Credit Risk Prediction

Jonggoo Kang

DePaul University

MS in Predictive Analytics

Abstract

The purpose of using ensemble-based classifiers is to integrate multiple models and establish a predictive model. Using ensemble methods is superior to using individual models by the following reasons - it averages out biases, reduces variance, is less likely to overfit, and provide performance boost from using ensemble methods.

In this report, we will address how credit risk analysis can be performed using machine learning algorithms to evaluate loan applications. Additionally, various classification techniques will be performed including ensemble-based classifiers to establish the model with the better performance. The following techniques will be used and discussed: Decision Tree, Naïve Bayes, Support Vector Machine (both linear and radial), Random Forest, Gradient Boosting, and XGBoost. Then the next step follows by performing cross validation and tuning hyper parameters in each of models to optimize and develop the models. Finally, the models will be evaluated in multiple ways: Accuracy, AUC (Area Under the ROC Curve), Log Loss, Sensitivity, Specificity, Precision, Recall, and the significance test with CV error.

Introduction

Credit risk is a primary risk factor in financial industry. According to David O Bleim, most of the systematic financial crisis and risks arise from enormous portfolios comprised of bad loans. Our questions are as follows: How can we mitigate the danger that arises from credit risk? What is the best model to predict this risk? When people submitted their loan applications, a bank needs to make a decision whether to approve or not. To reduce the loss from the bank's perspective, the bank needs a standard or decision rule regarding who to give approval of the loan and who not to. Thus, we analyzed the data by approaching various ways to investigate the problems and provide the solution or guideline to avoid the potential credit risk. In order to find the best prediction model, we constructed multiple classification models and tuned to reduce the cross-validation error with producing the best performance rate.

Data Exploration

Data Exploration is very important and required to figure out and summarize the characteristics of the dataset before modelling. The main objectives of this process are to understand the data and data type, check the missing values, and the balance of the data. The target variable is Creditability, and the features that are used for modelling are Account Balance, Payment Status of Previous Credit, Purpose, Value Savings Stocks, Length of current employment, Instalment per cent, Sex Marital Status, Guarantors, Duration in Current address, Most valuable available asset, Concurrent Credits, Type of apartment, No of Credits at this Bank, Occupation, No of dependents, Telephone, Foreign Worker, Duration of Credit month, Credit Amount, and years. There are total of 21 columns with 1,000 rows. Among them, there are 3

numerical columns and 17 categorical data columns. As shown below, Figure 1-1 represents the distribution of each variable in our dataset, and we scaled down the continuous variables: Age, Credit Amount, and Duration of Credit (Fig1-2). Next, by creating a sub dataset, continuous variables were also checked the independencies and the scale. (Fig 1-3).

Methodologies - Classification Algorithms

In this project, a comparative analysis was conducted to study the effectiveness of multiple classification approaches to predict a binary classification. Totally eight models were built and evaluated, including Decision Tree, Logistic Regression, Support Vector Machine (linear), Support Vector Machine (radial), Random Forest, AdaBoost, Gradient Boost, and XGBoost.

Before building models, the training dataset was randomly selected 67% from the dataset, and the remaining 33% was selected as a testing dataset. This split process was conducted 30 times and reduced the model variance of the random process. All models were built based on the training dataset and evaluated the performances on the testing dataset. According to Opitz and Maclin, in order to train the model, 5-fold cross validation was performed to reduce the variance of an accuracy score and accuracy scores were utilized as one of the metrics to select the final model. The parameters that performs the best will be delineated below, and the parameters details for tuning will be described in code appendix.

1. Decision Tree

The first classification model is Decision Tree learning algorithm. The objective of this algorithm is to create a model that predicts the number of target variable with the multiple input variables. Each branch contains a set of attributes, classification rules which are associated with

a class label. Initially, the decision tree was built with gain ratio. After the cross validation with tuning, the best performance decision tree model was built by tuning the parameters, where 5 minimum-samples-split, 8 max-depth, and entropy were chosen. The accuracy of this tuned decision tree is 0.723749, and the CV error with alpha at 5% is 0.021910.

