CS6301 MACHINE LEARNING LAB WEEK – 8 KNN, KMEANS, ENSEMBLE LEARNING

SRIHARI. S – 2018103601

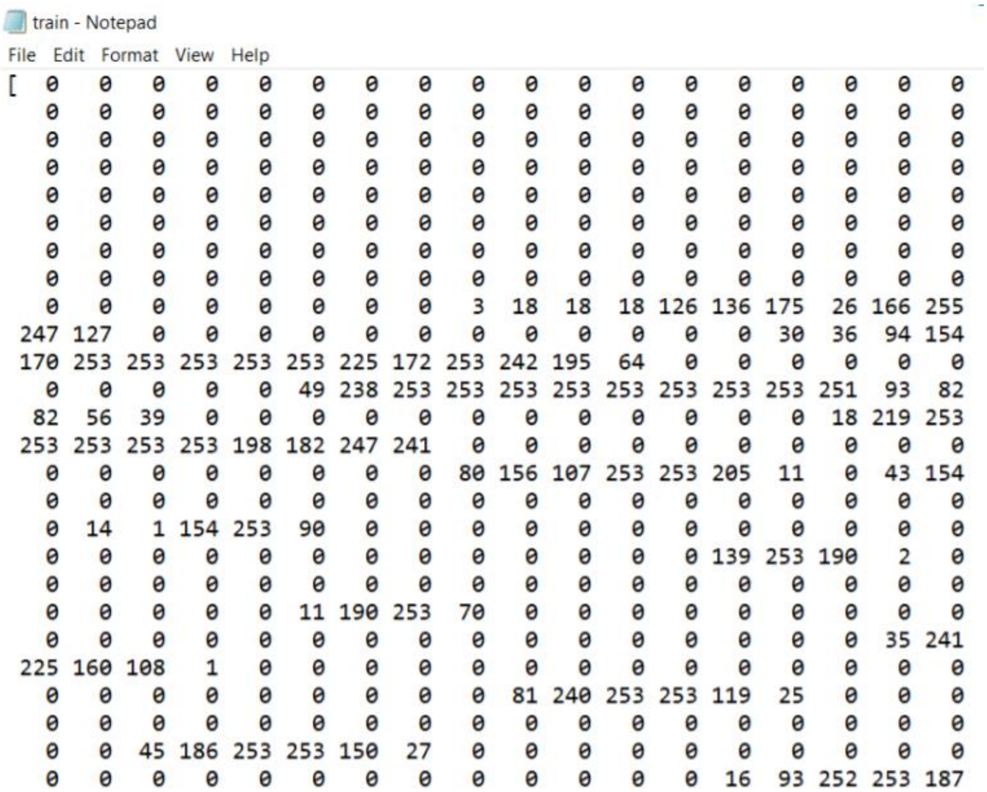
Date: 10-04-2021 Saturday

Aim: To implement ensemble learning algorithm using various classifiers – SVM, MLP, Decision Trees and KNN and determine the results.

Dataset: MNIST Dataset

The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems which contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

Input:


train - Notepad
File Edit Format View Help

```
[   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   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  18  18  18 126 136 175  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0  11 190 253  70   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  35 241
  225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
```

