cs109a_hw8_submission

November 21, 2017

1 CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 8

2 Ensemble methods

Harvard University Fall 2017 Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave

- 209 part surrogate split I partially consulted with my classmate Jiawen Tong.

Import libraries:

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        import sklearn.metrics as metrics
        from sklearn.model_selection import cross_val_score
        from sklearn import tree
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import AdaBoostClassifier
        from sklearn.linear_model import LogisticRegressionCV

        import random
        from sklearn.model_selection import KFold
        %matplotlib inline
```

3 Higgs Boson Discovery

The discovery of the Higgs boson in July 2012 marked a fundamental breakthrough in particle physics. The Higgs boson particle was discovered through experiments at the Large Hadron Collider at CERN, by colliding beams of protons at high energy. A key challenge in analyzing the results of these experiments is to differentiate between a collision that produces Higgs bosons and collisions thats produce only background noise. We shall explore the use of ensemble methods for this classification task.

You are provided with data from Monte-Carlo simulations of collisions of particles in a particle collider experiment. The training set is available in `Higgs_train.csv` and the test set is in

`Higgs_test.csv`. Each row in these files corresponds to a particle colision described by 28 features (columns 1-28), of which the first 21 features are kinematic properties measured by the particle detectors in the accelerator, and the remaining features are derived by physicists from the the first 21 features. The class label is provided in the last column, with a label of 1 indicating that the collision produces Higgs bosons (signal), and a label of 0 indicating that the collision produces other particles (background).

The data set provided to you is a small subset of the HIGGS data set in the UCI machine learning repository. The following paper contains further details about the data set and the predictors used: Baldi et al., Nature Communications 5, 2014.

3.1 Question 1 (2pt): Single Decision Tree

We start by building a basic model which we will use as our base model for comparison.

1. Fit a decision tree model to the training set and report the classification accuracy of the model on the test set. Use 5-fold cross-validation to choose the (maximum) depth for the tree. You will use the max_depth you find here throughout the homework.

```
In [2]: train = pd.read_csv("Higgs_train.csv")
        train.columns = train.columns.map(lambda x: x.replace(' ', '_'))
        X_train = train.iloc[:,:-1]
        y_train = train.iloc[:,-1]
        train.head()
```

```
Out[2]:    lepton_pT  _lepton_eta  _lepton_phi  _missing_energy_magnitude  \
        0   0.376816    -1.583727    -1.707552                   0.990897
        1   0.707330     0.087603    -0.399742                   0.918742
        2   0.617290     0.265839    -1.345227                   1.154581
        3   0.850992    -0.380876    -0.071264                   1.468704
        4   0.767540    -0.691572    -0.040191                   0.614843

           _missing_energy_phi  _jet_1_pt  _jet_1_eta  _jet_1_phi  _jet_1_b-tag  \
        0             0.114397   1.253553    0.619859   -1.479572      2.173076
        1            -1.229936   1.172847   -0.552574    0.886053      2.173076
        2             1.036646   0.954822    0.377252   -0.147960      0.000000
        3            -0.795133   0.691818    0.883260    0.496881      0.000000
        4             0.143765   0.748614    0.397057   -0.873640      0.000000

           _jet_2_pt   ...    _jet_4_phi  _jet_4_b-tag     _m_jj    _m_jjj     _m_lv  \
        0   0.753658   ...      0.397156      0.000000  0.522449  1.318622  0.982398
        1   1.298317   ...      0.236231      0.000000  0.439696  0.828885  0.992241
        2   1.063507   ...     -0.542413      0.000000  1.024506  1.026255  0.986289
        3   1.616349   ...     -1.520171      0.000000  1.197755  1.100534  0.987262
        4   1.147862   ...      0.502034      1.550981  0.921948  0.864080  0.982839

             _m_jlv     _m_bb    _m_wbb   _m_wwbb  _class
        0  1.359610  0.964809  1.309991  1.083203     1.0
        1  1.157820  2.215780  1.189586  0.937976     1.0
        2  0.927720  1.371080  0.981672  0.917436     1.0
```

```
3  1.353453  1.455383  0.994682  0.953553     1.0
4  1.373222  0.601492  0.918621  0.957063     0.0

[5 rows x 29 columns]
```

```
In [3]: test = pd.read_csv("Higgs_test.csv")
        test.columns = test.columns.map(lambda x: x.replace(' ', '_'))
        X_test = test.iloc[:,:-1]
        y_test = test.iloc[:,-1]
        test.head()
```

```
Out[3]:    lepton_pT  _lepton_eta  _lepton_phi  _missing_energy_magnitude  \
        0   0.883751    -0.461715     0.196283                   1.330217
        1   0.780168    -0.292245     0.897072                   0.384474
        2   0.352659    -1.073368    -1.741953                   1.174700
        3   0.757292     0.821975    -1.290851                   0.207558
        4   2.038721     2.025801    -0.471873                   0.423674

           _missing_energy_phi  _jet_1_pt  _jet_1_eta  _jet_1_phi  _jet_1_b-tag  \
        0             1.522737   1.039467   -1.519038   -1.457952      2.173076
        1             0.412700   1.215078   -0.466424   -0.920208      0.000000
        2            -0.198694   0.557980    0.057408   -1.491768      1.086538
        3            -0.150971   1.222224   -1.638856    1.531346      0.000000
        4            -1.497848   1.062186    0.798100    1.218678      2.173076

           _jet_2_pt   ...    _jet_4_phi  _jet_4_b-tag     _m_jj    _m_jjj     _m_lv  \
        0   0.361469   ...     -1.137282      0.000000  1.454688  0.790714  1.397708
        1   1.104803   ...     -0.091825      0.000000  0.812113  0.727805  0.975326
        2   0.911667   ...      1.545824      0.000000  0.829461  1.060325  0.992326
        3   1.683582   ...      0.116925      3.101961  4.290139  2.415188  0.994889
        4   0.805279   ...     -0.079062      3.101961  0.894990  0.936199  1.027161

             _m_jlv     _m_bb    _m_wbb   _m_wwbb  _class
        0  1.250985  0.712875  0.811940  0.820693     1.0
        1  0.636811  0.569234  0.776607  0.715494     1.0
        2  0.824898  0.365448  0.800015  0.765989     0.0
        3  0.923447  0.927035  1.755831  1.362970     1.0
        4  1.559567  1.148236  1.115536  1.157044     1.0

        [5 rows x 29 columns]
```

```
In [4]: depths = [i for i in range(2, 11)]
        train_scores = []
        test_scores = []
        for depth in depths:
            clf = DecisionTreeClassifier(max_depth = depth)
            clf.fit(X_train,y_train)
            train_scores.append(clf.score(X_train,y_train))
            test_scores.append(clf.score(X_test,y_test))
        print("For depth of 2, 3, ..., 10:")
        print("train scores are: \n", train_scores)
        print("test scores are: \n", test_scores)
```

```
For depth of 2, 3, ..., 10:
train scores are:
 [0.6385999999999995, 0.6421999999999999, 0.6602000000000001, 0.6820000000000005, 0.710600
test scores are:
 [0.6430000000000002, 0.6431999999999999, 0.6495999999999996, 0.6453999999999997, 0.65639
```

```
In [5]: total_scores = []
        for depth in depths:
            clf = DecisionTreeClassifier(max_depth = depth)
            # Perform 5-fold cross validation
            score = cross_val_score(estimator=clf, X=X_train, y=y_train, cv=5)
            total_scores.append((depth, np.mean(score)))
        total_scores
```

```
Out[5]: [(2, 0.6232074958074957 9),
```

```
Out[5]: [(2, 0.6232074958074957 9),
```

```
Out[5]: [(2, 0.62320749580749579),
         (3, 0.62041149381149385),
         (4, 0.63580570720570717),
         (5, 0.63961210841210847),
         (6, 0.62660650320650313),
         (7, 0.62840750420750424),
         (8, 0.62540109820109824),
         (9, 0.62220769800769804),
         (10, 0.61680469620469613)]
```

```
In [6]: # max(total_scores, key=lambda x:x[1])
        best_depth = max(total_scores, key=lambda x:x[1])[0]
        print("The depth of the tree using 5-fold cross-validation is:", best_depth)
```

```
The depth of the tree using 5-fold cross-validation is: 5
```

```
In [7]: clf = DecisionTreeClassifier(max_depth = best_depth)
        clf.fit(X_train,y_train)
        print("classification accuracy on the test set is:", clf.score(X_test, y_test))
```

```
classification accuracy on the test set is: 0.6454
```

3.2 Question 2 (15pt): Dropout-based Approach

We start with a simple method inspired from the idea of 'dropout' in machine learning, where we fit multiple decision trees on random subsets of predictors, and combine them through a majority vote. The procedure is described below.

- For each predictor in the training sample, set the predictor values to 0 with probability p (i.e. drop the predictor by setting it to 0). Repeat this for B trials to create B separate training sets.
- Fit decision tree models $\hat{h}^1(x), \ldots, \hat{h}^B(x) \in \{0, 1\}$ to the B training sets.
- Combine the decision tree models into a single classifier by taking a majority vote:

$$\hat{H}_{maj}(x) = majority\Big(\hat{h}^1(x), \ldots, \hat{h}^B(x)\Big).$$

We shall refer to the combined classifier as an ** *ensemble classifier* **. Implement the described dropout approach, and answer the following questions: 1. Apply the dropout procedure with $p = 0.5$ for different number of trees (say $2, 4, 8, 16, \ldots, 256$), and evaluate the training and test accuracy of the combined classifier. Does an increase in the number of trees improve the training and test performance? Explain your observations in terms of the bias-variance trade-off for the classifier. - Fix the number of trees to 64 and apply the dropout procedure with different dropout rates $p = 0.1, 0.3, 0.5, 0.7, 0.9$. Based on your results, explain how the dropout rate influences the bias and variance of the combined classifier. - Apply 5-fold cross-validation to choose the optimal combination of the dropout rate and number of trees. How does the test performance of an ensemble of trees fitted with the optimal dropout rate and number of trees compare with the single decision tree model in Question 1? [hint: Training with large number of trees can take long time. You may need to restrict the max number of trees.]

Question 2.1: Apply the dropout procedure with $p = 0.5$ for different number of trees (say $2, 4, 8, 16, \ldots, 256$), and evaluate the training and test accuracy of the combined classifier. Does an increase in the number of trees improve the training and test performance? Explain your observations in terms of the bias-variance trade-off for the classifier.

