Bagging

Data Set: bankloan.csv

Libraries

```
In [1]: # Jesus is my Saviour!!

In [2]: import os

In [3]: os.getcwd()
Out[3]: 'C:\\Users\\Dr Vinod\\Desktop\\WD_python'

In [4]: os.chdir("C:/Users/Dr Vinod/Desktop/WD_python")

In [5]: import pandas as pd

In [6]: import matplotlib.pyplot as plt

In [7]: import pylab as pl

In [8]: import sklearn

In [9]: from sklearn import tree

In [10]: from sklearn import metrics
```

Libraries

```
In [11]: from sklearn.tree import DecisionTreeClassifier

In [12]: from sklearn.model_selection import train_test_split

In [13]: from sklearn.metrics import confusion_matrix

In [14]: from sklearn.metrics import roc_curve, auc, roc_auc_score

In [15]: from sklearn.metrics import accuracy_score

In [16]: from sklearn.metrics import classification_report
```

Data

```
In [17]: bankloan = pd.read_csv("C:/Users/Dr Vinod/Desktop/DataSets1/bankloan.csv")

In [18]: bankloan = pd.DataFrame(bankloan)

In [19]: bankloan.shape
Out[19]: (700, 9)

In [20]: bankloan.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 9 columns):
age         700 non-null int64
ed          700 non-null int64
employ      700 non-null int64
address     700 non-null int64
income      700 non-null int64
debtinc     700 non-null float64
creddebt    700 non-null float64
othdebt     700 non-null float64
default     700 non-null int64
dtypes: float64(3), int64(6)
memory usage: 49.3 KB
```

Select useful variables

```
In [21]: bl = bankloan.ix[:, (0,4,6,7,8)]
__main__:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated

In [22]: # age, income, creddebt, othdebt, default

In [23]: bl.shape # 700 by 5
Out[23]: (700, 5)

In [24]: bl.info() # age, income, default as INTEGER; creddebt & othdebt as float
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 5 columns):
age         700 non-null int64
income      700 non-null int64
creddebt    700 non-null float64
othdebt     700 non-null float64
default     700 non-null int64
dtypes: float64(2), int64(3)
memory usage: 27.4 KB
```

X as set of predictors

```
In [25]: X = bl.ix[:, (0,1,2,3)] # age, income, creddebt, othdebt
__main__:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated

In [26]: X.info() # age, income as INTEGER; creddebt & othdebt as float
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 4 columns):
age         700 non-null int64
income      700 non-null int64
creddebt    700 non-null float64
othdebt     700 non-null float64
dtypes: float64(2), int64(2)
memory usage: 22.0 KB
```

y as response variable

```
In [27]: y = bl.ix[:, 4] # default as INTEGER
__main__:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated

In [28]: y.head(4) # 0,1,2 as 1, 0, 0, 0
Out[28]:
0    1
1    0
2    0
3    0
Name: default, dtype: int64
```

Partitioning

```
In [29]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,
random_state = 123)

In [30]: len(X_train) # 420
Out[30]: 420

In [31]: len(y_train) # 420
Out[31]: 420

In [32]: len(X_test) # 280
Out[32]: 280

In [33]: len(y_test) # 280
Out[33]: 280
```