K-MEANS ALGORITHM

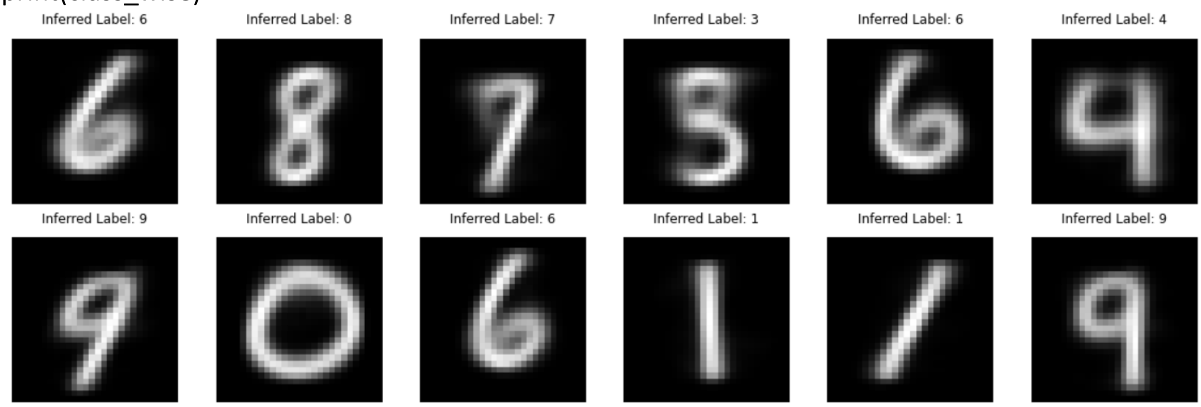
```
import sys
import sklearn
import matplotlib
```

```
import numpy as np

from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('Training Data: {}'.format(x_train.shape))
print('Training Labels: {}'.format(y_train.shape))

print('Testing Data: {}'.format(x_test.shape))
print('Testing Labels: {}'.format(y_test.shape))


import matplotlib.pyplot as plt

# python magic function
%matplotlib inline

# create figure with 3x3 subplots using matplotlib.pyplot
fig, axs = plt.subplots(3, 3, figsize = (12, 12))
plt.gray()

# loop through subplots and add mnist images
for i, ax in enumerate(axs.flat):
    ax.matshow(x_train[i])
    ax.axis('off')
    ax.set_title('Number {}'.format(y_train[i]))

# display the figure
fig.show()

# convert each image to 1 dimensional array
X = x_train.reshape(len(x_train),-1)
Y = y_train

# normalize the data to 0 - 1
X = X.astype(float) / 255.

print(X.shape)
print(X[0].shape)

from sklearn.cluster import MiniBatchKMeans
n_digits = len(np.unique(y_test))
print(n_digits)

# Initialize KMeans model
kmeans = MiniBatchKMeans(n_clusters = n_digits)

# Fit the model to the training data
kmeans.fit(X)
```

```python
def infer_cluster_labels(kmeans, actual_labels):
    inferred_labels = {}

    for i in range(kmeans.n_clusters):

        # find index of points in cluster
        labels = []
        index = np.where(kmeans.labels_ == i)

        # append actual labels for each point in cluster
        labels.append(actual_labels[index])

        # determine most common label
        if len(labels[0]) == 1:
            counts = np.bincount(labels[0])
        else:
            counts = np.bincount(np.squeeze(labels))

        # assign the cluster to a value in the inferred_labels dictionary
        if np.argmax(counts) in inferred_labels:
            # append the new number to the existing array at this slot
            inferred_labels[np.argmax(counts)].append(i)
        else:
            # create a new array in this slot
            inferred_labels[np.argmax(counts)] = [i]

        #print(labels)
        #print('Cluster: {}, label: {}'.format(i, np.argmax(counts)))

    return inferred_labels

def infer_data_labels(X_labels, cluster_labels):
    """
    Determines label for each array, depending on the cluster it has been assigned to.
    returns: predicted labels for each array
    """

    # empty array of len(X)
    predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)

    for i, cluster in enumerate(X_labels):
        for key, value in cluster_labels.items():
            if cluster in value:
                predicted_labels[i] = key

    return predicted_labels


# test the infer_cluster_labels() and infer_data_labels() functions
cluster_labels = infer_cluster_labels(kmeans, Y)
X_clusters = kmeans.predict(X)
```

```
predicted_labels = infer_data_labels(X_clusters, cluster_labels)
print(predicted_labels[:20])
print(Y[:20])

from sklearn import metrics

def calculate_metrics(estimator, data, labels):

    # Calculate and print metrics
    print('Number of Clusters: {}'.format(estimator.n_clusters))
    print('Inertia: {}'.format(estimator.inertia_))
    print('Homogeneity: {}'.format(metrics.homogeneity_score(labels, estimator.labels_)))

clusters = [10, 16, 36, 64, 144, 256]

# test different numbers of clusters
for n_clusters in clusters:
    estimator = MiniBatchKMeans(n_clusters = n_clusters)
    estimator.fit(X)

    # print cluster metrics
