Assignment 6, Boosting

November 14, 2018

1 Assignment 6 - Boosting

1.0.1 *Matthew Dunne*

```
In [1]: import pandas as pd
        import numpy as np
```

1.0.2 1. Data Processing

a) Import the data from the website directly: https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data (Links to an external site).

We can use pandas to read the data that is stored in csv format. Please note there is no header, so we will build column names in a later step. Also, we are going to remove leading white spaces (which just make things tough later).

```
In [2]: adult_df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/adul
```

b) There is no header included, but information on column names is here: https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names (Links to an external site).

```
In [3]: column_names = ['age', 'workclass', 'final_weight', 'education', 'education_num', 'mar
        adult_df.columns = column_names
        adult_df.head()
```

```
Out[3]:    age         workclass  final_weight  education  education_num  \
        0   39         State-gov         77516  Bachelors             13
        1   50  Self-emp-not-inc         83311  Bachelors             13
        2   38           Private        215646    HS-grad              9
        3   53           Private        234721       11th              7
        4   28           Private        338409  Bachelors             13

               marital_status         occupation   relationship   race     sex  \
        0       Never-married       Adm-clerical  Not-in-family  White    Male
        1  Married-civ-spouse    Exec-managerial        Husband  White    Male
        2            Divorced  Handlers-cleaners  Not-in-family  White    Male
        3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
```

```
   4  Married-civ-spouse     Prof-specialty          Wife  Black  Female

      capital_gain  capital_loss  hours_per_week native_country salary
   0          2174             0              40  United-States  <=50K
   1             0             0              13  United-States  <=50K
   2             0             0              40  United-States  <=50K
   3             0             0              40  United-States  <=50K
   4             0             0              40           Cuba  <=50K
```

c) Check your dataframe shape to verify that you have the correct # of rows and columns

```
In [4]: adult_df.shape

Out[4]: (32561, 15)
```

d) Drop the 3rd column from the data (it is referred to as "fnlwgt" on UCI's website and is not necessary in this homework)

```
In [5]: adult_df = adult_df.drop('final_weight', axis=1)
```

e) Note: There are random values of '?' that show up in the data - this is fine! These just refer to "unknown" and can be left as is. This data has no true NA values, so no need to check.

f) Use the .replace() method to make the following changes to the "salary" column:
"<=50K" should become 0
">50K" should become 1
Note: This step is essential to calculate the ROC_AUC score in model evaluation steps.

```
In [6]: adult_df.replace({'salary': {"<=50K": 0, ">50K": 1}}, inplace=True)
```

g) Create your X dataframe (just your predictors). It should include every feature except for the target variable which is "salary".

You should have the following shape: (32561, 13)

```
In [7]: X = adult_df.iloc[:, 0:13]
        X.shape

Out[7]: (32561, 13)
```

h) Create your y dataframe (just your target variable). It should only be "salary".

You should have the following shape: (32561,)
The values should only be 0 and 1.

```
In [8]: y = adult_df.iloc[:, 13]
        y.shape

Out[8]: (32561,)
```

i) For this homework we will try converting columns with factors to separate columns (i.e. one-hot encoding). It is not necessary for trees, but can be a very powerful tool to use. There are a variety of ways to do this, but we can use Pandas built-in method .get_dummies(). Pandas will automatically split out columns that are categorical. For now, just run across your full X dataframe.

```
In [9]: X_encoded = pd.get_dummies(X)
        X_encoded.shape

Out[9]: (32561, 107)
```

j) Split data into train / test set using an 70/30 split. Verify that you have the same number of columns in your X_train and X_test.

```
In [10]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=.30, rando
         print(X_train.shape, X_test.shape)

(22792, 107) (9769, 107)
```

1.0.3 2. Random Forest Classifier - Base Model:

Start by creating a simple Random Forest only using default parameters - this will let us compare Boosting methods to Random Forest in binary classification problems.

