ML HW6 Boosting Zoe Zhou

November 15, 2018

1 1. Data Processing

```
In [2]: import pandas as pd

In [153]: #a
          adult_df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/adu

In [154]: #b
          list_of_columns = ['age','workclass','fnlwgt','education','education-num','marital-s
          adult_df.columns = list_of_columns

In [155]: #c
          adult_df.shape

Out[155]: (32561, 15)

In [156]: #d
          adult_df=adult_df.drop(['fnlwgt'],axis=1)

In [159]: #e
          adult_df = adult_df.replace(to_replace = ['<=50K', '>50K'], value = [0, 1])

In [160]: adult_df.head()

Out[160]:    age         workclass  education  education-num      marital-status  \
          0   39         State-gov  Bachelors             13       Never-married
          1   50  Self-emp-not-inc  Bachelors             13  Married-civ-spouse
          2   38           Private    HS-grad              9            Divorced
          3   53           Private       11th              7  Married-civ-spouse
          4   28           Private  Bachelors             13  Married-civ-spouse

                    occupation   relationship   race     sex  capital-gain  \
          0       Adm-clerical  Not-in-family  White    Male          2174
          1    Exec-managerial        Husband  White    Male             0
          2  Handlers-cleaners  Not-in-family  White    Male             0
          3  Handlers-cleaners        Husband  Black    Male             0
          4     Prof-specialty           Wife  Black  Female             0
```

```
   capital-loss  hours-per-week native-country  salary
0             0              40  United-States       0
1             0              13  United-States       0
2             0              40  United-States       0
3             0              40  United-States       0
4             0              40           Cuba       0
```

In [161]:
```
#g
x_df=adult_df.iloc[:,0:13]
```

In [162]:
```
x_df.shape
```

Out[162]:
```
(32561, 13)
```

In [163]:
```
#h
y_df=adult_df.iloc[:,13]
```

In [164]:
```
y_df.shape
```

Out[164]:
```
(32561,)
```

In [13]:
```
#i
x_encoded = pd.get_dummies(x_df)
```

In [165]:
```
y_df
```

Out[165]:
```
0        0
1        0
2        0
3        0
4        0
5        0
6        0
7        1
8        1
9        1
10       1
11       1
12       0
13       0
14       1
15       0
16       0
17       0
18       0
19       1
20       1
21       0
22       0
```

```
23       0
24       0
25       1
26       0
27       1
28       0
29       0
        ..
32531    0
32532    1
32533    1
32534    0
32535    0
32536    1
32537    0
32538    1
32539    1
32540    0
32541    0
32542    0
32543    0
32544    0
32545    1
32546    0
32547    0
32548    0
32549    0
32550    0
32551    0
32552    0
32553    0
32554    1
32555    0
32556    0
32557    1
32558    0
32559    0
32560    1
Name: salary, Length: 32561, dtype: int64
```

In [166]:
```
x_encoded.shape
```

Out[166]:
```
(32561, 107)
```

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [168]:
```
#j
x_train, x_test = train_test_split(x_encoded,test_size = .3, random_state = 43)
y_train, y_test = train_test_split(y_df, test_size = .3, random_state =43)
```

In [169]: x_train.head()

Out[169]:
```
       age  education-num  capital-gain  capital-loss  hours-per-week  \
20717   24             13             0             0              35
11366   37              9             0             0              40
28940   46             13             0          1848              45
28302   50              9             0             0              40
10929   46              9             0             0              40

       workclass_?  workclass_Federal-gov  workclass_Local-gov  \
20717            0                      0                    1
11366            0                      0                    0
28940            0                      0                    0
28302            0                      0                    0
10929            0                      0                    0

       workclass_Never-worked  workclass_Private            ...                \
20717                       0                  0            ...
11366                       0                  0            ...
28940                       0                  1            ...
28302                       0                  1            ...
10929                       0                  1            ...

       native-country_Portugal  native-country_Puerto-Rico  \
20717                        0                           0
11366                        0                           0
28940                        0                           0
28302                        0                           0
10929                        0                           0

       native-country_Scotland  native-country_South  native-country_Taiwan  \
20717                        0                     0                      0
11366                        0                     0                      0
28940                        0                     0                      0
28302                        0                     0                      0
10929                        0                     0                      0

       native-country_Thailand  native-country_Trinadad&Tobago  \
20717                        0                               0
11366                        0                               0
28940                        0                               0
28302                        0                               0
10929                        0                               0

       native-country_United-States  native-country_Vietnam  \
20717                             1                       0
11366                             1                       0
28940                             1                       0
```

```
28302                            1                       0
10929                            1                       0

       native-country_Yugoslavia
20717                          0
11366                          0
28940                          0
28302                          0
10929                          0

[5 rows x 107 columns]
```