2. Logistic Regression

The second algorithm is logistic regression. The objective of this logistic regression model is to evaluate the probability of a discrete outcomes. Since the target variable is binary (Credible or Not Credible), this algorithm is quite efficient. By looking at the formula, the features, 'Account Balance', 'No of dependents', and 'Foreign Worker', contributed the most to determine whether the bank approves or not to. The accuracy of this algorithm is 0.750823, the CV error with alpha at 5% is 0.021511. The final logistic regression model was expressed as

logit [creditability = 1] = -4.6001 + (0.6337*Account Balance) + (-0.4369*Duration of Credit (month)) + (0.4543*Payment Status of Previous Credit) + (0.0322*Purpose) +

(-0.0789*Credit Amount) + (0.2574*Value Savings/Stocks) + (0.1313*Length of current employment) + (-0.3480*Instalment per cent) + (0.3604*Sex & Marital Status) + (0.5403*Guarantors) + (-0.0430*Duration in Current address) + (-0.2350*Most valuable available asset) + (0.3059*Age (years)) + (0.2285*Concurrent Credits) + (0.1043*Type of apartment) + (-0.2569*No of Credits at this Bank) + (0.0258*Occupation) + (-0.6543*No of dependents) + (0.1085*Telephone) + (1.3248*Foreign Worker).

3. Support Vector Machines – Linear

The third algorithm is Support Vector Machine linear. SVM (Support Vector Machine) is an algorithm that classifies the two different instances by the hyperplane. A hyper-plane separates the two classes as best as possible, and the vector of weights were coordinated to orthogonal to the hyper-plane to be given information about discrimination of the two classes referred by the research paper from Pohang University of Science and Technology.

SVM Linear is a method that linearly classifies the instances by hyperplane. This is quite fast but has difficulties to classify the complex data. The final model for support vector machines (linear) was designated with the parameter, where the cost was 0.5, gamma was 0.1 and 'linear' kernel was used.

4. Support Vector Machines – Radial

The forth algorithm is Support Vector Machine non-linear (radial). Similar to SVM linear, SVM radial also classifies the instances with the hyperplane, but this is a method that non-linearly classifies the instances by transforming the low dimensional space to higher dimensional space with the kernel equal to 'rbf' . With the cross validation and tuning, the best hyper-parameter tuning suggest that cost was equal to 0.5, and 0.1 for gamma.

5. Random Forest

The fifth model is Random Forest. Random Forest is one of the ensemble classifiers that creates multiple decision trees and decide the output or the predict the class by the mode of the outputs of each tree. With the cross validation with tuning, the parameters that has the best performance are 900 trees. This algorithm has 0.832098 accuracy with 0.024524 CV error. One of the benefit of Random Forest is to change the algorithm to the way that subtrees were learned, so it could result in achieving the highest accuracy and the lowest error. Fig 2-4 shows that the changes of error rate as number of trees increased.

6. AdaBoost

The sixth model is Adaboost which is also an ensemble technique and develops the weak classifiers to a strong classifier. Adaboost is one of the best used to boost the performance of decision trees on binary classification. The main parameters for this method are the number of iteration and the learning rate to develop the models. By computing the different numbers for tuning with the cross validation, the parameters that has best performance has 300 estimators (iteration) with 0.5 for the learning rate.

7. Gradient Boost

The seventh classification model is constructed using gradient boosting method. This algorithm also constructs a model with boosting method basis with optimizing the loss of the model by adding the weak learners using gradient descent like procedure. While the best gradient best model was building, the weak learners were combined, and prediction accuracies were improved. A lower weight was assigned to the observations where predicted correctly, a higher weight was assigned to the observations where miss-classified. Similar to AdaBoost, the weights also affects the feature importance. The parameters that produce the best performance with tuning have the number of estimators was equal to 400, and learning rate was equal to 0.05.

8. XGBoost

XGBoost offers several advanced features for model tuning and fast process. XGBoost is gradient boosting algorithm basis but can also perform the three main forms of gradient boosting: Gradient Boosting, Stochastic Gradient Boosting, and Regularized Gradient Boosting. This algorithm is robust enough to support fine tuning and regularized the parameters. For XGBoost, the main parameters for tuning (cross validation) for this algorithm are nround (number of

iteration), objective (either binary or linear), and early stopping (the larger the faster process but may cause the overfitting). The best optimized parameters are 30 for nround, linear for objective, and 20 for the early stopping value. This algorithm produced the highest accuracy which is 0.846, 0.919 for AUC, and the lowest log loss, 0.384.

