cs109-hw8

November 20, 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
Import libraries:

```
In [306]: 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
          from sklearn.model_selection import GridSearchCV
          from sklearn.ensemble import VotingClassifier
          from sklearn.metrics import accuracy_score
          from sklearn.model_selection import ParameterGrid
          from sklearn.model_selection import KFold
          from sklearn.model_selection import cross_val_score
          %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 [218]: train = pd.read_csv('Higgs_train.csv')
          test = pd.read_csv('Higgs_test.csv')
          x = list(train.columns)
          x.remove(' class')
          X_train = train[x]
          X_test = test[x]
          y_train = train[' class']
          y_test = test[' class']
```

```
In [16]: tree = DecisionTreeClassifier()
         # Use 5 fold cross validation to pick optimal tree depth
         parameter = {'max_depth': list(range(2,100))}
         tree_cv = GridSearchCV(tree, parameter, cv = 5)
         tree_cv.fit(X_train, y_train)
```

```
Out[16]: GridSearchCV(cv=5, error_score='raise',
                estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     presort=False, random_state=None, splitter='best'),
                fit_params={}, iid=True, n_jobs=1,
                param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=None, verbose=0)
```

```
In [21]: tree_cv.best_estimator_
```

```
Out[21]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     presort=False, random_state=None, splitter='best')
```

```
In [22]: max_depth = 5

In [269]: print('Train Score %f' %tree_cv.score(X_train, y_train))
          print('Test Score %f' %tree_cv.score(X_test, y_test))

Train Score 0.682000
Test Score 0.645400
```

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

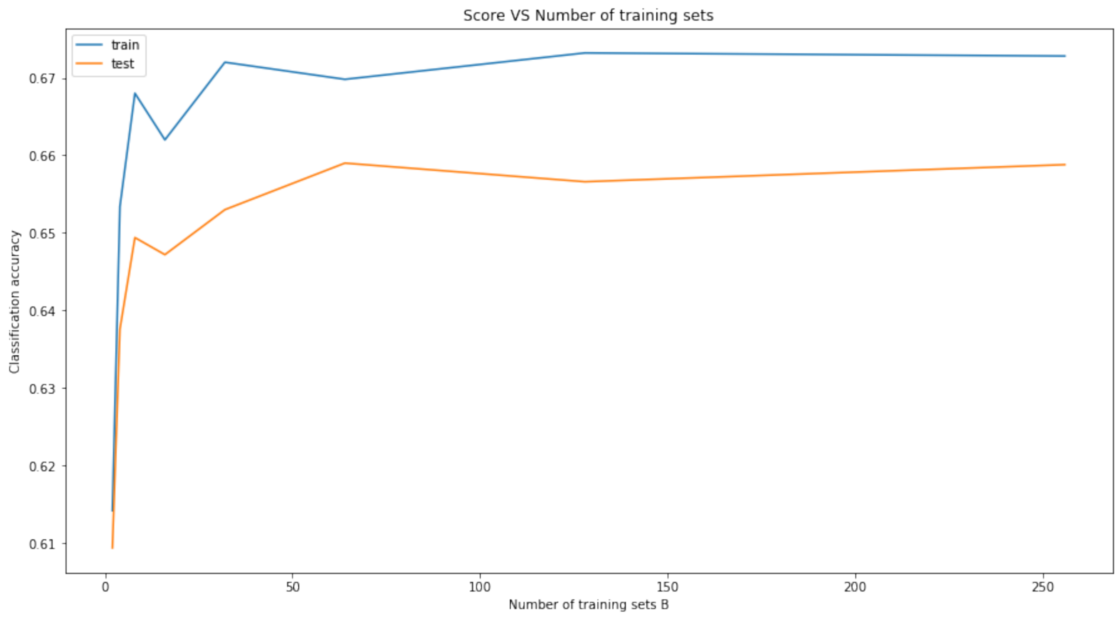
We start with a simple method inspired from the idea of 'dropout' in machine learning, where we fit multiple decision trees on random subsets of predictors, and combine them through a majority vote. The procedure is described below.