```
In [8]: tree_numbers = [2**i for i in range(1, 9)]
        true_train = y_train.tolist()
        true_test = y_test.tolist()
        N = len(true_train)

In [9]: accuracy_train = []
        accuracy_test = []
        for B in tree_numbers:

            trees = []
            for _ in range(B):
                X_train_new = X_train.copy()
                for c in list(X_train):
                    rand_number = random.random()
                    if rand_number < 0.5:
                        X_train_new[c] = 0
                dt = DecisionTreeClassifier(max_depth = 5, max_features=None, random_state=109
                trees.append(dt.fit(X_train_new, y_train))

            tree_preds_train = []
            tree_preds_test = []
            for tree in trees:
                tree_preds_train.append(tree.predict(X_train))
                tree_preds_test.append(tree.predict(X_test))

            predicted_train = pd.DataFrame(np.array(tree_preds_train)).T.mode(axis=1)[0].tolis
            predicted_test = pd.DataFrame(np.array(tree_preds_test)).T.mode(axis=1)[0].tolist(

            diff_train = [a - b for a, b in zip(true_train, predicted_train)]
            diff_test = [a - b for a, b in zip(true_test, predicted_test)]

            correct_train = sum(1 for x in diff_train if x == 0)
            correct_test = sum(1 for x in diff_test if x == 0)

            accuracy_train.append(correct_train / N)
            accuracy_test.append(correct_test / N)
```

```
In [10]: print("train accuracy for different number of trees 2, 4, 8, 16, ..., 256\n", accurac

         print("test accuracy for different number of trees 2, 4, 8, 16, ..., 256\n", accuracy_
```

```
train accuracy for different number of trees 2, 4, 8, 16, ..., 256
 [0.658, 0.6904, 0.722, 0.7242, 0.7318, 0.7392, 0.7352, 0.7288]
test accuracy for different number of trees 2, 4, 8, 16, ..., 256
 [0.6216, 0.637, 0.6738, 0.6706, 0.6862, 0.6834, 0.689, 0.6846]
```

Does an increase in the number of trees improve the training and test performance? Explain your observations in terms of the bias-variance trade-off for the classifier.

- Yes. We can see that as the number of trees increases, the accuracy for both training and test data set increases.
- Increase in the number of trees leads to the increase in varinace, since now we have different trees. It also leads to the decrease in bias because the probability to get the correct prediction increases, as we pick the majority vote among more trees' predictons.

Question 2.2: Fix the number of trees to 64 and apply the dropout procedure with different dropout rates $p = 0.1, 0.3, 0.5, 0.7, 0.9$. **Based on your results, explain how the dropout rate influences the bias and variance of the combined classifier.**

```
In [11]: rates = [0.1, 0.3, 0.5, 0.7, 0.9]

         accuracy_train_p = []
         accuracy_test_p = []

         for p in rates:

             trees = []
             for _ in range(64):
                 X_train_new = X_train.copy()
```

```
                for c in list(X_train):
                    rand_number = random.random()
                    if rand_number < p:
                        X_train_new[c] = 0
                dt = DecisionTreeClassifier(max_depth = 5, max_features=None, random_state=10
                trees.append(dt.fit(X_train_new, y_train))

            tree_preds_train = []
            tree_preds_test = []
            for tree in trees:
                tree_preds_train.append(tree.predict(X_train))
                tree_preds_test.append(tree.predict(X_test))

            predicted_train = pd.DataFrame(np.array(tree_preds_train)).T.mode(axis=1)[0].toli
            predicted_test = pd.DataFrame(np.array(tree_preds_test)).T.mode(axis=1)[0].tolist

            diff_train = [a - b for a, b in zip(true_train, predicted_train)]
            diff_test = [a - b for a, b in zip(true_test, predicted_test)]

            correct_train = sum(1 for x in diff_train if x == 0)
            correct_test = sum(1 for x in diff_test if x == 0)

            accuracy_train_p.append(correct_train / N)
            accuracy_test_p.append(correct_test / N)
```

```
In [12]: print("train accuracy with dropout rates 0.1, 0.3, 0.5, 0.7, 0.9: \n", accuracy_train
         print("test accuracy with dropout rates 0.1, 0.3, 0.5, 0.7, 0.9: \n", accuracy_test_p
```

```
train accuracy with dropout rates 0.1, 0.3, 0.5, 0.7, 0.9:
 [0.6818, 0.7168, 0.7338, 0.7308, 0.6888]
test accuracy with dropout rates 0.1, 0.3, 0.5, 0.7, 0.9:
 [0.6456, 0.6734, 0.688, 0.681, 0.6284]
```

Based on your results, explain how the dropout rate influences the bias and variance of the combined classifier.

- Simply looking at the trend, we can see that at first, as the dropout rate increases, the accuracy goes up; but as the dropout rate keeps increasing, the accuracy decreases.
- Therefore, at the first stage when we increase the dropout rate, the variance of the model increases and the bias decreases. At the second stage, as the dropout rate keeps increasing, every model will be more different from each other since only a few predictors are used in each tree. Hence the variance keeps increasing, and since trees with not enough predictors underfits the dataset, and therefore bias is high.

Question 2.3: Apply 5-fold cross-validation to choose the optimal combination of the dropout rate and number of trees. How does the test performance of an ensemble of trees fitted with the optimal dropout rate and number of trees compare with the single decision tree model in Question 1?

```
In [13]: combination = {}
         combination[(2,0.1)], combination[(2,0.3)], combination[(2,0.5)],  combination[(2,0.7
         combination[(4,0.1)], combination[(4,0.3)], combination[(4,0.5)],  combination[(4,0.7
         combination[(8,0.1)], combination[(8,0.3)], combination[(8,0.5)],  combination[(8,0.7
         combination[(16,0.1)], combination[(16,0.3)], combination[(16,0.5)],  combination[(16
         combination[(32,0.1)], combination[(32,0.3)], combination[(32,0.5)],  combination[(32
         combination[(64,0.1)], combination[(64,0.3)], combination[(64,0.5)],  combination[(64
         combination[(128,0.1)], combination[(128,0.3)], combination[(128,0.5)],  combination[
         combination[(256,0.1)], combination[(256,0.3)], combination[(256,0.5)],  combination[
         = ([] for i in range(40))

In [14]: for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9001).split(X_trai
             X_train_cv = X_train.iloc[itrain,:]
             y_train_cv = y_train.iloc[itrain]
             X_valid_cv = X_train.iloc[ivalid,:]
             y_valid_cv = y_train.iloc[ivalid]

             true_train_cv = y_train_cv.tolist()
             true_valid_cv = y_valid_cv.tolist()


             for B in tree_numbers:
         #        print("number of tree", B)
                 for p in rates:
         #            print("rate", p)
                     trees = []
                     for _ in range(B):
                         X_train_new = X_train_cv.copy()
                         for c in list(X_train_cv):
                             rand_number = random.random()
                             if rand_number < p:
                                 X_train_new[c] = 0
                         dt = DecisionTreeClassifier(max_depth = 5, max_features=None, random_
                         trees.append(dt.fit(X_train_new, y_train_cv))
         #            tree_preds_train = []
                     tree_preds_valid = []
                     for tree in trees:
         #                tree_preds_train.append(tree.predict(X_train_cv))
                         tree_preds_valid.append(tree.predict(X_valid_cv))

         #            predicted_train = pd.DataFrame(np.array(tree_preds_train)).T.mode(axis=
                     predicted_valid = pd.DataFrame(np.array(tree_preds_valid)).T.mode(axis=1)

         #            diff_train = [a - b for a, b in zip(true_train_cv, predicted_train)]
                     diff_valid = [a - b for a, b in zip(true_valid_cv, predicted_valid)]

         #            correct_train = sum(1 for x in diff_train if x == 0) / len(X_valid_cv)
                     correct_valid = sum(1 for x in diff_valid if x == 0) / len(X_valid_cv)
```

```
#             print(correct_valid)

                  combination[(B,p)].append(correct_valid)
```

```
In [15]: # combination
```

```
In [16]: for key, value in combination.items():
             combination[key] = sum(value) / len(value)
         best_combi = max(combination, key=combination.get)
         # combination
         # best_combi
```

```
In [17]: print("best combination is:", best_combi)
         best_tree_n = best_combi[0]
```