Conclusion and Discussion & Future work

The table below is the summary of our model evaluation after the cross validation and the tuning the parameters. As shown below, we evaluated the models in different ways: Accuracy, Log Loss, Precision, Recall, Area Under Curve from ROC, Sensitivity, and Specificity. Accuracy which is based on one specific cut point (or discrete measurement), ROC which captures all of the cut points and plots the sensitivity (True Positive rate) and specificity (True negative rate), Precision (percent of selected true positive to all selected elements), Recall (ratio scaled true positive to all true positives), and Logarithmic loss (quantified loss function).

Fig 2-1

	Accuracy	Lower Bound	Upper Bound	CV error (95%)	AUC	Log Loss	Sensitivity	Specificity	Precision	Recall
Decision Tree	0.723749	0.701839	0.745659	0.021910	0.807749	1.252750	0.646552	0.697959	0.729647	0.730794
Logistic Regression	0.750823	0.702238	0.772334	0.021511	0.811145	0.530084	0.689655	0.730612	0.754045	0.754313
Support Vector Machines (linear)	0.756246	0.705554	0.774441	0.018195	0.813291	0.527068	0.676724	0.730612	0.761737	0.756417
Support Vector Machines (radial)	0.837531	0.69577	0.86551	0.027979	0.926131	0.356092	0.869388	0.681034	0.808145	0.879121
Random Forest	0.832098	0.699225	0.856622	0.024524	0.927259	0.386443	0.775862	0.771429	0.830693	0.841958
AdaBoost	0.814759	0.698878	0.83963	0.024871	0.889478	0.684689	0.728448	0.787755	0.819104	0.816175
Gradient Boosting	0.837526	0.705015	0.85626	0.018734	0.918420	0.367137	0.741379	0.751020	0.848072	0.829170
XGBoost	0.86118	0.7	0.884929	0.023749	0.918713	0.384369	0.741379	0.751020	0.845022	0.854839

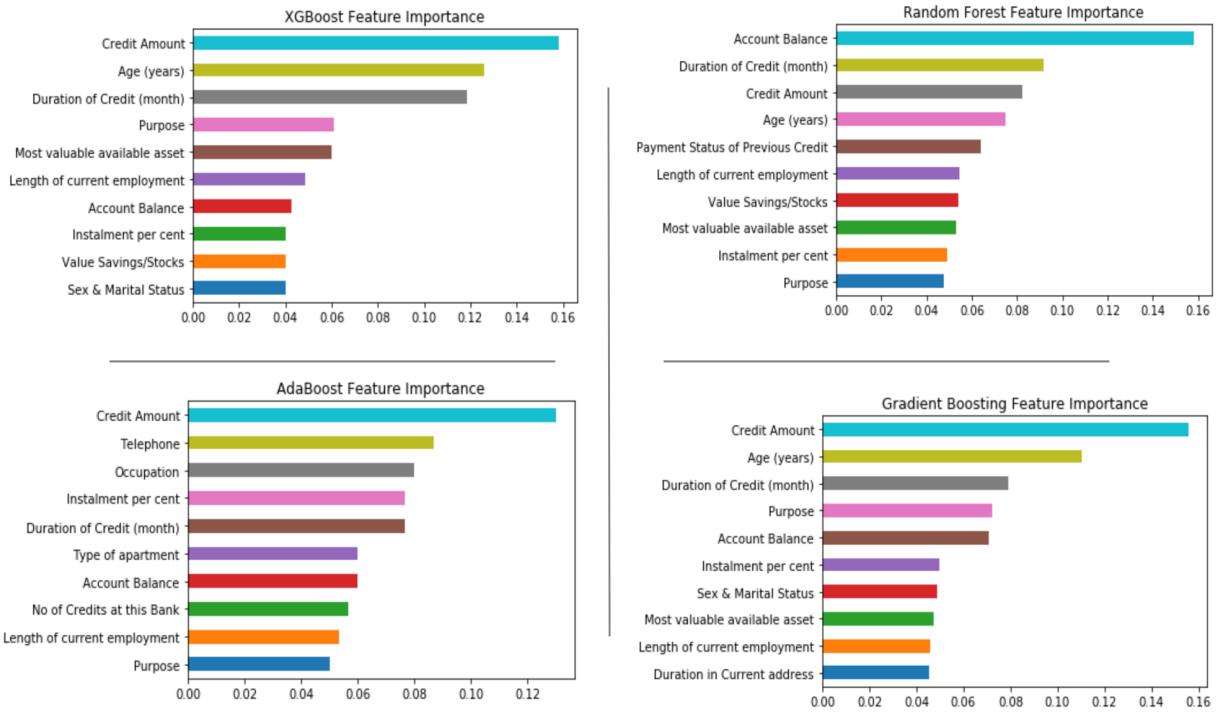
While models developed using ensemble techniques indicate great performance after tuning and validation, there were misclassifications found from testing. These misclassifications convey that probabilistic approaches can lead in predicting results quite accurately but not perfectly. It is because predicting features such as account balance, credit amount, or duration of credit is almost impossible since these features can fluctuate more easily compare to other features. The next section, variable importance, will discuss about which features considered to be important in each model to help the readers of this report understand what is behind the logic.