- 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. You may allow the trees to have unrestricted depth.
- Combine the decision tree models into a single classifier by taking a majority vote:

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

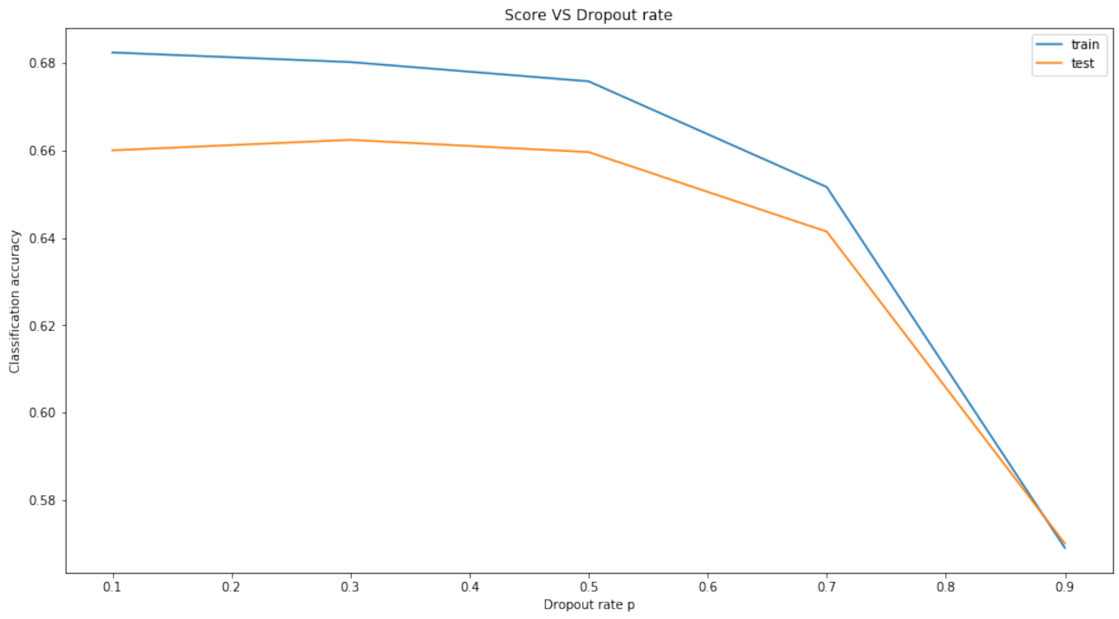
We shall refer to the combined classifier as an ** *ensemble classifier* **. Implement the described dropout approach, and answer the following questions: 1. Apply the dropout procedure with $p = 0.5$ for different number of trees (say $2, 4, 8, 16, \ldots, 256$), and evaluate the training and test accuracy 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.]

```
In [214]: def ensemble_classifier (p , b , X_train = X_train  , y_train = y_train, X_test= X_tes
              predictions_train = []
              predictions_test = []
              for i in range(b):
                  X_train_drop = X_train.applymap(lambda x: 0 if np.random.random() < p else x)
                  model = DecisionTreeClassifier(max_depth=max_depth)
                  model.fit(X_train_drop, y_train)
                  predictions_train.append(model.predict(X_train))
                  predictions_test.append(model.predict(X_test))
              predictions_train = np.mean(predictions_train, axis=0).round()
              predictions_test = np.mean(predictions_test, axis=0).round()
              return (accuracy_score(y_train,predictions_train), accuracy_score(y_test,predictio
In [215]: train_score = []
          test_score = []
          for i in np.power(2, list(range(1,9))):
              train, test = ensemble_classifier(.5, i)
              train_score.append(train)
              test_score.append(test)
          plt.figure(figsize = (15,8))
          plt.plot(np.power(2, list(range(1,9))), train_score, label = 'train')
          plt.plot(np.power(2, list(range(1,9))), test_score, label = 'test')
          plt.title('Score VS Number of training sets')
          plt.xlabel('Number of training sets B')
          plt.ylabel('Classification accuracy')
          plt.legend()
          plt.show()
```