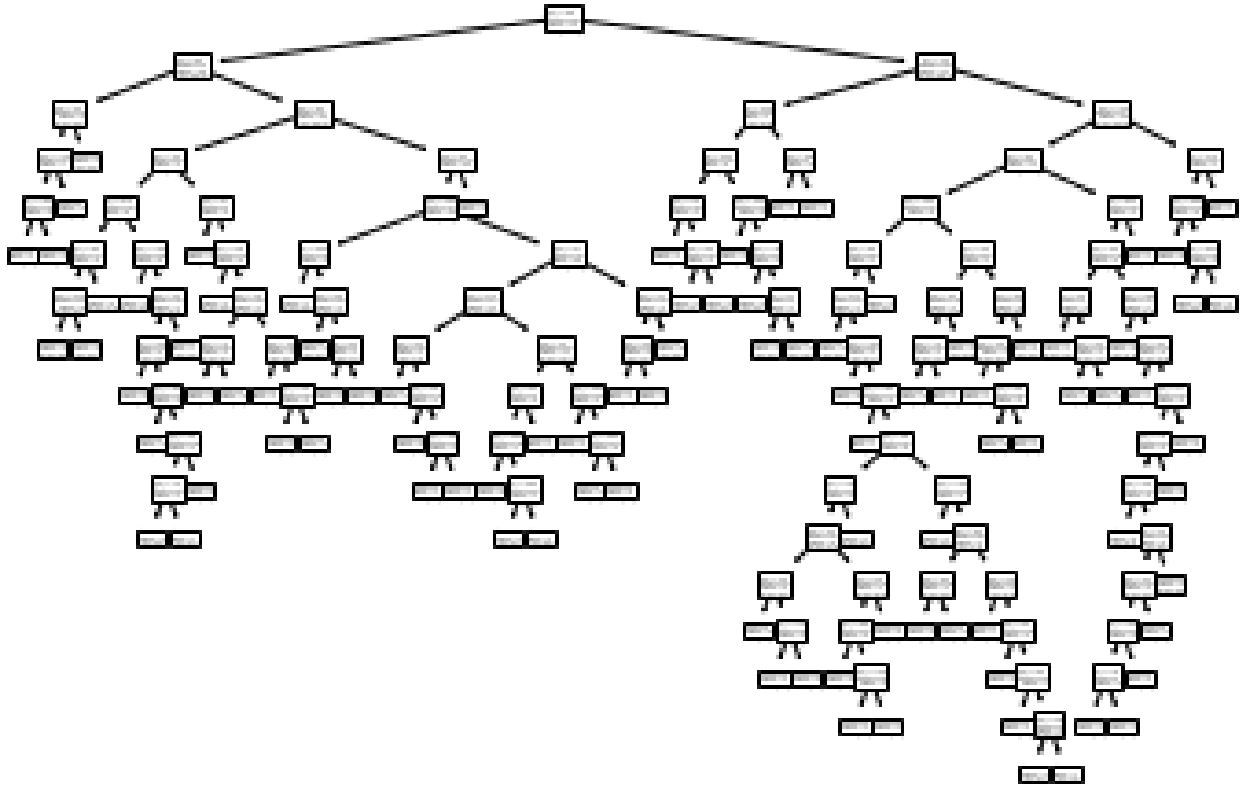
Build base estimator- tree

```
In [34]: clf = tree.DecisionTreeClassifier()

In [35]: clf
Out[35]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       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, presort=False,
                       random_state=None, splitter='best')

In [36]: clfFit = clf.fit(X_train, y_train)

In [37]: clfFit
Out[37]:
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       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, presort=False,
                       random_state=None, splitter='best')
```

Plot Tree

```
In [38]: #_____________________plot tree

In [39]: tree.plot_tree(clfFit)
Out[39]:
[Text(151.7750822368421, 211.04470588235293, 'X[2] <= 0.88\nentropy = 0.384\nsamples
= 420\nvalue = [311, 109]'),
 Text(50.522861842105264, 198.25411764705882, 'X[1] <= 15.5\nentropy = 0.26\nsamples
= 208\nvalue = [176, 32]'),
 Text(17.621052631578948, 185.4635294117647, 'X[0] <= 29.5\nentropy = 0.444\nsamples
```

Tree



Bagging-300 trees

```
In [40]: #_________________-Bagging

In [41]: from sklearn.ensemble import BaggingClassifier

In [42]: #base estimator is clf (tree build before)

In [43]: # Build BaggingClassifier 'bc'

In [44]: bc = BaggingClassifier(base_estimator= clf, n_estimators= 300,
oob_score=True, n_jobs=-1)
```

Fit bagging

```
In [45]: # Fit 'bc' to the training set

In [46]: bc.fit(X_train, y_train)


Out[46]:
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        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
,
                                                        presort=False,
                                                        random_state=None,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=300, n_jobs=-1, oob_score=True,
                  random_state=None, verbose=0, warm_start=False)
```

Predictions

```
In [47]: y_predB = bc.predict(X_test)

In [48]:

In [49]: y_predB
Out[49]:
array([0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

Classification Matrix & Report