Variable Importance

In addition to evaluating and comparing the models, we also noted couple different findings. The benefit of the ensemble of classifiers is to improve the performance or the accuracy of prediction, but the main drawback is that we cannot get the logic or the information behind the model. This indicates which variables are important when the model predicts whether the loan applicants are credible or not. So, we ranked and sorted the variables by importance, which were used in each model. The top most important variables are: Account Balance, Credit Amount, Duration of Credit month, and Age according to the results below (Fig2-3).

Feature/Model	XGBoost	Random Forest	AdaBoost	Gradient Boosting
Feature 1	Credit Amount	Account Balance	Credit Amount	Credit Amount
Feature 2	Age	Duration of Credit	Telephone	Age
Feature 3	Duration of Credit	Credit Amount	Occupation	Duration of Credit
Feature 4	Purpose	Age	Instalment percent	Purpose
Feature 5	Most Valuable available asset	Payment Status of Previous Credit	Duration of Credit	Account Balance

Fig 2-3



For future work, we can think of applying the neural network into the model to predict whether loan applicants would be able to pay back or not. Additionally, the model can be used further to evaluate loan applications of corporate clients, based on their financial position, company size, and current economy status. Finally, rather than classifying the risk of each applicant, measuring the credit risk of each applicant with different scale from 1 to 10 using Bayesian Network can be done in the future study, since our target variable is binary (credible or not credible).

References

David Opitz, Richard Maclin. 1999. Popular Ensemble Methods: An Empirical Study. p.188

Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung-Yang Bang. 2002. Support Vector Machine Ensemble with Bagging p.398

Tianqi Chen, Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. p.1

Thomas G Dietterich. Ensemble Methods in Machine Learning. Comparing Different Ensemble Methods. Oregon State University p.10

Prem Melville. 2003. Creating Diverse Ensemble Classifiers. Influence of Ensemble Size. The University of Texas at Austin. P.21

Moacir P. Ponti Jr. Combining Classifiers: from the creation of ensembles to the decision fusion. Why use a multiple classifier.system?. University of São Paulo at São Carlos. P.2

Appendix

Fig 1-1

Fig 1-2


Account Balance
Age (years)
Concurrent Credits
Credit Amount
Creditability
Duration in Current address
Duration of Credit (month)
Foreign Worker
Guarantors
Instalment per cent
Length of current employment
Most valuable available asset
No of Credits at this Bank
No of dependents
Occupation
Payment Status of Previous Credit
Purpose
Sex & Marital Status
Telephone
Type of apartment
Value Savings/Stocks