1. 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. An increase in the number of trees increases variance, but reduces bias. There is very little difference that occurs past 50 trees.

```
In [216]: train_score = []
          test_score = []
          for i in [0.1,0.3,0.5,0.7,0.9]:
              train, test = ensemble_classifier(i, 64)
              train_score.append(train)
              test_score.append(test)
          plt.figure(figsize = (15,8))
          plt.plot([0.1,0.3,0.5,0.7,0.9], train_score, label = 'train')
          plt.plot([0.1,0.3,0.5,0.7,0.9], test_score, label = 'test')
          plt.title('Score VS Dropout rate')
          plt.xlabel('Dropout rate p')
          plt.ylabel('Classification accuracy')
          plt.legend()
          plt.show()
```



2. explain how the dropout rate influences the bias and variance of the combined classifier.
An increase in the dropout rate decreases variance but increases bias. This effect seems to increase in a faster non-linearly fashion as the dropout rate increases.

```
In [219]: kf = KFold(n_splits=5)
          parameters = {'p':[0.1,0.3,0.5,0.7,0.9], 'b':np.power(2, list(range(1,9)))}
          results = []
          for i in ParameterGrid(parameters):
              scores = []
              for train_index, test_index in kf.split(train):
                  train_set = train.iloc[train_index]
                  test_set = train.iloc[test_index]
                  yK = train_set[' class']
                  yK_test = test_set[' class']
                  xK = list(train_set.columns)
                  xK.remove(' class')
                  XK = train_set[xK]
                  XK_test = test_set[xK]
                  tr, score = ensemble_classifier(**i, X_train =XK, y_train = yK, X_test= XK_tes
                  scores.append(score)
              i['score'] = np.mean(scores)
              print(i)
              results.append(i)
```

```
{'b': 2, 'p': 0.1, 'score': 0.6400000000000001}
{'b': 2, 'p': 0.3, 'score': 0.6310000000000001}
{'b': 2, 'p': 0.5, 'score': 0.6161999999999997}
{'b': 2, 'p': 0.7, 'score': 0.5911999999999995}
{'b': 2, 'p': 0.9, 'score': 0.5123999999999997}
{'b': 4, 'p': 0.1, 'score': 0.6450000000000002}
{'b': 4, 'p': 0.3, 'score': 0.6452}
{'b': 4, 'p': 0.5, 'score': 0.6318000000000003}
{'b': 4, 'p': 0.7, 'score': 0.6145999999999992}
{'b': 4, 'p': 0.9, 'score': 0.5130000000000012}
{'b': 8, 'p': 0.1, 'score': 0.6445999999999995}
{'b': 8, 'p': 0.3, 'score': 0.6444000000000008}
{'b': 8, 'p': 0.5, 'score': 0.63}
{'b': 8, 'p': 0.7, 'score': 0.6284000000000007}
{'b': 8, 'p': 0.9, 'score': 0.5322000000000012}
{'b': 16, 'p': 0.1, 'score': 0.6466000000000006}
{'b': 16, 'p': 0.3, 'score': 0.6508000000000005}
{'b': 16, 'p': 0.5, 'score': 0.6430000000000002}
{'b': 16, 'p': 0.7, 'score': 0.6320000000000001}
{'b': 16, 'p': 0.9, 'score': 0.5412000000000001}
{'b': 32, 'p': 0.1, 'score': 0.6518000000000005}
{'b': 32, 'p': 0.3, 'score': 0.6475999999999995}
{'b': 32, 'p': 0.5, 'score': 0.6493999999999998}
{'b': 32, 'p': 0.7, 'score': 0.6256000000000004}
{'b': 32, 'p': 0.9, 'score': 0.5442000000000002}
{'b': 64, 'p': 0.1, 'score': 0.6473999999999998}
{'b': 64, 'p': 0.3, 'score': 0.6518000000000005}
{'b': 64, 'p': 0.5, 'score': 0.6435999999999995}
{'b': 64, 'p': 0.7, 'score': 0.6310000000000001}
{'b': 64, 'p': 0.9, 'score': 0.5467999999999995}
{'b': 128, 'p': 0.1, 'score': 0.6482000000000011}
{'b': 128, 'p': 0.3, 'score': 0.6510000000000002}
{'b': 128, 'p': 0.5, 'score': 0.6427999999999993}
{'b': 128, 'p': 0.7, 'score': 0.63}
{'b': 128, 'p': 0.9, 'score': 0.5347999999999994}
{'b': 256, 'p': 0.1, 'score': 0.6487999999999993}
{'b': 256, 'p': 0.3, 'score': 0.6510000000000002}
{'b': 256, 'p': 0.5, 'score': 0.6470000000000002}
{'b': 256, 'p': 0.7, 'score': 0.6308000000000003}
{'b': 256, 'p': 0.9, 'score': 0.5308000000000005}
```

```
In [233]: data = pd.DataFrame(results)
          data.sort_values('score', ascending = False).head(1)
```

```
Out[233]:      b    p   score
          20  32  0.1  0.6518
```

```
In [234]: train_score, test_score = ensemble_classifier(.1,32)
```

```
In [235]: print('Train Score %f' %train_score)
          print('Test Score %f' %test_score)
```

```
Train Score 0.688000
Test Score 0.659000
```