a) Use the RandomForestClassifier in sklearn. Fit your model on the training data.

```
In [11]: from sklearn.ensemble import RandomForestClassifier
         forest_clf = RandomForestClassifier(random_state=1007)
         rf_model=forest_clf.fit(X_train, y_train)
```

b) Use the fitted model to predict on test data. Use the .predict_proba() and the .predict() methods to get predicted probabilities as well as predicted classes.

```
In [12]: #the class predictions
         rf_pred_class=rf_model.predict(X_test)
         #the probability predictions
         rf_pred_prob=rf_model.predict_proba(X_test)
         #the probability predictions return a n x 2 array. First column = prob. of 0. Second 
         #for the roc_auc_score in step d you can only use one column - the prob. of being 1. 
         rf_prob_of_one=rf_pred_prob[:,1]
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

```
In [13]: from sklearn.metrics import confusion_matrix, classification_report
         print('Confusion Matrix\n')
         print(confusion_matrix(rf_pred_class, y_test))
         print('\nClassification Report\n')
         print(classification_report(rf_pred_class, y_test))
```

```
Confusion Matrix

[[6837  932]
 [ 632 1368]]

Classification Report

             precision    recall  f1-score   support

          0       0.92      0.88      0.90      7769
          1       0.59      0.68      0.64      2000

avg / total       0.85      0.84      0.84      9769
```

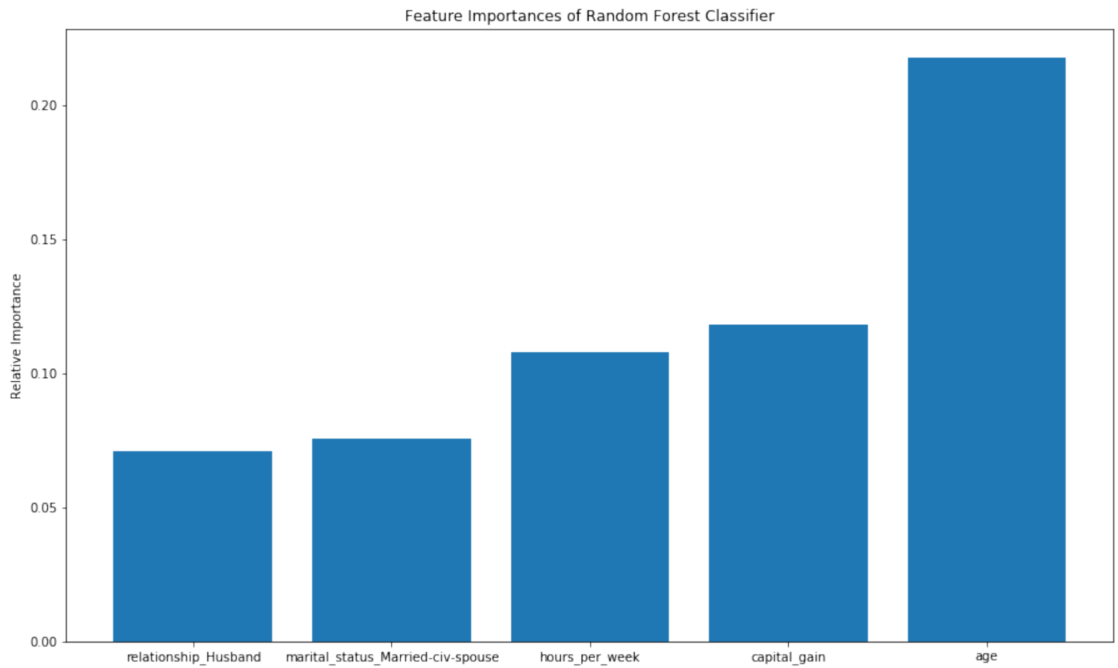
d) Calculate the AUC score (we did this in HW #4 many times).

```
In [14]: from sklearn.metrics import roc_auc_score
         roc_auc_score(y_test, rf_prob_of_one)

Out[14]: 0.861122378294O5014
```

e) Identify the top 5 features. Feel free to print a list OR to make a plot.