```
best combination is: (128, 0.5)
```

```
In [18]: trees = []
         for _ in range(best_tree_n):
             X_train_new = X_train.copy()
             for c in list(X_train):
                 rand_number = random.random()
                 if rand_number < 0.5:
                     X_train_new[c] = 0
             dt = DecisionTreeClassifier(max_depth = 5, max_features=None, random_state=109)
             trees.append(dt.fit(X_train_new, y_train))

         tree_preds_train = []
         tree_preds_test = []
         for tree in trees:
             tree_preds_train.append(tree.predict(X_train))
             tree_preds_test.append(tree.predict(X_test))

         predicted_train = pd.DataFrame(np.array(tree_preds_train)).T.mode(axis=1)[0].tolist()
         predicted_test = pd.DataFrame(np.array(tree_preds_test)).T.mode(axis=1)[0].tolist()

         diff_train = [a - b for a, b in zip(true_train, predicted_train)]
         diff_test = [a - b for a, b in zip(true_test, predicted_test)]

         correct_train = sum(1 for x in diff_train if x == 0)
         correct_test = sum(1 for x in diff_test if x == 0)

         accuracy_train_best = correct_train / N
         accuracy_test_best = correct_test / N
```

```
In [19]: print("an ensemble of trees fitted with the optimal dropout rate 0.5 and number of tr
         print("training accuracy:", accuracy_train_best)
         print("test accuracy:", accuracy_test_best)
```

```
an ensemble of trees fitted with the optimal dropout rate 0.5 and number of trees 128:
training accuracy: 0.7368
test accuracy: 0.6824
```

How does the test performance of an ensemble of trees fitted with the optimal dropout rate and number of trees compare with the single decision tree model in Question 1?ű

- The single decision tree model in Question 1 gives accuracy 0.645 on the test set, whereas this ensemble of trees gives accuracy 0.682 on the test set, which is better than the former method.

3.3 Question 3 (15pt): Random Forests

We now move to a more sophisticated ensemble technique, namely random forest: 1. How does a random forest approach differ from the dropout procedure described in Question 2?

- Fit random forest models to the training set for different number of trees (say $2, 4, 8, 16, \ldots, 256$), and evaluate the training and test accuracies of the models. You may set the number of predictors for each tree in the random forest model to $\sqrt{p}$, where p is the total number of predictors.

- Based on your results, do you find that a larger number of trees necessarily improves the test accuracy of a random forest model? Explain how the number of trees effects the training and test accuracy of a random forest classifier, and how this relates to the bias-variance trade-off for the classifier.

- Fixing the number of trees to a reasonable value, apply 5-fold cross-validation to choose the optimal value for the number of predictors. How does the test performance of random forest model fitted with the optimal number of trees compare with the dropout approach in Question 2?

Quesrion 3.1: How does a random forest approach differ from the dropout procedure described in Question 2?

- Random forest first bootstraps samples from the entire training set, but the dropout procedure uses full dataset.
- Random forest randomly picks a certain number of predictors ($\sqrt{p}$ in this problem), whereas the dropout procedure has a tunable probability to select each predictor.

Quesrion 3.2: Fit random forest models to the training set for different number of trees (say $2, 4, 8, 16, \ldots, 256$), and evaluate the training and test accuracies of the models. You may set the number of predictors for each tree in the random forest model to $\sqrt{p}$, where p is the total number of predictors.

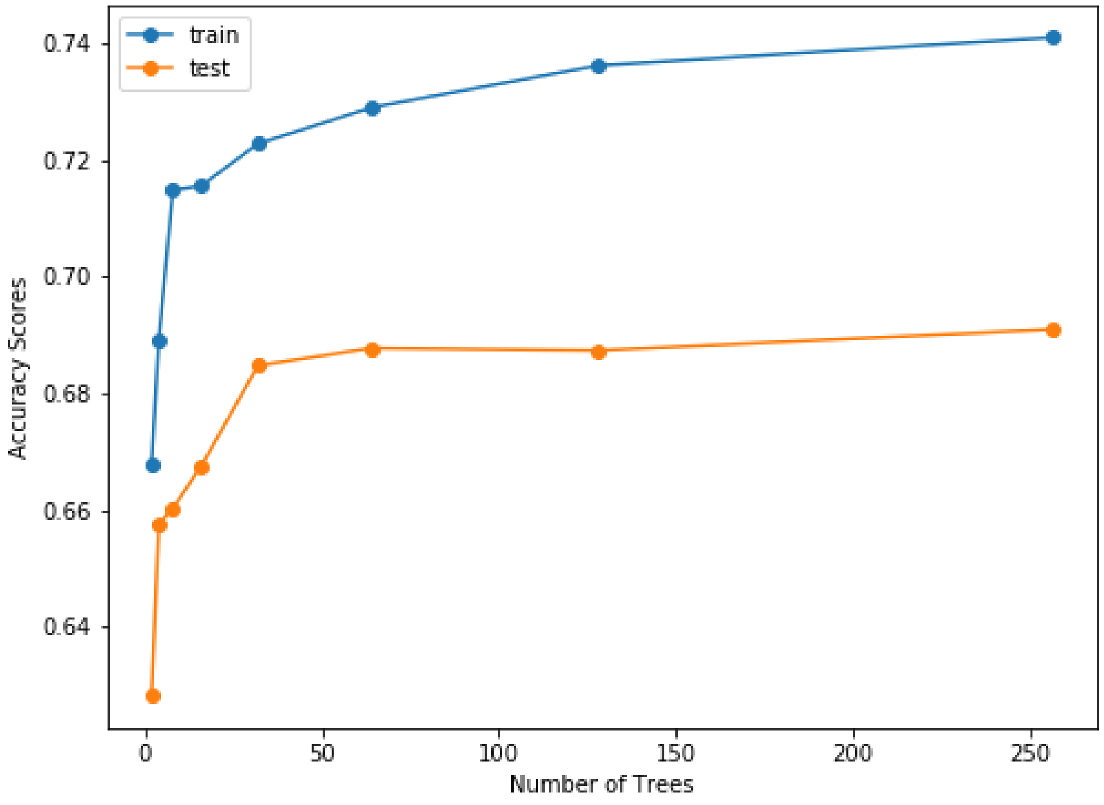
```
In [20]: rf_train_scores = []
         rf_test_scores = []
         for t in tree_numbers:
             print("Number of trees:", t)
```

```
            rf = RandomForestClassifier(n_estimators=t, max_depth=5, max_features="sqrt", ranc
            rf.fit(X_train, y_train)
            rf_train_scores.append(rf.score(X_train, y_train))
            print("Train score:", rf.score(X_train, y_train))
            rf_test_scores.append(rf.score(X_test, y_test))
            print("Test score:", rf.score(X_test, y_test))
```

```
Number of trees: 2
Train score: 0.668
Test score: 0.6284
Number of trees: 4
Train score: 0.6892
Test score: 0.6576
Number of trees: 8
Train score: 0.7148
Test score: 0.6604
Number of trees: 16
Train score: 0.7156
Test score: 0.6676
Number of trees: 32
Train score: 0.7228
Test score: 0.6848
Number of trees: 64
Train score: 0.729
Test score: 0.6878
Number of trees: 128
Train score: 0.7362
Test score: 0.6874
Number of trees: 256
Train score: 0.741
Test score: 0.691
```

Question 3.3: Based on your results, do you find that a larger number of trees necessarily improves the test accuracy of a random forest model? Explain how the number of trees effects the training and test accuracy of a random forest classifier, and how this relates to the bias-variance trade-off for the classifier.

```
In [21]: plt.figure(figsize=(8, 6))
         # x = [1,2,3,4,5,6,7,8]
         plt.plot(tree_numbers, rf_train_scores, marker='o', label="train")
         plt.plot(tree_numbers, rf_test_scores, marker='o', label="test")
         plt.xlabel("Number of Trees")
         plt.ylabel("Accuracy Scores")
         # plt.xticks(x, tree_numbers)
         plt.legend()
         plt.show()
```



- A larger number of trees improves the test accuracy. The improvement is rapid at first, but slows down when the number of trees is already large enough.
- As the number of trees increases, both of the training accuracy and test accuracy goes up. The increase is initially very fast; however, the increase slows down later and tends to stay at the same level.
- Bias-variance trade-off: Random forest can only decrease variance, not bias, and we know that error = variance + bias. This is because the error of the entire forest is roughly equal to the bias of a single tree, and to lower the bias, we need to let the tree grow very deep. In random forests, we usually allow the depth of the individual trees to be unrestricted. But in this problem, we restrict the max depth to be 5, so the bias is not low initially. As we increase the number of trees, the variance is decreasing. But at some point, there is no need to add more trees because it cannot decrease variance anymore. So the accuracy score will converge.

Question 3.4: Fixing the number of trees to a reasonable value, apply 5-fold cross-validation to choose the optimal value for the number of predictors. How does the test performance of random forest model fitted with the optimal number of trees compare with the dropout approach in Question 2?