2 2. Random Forest Classifier - Base Model:

In [171]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
```

In [172]:
```
#create Gaussian classifier
rfbase_clf=RandomForestClassifier(random_state=43,n_estimators=1000)
#train the model using the training set
rfbase_clf.fit(x_train,y_train)
```

Out[172]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=None,
            oob_score=False, random_state=43, verbose=0, warm_start=False)
```

In [174]:
```
#limit to probability for class = 1
y_test_p=rfbase_clf.predict(x_test)
```

In [175]:
```
base_probs=rfbase_clf.predict_proba(x_test)[:,1]
base_probs_train=rfbase_clf.predict_proba(x_train)[:,1]
```

In [176]:
```
#confusion matrix
confusion_matrix(y_test,y_test_p)
```

Out[176]:
```
array([[6833,  601],
       [ 854, 1481]])
```

In [177]:
```
#classification report
print(classification_report(y_test,y_test_p))
```

```
    precision    recall  f1-score   support

 0       0.89      0.92      0.90      7434
```

```
           1       0.71      0.63      0.67      2335

   micro avg       0.85      0.85      0.85      9769
   macro avg       0.80      0.78      0.79      9769
weighted avg       0.85      0.85      0.85      9769
```

In [179]:
```
roc_auc_score(y_train,base_probs_train)
```

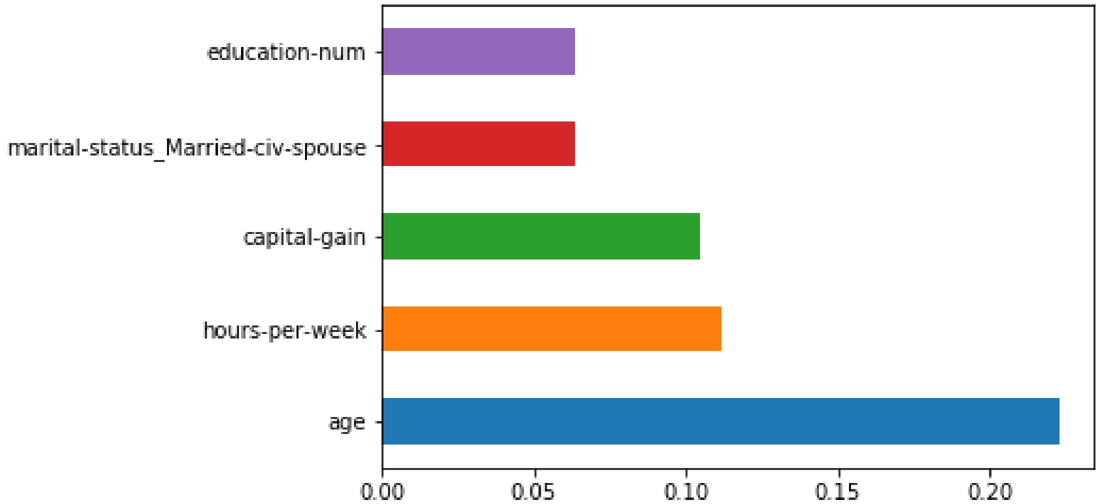
Out[179]: 0.998081988876355

In [180]:
```
roc_auc_score(y_test,base_probs)
```

Out[180]: 0.8946818512546383

In [181]:
```
import matplotlib.pyplot as plt
%matplotlib inline

feature_importances = pd.Series(rfbase_clf.feature_importances_, index = x_test.colu
feature_importances.nlargest(5).plot(kind='barh')
```

Out[181]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ee5ae80>



In [182]:
```
y_rf_train=rfbase_clf.predict(x_train)
```

In [183]:
```
print(classification_report(y_train,y_rf_train))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     17286
```

```
           1       0.97      0.95      0.96      5506

   micro avg       0.98      0.98      0.98     22792
   macro avg       0.98      0.97      0.97     22792
weighted avg       0.98      0.98      0.98     22792
```

There is an overfitting in Random Forest since the performance for the training classification report has weighted average of 0.98 for both precision, recall and f1-score.