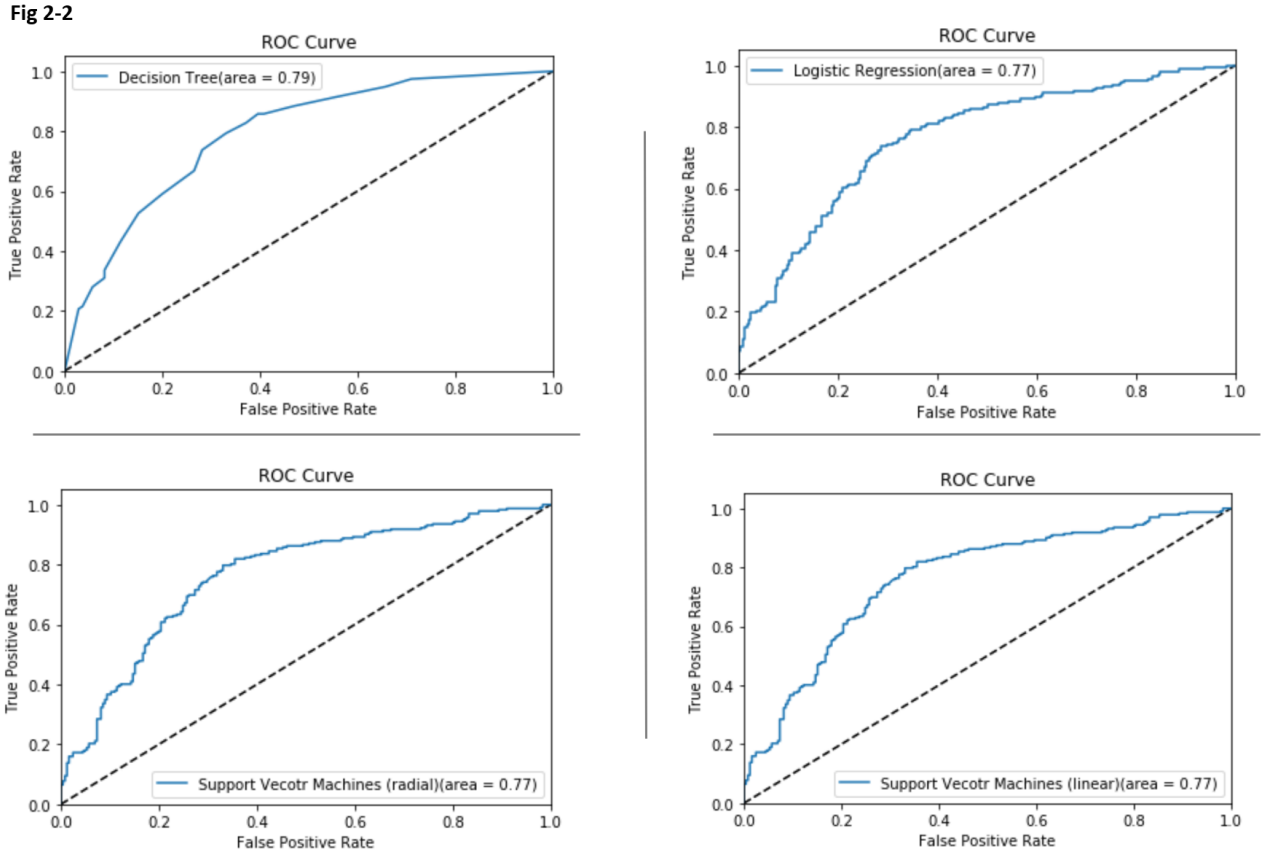
Fig 1-3


Age (years)
Credit Amount
Duration of Credit (month)
1
0.032
-0.038
0.032
1
0.62
-0.038
0.62
1
1.0
0.8
0.6
0.4
0.2
0.0
Age (years)
Credit Amount
Duration of Credit (month)

Fig 2-1

	Lower Bound	Accuracy	Upper Bound	Std.dev (95%)	AUC	Log Loss	Sensitivity	Specificity	Precision	Recall
Decision Tree	0.701839	0.723749	0.745659	0.021910	0.807749	1.252750	0.646552	0.697959	0.729647	0.730794
Logistic Regression	0.702238	0.750823	0.772334	0.021511	0.811145	0.530084	0.689655	0.730612	0.754045	0.754313
Support Vector Machines (linear)	0.705554	0.756246	0.774441	0.018195	0.813291	0.527068	0.676724	0.730612	0.761737	0.756417
Support Vector Machines (radial)	0.69577	0.837531	0.86551	0.027979	0.926131	0.356092	0.869388	0.681034	0.808145	0.879121
Random Forest	0.699225	0.832098	0.856622	0.024524	0.927259	0.386443	0.775862	0.771429	0.830693	0.841958
AdaBoost	0.698878	0.814759	0.83963	0.024871	0.889478	0.684689	0.728448	0.787755	0.819104	0.816175
Gradient Boosting	0.705015	0.837526	0.85626	0.018734	0.918420	0.367137	0.741379	0.751020	0.848072	0.829170
XGBoost	0.7	0.86118	0.884929	0.023749	0.918713	0.384369	0.741379	0.751020	0.845022	0.854839