```
In [50]: pd.crosstab(y_test, y_predB, margins = True,
    ...:                 rownames = ["Actual"], colnames = ["Predict"])
Out[50]:
Predict    0   1  All
Actual
0        171  35  206
1         49  25   74
All      220  60  280

In [51]: print(classification_report(y_test, y_predB))
              precision    recall  f1-score   support

           0       0.78      0.83      0.80       206
           1       0.42      0.34      0.37        74

    accuracy                           0.70       280
   macro avg       0.60      0.58      0.59       280
weighted avg       0.68      0.70      0.69       280
```

Probabilities

```
In [52]: predPB = bc.predict_proba(X_test)[:,1]

In [53]: predPB
Out[53]:
array([0.04666667, 0.14333333, 0.62666667, 0.05333333, 0.39666667,
       0.81333333, 0.55333333, 0.01666667, 0.01666667, 0.91333333,
       0.15666667, 0.33333333, 0.11      , 0.45333333, 0.08666667,
       0.72666667, 0.01333333, 0.44666667, 0.11666667, 0.07333333,
       0.63      , 0.00333333, 0.48      , 0.33666667, 0.01666667,
       0.58333333, 0.00666667, 0.55333333, 0.82      , 0.24666667,
```

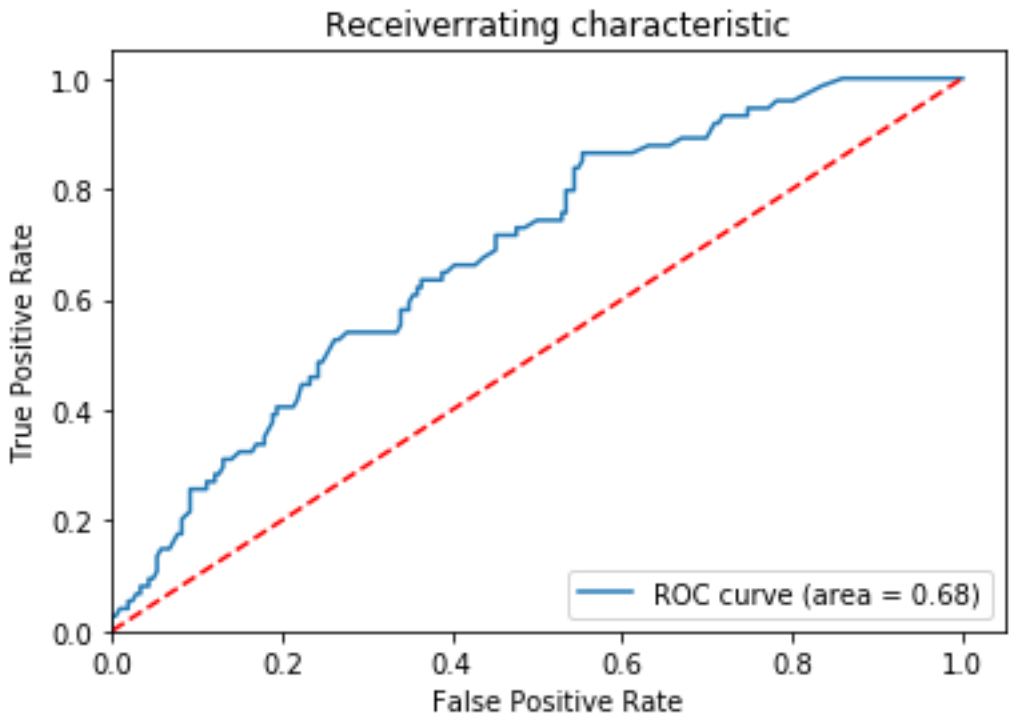
ROC Curve

```
In [54]: false_positive_rateB, true_positive_rateB, thresholdsB =
sklearn.metrics.roc_curve(y_test, predPB)

In [55]: roc_aucB = auc(false_positive_rateB, true_positive_rateB)

