	242	0	0	0	0	0	0	0	0
2	1	2	195	0	2	0	1	7	0	0
3	0	2	0	210	0	1	0	1	0	0
4	1	4	0	0	195	0	0	0	0	5
5	0	2	0	3	0	175	0	0	0	2
6	0	0	0	0	0	1	202	0	0	0
7	0	3	1	0	0	0	0	205	0	4
8	1	4	0	3	0	1	1	0	180	1
9	0	0	0	1	1	2	0	4	0	221

3.6 k=10

3.6.1 60:40

Accuracy : 0.9610714285714286
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1622	0	1	0	0	0	5	1	0	1
1	0	1880	4	1	0	0	3	3	0	0
2	18	38	1579	8	2	4	4	29	5	1
3	2	11	9	1730	0	18	0	10	10	11
4	2	24	0	0	1566	0	9	4	0	43
5	4	6	1	29	2	1428	17	2	1	10
6	12	3	1	0	2	7	1624	0	0	0
7	2	29	5	0	4	0	0	1706	0	17
8	13	24	4	32	7	27	11	6	1419	19
9	6	6	2	11	16	4	1	28	2	1592

3.6.2 70:30

Accuracy : 0.9645238095238096
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1236	0	0	0	0	0	3	1	0	1
1	0	1399	2	1	0	0	1	3	0	0
2	16	22	1159	4	1	4	3	25	2	1
3	1	7	7	1277	0	18	0	6	6	10
4	0	15	0	0	1191	0	5	2	0	27
5	1	4	1	20	1	1079	11	1	0	6
6	6	2	0	0	2	7	1225	0	0	0
7	1	21	3	0	2	0	0	1305	0	11
8	7	17	3	27	6	14	11	4	1058	16
9	3	2	2	8	10	3	1	18	1	1224

3.6.3 75:25

Accuracy : 0.9654285714285714
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1025	0	0	0	0	0	2	1	0	1
1	0	1162	2	0	0	0	1	1	0	0
2	12	18	977	3	1	2	3	16	2	1
3	0	6	6	1086	0	16	0	5	7	6
4	0	13	0	0	974	0	5	3	0	24
5	1	3	0	18	0	903	8	1	0	6
6	3	2	0	0	2	4	1004	0	0	0
7	0	18	2	0	1	0	0	1106	0	8
8	5	14	3	17	3	13	9	4	872	16
9	3	3	1	8	6	2	1	20	1	1028

3.6.4 80:20

Accuracy : 0.9658333333333333
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	814	0	0	0	0	0	2	1	0	1
1	0	948	1	0	0	0	1	0	0	0
2	8	13	796	2	1	1	4	15	2	1
3	0	5	4	869	0	10	0	2	6	6
4	0	13	0	0	784	0	3	2	0	18
5	1	2	0	11	1	721	6	1	0	3
6	2	1	0	0	2	2	807	0	0	0
7	0	18	2	0	1	0	0	881	0	8
8	4	12	3	18	3	9	9	4	672	12
9	3	2	1	5	2	3	1	12	1	821

3.6.5 90:10

Accuracy : 0.9685714285714285
Confusion matrix :

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	402	0	0	0	0	0	1	0	0	1
1	0	483	0	0	0	0	1	0	0	0
2	2	5	404	0	1	0	1	10	1	0
3	0	2	2	439	1	2	0	1	2	1
4	0	9	0	0	389	0	1	2	0	8
5	2	2	0	8	0	367	1	1	0	1
6	2	0	0	0	1	2	401	0	0	0
7	0	6	2	0	0	0	0	449	0	2
8	2	4	2	9	0	5	4	1	327	5
9	1	0	0	3	1	2	1	8	0	407

3.6.6 95:5

Accuracy : 0.9680952380952381
Confusion matrix :

Actual \ Predicted	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	212	0	0	0	0	0	0	0	0	1
1	0	242	0	0	0	0	0	0	0	0
2	1	4	195	0	1	0	1	6	0	0
3	0	2	0	210	0	1	0	1	0	0
4	0	5	0	0	195	0	1	0	0	4
5	1	2	0	3	0	173	2	0	0	1
6	1	0	0	0	0	1	201	0	0	0
7	0	3	1	0	0	0	0	208	0	1
8	1	4	2	3	0	3	2	0	175	1
9	0	0	0	2	0	1	0	4	0	222

3.7 Graphical analysis