    calculate_metrics(estimator, X, Y)

    # determine predicted labels
    cluster_labels = infer_cluster_labels(estimator, Y)
    predicted_Y = infer_data_labels(estimator.labels_, cluster_labels)

    # calculate and print accuracy
    print('Accuracy: {}\n'.format(metrics.accuracy_score(Y, predicted_Y)))

# test kmeans algorithm on testing dataset
# convert each image to 1 dimensional array
X_test = x_test.reshape(len(x_test),-1)

# normalize the data to 0 - 1
X_test = X_test.astype(float) / 255.

# initialize and fit KMeans algorithm on training data
kmeans = MiniBatchKMeans(n_clusters = 256)
kmeans.fit(X)
cluster_labels = infer_cluster_labels(kmeans, Y)

# predict labels for testing data
test_clusters = kmeans.predict(X_test)
predicted_labels = infer_data_labels(kmeans.predict(X_test), cluster_labels)

# calculate and print accuracy
print('Accuracy: {}\n'.format(metrics.accuracy_score(y_test, predicted_labels)))

kmeans = MiniBatchKMeans(n_clusters = 36)
kmeans.fit(X)
```

```
# record centroid values
centroids = kmeans.cluster_centers_

# reshape centroids into images
images = centroids.reshape(36, 28, 28)
images *= 255
images = images.astype(np.uint8)

# determine cluster labels
cluster_labels = infer_cluster_labels(kmeans, Y)

# create figure with subplots using matplotlib.pyplot
fig, axs = plt.subplots(6, 6, figsize = (20, 20))
plt.gray()

# loop through subplots and add centroid images
for i, ax in enumerate(axs.flat):

    # determine inferred label using cluster_labels dictionary
    for key, value in cluster_labels.items():
        if i in value:
            ax.set_title('Inferred Label: {}'.format(key))

    # add image to subplot
    ax.matshow(images[i])
    ax.axis('off')

# display the figure
fig.show()
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=predicted_labels)*100)
class_wise = metrics.classification_report(y_true=y_test, y_pred=predicted_labels)
print(class_wise)
```



```
Number of Clusters: 10
Inertia: 2357144.748114637
Homogeneity: 0.4816337082074267
Accuracy: 0.5894166666666667

Number of Clusters: 16
Inertia: 2202709.811383712
Homogeneity: 0.5484525982876197
Accuracy: 0.6318833333333334

Number of Clusters: 36
Inertia: 1960408.829054009
Homogeneity: 0.6793240346887368
Accuracy: 0.7580333333333333

Number of Clusters: 64
Inertia: 1814522.8736466381
Homogeneity: 0.7274813229069543
Accuracy: 0.7906833333333333

Number of Clusters: 144
Inertia: 1631617.6508676028
Homogeneity: 0.8064918087256727
Accuracy: 0.86935

Number of Clusters: 256
Inertia: 1514100.8151359544
Homogeneity: 0.8426510658055861
Accuracy: 0.89795
```

```
10
MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
                init_size=None, max_iter=100, max_no_improvement=10,
                n_clusters=10, n_init=3, random_state=None,
                reassignment_ratio=0.01, tol=0.0, verbose=0)
```

```
Accuracy =  89.84
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       980
           1       0.96      0.99      0.98      1135
           2       0.92      0.93      0.93      1032
           3       0.84      0.91      0.87      1010
           4       0.90      0.79      0.84       982
           5       0.88      0.82      0.85       892
           6       0.94      0.96      0.95       958
           7       0.91      0.91      0.91      1028
           8       0.88      0.85      0.86       974
           9       0.79      0.84      0.81      1009

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

KNN