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? Using 5-fold cross-validation we see that a $B = 32, p = 0.1$ is optimal. This model has a better train score (very small difference) and test score (comparitvely larger effect than train).

3.3 Question 3 (15pt): Random Forests

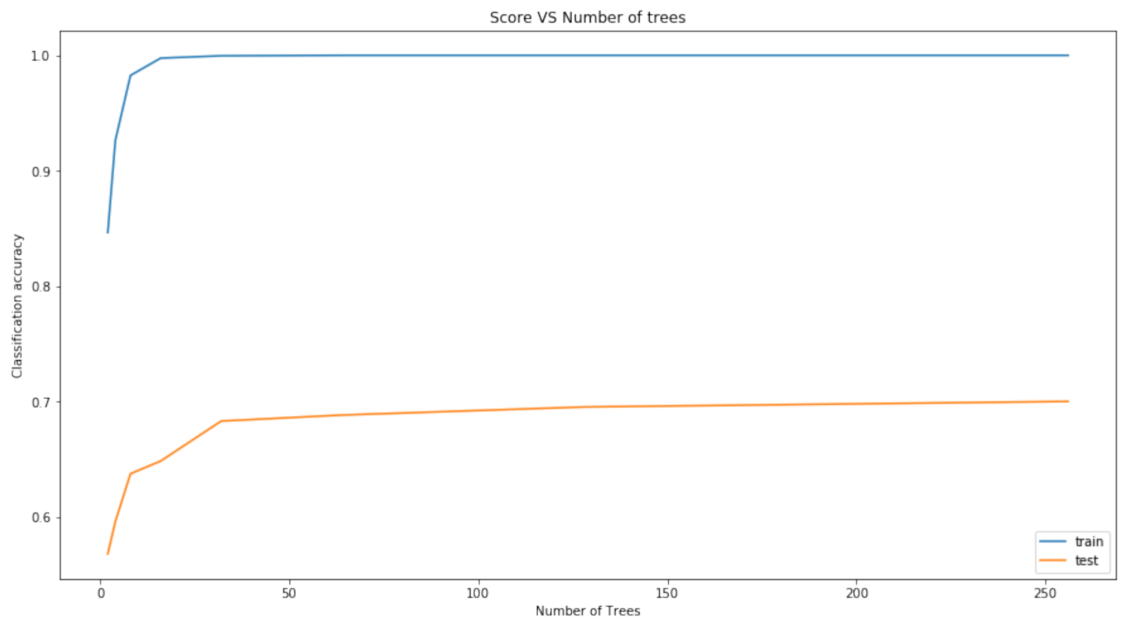
We now move to a more sophisticated ensemble technique, namely random forest: 1. How does a random forest approach differ from the dropout procedure described in Question 2?

- 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?

