**APPENDIX: CODE**

"""

EECS 445 - Introduction to Machine Learning

Winter 2019

Homework 2, Ensemble Methods

Skeleton Code

"""

import random

import numpy as np

import matplotlib.pyplot as plt

from collections import Counter

from sklearn import metrics, utils

from sklearn.datasets import fetch\_mldata

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

def load\_mnist(classes):

"""

Load MNIST dataset for classes

Every 25th sample is used to reduce computational resources

Input:

classes : list of ints

Returns:

X : np.array (num\_samples, num\_features)

y : np.array (num\_samples)

"""

print('Fetching MNIST data...')

mnist = fetch\_mldata('MNIST original')

X\_all = np.array(mnist.data)[::25]

y\_all = np.array(mnist.target)[::25]

desired\_idx = np.isin(y\_all, classes)

return X\_all[desired\_idx], y\_all[desired\_idx]

def get\_avg\_performance(X, y, m\_vals, n\_splits=50):

"""

Compare the average performance of bagging and random forest across 50

random splits of X and y

Input:

X : np.array (num\_samples, num\_features)

y : np.array (num\_samples)

m\_vals: list - list of values for m

n\_splits: int - number of random splits

Returns:

bag\_results : np.array (len(m\_vals)) - estimate of bagging performance

rf\_results : np.array (len(m\_vals)) - estimate of random forest performance

"""

print('Getting bagging and random forest scores...')

rf\_results = []

bag\_results = []

for m in m\_vals:

print('m = {}'.format(m))

bagging\_scores = []

random\_forest\_scores = []

for i in range(n\_splits):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

random\_forest\_scores.append(random\_forest(X\_train, y\_train, X\_test, y\_test, m))

bagging\_scores.append(bagging\_ensemble(X\_train, y\_train, X\_test, y\_test))

rf\_results.append(np.median(np.array(random\_forest\_scores)))

bag\_results.append(np.median(np.array(bagging\_scores)))

return bag\_results, rf\_results

def plot\_data(bagging\_scores, random\_forest\_scores, m\_vals):

"""

Plot bagging and random forest accuracies

Input:

bagging\_scores : np.array - array containing accuracies for bagging ensemble classifiers

random\_forest\_scores : np.array - array containing accuracies for random forest classifiers

"""

plt.figure()

plt.plot(list(m\_vals), bagging\_scores, '--', label='bagging')

plt.plot(list(m\_vals), random\_forest\_scores, '--', label='random forest')

plt.xlabel('m')

plt.ylabel('Accuracy')

plt.legend(loc='upper right')

plt.savefig('ensemble.png', dpi=300)

plt.show()

def random\_forest(X\_train, y\_train, X\_test, y\_test, m, n\_clf=10):

"""

Returns accuracy on the test set X\_test with corresponding labels y\_test

using a random forest classifier with n\_clf decision trees trained with

training examples X\_train and training labels y\_train.

Input:

X\_train : np.array (n\_train, d) - array of training feature vectors

y\_train : np.array (n\_train) - array of labels corresponding to X\_train samples

X\_test : np.array (n\_test,d) - array of testing feature vectors

y\_test : np.array (n\_test) - array of labels corresponding to X\_test samples

m : int - number of features to consider when splitting

n\_clf : int - number of decision tree classifiers in the random forest, default is 10

Returns:

accuracy : float - accuracy of random forest classifier on X\_test samples

"""

# TODO: Implement this function

y\_predict = np.zeros((10,X\_test.shape[0]))

boot\_size = X\_train.shape[0]

for i in range(10):

X\_boot, y\_boot = utils.resample(X\_train, y\_train, n\_samples = boot\_size)

clf = DecisionTreeClassifier(criterion = 'entropy', max\_features = m)

clf.fit(X\_boot, y\_boot)

y\_pred = clf.predict(X\_test)

y\_predict[i] = y\_pred

y\_pred = []

for i in range(X\_test.shape[0]):

y\_pred = np.append(y\_pred,Counter(y\_predict[:,i]).most\_common(1)[0][0])

return metrics.accuracy\_score(y\_test, y\_pred)

def bagging\_ensemble(X\_train, y\_train, X\_test, y\_test, n\_clf=10):

"""

Returns accuracy on the test set X\_test with corresponding labels y\_test

using a bagging ensemble classifier with n\_clf decision trees trained with

training examples X\_train and training labels y\_train.

Input:

X\_train : np.array (n\_train, d) - array of training feature vectors

y\_train : np.array (n\_train) - array of labels corresponding to X\_train samples

X\_test : np.array (n\_test,d) - array of testing feature vectors

y\_test : np.array (n\_test) - array of labels corresponding to X\_test samples

n\_clf : int - number of decision tree classifiers in the random forest, default is 10

Returns:

accuracy : float - accuracy of random forest classifier on X\_test samples

"""

# TODO: Implement this function

y\_predict = np.zeros((10,X\_test.shape[0]))

boot\_size = X\_train.shape[0]

for i in range(10):

X\_boot, y\_boot = utils.resample(X\_train, y\_train, n\_samples = boot\_size)

clf = DecisionTreeClassifier(criterion = 'entropy')

clf.fit(X\_boot, y\_boot)

y\_pred = clf.predict(X\_test)

y\_predict[i] = y\_pred

y\_pred = []

for i in range(X\_test.shape[0]):

y\_pred = np.append(y\_pred,Counter(y\_predict[:,i]).most\_common(1)[0][0])

return metrics.accuracy\_score(y\_test, y\_pred)

def main():

"""

Analyze how the performance of bagging and random forest changes with m.

"""

X, y = load\_mnist([1,2,3,4])

# Plot accuracies

m\_vals = [1] + list(range(56, 785, 56))

bagging\_scores, random\_forest\_scores = get\_avg\_performance(X, y, m\_vals)

plot\_data(bagging\_scores, random\_forest\_scores, m\_vals)

if \_\_name\_\_ == '\_\_main\_\_':

main()