```
In [15]: import matplotlib.pyplot as plt
         features=X_train.columns
         importances=rf_model.feature_importances_
         #get the indices of the importances, ordered by the underlying value, take top 5 (np.
         indices = np.argsort(importances)[-5:]
         plt.figure(figsize=(15,9))
         plt.title('Feature Importances of Random Forest Classifier')
         plt.bar(range(len(indices)), importances[indices], align='center')
         plt.xticks(range(len(indices)), features[indices])
         plt.ylabel('Relative Importance')
         plt.show()
```



f) Using the model from part B, predict for the train data. Look at the classification report for the train data - is there overfitting for the RandomForest model happening?

```
In [16]: rf_predclass_on_train = rf_model.predict(X_train)
         rf_predprob_on_train = rf_model.predict_proba(X_train)
         rf_prob_of_one=rf_predprob_on_train[:,1]
         print('\nClassification Report\n')
         print(classification_report(rf_predclass_on_train, y_train))
         print('\nAUC Score\n')
         print(roc_auc_score(y_train, rf_prob_of_one))
```

```
Classification Report

             precision    recall  f1-score   support

          0       0.99      0.97      0.98     17497
          1       0.91      0.96      0.94      5295

avg / total       0.97      0.97      0.97     22792


AUC Score

0.995754724576
```

Clearly there is an overfitting problem given the drop of in F1-Score and AUC Score from Train to Test (0.97 to 0.84 and 0.995 to 0.86 respectively).

1.0.4 3. AdaBoost Classifier - GridSearch:

Start by creating a simple AdaBoostClassifier only using default parameters.
(Note: sklearn defaults to a max_depth of 1 for AdaBoost. Read more in the documentation)

a) Use the AdaBoostClassifier along with the GridSearchCV tool. Run the GridSearchCV using the following:
n_estimators: 100, 200, 300, 400
learning_rate: 0.2,0.4,0.6,0.8,1, 1.2

Note: Feel free to try out more parameters, the above is the bare minimum for this assignment.
Use 5 cross-fold and for scoring use "roc_auc" (this is the score that will be referenced when identifying the best parameters). This run took 8 minutes for your TA.

```
In [17]: from sklearn.ensemble import AdaBoostClassifier
         from sklearn.model_selection import GridSearchCV
         ada_clf = AdaBoostClassifier()
         param_grid={'n_estimators':[100,200,300,400,500], 'learning_rate':[0.2,0.4,0.6,0.8,1,
         grid_ada=GridSearchCV(ada_clf, param_grid, cv = 5, scoring='roc_auc', refit = True, n_
         grid_ada_model = grid_ada.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits


[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 175 out of 175 | elapsed:  3.6min finished
```

b) Use the best estimator from GridSearchCV to predict on test data. Use the .predict_proba() and the .predict() methods to get predicted probabilities as well as predicted classes.

```
In [18]: best_ada_model=grid_ada_model.best_estimator_
         best_ada_class = best_ada_model.predict(X_test)
         best_ada_prob = best_ada_model.predict_proba(X_test)
         best_ada_prob_of_one=best_ada_prob[:,1]
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

```
In [19]: print('\nConfusion Matrix\n')
         print(confusion_matrix(best_ada_class, y_test))
         print('\nClassification Report\n')
         print(classification_report(best_ada_class, y_test))
```

```
Confusion Matrix

[[6983  822]
 [ 486 1478]]

Classification Report

             precision    recall  f1-score   support

          0       0.93      0.89      0.91      7805
          1       0.64      0.75      0.69      1964

avg / total       0.88      0.87      0.87      9769
```