1. How does a random forest approach differ from the dropout procedure described in Question 2? In two ways 1. It uses bagging, so a bootstrap method is used for data selection rather than randomly dropping a precentage of the data 2. to decorollate trees, random forests then selects a random subset of the predictors to use in the creation of the tree that is then aggregated.

```
In [236]: train_score = []
          test_score = []
          for i in np.power(2, list(range(1,9))):
              forest = RandomForestClassifier(i)
              forest.fit(X_train, y_train)
              train_score.append(forest.score(X_train, y_train))
```

```
    test_score.append(forest.score(X_test, y_test))
plt.figure(figsize = (15,8))
plt.plot(np.power(2, list(range(1,9))), train_score, label = 'train')
plt.plot(np.power(2, list(range(1,9))), test_score, label = 'test')
plt.title('Score VS Number of trees')
plt.xlabel('Number of Trees')
plt.ylabel('Classification accuracy')
plt.legend()
plt.show()
```



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. It increases the test accuracy with more trees but to a point (more than 50 has very little return). Increased trees lowers bias with little apparent increase in variance.

```
In [238]: # Use 5 fold cross validation to number of predictors
          forest = RandomForestClassifier(50)
          parameter = {'max_features': list(range(1,29))}
          forest_cv = GridSearchCV(forest, parameter, cv = 5)
          forest_cv.fit(X_train, y_train)

Out[238]: GridSearchCV(cv=5, error_score='raise',
                 estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=50, n_jobs=1, oob_score=False, random_state=None,
                      verbose=0, warm_start=False),
                 fit_params={}, iid=True, n_jobs=1,
                 param_grid={'max_features': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                 scoring=None, verbose=0)
```

```
In [239]: forest_cv.best_estimator_
```

```
Out[239]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features=7, max_leaf_nodes=None,
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=50, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)
```

```
In [241]: print('Train Score = %f'%forest_cv.score(X_train, y_train))
          print('Test Score = %f'%forest_cv.score(X_test, y_test))
```

```
Train Score = 0.999800
Test Score = 0.695400
```

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? The optimal values for number of predictors is 7, and this model has a much better train and test score when compared to the drop out approach

3.4 Question 4 (15pt): Boosting

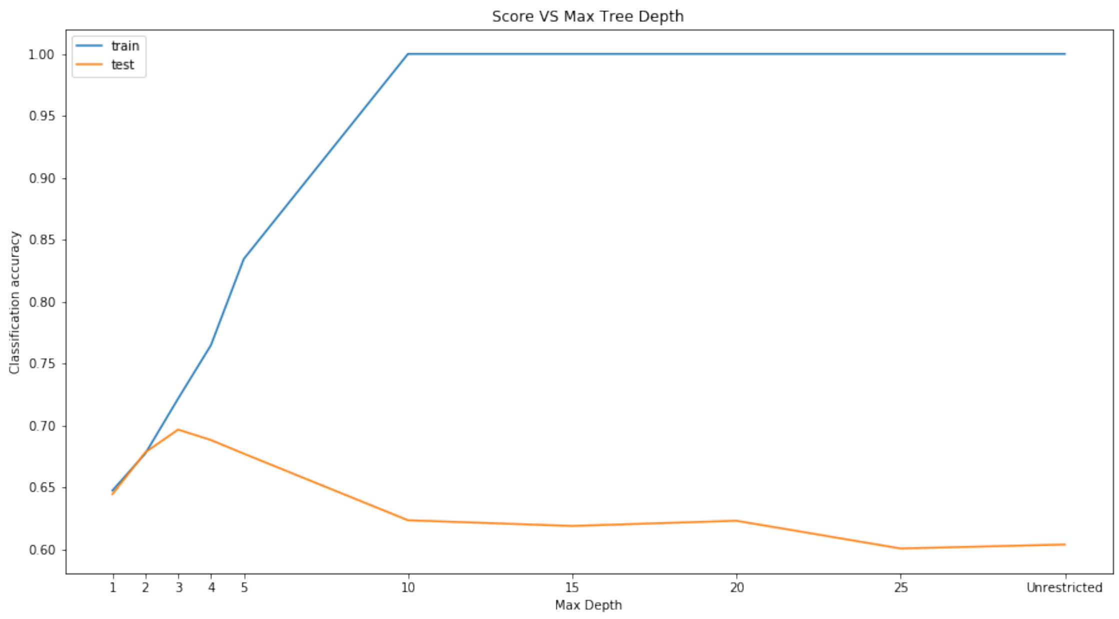
We next compare the random forest model with the approach of boosting:

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 $2, 4, 8, 16, \ldots, 256$).