In [56]: roc_aucB
Out[56]: 0.6834164261348727
```

Let’s plot

```
In [57]: %matplotlib inline
    ...: plt.figure()
    ...: plt.plot(false_positive_rateB, true_positive_rateB, label='ROC curve (area
= %0.2f)' % roc_aucB)
    ...: plt.plot([0, 1], [0, 1], 'r--') # k for black, r for red, b for blue, g for
green
    ...: plt.xlim([0.0, 1.05])
    ...: plt.ylim([0.0, 1.05])
    ...: plt.xlabel('False Positive Rate')
    ...: plt.ylabel('True Positive Rate')
    ...: plt.title('Receiverrating characteristic')
    ...: plt.legend(loc="lower right")
    ...: plt.show()
```

ROC Curve-Bagging



Random Forest

Fit random forest

```
In [58]: from sklearn.ensemble import RandomForestClassifier

In [59]: # Create the model with 100 trees

In [60]: rf = RandomForestClassifier(n_estimators=100,
    ...:                             bootstrap = True,
    ...:                             max_features = 'sqrt')
    ...:

In [61]: rfFit = rf.fit(X_train, y_train)
```

Predictions

```
In [62]: # Actual class predictions

In [63]: y_predRF = rfFit.predict(X_test)

In [64]: len(y_predRF) #280
Out[64]: 280

In [65]: y_predRF # 0 and 1
Out[65]:
array([0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

Result & Report

```
In [66]: # Cross Tab

In [67]: pd.crosstab(y_test, y_predRF, margins = True,
    ...:                 rownames = ["Actual"], colnames = ["Predict"])
    ...:
Out[67]:
Predict    0   1  All
Actual
0        176  30  206
1         50  24   74
All      226  54  280

In [68]: print(classification_report(y_test, y_predRF))
              precision    recall  f1-score   support

           0       0.78      0.85      0.81       206
           1       0.44      0.32      0.38        74

    accuracy                           0.71       280
   macro avg       0.61      0.59      0.59       280
weighted avg       0.69      0.71      0.70       280
```

Probabilities

```
In [69]: # Probabilities for each class

In [70]: predP_RF = rfFit.predict_proba(X_test)[:, 1]

In [71]: predP_RF # actual numbers
Out[71]:
array([0.04, 0.14, 0.53, 0.15, 0.37, 0.69, 0.51, 0.03, 0.  , 0.8 , 0.19,
       0.38, 0.13, 0.55, 0.15, 0.7 , 0.03, 0.46, 0.21, 0.17, 0.58, 0.  ,
       0.33, 0.29, 0.06, 0.59, 0.01, 0.43, 0.77, 0.29, 0.29, 0.27, 0.16,
       0.01, 0.11, 0.02, 0.05, 0.38, 0.11, 0.03, 0.01, 0.  , 0.  , 0.02,
```

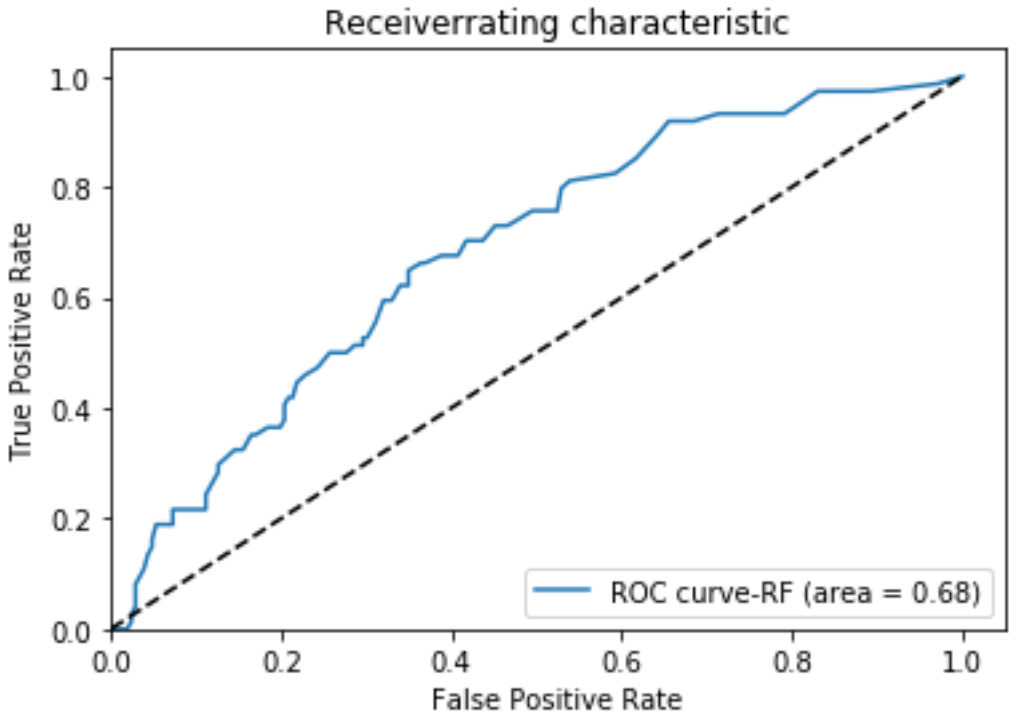
ROC Curve

```
In [72]: # Find fpr, tpr, threshold

In [73]: false_positive_rateRF, true_positive_rateRF, thresholdsRF =
sklearn.metrics.roc_curve(y_test, predP_RF)

In [74]: roc_aucRF = auc(false_positive_rateRF, true_positive_rateRF)

In [75]: roc_aucRF
Out[75]: 0.684203621096825
```