d) Calculate the AUC score

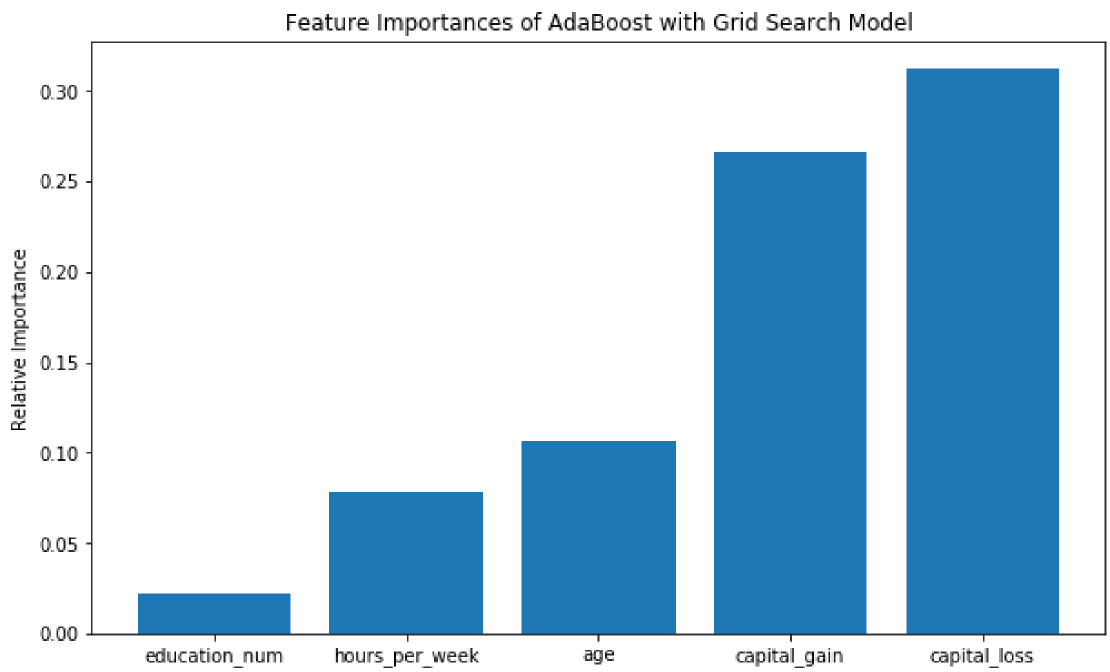
```
In [20]: print('\nAUC Score\n')
         print(roc_auc_score(y_test, best_ada_prob_of_one))
```

```
AUC Score

0.918731918015
```

e) Identify the top 5 features. Feel free to print a list OR to make a plot.

```
In [21]: features=X_train.columns
         ada_importances=best_ada_model.feature_importances_
         #get the indices of the importances, ordered by the underlying value, take top 5 (np.
         ada_indices = np.argsort(ada_importances)[-5:]
         plt.figure(figsize=(10,6))
         plt.title('Feature Importances of AdaBoost with Grid Search Model')
         plt.bar(range(len(ada_indices)), ada_importances[ada_indices], align='center')
         plt.xticks(range(len(ada_indices)), features[ada_indices])
         plt.ylabel('Relative Importance')
         plt.show()
```



f) Using the model from part (b), predict for the train data. Look at the classification report for the train data - is there overfitting for the best estimator?

```
In [22]: ada_class_on_train = best_ada_model.predict(X_train)
         ada_prob_on_train = best_ada_model.predict_proba(X_train)
         ada_prob_of_one_on_train=ada_prob_on_train[:,1]
         print('\nClassification Report\n')
         print(classification_report(ada_class_on_train, y_train))
         print('\nAUC Score\n')
         print(roc_auc_score(y_train, ada_prob_of_one_on_train))
```

```
Classification Report

             precision    recall  f1-score   support

          0       0.94      0.90      0.92     18079
          1       0.67      0.79      0.72      4713

avg / total       0.89      0.88      0.88     22792


AUC Score

0.93607582165
```

Well would you look at that! This model performs better on the training data, but only slightly. From Train to Test, the F-1 score goes from 0.88 to 0.87, and the AUC from 0.936 to 0.919. Minimal overfitting.