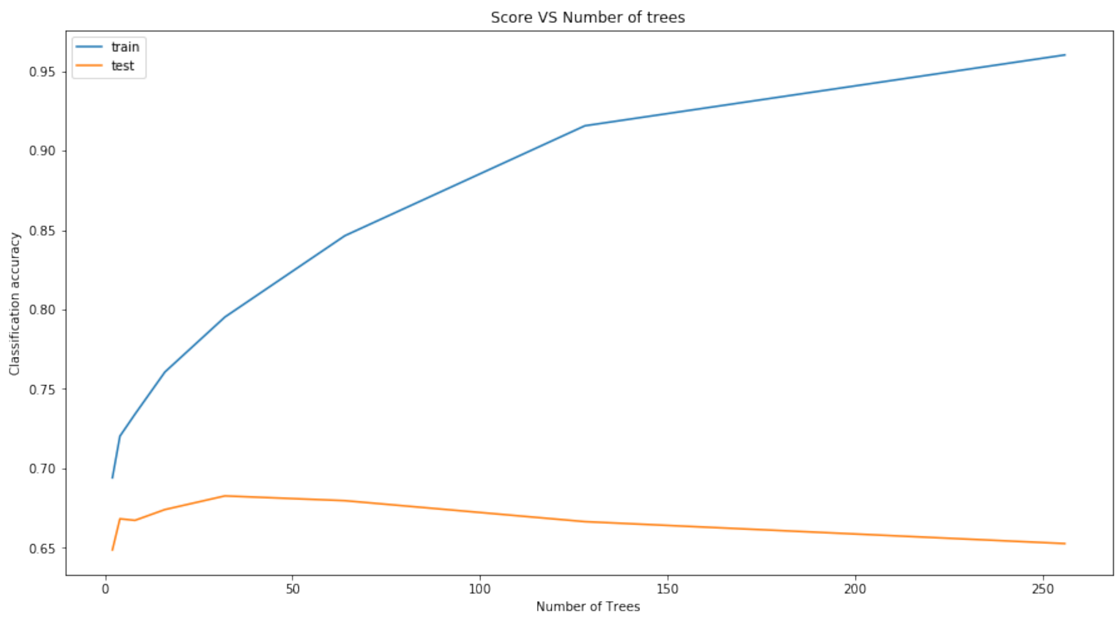
- How does the number of trees influence the training and test performance? Compare and contrast between the trends you see in the training and test performance of AdaBoost and that of the random forest models in Question 3. Give an explanation for your observations.
- How does the tree depth of the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

```
In [261]: train_score = []
          test_score = []
          for i in [1,2,3,4,5,10,15,20,25,None]:
              adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth=i), learning_rate=0
              adaboost.fit(X_train, y_train)
              train_score.append(adaboost.score(X_train, y_train))
              test_score.append(adaboost.score(X_test, y_test))

In [264]: plt.figure(figsize = (15,8))
          plt.plot([1,2,3,4,5,10,15,20,25,30], train_score, label = 'train')
          plt.plot([1,2,3,4,5,10,15,20,25,30], test_score, label = 'test')
          plt.title('Score VS Max Tree Depth')
          plt.xlabel('Max Depth')
          plt.xticks([1,2,3,4,5,10,15,20,25,30], [1,2,3,4,5,10,15,20,25,'Unrestricted'])
          plt.ylabel('Classification accuracy')
          plt.legend()
          plt.show()
```



```
In [272]: train_score = []
          test_score = []
          for i in np.power(2, list(range(1,9))):
              adaboost = AdaBoostClassifier(DecisionTreeClassifier(max_depth = max_depth), learn
              adaboost.fit(X_train, y_train)
              train_score.append(adaboost.score(X_train, y_train))
              test_score.append(adaboost.score(X_test, y_test))
          plt.figure(figsize = (15,8))
          plt.plot(np.power(2, list(range(1,9))), train_score, label = 'train')
          plt.plot(np.power(2, list(range(1,9))), test_score, label = 'test')
          plt.title('Score VS Number of trees')
          plt.xlabel('Number of Trees')
          plt.ylabel('Classification accuracy')
          plt.legend()
          plt.show()
```



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. Increasd number of trees decreases bias to a point (around 75) whilst increasing variance. The adaboost model has much lower variance at low numbers of trees (less than 50) compared to the random forests model, but the training model has much higher bias. The test performance for the adaboost and random forest model seems very similar.