Let's Plot

```
In [76]: %matplotlib inline
    ...: plt.figure()
    ...: plt.plot(false_positive_rateRF, true_positive_rateRF, label='ROC curve-RF
(area = %0.2f)' % roc_aucRF)
    ...: plt.plot([0, 1], [0, 1], 'k--') # k for black, r for red, b for blue, g for
green
    ...: plt.xlim([0.0, 1.05])
    ...: plt.ylim([0.0, 1.05])
    ...: plt.xlabel('False Positive Rate')
    ...: plt.ylabel('True Positive Rate')
    ...: plt.title('Receiverrating characteristic')
    ...: plt.legend(loc="lower right")
    ...: plt.show()
```

Plot


Receiverrating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve-RF (area = 0.68)

Importance of variables

```
In [77]: import pandas as pd

In [78]: # Extract feature importances

In [79]: fi = pd.DataFrame({'feature': list(X_train.columns),
    ...:                    'importance': rfFit.feature_importances_}).\
    ...:                     sort_values('importance', ascending = False)

In [80]: # Display

In [81]: fi.head() # creddebt=0.31, othdebt=0.25, income=0.25, age=0.19
Out[81]:
    feature  importance
2  creddebt    0.307404
1    income    0.256796
3   othdebt    0.247423
0       age    0.188377
```

Adaptive Boosting

Fit adaB

```
In [82]: from sklearn.ensemble import AdaBoostClassifier

In [83]: from sklearn.tree import DecisionTreeClassifier

In [84]: ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=200)

In [85]: adaFit = ada.fit(X_train, y_train)

In [86]: adaFit
Out[86]:
AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(class_weight=None,
                                                         criterion='gini',
                                                         max_depth=1,
                                                         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,
                                                         presort=False,
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1.0, n_estimators=200, random_state=None)
```

Predict

```
In [87]: y_predADA = ada.predict(X_test)

In [88]: len(y_predADA) #280
Out[88]: 280

In [89]: y_predADA # 0 and 1
Out[89]:
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

Results and Report

```
In [90]: # Cross Tab

In [91]: pd.crosstab(y_test, y_predADA, margins = True,
    ...:                 rownames = ["Actual"], colnames = ["Predict"])
Out[91]:
Predict    0   1  All
Actual
0        182  24  206
1         47  27   74
All      229  51  280

In [92]: print(classification_report(y_test, y_predADA))
              precision    recall  f1-score   support

           0       0.79      0.88      0.84       206
           1       0.53      0.36      0.43        74

    accuracy                           0.75       280
   macro avg       0.66      0.62      0.63       280
weighted avg       0.72      0.75      0.73       280
```

Probabilities

```
In [93]: predP_ADA = adaFit.predict_proba(X_test)[:, 1]