```
In [22]: # choose optimum as smallest number of trees "sufficiently" close to maximum train ac
         for i in range(len(rf_test_scores)):
             if np.max(rf_test_scores)-rf_test_scores[i]<0.005:
                 n_trees_optimal=i+1
                 break

         print("reasonable number of trees:", 2**n_trees_optimal)
```

```
reasonable number of trees: 64
```

```
In [23]: accuracy_train_cv = np.zeros((29,5))
         accuracy_valid_cv = np.zeros((29,5))

         fold_ctr = 0

         for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9001).split(X_trai
             X_train_cv = X_train.iloc[itrain,:]
             y_train_cv = y_train.iloc[itrain]
             X_valid_cv = X_train.iloc[ivalid,:]
             y_valid_cv = y_train.iloc[ivalid]

             for i in range(1,29):
         #         print("max_feature =", i)
                 rf = RandomForestClassifier(n_estimators=64, max_depth = 5, max_features=i, r
                 rf.fit(X_train_cv, y_train_cv)

                 accuracy_train_cv[i,fold_ctr] = rf.score(X_train_cv, y_train_cv)
                 accuracy_valid_cv[i,fold_ctr] = rf.score(X_valid_cv, y_valid_cv)

             fold_ctr += 1

In [24]: # accuracy_train_cv[1:], accuracy_valid_cv[1:]

In [25]: scores_train = np.mean(accuracy_train_cv, axis=1)[1:]
         scores_valid = np.mean(accuracy_valid_cv, axis=1)[1:]

In [26]: # choose optimum as smallest number of components "sufficiently" close to maximum val
         for i in range(len(scores_valid)):
             if np.max(scores_valid)-scores_valid[i]<0.005:
                 n_predictors_optimal=i+1
                 break

         print("maximum validation accuracy using %i predictors" % n_predictors_optimal)
```

```
maximum validation accuracy using 7 predictors
```

```
In [27]: rf = RandomForestClassifier(n_estimators=64, max_depth = 5, max_features=n_predictors
         rf.fit(X_train, y_train)
         print("Test accuracy with 64 trees and 7 features:", rf.score(X_test, y_test))
```

```
Test accuracy with 64 trees and 7 features: 0.6954
```

How does the test performance of random forest model fitted with the optimal number of trees compare with the dropout approach in Question 2?

- The test performance of random forest model with the optimal number of trees on the test set is 0.695, whereas the perfomance of the dropout approach on the test set is 0.682. So the random forest model is a little better than the dropout approach.

3.4 Question 4 (15pt): Boosting

We next compare the random forest model with the approach of boosting:

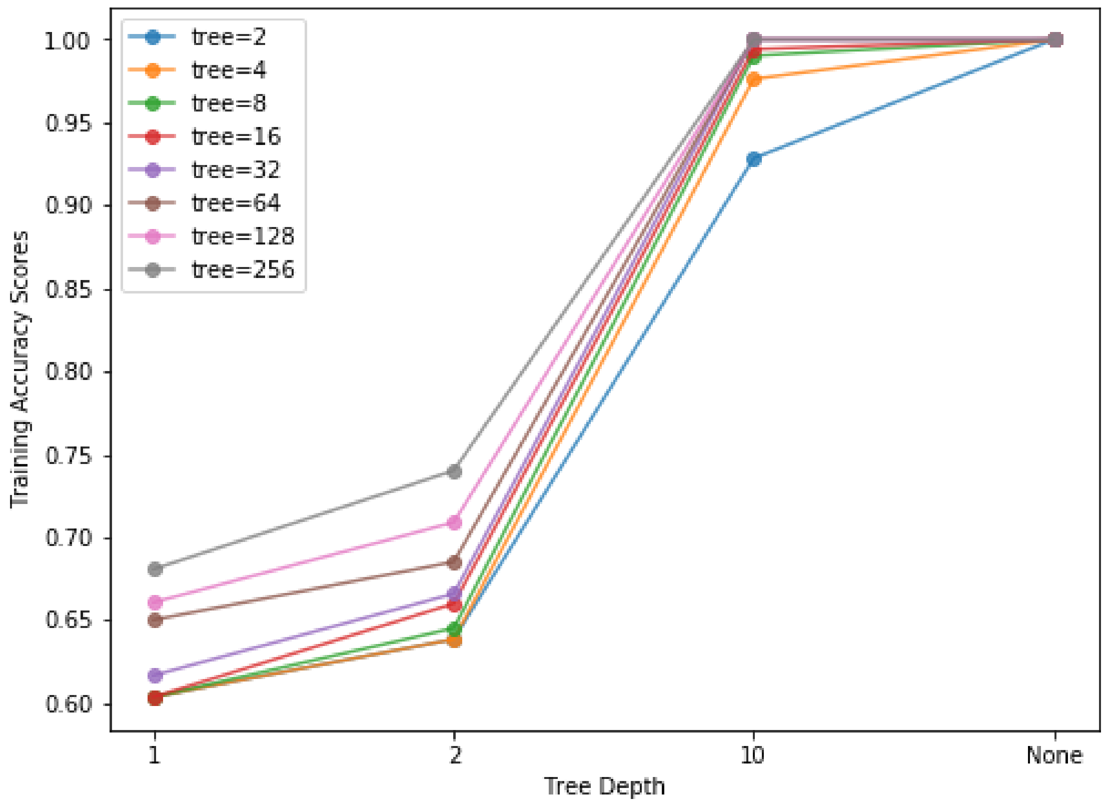
1. Apply the AdaBoost algorithm to fit an ensemble of decision trees. Set the learning rate to 0.05, and try out different tree depths for the base learners: 1, 2, 10, and unrestricted depth. Make a plot of the training accuracy of the ensemble classifier as a function of tree depths. Make a similar plot of the test accuracies as a function of number of trees (say $2, 4, 8, 16, \ldots, 256$).

- How does the number of trees influence the training and test performance? Compare and contrast between the trends you see in the training and test performance of AdaBoost and that of the random forest models in Question 3. Give an explanation for your observations.
- How does the tree depth of the base learner impact the training and test performance? Recall that with random forests, we allow the depth of the individual trees to be unrestricted. Would you recommend the same strategy for boosting? Explain your answer.
- Apply 5-fold cross-validation to choose the optimal number of trees B for the ensemble and the optimal tree depth for the base learners. How does an ensemble classifier fitted with the optimal number of trees and the optimal tree depth compare with the random forest model fitted in Question 3.4?

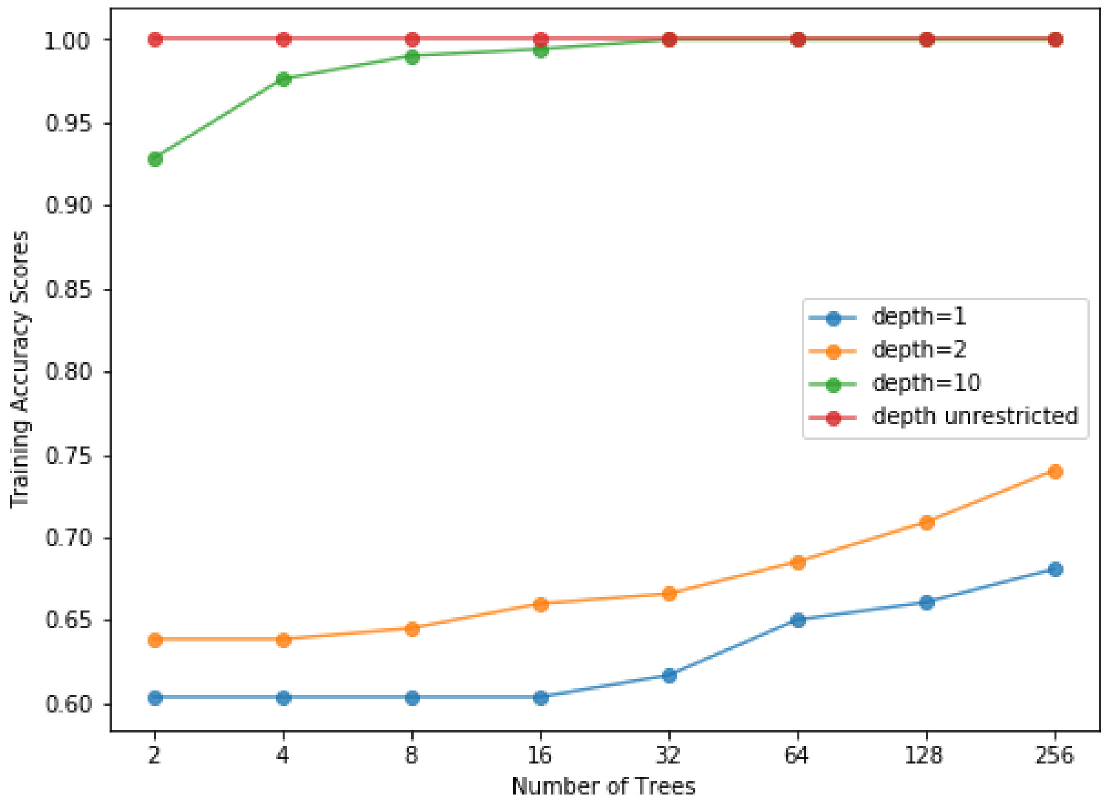
Question 4.1: Apply the AdaBoost algorithm to fit an ensemble of decision trees. Set the learning rate to 0.05, and try out different tree depths for the base learners: 1, 2, 10, and unrestricted depth. Make a plot of the training accuracy of the ensemble classifier as a function of tree depths. Make a similar plot of the test accuracies as a function of number of trees (say $2, 4, 8, 16, \ldots, 256$**).**