```
import csv
import math
import random
def loadDataset(filename, split, trainingset = [], testset = []):
   with open(filename,'r') as csvfile:
      lines = csv.reader(csvfile)
      dataset = list(lines)
      for x in range(1,math.ceil((len(dataset)+1)/10)):
         for y in range(1,len(dataset[x])):
            dataset[x][y] = float(dataset[x][y])
         if random.random() < split:
            trainingset.append(dataset[x])
         else:
            testset.append(dataset[x])

trainingset = []
testset = []
loadDataset('/content/sample_data/mnist_train_small.csv',0.66,trainingset,testset)
print("train "+repr(len(trainingset)))
print("test "+repr(len(testset)))

import math
def euclideandistance(instance1,instance2,length):
   distance = 0
   for x in range(1,length):
      distance += pow((instance1[x]-instance2[x]),2)
   return math.sqrt(distance)

import operator
def getNeighbours(trainingset,testinstance,k):
   distances = []
   length = len(testinstance)-1
   for x in range(1,len(trainingset)):
      dist = euclideandistance(testinstance, trainingset[x],length)
      distances.append((trainingset[x],dist))
   distances.sort(key=operator.itemgetter(1))
   neighbours = []
   for x in range(k):
      neighbours.append(distances[x][0])
   return neighbours

import operator
def getResponse(neighbours):
   classvotes = {}
   for x in range(len(neighbours)):
      response = neighbours[x][0]
      if response in classvotes:
         classvotes[response] += 1
      else:
         classvotes[response] = 1
```

```
    sortedvotes = sorted(classvotes.items(),key = operator.itemgetter(1),reverse = True)
    return sortedvotes[0][0]

def getAccuracy(testset, predictions):
    correct = 0
    for x in range(len(testset)):
        if testset[x][0] == predictions[x]:
            correct= correct + 1
            #print("Srihari",correct)
    return (correct/float(len(testset)))*100.0


from sklearn import metrics
trainingset = []
testset = []
split = 0.66
loadDataset('/content/sample_data/mnist_train_small.csv',split,trainingset,testset)

print("train "+repr(len(trainingset)))
print("test "+repr(len(testset)))
predictions = []
y_test = []
k = 1
i=1
for x in range(len(testset)):
    neighbours = getNeighbours(trainingset,testset[x],k)
    result = getResponse(neighbours)
    predictions.append(result)
    y_test.append(testset[x][0])
    print(i)
    i = i + 1
    #print("Predicted = "+repr(result)+' actual = '+repr(testset[x][-1]))
#accuracy = getAccuracy(testset, predictions)
#print("accuracy = ",repr(accuracy),"%")


confusion = metrics.confusion_matrix(y_true = y_test, y_pred = predictions)
confusion

print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=predictions)*100)
class_wise = metrics.classification_report(y_true=y_test, y_pred=predictions)
print(class_wise)
```

```
Accuracy = 90.24

              precision    recall  f1-score   support

           0       0.97      1.00      0.98        64
           1       0.80      0.99      0.89        67
           2       0.95      0.84      0.89        73
           3       0.90      0.91      0.90        57
           4       0.87      0.83      0.85        66
           5       0.92      0.90      0.91        68
           6       0.97      0.94      0.95        66
           7       0.85      0.91      0.88        70
           8       0.94      0.79      0.86        62
           9       0.78      0.81      0.80        70

    accuracy                           0.89       663
   macro avg       0.90      0.89      0.89       663
weighted avg       0.90      0.89      0.89       663
```

```
confusion = metrics.confusion_matrix(y_true = y_test, y_pred = predictions)
confusion
```