1.0.5 4. Gradient Boosting Classifier - GridSearch:

a) Use GradientBoostingClassifier along with the GridSearchCV tool. Run the GridSearchCV using the following hyperparameters:
n_estimators: 100,200, 300 & 400
learning_rate: choose 3 learning rates of your choice
max_depth: 1, 2 (you can try deeper, but remember part of the value of boosting stems from minimal complexity of trees)

Note: Feel free to try out more parameters, the above is the bare minimum for this assignment. Use 5 cross-fold and for scoring use "roc_auc" (this is the score that will be referenced when identifying the best parameters).

```
In [23]: from sklearn.model_selection import GridSearchCV
         from sklearn.ensemble import GradientBoostingClassifier
         gb_param_grid={'n_estimators':[100,200,300,400,500], 'learning_rate':[0.5,1, 1.5], 'ma
         gbc_clf = GradientBoostingClassifier()
         grid_gb=GridSearchCV(gbc_clf, gb_param_grid, cv = 5, scoring='roc_auc', refit = True,
         grid_gb_model = grid_gb.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits


[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    3.0s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:   54.9s
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  2.4min finished
```

b) Use the best estimator from GridSearchCV to predict on test data. Use the .predict_proba() and the .predict() methods to get predicted probabilities as well as predicted classes.

```
In [24]: best_gb_model=grid_gb_model.best_estimator_
         best_gb_pred_class = best_gb_model.predict(X_test)
         best_gb_pred_prob = best_gb_model.predict_proba(X_test)
         best_gb_prob_of_one = best_gb_pred_prob[:,1]
```

c) Calculate the confusion matrix and classification report (both are in sklearn.metrics).

```
In [25]: from sklearn.metrics import confusion_matrix, classification_report
         print('\nConfusion Matrix\n')
         print(confusion_matrix(best_gb_pred_class, y_test))
         print('\nClassification Report\n')
         print(classification_report(best_gb_pred_class, y_test))
```

```
Confusion Matrix

[[6974  802]
 [ 495 1498]]

Classification Report

             precision    recall  f1-score   support

          0       0.93      0.90      0.91      7776
          1       0.65      0.75      0.70      1993

avg / total       0.88      0.87      0.87      9769
```

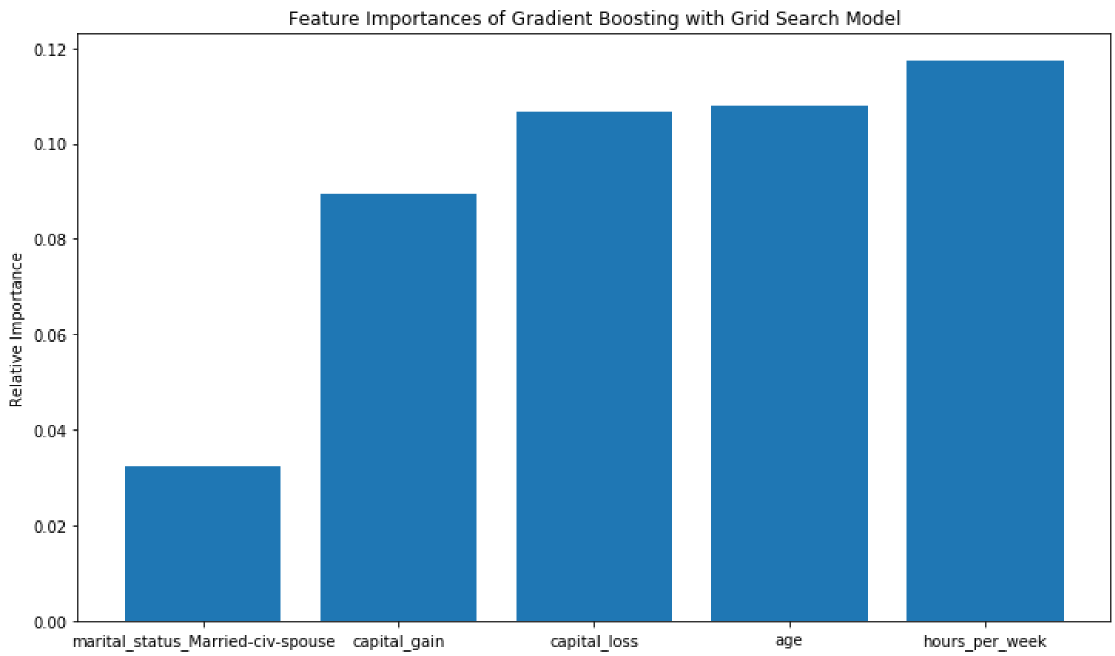
d) Calculate the AUC score

```
In [26]: from sklearn.metrics import roc_auc_score
         print(roc_auc_score(y_test, best_gb_prob_of_one))
```

```
0.920066448567
```