3 3. AdaBoost Classifier - GridSearch:

In [185]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
```

In [203]:
```
param_grid = {'n_estimators' : [100,200,300,400] ,
              'learning_rate': [0.2,0.4,0.6,0.8,1,1.2]}
```

In [204]:
```
ada_obj = AdaBoostClassifier()

ada_Grid = GridSearchCV(ada_obj, param_grid, cv = 5, refit = True, verbose = 0, scor

ada_Grid.fit(x_train, y_train)
```

Out[204]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [100, 200, 300, 400], 'learning_rate': [0.2, 0.4,
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [205]:
```
ada_Grid_Best = ada_Grid.best_estimator_
```

In [206]:
```
ada_Grid_Para = ada_Grid.best_params_
```

In [207]:
```
print(ada_Grid_Best)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.2, n_estimators=400, random_state=None)
```

In [208]:
```
y_ada_test= ada_Grid_Best.predict(x_test)
```

In [210]:
```
ada_probs=ada_Grid_Best.predict_proba(x_test)[:,1]
```

In [211]:
```
ada_probs_train=ada_Grid_Best.predict_proba(x_train)[:,1]
```

In [212]: #confusion matrix
```
confusion_matrix(y_test,y_ada_test)
```

Out[212]:
```
array([[7026,  408],
       [ 834, 1501]])
```

In [213]: #classification report
```
print(classification_report(y_test,y_ada_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      7434
           1       0.79      0.64      0.71      2335

   micro avg       0.87      0.87      0.87      9769
   macro avg       0.84      0.79      0.81      9769
weighted avg       0.87      0.87      0.87      9769
```

In [214]:
```
roc_auc_score(y_train,ada_probs_train)
```

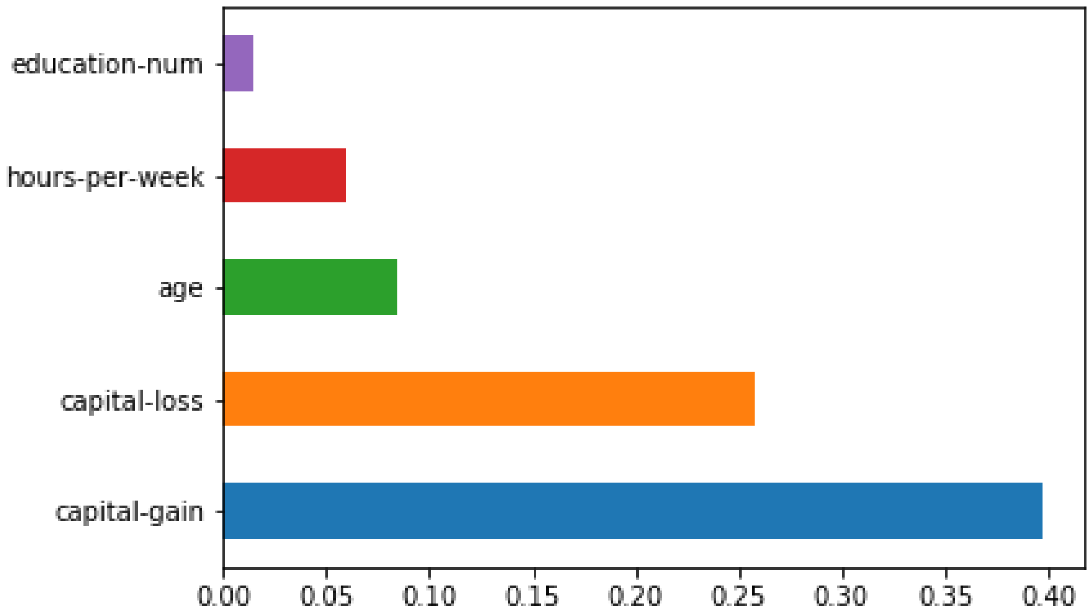
Out[214]: 0.9304553699877604

In [215]: #auc score
```
roc_auc_score(y_test,ada_probs)
```

Out[215]: 0.927774436454072

In [216]:
```
import matplotlib.pyplot as plt
%matplotlib inline

feature_importances = pd.Series(ada_Grid_Best.feature_importances_, index = x_test.c
feature_importances.nlargest(5).plot(kind='barh')
```

Out[216]: <matplotlib.axes._subplots.AxesSubplot at 0x11a386dd8>




```
In [217]: y_ada_train= ada_Grid_Best.predict(x_train)
          #classification report
          print(classification_report(y_train,y_ada_train))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     17286
           1       0.78      0.65      0.71      5506

   micro avg       0.87      0.87      0.87     22792
   macro avg       0.84      0.79      0.81     22792
weighted avg       0.87      0.87      0.87     22792
```

There is no overfitting for the best estimator because the classification reports for training and testing data shows similar results.

4 4. Gradient Boosting Classifier - GridSearch:

```
In [218]: from sklearn.ensemble import GradientBoostingClassifier
```

```
In [219]: param_grid_gradient = {'n_estimators' : [100,200,300,400] ,
                          'learning_rate': [0.4,0.6,0.8],
                                'max_depth' :[1,2]
                         }
```

In [220]:
```
gbrt_obj = GradientBoostingClassifier()

gbrt_Grid = GridSearchCV(gbrt_obj, param_grid_gradient, cv = 5, refit = True, verbose

gbrt_Grid.fit(x_train, y_train)
```

Out[220]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001, valida
              verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [100, 200, 300, 400], 'learning_rate': [0.4, 0.6,
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [221]:
```
gbrt_Grid_Best = gbrt_Grid.best_estimator_
gbrt_Grid_Best
```

Out[221]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.4, loss='deviance', max_depth=2,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=300,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

In [222]:
```
gbrt_Grid_Para = gbrt_Grid.best_params_
```

In [223]:
```
y_gbrt_test= gbrt_Grid_Best.predict(x_test)
```

In [224]:
```
gbrt_probs_train=gbrt_Grid_Best.predict_proba(x_train)[:,1]
```

In [225]:
```
gbrt_probs_train
```

Out[225]:
```
array([0.01599939, 0.00597978, 0.98409291, ..., 0.52627481, 0.00191023,
       0.99306095])
```

In [226]:
```
y_gbrt_test
```

Out[226]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [227]:
```
gbrt_probs=gbrt_Grid_Best.predict_proba(x_test)[:,1]
```

In [228]:
```
#confusion matrix
confusion_matrix(y_test,y_gbrt_test)
```

Out[228]:
```
array([[7023,  411],
       [ 795, 1540]])
```

In [229]:
```
#classification report
print(classification_report(y_test,y_gbrt_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      7434
           1       0.79      0.66      0.72      2335

   micro avg       0.88      0.88      0.88      9769
   macro avg       0.84      0.80      0.82      9769
weighted avg       0.87      0.88      0.87      9769
```

In [230]:
```
roc_auc_score(y_train,gbrt_probs_train)
```

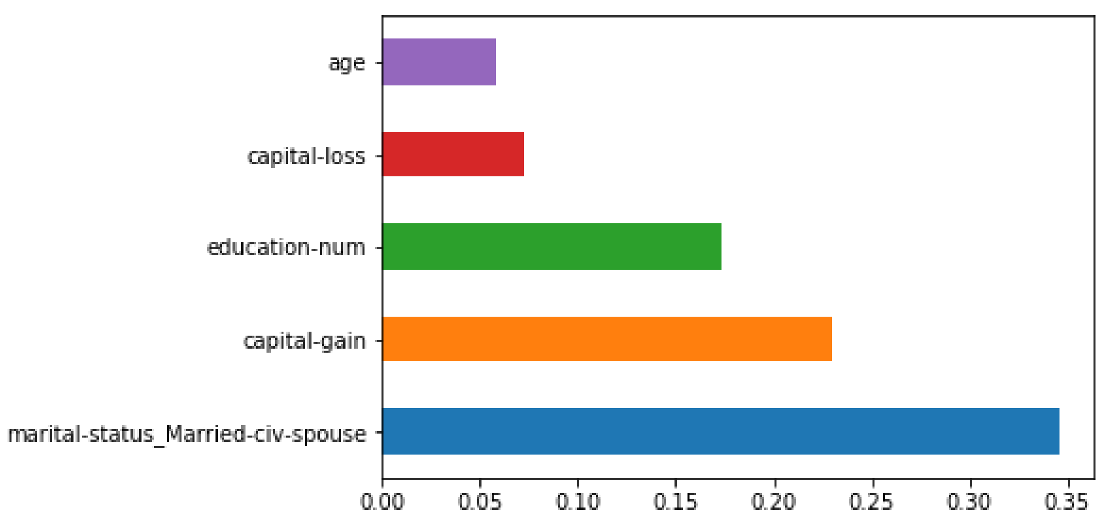
Out[230]:
```
0.9401653604018025
```

In [231]:
```
#auc score
roc_auc_score(y_test,gbrt_probs)
```

Out[231]:
```
0.9298749480798623
```

In [232]:
```
import matplotlib.pyplot as plt
%matplotlib inline

feature_importances = pd.Series(gbrt_Grid_Best.feature_importances_, index = x_test.
feature_importances.nlargest(5).plot(kind='barh')
```

Out[232]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x118b29390>
```



```
In [233]: y_gbrt_train= gbrt_Grid_Best.predict(x_train)
          #classification report
          print(classification_report(y_train,y_gbrt_train))
```

```
              precision    recall  f1-score   support

           0       0.90      0.95      0.92     17286
           1       0.81      0.67      0.73      5506

   micro avg       0.88      0.88      0.88     22792
   macro avg       0.86      0.81      0.83     22792
weighted avg       0.88      0.88      0.88     22792
```

There is no overfitting for the best estimator because the classification reports for training and testing data shows similar results.