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. Tree depth improves training and test performance but only up to a point (around 3) afterwhich the variance increase dramatically and so does the bias. I would not recommend unrestricting tree depth as this is causing overfitting to the training data. Fitting to residuals with unrestricted depths starts to fit the model to the variance, causing overfitting.

```
In [265]: # Use 5 fold cross validation to pick number of trees and tree depth
          adaboost = AdaBoostClassifier(DecisionTreeClassifier(), learning_rate=0.05)
          parameter = {'base_estimator__max_depth':list(range(1,10)), 'n_estimators': np.power(2
          adaboost_cv = GridSearchCV(adaboost, parameter, cv = 5)
          adaboost_cv.fit(X_train, y_train)

Out[265]: GridSearchCV(cv=5, error_score='raise',
                 estimator=AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', m
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best'),
                    learning_rate=0.05, n_estimators=50, random_state=None),
                 fit_params={}, iid=True, n_jobs=1,
                 param_grid={'base_estimator__max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9], 'n_estima
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                 scoring=None, verbose=0)

In [267]: adaboost_cv.best_estimator_

Out[267]: AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', m
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best'),
                    learning_rate=0.05, n_estimators=256, random_state=None)

In [280]: print('Train Score = %f'%adaboost_cv.score(X_train, y_train))
          print('Test Score = %f'%adaboost_cv.score(X_test, y_test))

Train Score = 0.740200
Test Score = 0.694000
```

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? The optimal number of trees is $B = 256$ and the optimal max_depth is 2. This model has a worse training performance and a very slightly worse test preformance which is strange as our expectation would be improved performance with adaboosting

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

```
In [304]: scores = list(map(lambda x: x.score(X_train, y_train), models))
          weights = list(map(lambda x:(x-min(scores))/(max(scores)-min(scores)), scores))
```

```
In [343]: predictions_train = []
          predictions_test = []
          for model in models:
              predictions_train.append(model.predict(X_train))
              predictions_test.append(model.predict(X_test))
          X_train_meta = np.matrix(predictions_train).T
          X_test_meta = np.matrix(predictions_test).T
```

```
In [389]: print('Majority vote Train Score = %f'%accuracy_score(y_train,np.mean(predictions_trai
          print('Majority vote Test Score = %f'%accuracy_score(y_test,np.mean(predictions_test,
```

```
Majority vote Train Score = 0.657800
Majority vote Test Score = 0.664600
```

```
In [374]: adaboost = AdaBoostClassifier(DecisionTreeClassifier(), learning_rate=0.05)
          parameter = {'base_estimator__max_depth':list(range(1,5)), 'n_estimators': np.power(2,
          adaboost_cv = GridSearchCV(adaboost, parameter, cv = 3)
          adaboost_cv.fit(X_train_meta, y_train)
```

```
Out[374]: GridSearchCV(cv=3, error_score='raise',
                 estimator=AdaBoostClassifier(algorithm='SAMME.R',
                    base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', m
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best'),
                    learning_rate=0.05, n_estimators=50, random_state=None),
                 fit_params={}, iid=True, n_jobs=1,
                 param_grid={'base_estimator__max_depth': [1, 2, 3, 4], 'n_estimators': array([
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                 scoring=None, verbose=0)
```

```
In [390]: print('Meta Classifier Train Score = %f'%adaboost_cv.score(X_train_meta, y_train))
          print('Meta Classifier Test Score = %f'%adaboost_cv.score(X_test_meta, y_test))
```

```
Meta Classifier Train Score = 0.697600
Meta Classifier Test Score = 0.686000
```

The stacked method performs better than the majority vote for both the training and test set. The stacked method I chose takes the predicted classifications of all of the models and builds predictors out of them. A random forests model with adaboosting which has its tree depth and number of trees tuned using cross validation is then used on the new set of predictors to create the final stacked model.