e) Identify the top 5 features. Feel free to print a list OR to make a plot.

```
In [27]: import matplotlib.pyplot as plt
         features=X_train.columns
         gb_importances=best_gb_model.feature_importances_
         #get the indices of the importances, ordered by the underlying value, take top 5 (np.
         gb_indices = np.argsort(gb_importances)[-5:]
         plt.figure(figsize=(12,7))
         plt.title('Feature Importances of Gradient Boosting with Grid Search Model')
         plt.bar(range(len(gb_indices)), gb_importances[gb_indices], align='center')
         plt.xticks(range(len(gb_indices)), features[gb_indices])
         plt.ylabel('Relative Importance')
         plt.show()
```



f) Using the model from part (b), predict for the train data. Look at the classification report for the train data - is there overfitting for the best estimator?

```
In [28]: gb_class_on_train = best_gb_model.predict(X_train)
         gb_prob_on_train = best_gb_model.predict_proba(X_train)
         gb_prob_of_on_on_train=gb_prob_on_train[:,1]
         print('\nClassification Report\n')
         print(classification_report(gb_class_on_train, y_train))
         print('\nAUC Score\n')
         print(roc_auc_score(y_train, gb_prob_of_on_on_train))
```

```
Classification Report

             precision    recall  f1-score   support

          0       0.95      0.91      0.93     18006
          1       0.71      0.82      0.76      4786

avg / total       0.90      0.89      0.89     22792


AUC Score

0.945620183858
```

The model performs better on the training data but only slightly so. From Train to Test the F1-Score falls from 0.89 to 0.87 and the AUC Score falls from 0.95 to 0.92. **As with the Adaboost, this is minimal overfitting.**

1.0.6 Moving into Conceptual Problems:

5) What does the alpha parameter represent in AdaBoost? Please refer to chapter 7 of the Hands-On ML book if you are struggling.

Alpha is a parameter in the AdaBoost algorithm, not the sci-kit learn function. It is the weight given to a particular predictor (a predictor can be thought of as an instance of a fit of a classifier model, i.e. a prediction model). It is determined using the learning rate and the weighted error rate of the *previous* predictor. It is also used to determine (1) the re-adjusted weights of misclassified observations in the *next* predictor and (2) the weights for the final prediction output.

6) In AdaBoost explain how the final predicted class is determined. Be sure to reference the alpha term in your explanation.

To make a final prediction, AdaBoost will look at the predictions for a given observation across all predictors, weighted by the alpha of that given predictor. The class that gets the majority of votes (after weighting provided by the alpha of that predictor) will be the output prediction of that observation for the entire AdaBoost model.

7) In Gradient Boosting, what is the role of the max_depth parameter? Why is it important to tune on this parameter?

The max_depth parameter **controls how deep the tree is (number of nodes)**. Limiting this depth **prevents overfitting** because if the depth is too big the model will learn the relations that work well for that particular sample but may not generalize. Gradient Boosting works by fitting a new model to the errors of a previous model, but if the previous model overfits then it means errors will be minimized thus defeating the purpose of using Gradient Boosting.