5 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. Answer: Alpha parameter represent the predictor's weight. The more accurate the predictor is, the higher the weight will be.

6) In AdaBoost explain how the final predicted class is determined. Be sure to reference the alpha term in your explanation. Answer: AdaBoost simply computes the predictions of all the predictions and weighs them using the predictor weights alpha. The predicted class is the one that receives the majority of weighted votes.

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

Answer: The role of the maximum depth is to limit the number of nodes in the tree. It will control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.

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).

Answer: Education number shows all three times. In real life, higher the education, more likely a person is gonna get >50k salary. Those two are correlated.

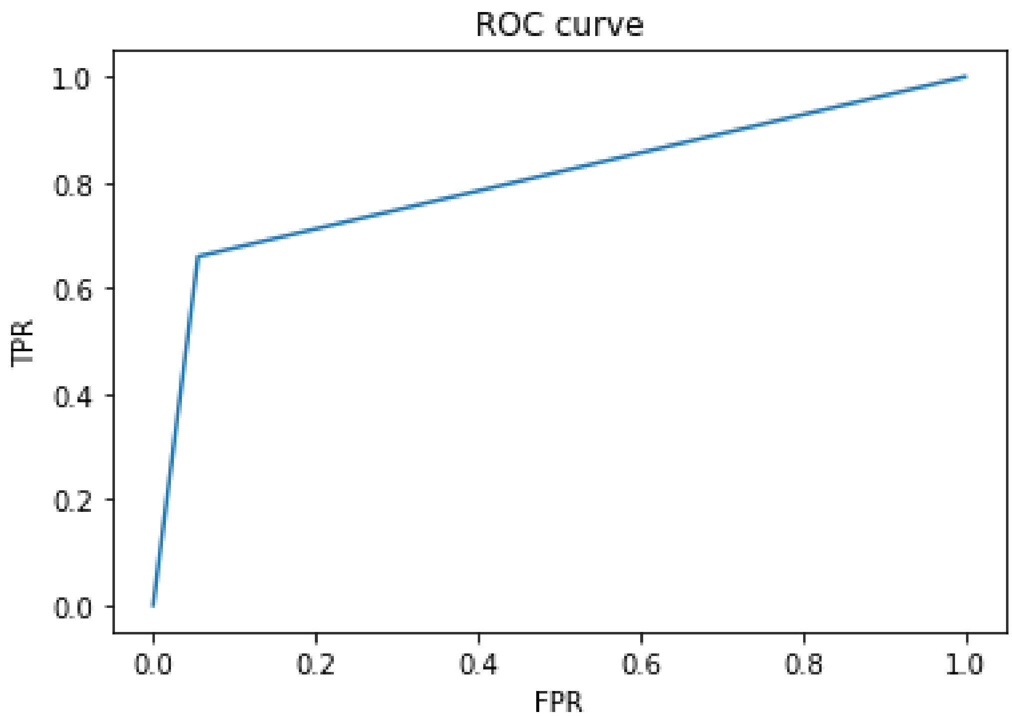
9) From the models run in steps 2-4, which performs the best based on the Classification Report? Support your reasoning with evidence from your test data and be sure to share the optimal hyperparameters found from your grid search. Answer: Gradient Boosting Classifier - GridSearch is the best model in this case. Base model of the Random Forest overfit.

The performance between Ada and Gradient on the training set are pretty much the same. However, Gradient does produce a slightly better test result.

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

```
In [245]: from sklearn.metrics import roc_curve
          import matplotlib.pyplot as plt


        ---------------------------------------------------------------------------

        ModuleNotFoundError                       Traceback (most recent call last)

        <ipython-input-245-9d6534b60aa5> in <module>()
          2 import matplotlib.pyplot as plt
          3 from sklearn import metrics
    ----> 4 from ggplot import *
          5 #fpr, tpr, _ = roc_curve(y_test, gbrt_probs)


        ModuleNotFoundError: No module named 'ggplot'
```

```
In [247]: fpr, tpr, thresholds = roc_curve(y_test,y_gbrt_test)

          plt.clf()
          plt.plot(fpr, tpr)
          plt.xlabel('FPR')
          plt.ylabel('TPR')
          plt.title('ROC curve')
          plt.show()
```


ROC curve
TPR
FPR
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0