```
In [28]: s1_train = []
         s1_test = []
         for b in [1, 2, 10, None]:
         #     print("Tree depths for the base learners:", b)
             s2_train = []
             s2_test = []
             for n in tree_numbers:
         #         print("Number of trees:", n)
                 adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=b), n_estimato
                 adaboost.fit(X_train, y_train)
                 s2_train.append(adaboost.score(X_train, y_train))
                 s2_test.append(adaboost.score(X_test, y_test))
         #         print("Train score:", adaboost.score(X_train, y_train))
         #         print("Test score:", adaboost.score(X_test, y_test))
             s1_train.append(s2_train)
             s1_test.append(s2_test)
```

```
In [29]: # s1_train

In [30]: # as a function of tree depth (train set):
         n_trees_2_train = [item[0] for item in s1_train]
         n_trees_4_train = [item[1] for item in s1_train]
         n_trees_8_train = [item[2] for item in s1_train]
         n_trees_16_train = [item[3] for item in s1_train]
         n_trees_32_train = [item[4] for item in s1_train]
         n_trees_64_train = [item[5] for item in s1_train]
         n_trees_128_train = [item[6] for item in s1_train]
         n_trees_256_train = [item[7] for item in s1_train]

         plt.figure(figsize=(8, 6))
         labels = ['1', '2', '10', 'None']
         x = [1,2,3,4]
         plt.plot(x, n_trees_2_train, marker='o', label="tree=2", alpha=0.8)
         plt.plot(x, n_trees_4_train, marker='o', label="tree=4", alpha=0.8)
         plt.plot(x, n_trees_8_train, marker='o', label="tree=8", alpha=0.8)
         plt.plot(x, n_trees_16_train, marker='o', label="tree=16", alpha=0.8)
         plt.plot(x, n_trees_32_train, marker='o', label="tree=32", alpha=0.8)
         plt.plot(x, n_trees_64_train, marker='o', label="tree=64", alpha=0.8)
         plt.plot(x, n_trees_128_train, marker='o', label="tree=128", alpha=0.8)
         plt.plot(x, n_trees_256_train, marker='o', label="tree=256", alpha=0.8)
         plt.xticks(x, labels)
         plt.xlabel("Tree Depth")
         plt.ylabel("Training Accuracy Scores")
         plt.legend()
         plt.show()
```



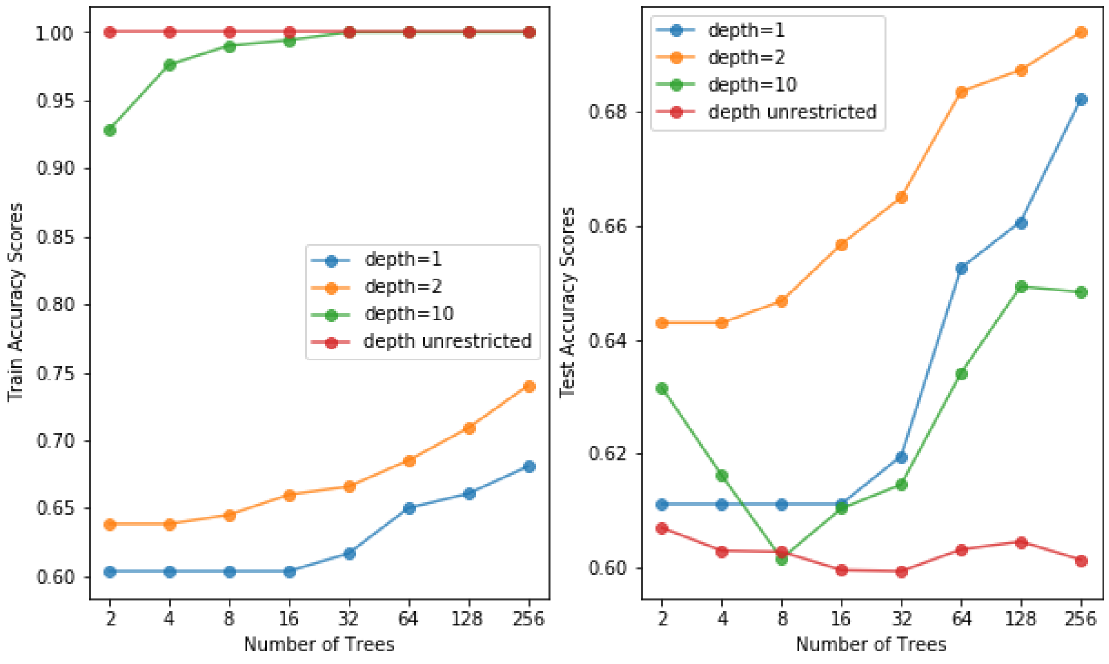
```
In [31]: # as a function of number of trees:
         plt.figure(figsize=(8, 6))
         labels = ['2', '4', '8', '16', '32', '64', '128', '256']
         x = [1,2,3,4,5,6,7,8]
         plt.plot(x, s1_train[0], marker='o', label="depth=1", alpha=0.8)
         plt.plot(x, s1_train[1], marker='o', label="depth=2", alpha=0.8)
         plt.plot(x, s1_train[2], marker='o', label="depth=10", alpha=0.8)
         plt.plot(x, s1_train[3], marker='o', label="depth unrestricted", alpha=0.8)

         plt.xticks(x, labels)
         plt.xlabel("Number of Trees")
         plt.ylabel("Training Accuracy Scores")
         plt.legend()
         plt.show()
```



Question 4.2: How does the number of trees influence the training and test performance? Compare and contrast between the trends you see in the training and test performance of AdaBoost and that of the random forest models in Question 3. Give an explanation for your observations.

```
In [32]: f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))

         labels = ['2', '4', '8', '16', '32', '64', '128', '256']
         x = [1,2,3,4,5,6,7,8]
         ax1.plot(x, s1_train[0], marker='o', label="depth=1", alpha=0.8)
         ax1.plot(x, s1_train[1], marker='o', label="depth=2", alpha=0.8)
         ax1.plot(x, s1_train[2], marker='o', label="depth=10", alpha=0.8)
         ax1.plot(x, s1_train[3], marker='o', label="depth unrestricted", alpha=0.8)

         ax1.set_xticks(x)
         ax1.set_xticklabels(labels)
         ax1.set_xlabel("Number of Trees")
         ax1.set_ylabel("Train Accuracy Scores")
         ax1.legend()


         ax2.plot(x, s1_test[0], marker='o', label="depth=1", alpha=0.8)
```

```
ax2.plot(x, s1_test[1], marker='o', label="depth=2", alpha=0.8)
ax2.plot(x, s1_test[2], marker='o', label="depth=10", alpha=0.8)
ax2.plot(x, s1_test[3], marker='o', label="depth unrestricted", alpha=0.8)

ax2.set_xticks(x)
ax2.set_xticklabels(labels)
ax2.set_xlabel("Number of Trees")
ax2.set_ylabel("Test Accuracy Scores")
ax2.legend()
plt.show()
```



How does the number of trees influence the training and test performance?

- As the number of trees increases, the training accuracy increases as well.
- As the number of trees increases, the test accuracy for models with restricted depths has a general trend of increasing, but for example, when the depth is 10, there is a drop in the middle followed by an increase again. For the case of unrestricted depth, the test accuracy stays about the same no matter how big the number of trees is.

Compare and contrast between the trends you see in the training and test performance of AdaBoost and that of the random forest models in Question 3. Give an explanation for your observations.

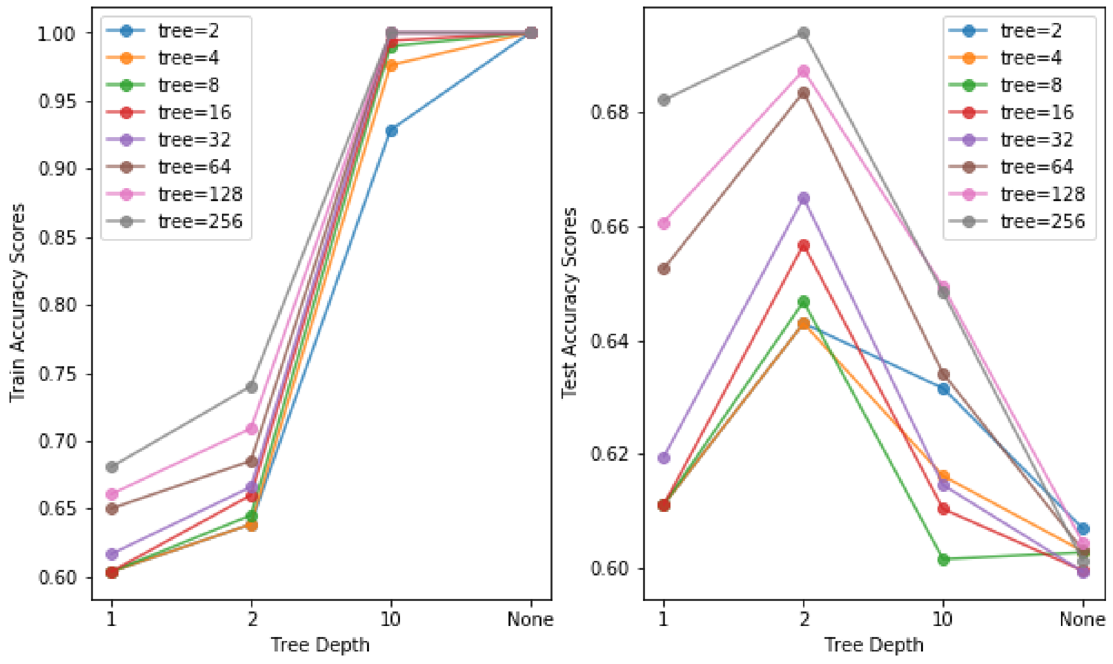
- In the random forest models, the training and test performance have similar trend that as the number of trees increases, the accuracy score increases fast at the beginning and soon converges to a certain value.

- In the AdaBoost models, we can see that the training performance has a similar trend as in the random forest models; however, the test performance is different. For small depth like 1 or 2, the test score increases as the number of trees increases; but when depth gets larger, its test score is not as good as before, and sometimes decreases and the number of trees goes up. Especially, the AdaBoost model with unrestricted depth has the lowest test score among all the four depths.
- In practice, random forest seldom overfits the data due to the increase of the number of trees. This is because it randomly selects predictors for each tree model, and while some tree may overfit, some other tree can underfit and therefore the result cancels each other out.
- For AdaBoost, it is robust for overfitting because it is a boosting ensemble method, and therefore the test performance seldom decreases as the number of trees increases. But we can see that for AdaBoost, maybe a simple base learner such as depth of 1 or 2 decision trees will perform better.

Question 4.3: How does the tree depth of the base learner impact the training and test performance? Recall that with random forests, we allow the depth of the individual trees to be unrestricted. Would you recommend the same strategy for boosting? Explain your answer.