8) In Part (e) of Steps 2-4 you determined the top 5 predictors across each model. Do any predictors show up in the top 5 predictors for all three models? If so, comment on if this predictor makes sense given what you are attempting to predict. (Note: If you don't have any predictors showing up across all 3 predictors, explain one that shows up in 2 of them).

```
In [29]: print('Random Forest Important Features\n')
         print(features[indices])
         print('\nAdaBoost Important Features\n')
         print(features[ada_indices])
         print('\nGradient Boosting Important Features\n')
         print(features[gb_indices])
```

```
Random Forest Important Features

Index(['relationship_Husband', 'marital_status_Married-civ-spouse',
       'hours_per_week', 'capital_gain', 'age'],
      dtype='object')

AdaBoost Important Features

Index(['education_num', 'hours_per_week', 'age', 'capital_gain',
       'capital_loss'],
```

```
      dtype='object')

Gradient Boosting Important Features

Index(['marital_status_Married-civ-spouse', 'capital_gain', 'capital_loss',
       'age', 'hours_per_week'],
      dtype='object')
```

Hours per week, age, and capital gain are in all three models. This makes sense as each would tend to have a postive correlation with income: the more you work the more you make, the older you are the more your salary tends to increase, and the more capital gains income you have the more income you have in general.

9) From the models run in steps 2-4, which performs the best based on the Classification Report? Support your reasoning with evidence from your test data and be sure to share the optimal hyperparameters found from your grid search.

The AdaBoost and Gradient Boosting models performed slightly better than the Random Forest. But **there is little to distinguish between AdaBoost and Gradient Boosting in their Classification Reports**. The AUC Score is ever so slightly higher for Gradient Boosting. To choose between the two one would need to know more about the value of precision and recall for the various classes, as there there are slight differences here.

```
In [30]: print('\nAdaBoost Classification Report\n')
         print(classification_report(best_ada_class, y_test))
         print('\nGradient Boosting Classification Report\n')
         print(classification_report(best_gb_pred_class, y_test))
```

```
AdaBoost Classification Report

             precision    recall  f1-score   support

          0       0.93      0.89      0.91      7805
          1       0.64      0.75      0.69      1964

avg / total       0.88      0.87      0.87      9769


Gradient Boosting Classification Report

             precision    recall  f1-score   support

          0       0.93      0.90      0.91      7776
          1       0.65      0.75      0.70      1993

avg / total       0.88      0.87      0.87      9769
```

```
In [31]: print('\nAdaBoost Model Best Parameters\n')
         print(grid_ada_model.best_params_)
         print('\nGradient Boosting Model Best Parameters\n')
         print(grid_gb_model.best_params_)
```

```
AdaBoost Model Best Parameters

{'learning_rate': 1.5, 'n_estimators': 500, 'random_state': 1320}

Gradient Boosting Model Best Parameters

{'learning_rate': 0.5, 'max_depth': 2, 'n_estimators': 300, 'random_state': 11102018}
```