```
array([[64,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 66,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 2,  4, 61,  1,  0,  0,  0,  2,  2,  1],
       [ 0,  0,  2, 52,  0,  2,  0,  0,  1,  0],
       [ 0,  1,  0,  0, 55,  0,  0,  2,  0,  8],
       [ 0,  2,  0,  2,  0, 61,  2,  0,  0,  1],
       [ 0,  1,  1,  0,  0,  2, 62,  0,  0,  0],
       [ 0,  1,  0,  0,  2,  0,  0, 64,  0,  3],
       [ 0,  6,  0,  2,  1,  1,  0,  0, 49,  3],
       [ 0,  1,  0,  1,  5,  0,  0,  6,  0, 57]])
```

TABULAR INFERENCE

ALGORITHM	K-MEANS	K-NN
PRECISION	0.90	0.90
RECALL	0.90	0.89
F1-SCORE	0.90	0.89
ACCURACY	89.84 %	90.24 %

ALGORITHM 1 – K-MEANS

VALUE OF K	10	16	36	64	144	256
ACCURACY	58.9%	63.1%	75.8%	79.06%	86.9%	89.84%

ALGORITHM 2 – K-NN

VALUE OF K	1	8	10	12	20
ACCURACY	90.24%	88.19%	89.6%	87.15%	84.2%

Inference: Thus, the K-Means Clustering and K-Nearest Neighbours algorithms were implemented. It was applied on the MNIST dataset as well. The accuracies obtained by varying the value of k is tabulated.

ENSEMBLE METHODS

CLASSIFIER 1: Multi-Layer Perceptron

```
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier()
params_mlp = {'max_iter':[500,1000], 'activation': ['tanh','relu','identity','logistic']}
mlp_gs = GridSearchCV(mlp, params_mlp, cv=5)
mlp_gs.fit(X_train, y_train)
#save best model
mlp_best = mlp_gs.best_estimator_
#check best n_neigbors value
print(mlp_gs.best_params_)
pred_mlp_gs = mlp_gs.predict(X_test)
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_mlp_gs))
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_mlp_gs)
print(class_wise)
print('mlp: {}'.format(mlp_gs.score(X_test, y_test)))
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=200, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state=None, shuffle=True,
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'activation': ['tanh', 'relu', 'identity', 'logistic'],
                         'max_iter': [500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
{'activation': 'logistic', 'max_iter': 500}
```

```
Accuracy =  0.895
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        57
           1       0.93      0.98      0.96        66
           2       0.93      0.87      0.90        60
           3       0.83      0.91      0.87        57
           4       0.88      0.92      0.90        64
           5       0.83      0.72      0.77        54
           6       0.95      0.95      0.95        60
           7       0.93      0.85      0.89        67
           8       0.83      0.85      0.84        52
           9       0.86      0.90      0.88        63

    accuracy                           0.90       600
   macro avg       0.89      0.89      0.89       600
weighted avg       0.90      0.90      0.89       600

mlp: 0.895
```

CLASSIFIER 2: Support Vector Machine

```
import numpy as np
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
svm_rbf = SVC(kernel='rbf')
params_svm = {'C':[1, 10, 100], 'gamma': ['scale',1e-2, 1e-3, 1e-4]}
svm_gs = GridSearchCV(svm_rbf, params_svm, cv=5)
svm_gs.fit(X_train, y_train)
#save best model
svm_best = svm_gs.best_estimator_
#check best n_neigbors value
print(svm_gs.best_params_)
pred_svm_gs = svm_gs.predict(X_test)
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_svm_gs))
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_svm_gs)
print(class_wise)
print('svm: {}'.format(svm_gs.score(X_test, y_test)))
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [1, 10, 100],
                         'gamma': ['scale', 0.01, 0.001, 0.0001]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
{'C': 10, 'gamma': 'scale'}
```