Fig 2-2



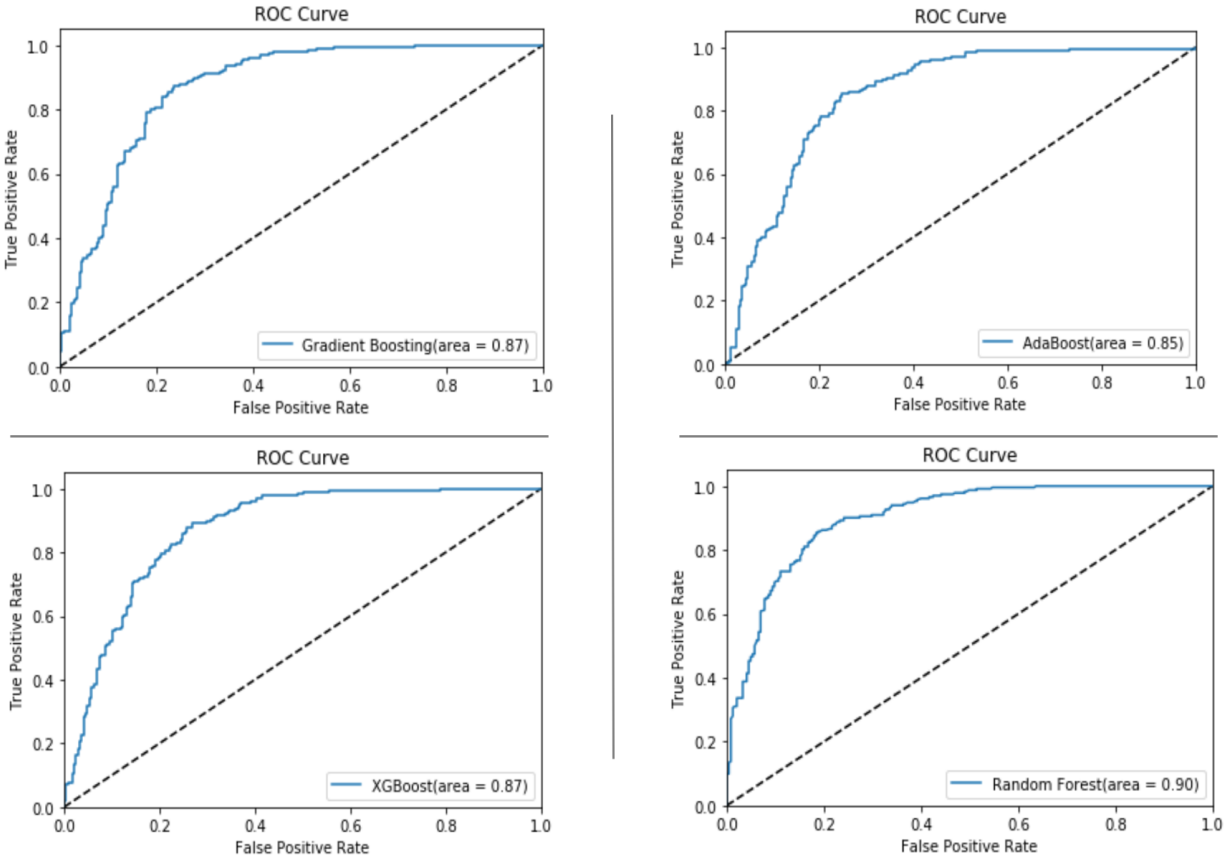

ROC Curve
True Positive Rate
False Positive Rate
Gradient Boosting(area = 0.87)
ROC Curve
True Positive Rate
False Positive Rate
AdaBoost(area = 0.85)
ROC Curve
True Positive Rate
False Positive Rate
XGBoost(area = 0.87)
ROC Curve
True Positive Rate
False Positive Rate
Random Forest(area = 0.90)