```
In [33]: # as a function of tree depth (test set):
         n_trees_2_test = [item[0] for item in s1_test]
         n_trees_4_test = [item[1] for item in s1_test]
         n_trees_8_test = [item[2] for item in s1_test]
         n_trees_16_test = [item[3] for item in s1_test]
         n_trees_32_test = [item[4] for item in s1_test]
         n_trees_64_test = [item[5] for item in s1_test]
         n_trees_128_test = [item[6] for item in s1_test]
         n_trees_256_test = [item[7] for item in s1_test]

         f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,6))
         labels = ['1', '2', '10', 'None']
         x = [1,2,3,4]

         ax1.plot(x, n_trees_2_train, marker='o', label="tree=2", alpha=0.8)
         ax1.plot(x, n_trees_4_train, marker='o', label="tree=4", alpha=0.8)
         ax1.plot(x, n_trees_8_train, marker='o', label="tree=8", alpha=0.8)
         ax1.plot(x, n_trees_16_train, marker='o', label="tree=16", alpha=0.8)
         ax1.plot(x, n_trees_32_train, marker='o', label="tree=32", alpha=0.8)
         ax1.plot(x, n_trees_64_train, marker='o', label="tree=64", alpha=0.8)
         ax1.plot(x, n_trees_128_train, marker='o', label="tree=128", alpha=0.8)
         ax1.plot(x, n_trees_256_train, marker='o', label="tree=256", alpha=0.8)
         ax1.set_xticks(x)
         ax1.set_xticklabels(labels)
         ax1.set_xlabel("Tree Depth")
         ax1.set_ylabel("Train Accuracy Scores")
         ax1.legend()

         ax2.plot(x, n_trees_2_test, marker='o', label="tree=2", alpha=0.8)
         ax2.plot(x, n_trees_4_test, marker='o', label="tree=4", alpha=0.8)
```

```
ax2.plot(x, n_trees_8_test, marker='o', label="tree=8", alpha=0.8)
ax2.plot(x, n_trees_16_test, marker='o', label="tree=16", alpha=0.8)
ax2.plot(x, n_trees_32_test, marker='o', label="tree=32", alpha=0.8)
ax2.plot(x, n_trees_64_test, marker='o', label="tree=64", alpha=0.8)
ax2.plot(x, n_trees_128_test, marker='o', label="tree=128", alpha=0.8)
ax2.plot(x, n_trees_256_test, marker='o', label="tree=256", alpha=0.8)
ax2.set_xticks(x)
ax2.set_xticklabels(labels)
ax2.set_xlabel("Tree Depth")
ax2.set_ylabel("Test Accuracy Scores")
ax2.legend()
plt.show()
```



- We can see that as tree depth increases, trian performance increases but test performance decreases. So a complicated base learner model can cause overfitting in AdaBoost models. I think the explanation is that if the base learner model is already complex enough, then there is not much to improve at each step, which leads to overfitting.
- We allow Random Forest to have fully-grown trees, but for AdaBoost, I don't think this is a good idea, based on the above plots. Fully-grown trees in AdaBoost causes overfitting and drop in test score.

Question 4.4: Apply 5-fold cross-validation to choose the optimal number of trees B for the ensemble and the optimal tree depth for the base learners. How does an ensemble classifier fitted with the optimal number of trees and the optimal tree depth compare with the random forest model fitted in Question 3.4?

```
In [34]: train_cv_n_tree = np.zeros((9,5))
         valid_cv_n_tree = np.zeros((9,5))

         fold_ctr = 0

         for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9001).split(X_trai
             X_train_cv = X_train.iloc[itrain,:]
             y_train_cv = y_train.iloc[itrain]
             X_valid_cv = X_train.iloc[ivalid,:]
             y_valid_cv = y_train.iloc[ivalid]

             for i, t in enumerate(tree_numbers):
         #         print("max_feature =", i)
                 adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=None), n_estima
                 adaboost.fit(X_train_cv, y_train_cv)

                 train_cv_n_tree[i,fold_ctr] = adaboost.score(X_train_cv, y_train_cv)
                 valid_cv_n_tree[i,fold_ctr] = adaboost.score(X_valid_cv, y_valid_cv)

             fold_ctr += 1

In [35]: # train_cv_n_tree[:-1], valid_cv_n_tree[:-1]

In [36]: scores_train_tree = np.mean(train_cv_n_tree, axis=1)[:-1]
         scores_valid_tree = np.mean(valid_cv_n_tree, axis=1)[:-1]

In [37]: # choose optimum as smallest number of trees "sufficiently" close to maximum validati
         for i in range(len(scores_valid_tree)):
             if np.max(scores_valid_tree)-scores_valid_tree[i]<0.002:
                 n_trees_optimal=i+1
                 break

         optimal_n_tree = 2**n_trees_optimal
         print("the optimal number of trees: ", optimal_n_tree)

the optimal number of trees:  128


In [38]: train_cv_n_depth = np.zeros((5,5))
         valid_cv_n_depth = np.zeros((5,5))

         fold_ctr = 0

         for itrain, ivalid in KFold(n_splits=5, shuffle=True, random_state=9001).split(X_trai
             X_train_cv = X_train.iloc[itrain,:]
             y_train_cv = y_train.iloc[itrain]
             X_valid_cv = X_train.iloc[ivalid,:]
             y_valid_cv = y_train.iloc[ivalid]
```

```
        for i, d in enumerate([1, 2, 10, None]):
#             print("max_feature =", i)
            adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=d), n_estimato
            adaboost.fit(X_train_cv, y_train_cv)

            train_cv_n_depth[i,fold_ctr] = adaboost.score(X_train_cv, y_train_cv)
            valid_cv_n_depth[i,fold_ctr] = adaboost.score(X_valid_cv, y_valid_cv)

        fold_ctr += 1
```

```
In [39]: # train_cv_n_depth, valid_cv_n_depth
```

```
In [40]: scores_train_depth = np.mean(train_cv_n_depth, axis=1)[:-1]
         scores_valid_depth = np.mean(valid_cv_n_depth, axis=1)[:-1]
```

```
In [41]: # choose optimum as smallest number of depths "sufficiently" close to maximum validat
         for i in range(len(scores_valid_depth)):
             if np.max(scores_valid_depth)-scores_valid_depth[i]<0.002:
                 n_predictors_optimal=i
                 break

         optimal_n_depth = [1, 2, 10, None][i]
         print("optimal depth for the base learner: ", optimal_n_depth)
```

```
optimal depth for the base learner:  2
```

```
In [42]: adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=optimal_n_depth), n_es
         adaboost.fit(X_train, y_train)
         print("Test score with the optimal number of tree and the optimal tree depth:", adabo
```

```
Test score with the optimal number of tree and the optimal tree depth: 0.6874
```

How does an ensemble classifier fitted with the optimal number of trees and the optimal tree depth compare with the random forest model fitted in Question 3.4?

- The ensemble classifier fitted with the optimal number of trees and the optimal tree depth has test score of 0.687, whereas the random forest model fitted in Question 3.4 has test score of 0.6968, so it is a little worse than the random forest model, but they do not differ much.

3.5 Question 5 (3pt): Meta-classifier

We have so far explored techniques that grow a collection of trees either by creating multiple copies of the original training set, or through a sequential procedure, and then combines these trees into a single classifier. Consider an alternate scenario where you are provided with a pre-trained collection of trees, say from different participants of a data science competition for Higgs boson discovery. What would be a good strategy to combine these pre-fitted trees into a single powerful classifier? Of course, a simple approach would be to take the majority vote from the individual trees. Can we do better than this simple combination strategy?

A collection of 100 decision tree classifiers is provided in the file `models.npy` and can be loaded into an array by executing:

`models = np.load('models.npy')`

You can make predictions using the i^{th} model on an array of predictors `x` by executing:

`model[i].predict(x)` or `model[i].predict_proba(x)`

and score the model on predictors `x` and labels `y` by using:

`model[i].score(x, y)`.

1. Implement a strategy to combine the provided decision tree classifiers, and compare the test perfomance of your approach with the majority vote classifier. Explain your strategy/algorithm.