```
 Accuracy =  0.9183333333333333
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        57
           1       0.94      0.98      0.96        66
           2       0.91      0.88      0.90        60
           3       0.90      0.91      0.90        57
           4       0.88      0.91      0.89        64
           5       0.86      0.89      0.87        54
           6       0.94      0.98      0.96        60
           7       0.94      0.88      0.91        67
           8       0.96      0.87      0.91        52
           9       0.89      0.87      0.88        63

    accuracy                           0.92       600
   macro avg       0.92      0.92      0.92       600
weighted avg       0.92      0.92      0.92       600

 svm: 0.9183333333333333
```

CLASSIFIER 3: Decision Tree

```
import numpy as np
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
params_dt = {'max_depth': np.arange(1, 10),'criterion':['gini','entropy'],'splitter':['best','random']}
dt_gs = GridSearchCV(dt, params_dt, cv=10)
#fit model to training data
dt_gs.fit(X_train, y_train)
dt_best = dt_gs.best_estimator_
print(dt_gs.best_params_)
pred_dt_gs = dt_gs.predict(X_test)
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_dt_gs))
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_dt_gs)
print(class_wise)
print('dt: {}'.format(dt_gs.score(X_test, y_test)))
```

```
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
{'criterion': 'entropy', 'max_depth': 9, 'splitter': 'best'}
```

```
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_dt_gs))
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_dt_gs)
print(class_wise)
```

```
Accuracy =  0.7066666666666667
              precision    recall  f1-score   support

           0       0.85      0.77      0.81        57
           1       0.78      0.91      0.84        66
           2       0.59      0.57      0.58        60
           3       0.71      0.74      0.72        57
           4       0.72      0.75      0.73        64
           5       0.51      0.50      0.50        54
           6       0.75      0.75      0.75        60
           7       0.81      0.75      0.78        67
           8       0.59      0.56      0.57        52
           9       0.71      0.71      0.71        63

    accuracy                           0.71       600
   macro avg       0.70      0.70      0.70       600
weighted avg       0.71      0.71      0.71       600
```

```
print('dt: {}'.format(dt_gs.score(X_test, y_test)))
```

```
dt: 0.7066666666666667
```

CLASSIFIER 4: K-Nearest Neighbours

```
import numpy as np
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
#create new a knn model
knn = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
params_knn = {'n_neighbors': np.arange(1, 25)}
#use gridsearch to test all values for n_neighbors
knn_gs = GridSearchCV(knn, params_knn, cv=5)
#fit model to training data
knn_gs.fit(X_train, y_train)
#save best model
knn_best = knn_gs.best_estimator_
#check best n_neigbors value
print(knn_gs.best_params_)
pred_knn_gs = knn_gs.predict(X_test)
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_knn_gs))
```

```
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_knn_gs)
print(class_wise)
print('knn: {}'.format(knn_gs.score(X_test, y_test)))
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
{'n_neighbors': 1}
```

```
Accuracy =  0.8983333333333333
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        57
           1       0.80      0.98      0.88        66
           2       0.92      0.80      0.86        60
           3       0.91      0.88      0.89        57
           4       0.90      0.86      0.88        64
           5       0.88      0.80      0.83        54
           6       0.97      0.98      0.98        60
           7       0.88      0.90      0.89        67
           8       0.89      0.92      0.91        52
           9       0.87      0.86      0.86        63

    accuracy                           0.90       600
   macro avg       0.90      0.90      0.90       600
weighted avg       0.90      0.90      0.90       600