In [94]: predP_ADA # actual numbers
Out[94]:
array([0.4913498 , 0.46867653, 0.49980017, 0.4912161 , 0.49421071,
       0.5034588 , 0.49910492, 0.48245949, 0.49016297, 0.51228893,
       0.5035122 , 0.50035409, 0.49674835, 0.49847706, 0.49553244,
       0.50097243, 0.49580815, 0.49887398, 0.4927394 , 0.47162318,
       0.49954554, 0.48957278, 0.49870869, 0.49651279, 0.49037143,
```

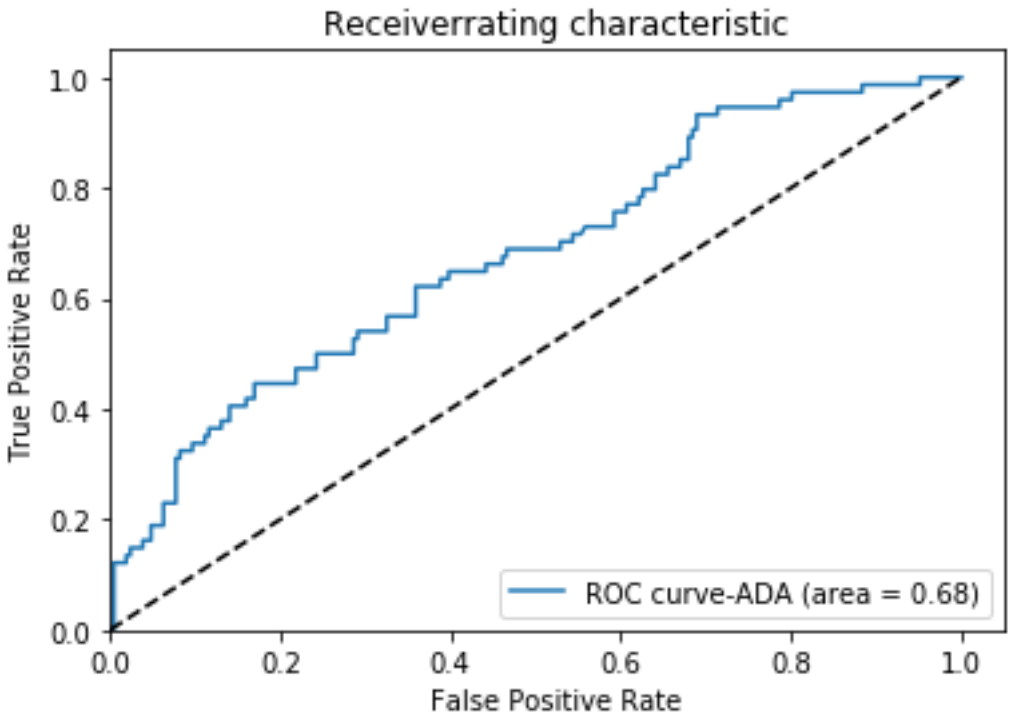
ROC Curve

```
In [95]: false_positive_rateADA, true_positive_rateADA, thresholdsADA =
sklearn.metrics.roc_curve(y_test, predP_ADA)

In [96]: roc_aucADA = auc(false_positive_rateADA, true_positive_rateADA)

In [97]: roc_aucADA
Out[97]: 0.6791852532143794
```

Let's Plot

```
In [98]: %matplotlib inline
    ...: plt.figure()
    ...: plt.plot(false_positive_rateADA, true_positive_rateADA, label='ROC curve-
ADA (area = %0.2f)' % roc_aucADA)
    ...: plt.plot([0, 1], [0, 1], 'k--') # k for black, r for red, b for blue, g for
green
    ...: plt.xlim([0.0, 1.05])
    ...: plt.ylim([0.0, 1.05])
    ...: plt.xlabel('False Positive Rate')
    ...: plt.ylabel('True Positive Rate')
    ...: plt.title('Receiverrating characteristic')
    ...: plt.legend(loc="lower right")
    ...: plt.show()
    ...:
```

ROC Curve


Receiverrating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve-ADA (area = 0.68)


Happy
Learning


Happy
Learning