```
In [43]: models = np.load('models.npy', encoding='latin1')
```

```
/Users/yijunshen/anaconda3/lib/python3.6/site-packages/sklearn/base.py:311: UserWarning: Tryin
  UserWarning)
```

Majority Vote Classifier

```
In [44]: majority_test = []
         for i in range(100):
             majority_test.append(models[i].predict(X_test))

         major_test_pred = pd.DataFrame(np.array(majority_test)).T.mode(axis=1)[0].tolist()
```

```
In [45]: test_true = y_test.tolist()
         diff_test_major = [a - b for a, b in zip(test_true, major_test_pred)]
         correct_test_major = sum(1 for x in diff_test_major if x == 0)
         N = len(test_true)
         score_test_major = correct_test_major / N
         print("The majority classifier gives test performance of", score_test_major)
```

```
The majority classifier gives test performance of 0.6646
```

Meta-Classifier:

```
In [46]: models_scores = []
         for i in range(100):
             models_scores.append(models[i].score(X_train, y_train))
         sum_scores = sum(models_scores)
         for n, score in enumerate(models_scores):
             models_scores[n] = score/sum_scores
         # models_scores
```

```
In [47]: meta = []
         for i, score in enumerate(models_scores):
             meta.append(models[i].predict(X_test) * score)
         meta_array = np.asarray(meta)
         # meta_array
         weighted_result = np.sum(meta_array, axis=0)
         # weighted_result

In [48]: threshold_result = weighted_result
         threshold_result[threshold_result < 0.5] = 0
         threshold_result[threshold_result > 0.5] = 1
         # threshold_result

In [49]: test_true = y_test.tolist()
         diff_test_meta = [a - b for a, b in zip(test_true, threshold_result)]
         correct_test_meta = sum(1 for x in diff_test_meta if x == 0)
         N = len(test_true)
         score_test_meta = correct_test_meta / N
         print("The meta-classifier gives test performance of", score_test_meta)
```

```
The meta-classifier gives test performance of 0.6658
```

Compare the test perfomance of your approach with the majority vote classifier. Explain your strategy/algorithm.

- My strategy is first apply each of the 100 models to the training dataset, and get the score of each model. To calculate the weight for each model, I divide the score by the sum of the 100 scores to get a ratio.
- The meta model combines the 100 models with their weights respectively. Each model will be used to predict the test set, and the result will be multiplied by that model's weight. Then I add up the weighted prediction from the 100 models to get a final weighted prediction for each observation.
- I set the threshold for the weighted prediction at 0.5. So above 0.5, the observation will be classified to class 1 and below to be class 0.
- The meta-classifier gives test performance of 0.6658, whereas the majority vote classifier gives test performance of 0.6646. So it improves a little bit.

3.6 APCOMP209a - Homework Question

We've worked with imputation methods on missing data in Homework 6. We've worked with Decision Trees in HW7 and here. Now let's see what happens if we try to work with Decision Trees and Missing Data at the same time! We'll be working with a dataset from the UCI Machine Learning Repository that uses a variety of wine chemical predictors to classify wines grown in the same region in Italy. Each line represents 13 (mostly chemical) predictors of the response variable wine class, including things like alcohol content, hue , and phenols. Unfortunately some of the predictor values were lost in measurement. Please load `wine_quality_missing.csv`. *Note*: As in HW6 be careful of reading/treating column names and row names in this data set. 1. Remove

all observations that contain and missing values, split the dataset into a 75-25 train-test split, and fit the sklearn DecisionTreeClassifier and RandomForestClassifier. Use cross-validation to find the optimal tree depth for each method. Report the optimal tree-depth, overall classification rate and confusion matrix on the test set for each method. 2. Restart with a fresh copy of the data and impute the missing data via mean imputation. Split the data 75-25 and again fit DecisionTreeClassifier and RandomForestClassifier using cross-validation to find the optimal tree depth. Report the optimal tree depth, overall classification rate and confusion matrix on the test set for each method.
3. Again restart with a fresh copy of the data but this time let's try something different. As discussed in section, CART Decision Trees can take advantage of surrogate splits to handle missing data. Split the data 75-25 and construct a **custom** decision tree model and train it on the training set with missing data. Report the optimal tree depth, overall classification rate and confusion matrix on the test set and compare your results to the Imputation and DecisionTree model results in part 1 & 2.

```
In [50]: wine_data = pd.read_csv("wine_quality_missing.csv")
         wine_data.columns = wine_data.columns.map(lambda x: x.replace(' ', '_'))
         wine_data.head()
```

```
Out[50]:    Alcohol  Malic_acid   Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
         0    14.23        1.71  2.43               15.6      127.0           2.80
         1    13.20        1.78  2.14                NaN      100.0           2.65
         2    13.16        2.36  2.67               18.6      101.0           2.80
         3    14.37        1.95  2.50                NaN      113.0           3.85
         4    13.24        2.59  2.87               21.0      118.0           2.80

            Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity   Hue  \
         0        3.06                  0.28             2.29             5.64  1.04
         1        2.76                  0.26             1.28             4.38  1.05
         2        3.24                  0.30             2.81             5.68  1.03
         3        3.49                  0.24             2.18             7.80  0.86
         4        2.69                  0.39              NaN             4.32  1.04

            OD280/OD315_of_diluted_wines  Proline  Class
         0                          3.92   1065.0      1
         1                          3.40   1050.0      1
         2                          3.17   1185.0      1
         3                          3.45   1480.0      1
         4                          2.93    735.0      1
```

```
In [51]: # wine_X = wine_data.iloc[:,:-1]
         # wine_y = wine_data.iloc[:,-1]
```

1. Remove all observations that contain and missing values, split the dataset into a 75-25 train-test split, and fit the sklearn DecisionTreeClassifier and RandomForestClassifier. Use cross-validation to find the optimal tree depth for each method. Report the optimal tree-depth, overall classification rate and confusion matrix on the test set for each method.

```
In [52]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import confusion_matrix

In [53]: # remove all missing values
         wine_data1 = wine_data.dropna()
         wine_X = wine_data1.iloc[:,:-1]
         wine_y = wine_data1.iloc[:,-1]
         wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(wine_X, wine_
```

DecisionTreeClassifier

```
In [54]: # DecisionTreeClassifier
         dt_scores = []
         for depth in range(1,10):
             clf = DecisionTreeClassifier(max_depth = depth, random_state=50)
             # Perform 5-fold cross validation
             score = cross_val_score(estimator=clf, X=wine_X_train, y=wine_y_train, cv=5)
             dt_scores.append((depth, np.mean(score)))

         best_depth_dt = max(dt_scores, key=lambda x:x[1])[0]
         print("The depth of decision tree model using 5-fold cross-validation is:", best_dept

The depth of decision tree model using 5-fold cross-validation is: 2


In [55]: clf_dt = DecisionTreeClassifier(max_depth = best_depth_dt, random_state=50)
         clf_dt.fit(wine_X_train,wine_y_train)
         print("classification accuracy on the test set is:", clf_dt.score(wine_X_test, wine_y

classification accuracy on the test set is: 0.636363636364


In [56]: expected = wine_y_test
         predicted = clf_dt.predict(wine_X_test)

         conf_mat = confusion_matrix(expected, predicted)
         conf_df = pd.DataFrame(conf_mat, columns = ['y_hat=1', 'y_hat = 2', 'y_hat = 3'], inde
         conf_df

Out[56]:      y_hat=1  y_hat = 2  y_hat = 3
         y=1        3          2          0
         y=2        0          2          0
         y=3        0          2          2
```

RandomForestClassifier

```
In [57]: # RandomForestClassifier
         rf_scores = []
         for depth in range(1,10):
             clf = RandomForestClassifier(max_depth = depth, random_state=50)
             # Perform 5-fold cross validation
             score = cross_val_score(estimator=clf, X=wine_X_train, y=wine_y_train, cv=5)
             rf_scores.append((depth, np.mean(score)))

         best_depth_rf = max(rf_scores, key=lambda x:x[1])[0]
         print("The depth of random forest model using 5-fold cross-validation is:", best_dept
```

```
The depth of random forest model using 5-fold cross-validation is: 3
```

```
In [58]: clf_rf = RandomForestClassifier(max_depth = best_depth_rf, random_state=50)
         clf_rf.fit(wine_X_train,wine_y_train)
         print("classification accuracy on the test set is:", clf_rf.score(wine_X_test, wine_y
```

```
classification accuracy on the test set is: 0.818181818182
```

```
In [59]: expected = wine_y_test
         predicted = clf_rf.predict(wine_X_test)

         conf_mat = confusion_matrix(expected, predicted)
         conf_df = pd.DataFrame(conf_mat, columns = ['y_hat=1', 'y_hat = 2', 'y_hat = 3'], inde
         conf_df
```

```
Out[59]:      y_hat=1  y_hat = 2  y_hat = 3
         y=1        5          0          0
         y=2        0          1          1
         y=3        0          1          3
```

2. Restart with a fresh copy of the data and impute the missing data via mean imputation. Split the data 75-25 and again fit DecisionTreeClassifier and RandomForestClassifier using cross-validation to find the optimal tree depth. Report the optimal tree depth, overall classification rate and confusion matrix on the test set for each method.