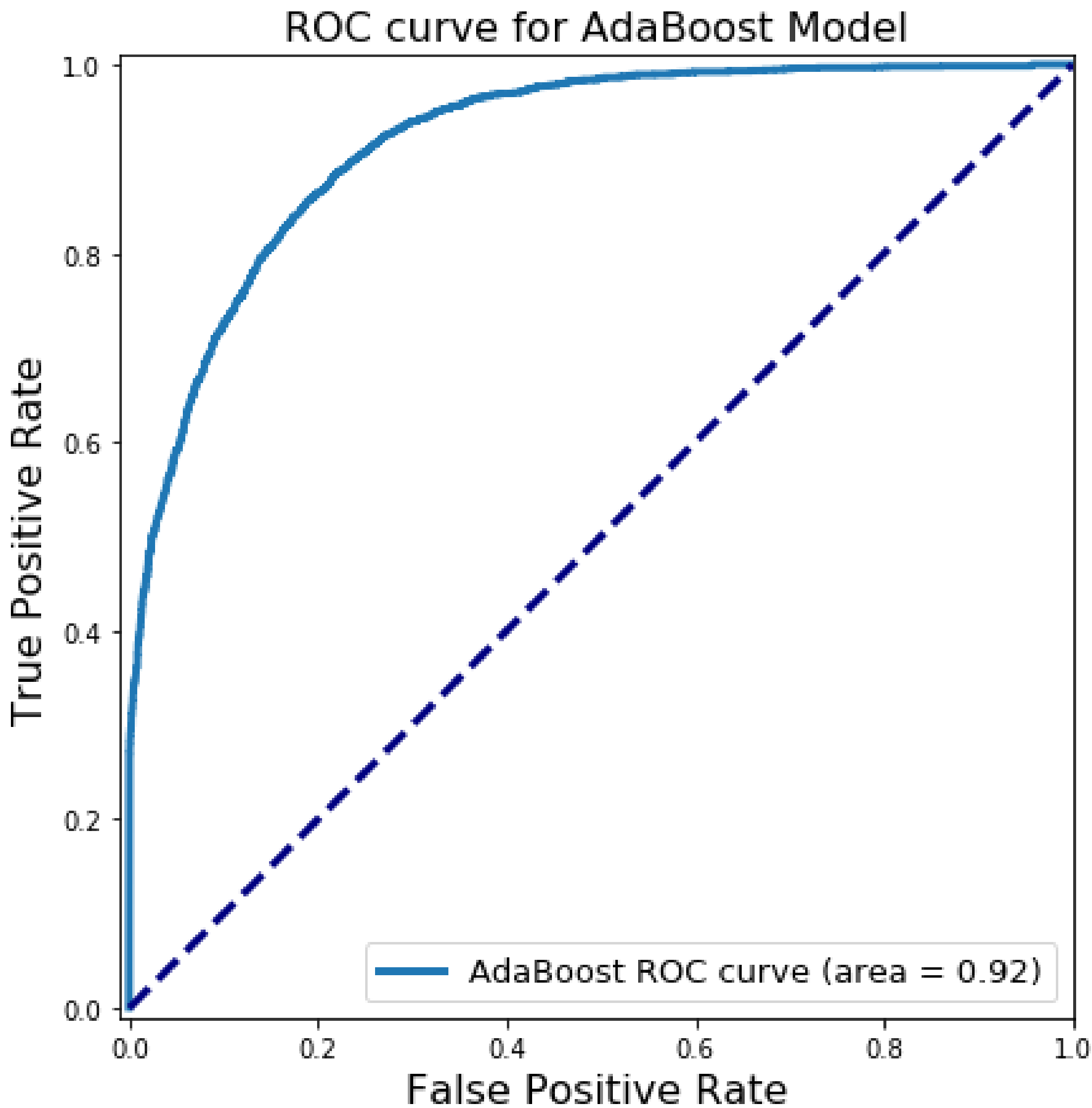
10) For your best performing model, plot out an ROC curve. Feel free to use sklearn, matplotlib or any other method in python.

We will deem the **AdaBoost** as our best performing model for purposes of this question.

```
In [32]: from sklearn.metrics import roc_curve, auc
         y_score_ada = best_ada_model.decision_function(X_test)
         fpr_ada, tpr_ada, _ = roc_curve(y_test, y_score_ada)
         roc_auc_ada = auc(fpr_ada, tpr_ada)
```

```
In [33]: plt.figure(figsize=(12,7))
         plt.xlim([-0.01, 1.00])
         plt.ylim([-0.01, 1.01])
         plt.plot(fpr_ada, tpr_ada, lw=3, label='AdaBoost ROC curve (area = {:0.2f})'.format(ro
         plt.xlabel('False Positive Rate', fontsize=16)
         plt.ylabel('True Positive Rate', fontsize=16)
         plt.title('ROC curve for AdaBoost Model', fontsize=16)
         plt.legend(loc='lower right', fontsize=13)
         plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
         plt.axes().set_aspect('equal')
         plt.show()
```

```
C:\Users\mjdun\Anaconda\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprec
  warnings.warn(message, mplDeprecation, stacklevel=1)
```


ROC curve for AdaBoost Model
True Positive Rate
False Positive Rate
AdaBoost ROC curve (area = 0.92)
0.0
0.2
0.4
0.6
0.8
1.0