knn: 0.8983333333333333
```

VOTING CLASSIFIER

```
1 ensemble
```

```
VotingClassifier(estimators=[('knn',
                              KNeighborsClassifier(algorithm='auto',
                                                   leaf_size=30,
                                                   metric='minkowski',
                                                   metric_params=None,
                                                   n_jobs=None, n_neighbors=1,
                                                   p=2, weights='uniform')),
                             ('mlp',
                              MLPClassifier(activation='logistic', alpha=0.0001,
                                            batch_size='auto', beta_1=0.9,
                                            beta_2=0.999, early_stopping=False,
                                            epsilon=1e-08,
                                            hidden_layer_sizes=(100,),
                                            learning_rate='c...
                              DecisionTreeClassifier(ccp_alpha=0.0,
                                                     class_weight=None,
                                                     criterion='entropy',
                                                     max_depth=7,
                                                     max_features=None,
                                                     max_leaf_nodes=None,
                                                     min_impurity_decrease=0.0,
                                                     min_impurity_split=None,
                                                     min_samples_leaf=1,
                                                     min_samples_split=2,
                                                     min_weight_fraction_leaf=0.0,
                                                     presort='deprecated',
                                                     random_state=None,
                                                     splitter='best'))],
                 flatten_transform=True, n_jobs=None, voting='hard',
                 weights=None)
```

Hard Voting:

```
from sklearn.ensemble import VotingClassifier
#create a dictionary of our models
estimators=[('knn', knn_best), ('mlp', mlp_best),  ('svm', svm_best), ('dt', dt_best)]
#create our voting classifier, inputting our models
ensemble = VotingClassifier(estimators, voting='hard')
ensemble.fit(X_train, y_train)
pred_em = ensemble.predict(X_test)
print("Accuracy = ",metrics.accuracy_score(y_true=y_test, y_pred=pred_em))
class_wise = metrics.classification_report(y_true=y_test, y_pred=pred_em)
print(class_wise)
print('EM: {}'.format(ensemble.score(X_test, y_test)))
```

```
Accuracy =  0.9233333333333333
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        57
           1       0.92      0.98      0.95        66
           2       0.91      0.88      0.90        60
           3       0.88      0.93      0.91        57
           4       0.88      0.92      0.90        64
           5       0.94      0.85      0.89        54
           6       0.97      0.97      0.97        60
           7       0.92      0.91      0.92        67
           8       0.94      0.90      0.92        52
           9       0.93      0.87      0.90        63

    accuracy                           0.92       600
   macro avg       0.92      0.92      0.92       600
weighted avg       0.92      0.92      0.92       600

EM: 0.9233333333333333
```

Soft Voting:

```
from sklearn.ensemble import VotingClassifier
estimators=[('knn', knn_best), ('mlp', mlp_best),  ('svm', svm_best), ('dt', dt_best)]
ensemble = VotingClassifier(estimators, voting='soft')
ensemble.fit(X_train, y_train)
#test our model on the test data
ensemble.score(X_test, y_test)
```

```
Accuracy =  0.92
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        57
           1       0.93      0.98      0.96        66
           2       0.91      0.87      0.89        60
           3       0.91      0.91      0.91        57
           4       0.91      0.91      0.91        64
           5       0.90      0.85      0.88        54
           6       0.97      0.97      0.97        60
           7       0.90      0.91      0.90        67
           8       0.91      0.92      0.91        52
           9       0.89      0.87      0.88        63

    accuracy                           0.92       600
   macro avg       0.92      0.92      0.92       600
weighted avg       0.92      0.92      0.92       600

EM: 0.92
```

TABULAR INFERENCE

CLASSIFIER	PRECISION	RECALL	F1-SCORE	ACCURACY
MLP	0.89	0.89	0.89	89.5%
SVM	0.92	0.92	0.92	91.8%
Decision Tree	0.70	0.70	0.70	70.6%
KNN	0.90	0.90	0.90	89.8%
Hard Voting Classifier	0.92	0.92	0.92	**92.3%**
Soft Voting Classifier	0.92	0.92	0.92	**92%**

Inference: Thus, using four different classifiers – MLP, SVM, DT and KNN ensemble methods were implemented. It was applied on the MNIST dataset as well. The accuracies obtained are found to be higher than the individual accuracies of the classifiers using both hard voting and soft voting.