```
In [60]: from sklearn.preprocessing import Imputer
         wine_data2 = wine_data.copy()
         values = wine_data2.values
         imputer = Imputer( strategy='mean')
         wine_data2 = imputer.fit_transform(values)
         msk = np.random.rand(len(wine_data2)) < 0.75
         wine_train2 = wine_data2[msk]
         wine_test2 = wine_data2[~msk]
```

```
In [61]: wine_X_train2 = wine_train2[:, 0:-1]
         wine_y_train2 = wine_train2[:, -1]
         wine_X_test2 = wine_test2[:, 0:-1]
         wine_y_test2 = wine_test2[:, -1]
```

DecisionTreeClassifier

```
In [62]: # DecisionTreeClassifier
         dt2_scores = []
         for depth in range(1,20):
             clf = DecisionTreeClassifier(max_depth = depth, random_state=50)
             # Perform 5-fold cross validation
             score = cross_val_score(estimator=clf, X=wine_X_train2, y=wine_y_train2, cv=5)
             dt2_scores.append((depth, np.mean(score)))

         best_depth_dt2 = max(dt2_scores, key=lambda x:x[1])[0]
         print("The depth of decision tree model using 5-fold cross-validation is:", best_dept
```

```
The depth of decision tree model using 5-fold cross-validation is: 5
```

```
In [63]: clf_dt2 = DecisionTreeClassifier(max_depth = best_depth_dt2, random_state=50)
         clf_dt2.fit(wine_X_train2,wine_y_train2)
         print("classification accuracy on the test set is:", clf_dt2.score(wine_X_test2, wine_
```

```
classification accuracy on the test set is: 0.772727272727
```

```
In [64]: expected = wine_y_test2
         predicted = clf_dt2.predict(wine_X_test2)

         conf_mat = confusion_matrix(expected, predicted)
         conf_df = pd.DataFrame(conf_mat, columns = ['y_hat=1', 'y_hat = 2', 'y_hat = 3'], inde
         conf_df
```

```
Out[64]:      y_hat=1  y_hat = 2  y_hat = 3
         y=1       12          1          1
         y=2        0         15          2
         y=3        1          5          7
```

RandomForestClassifier

```
In [65]: # RandomForestClassifier
         rf2_scores = []
         for depth in range(1,10):
             clf = RandomForestClassifier(max_depth = depth, random_state=50)
             # Perform 5-fold cross validation
             score = cross_val_score(estimator=clf, X=wine_X_train2, y=wine_y_train2, cv=5)
             rf2_scores.append((depth, np.mean(score)))

         best_depth_rf2 = max(rf2_scores, key=lambda x:x[1])[0]
         print("The depth of random forest model using 5-fold cross-validation is:", best_dept
```

```
The depth of random forest model using 5-fold cross-validation is: 2
```

```
In [66]: clf_rf2 = RandomForestClassifier(max_depth = best_depth_rf2, random_state=50)
         clf_rf2.fit(wine_X_train2,wine_y_train2)
         print("classification accuracy on the test set is:", clf_rf2.score(wine_X_test2, wine_

classification accuracy on the test set is: 0.863636363636
```

```
In [67]: expected = wine_y_test2
         predicted = clf_rf2.predict(wine_X_test2)

         conf_mat = confusion_matrix(expected, predicted)
         conf_df = pd.DataFrame(conf_mat, columns = ['y_hat=1', 'y_hat = 2', 'y_hat = 3'], inde
         conf_df

Out[67]:      y_hat=1  y_hat = 2  y_hat = 3
         y=1       12          2          0
         y=2        0         16          1
         y=3        0          3         10
```

3. Again restart with a fresh copy of the data but this time let's try something different. As discussed in section, CART Decision Trees can take advantage of surrogate splits to handle missing data. Split the data 75-25 and construct a custom decision tree model and train it on the training set with missing data. Report the optimal tree depth, overall classification rate and confusion matrix on the test set and compare your results to the Imputation and DecisionTree model results in part 1 & 2.

```
In [79]: wine_data3 = wine_data.copy()

In [80]: nans = lambda df: df[df.isnull().any(axis=1)]
         wine3_missing = nans(wine_data3)

In [81]: msk = np.random.rand(len(wine_data3)) < 0.75
         wine_train3 = wine_data3[msk]
         wine_test3 = wine_data3[~msk]

In [82]: wine_X_train3 = wine_train3.loc[:, wine_train3.columns != 'Class']
         wine_y_train3 = wine_train3['Class']
         wine_X_test3 = wine_test3.loc[:, wine_test3.columns != 'Class']
         wine_y_test3 = wine_test3['Class']

In [83]: def surrogate_fit(df_train, max_depth):
             X_train = df_train.drop('Class', axis=1)
             y_train = df_train['Class']

             def gini_coeff(left, right):
                 gini_1 = 1 - np.sum([(i/left.shape[0])**2 for i in left['Class'].value_counts
                 gini_2 = 1 - np.sum([(i/right.shape[0])**2 for i in right['Class'].value_coun
                 n = left.shape[0] + right.shape[0]
                 gini_value = (gini_1 * left.shape[0] + gini_2 * right.shape[0]) / n
```

```
        return gini_value
    num_cols = list(range(X_train.shape[1]))
    col_names = X_train.columns.values

    def best_split(data):
        best_gini = 10000
        best_col = None
        best_t = None
        for col in num_cols:
            dropped = data.dropna(subset=[col_names[col]])
            for t in range(dropped.shape[0]):
                left = dropped[dropped.iloc[:,col] <= dropped.iloc[t,col]]
                right = dropped[dropped.iloc[:,col] > dropped.iloc[t,col]]
                gini = gini_coeff(left, right)
                if gini < best_gini:
                    best_gini = gini
                    best_col = col
                    best_t = dropped.iloc[t,col]
#          dropped = data.dropna(subset=[col_names[best_col]])
        return best_col, best_t, dropped

    tree = {}
    stack = [{'data':df_train, 'depth':1}]
    n_nodes = int(np.sum([2**i for i in range(max_depth)]))
    groups = [0 for i in range(n_nodes)]
    while stack:
        data = stack.pop()
        if data['depth'] == max_depth:
            tree[len(tree)] = {'class': data['data']['Class'].value_counts().idxmax()
        else:
            n = len(tree)
            best_col, best_t, dropped = best_split(data['data'])
            if len(dropped['Class'].unique()) == 1:
                for i in range(n, n+1+np.sum([2**i for i in range(1, max_depth-data['
                    tree[i] = {'class': dropped['Class'].value_counts().idxmax()}
            else:
                left = dropped[dropped.iloc[:,best_col] <= best_t]
                right = dropped[dropped.iloc[:,best_col] > best_t]
                tree[n] = {'feature': best_col, 'threshold': best_t, 'left_node': n+1
                           'left': set(left.index), 'right': set(right.index)}
                groups[n] = dropped
                stack.extend([{'data':right, 'depth':data['depth']+1}, {'data':left,

    for i in range(n_nodes):
        if 'class' not in tree[i]:
            other_columns = num_cols[:tree[i]['feature']] + num_cols[tree[i]['feature
            closeness = []
            for col in other_columns:
```

```
                         dropped = groups[i].dropna(subset=[col_names[col]])
                         best_c = 0
                         best_t = None
                         for t in range(dropped.shape[0]):
                             left = set(dropped[dropped.iloc[:,col] <= dropped.iloc[t,col]].in
                             right = set(dropped[dropped.iloc[:,col] > dropped.iloc[t,col]].in
                             c = (len(tree[i]['left'].intersection(left)) + len(tree[i]['right
                             if c > best_c:
                                 best_c = c
                                 best_t = dropped.iloc[t,col]
                         closeness.append([best_c, col, best_t])
                     closeness = sorted(closeness, reverse=True, key = lambda x: x[0])
                     tree[i]['surrogate'] = closeness
             return tree

In [84]: def surrogate_predict(X, tree):
             pred_test = []
             for i in range(X.shape[0]):
                 current = 0
                 while 'class' not in tree[current]:
                     value = X.iloc[i, tree[current]['feature']]
                     if not np.isnan(value):
                         if value <= tree[current]['threshold']:
                             current = tree[current]['left_node']
                         else:
                             current = tree[current]['right_node']
                         continue
         #                 cur = tree[cur]['left_node'] if value <= tree[cur]['threshold'] els
                     else:
                         for surrogate in tree[current]['surrogate']:
                             value = X.iloc[i, surrogate[1]]
                             if not np.isnan(value):
                                 if value <= surrogate[2]:
                                     current = tree[current]['left_node']
                                 else:
                                     current = tree[current]['right_node']
         #                         cur = tree[cur]['left_node'] if value <= sur[2] else tree[c
                                 break
                 pred_test.append(tree[current]['class'])
             return pred_test

In [85]: tree = surrogate_fit(wine_train3, 5)

In [86]: surro_pred = surrogate_predict(wine_X_test3, tree)

In [87]: wine_true_y = wine_y_test3.tolist()

In [88]: diff_surro = [a - b for a, b in zip(wine_true_y, surro_pred)]
         correct_surro = sum(1 for x in diff_surro if x == 0)
         accuracy_surro = correct_surro / len(wine_true_y)

         print("surrogate accuracy:", accuracy_surro)

surrogate accuracy: 0.8604651162790697


In [89]: conf_mat = confusion_matrix(wine_true_y, surro_pred)
         conf_df = pd.DataFrame(conf_mat, columns = ['y_hat=1', 'y_hat = 2', 'y_hat = 3'], inde
         conf_df

Out[89]:      y_hat=1  y_hat = 2  y_hat = 3
         y=1       16          1          0
         y=2        2         10          1
         y=3        0          2         11
```