Fig 2-3

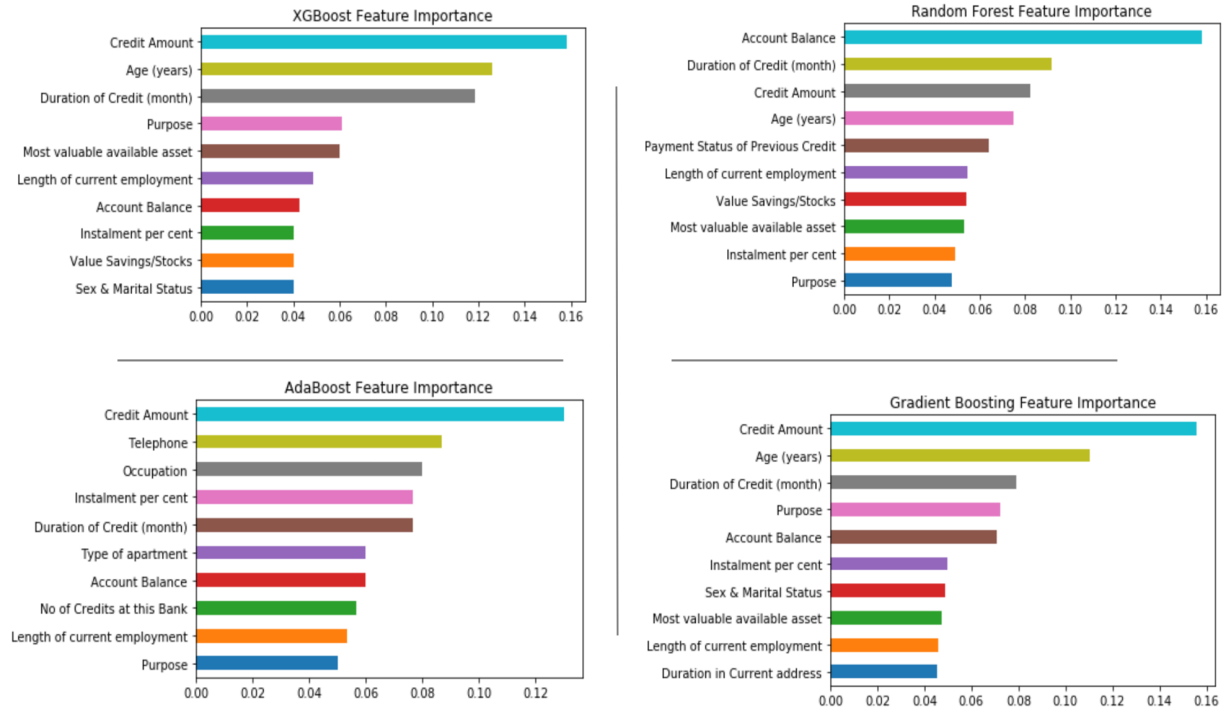
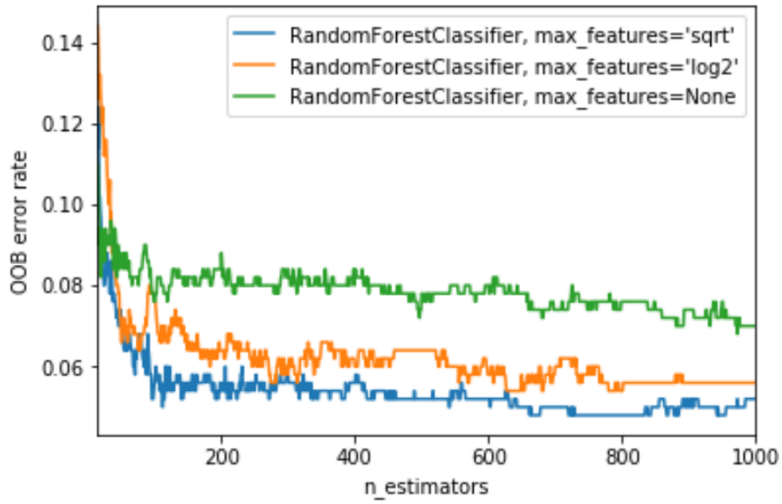
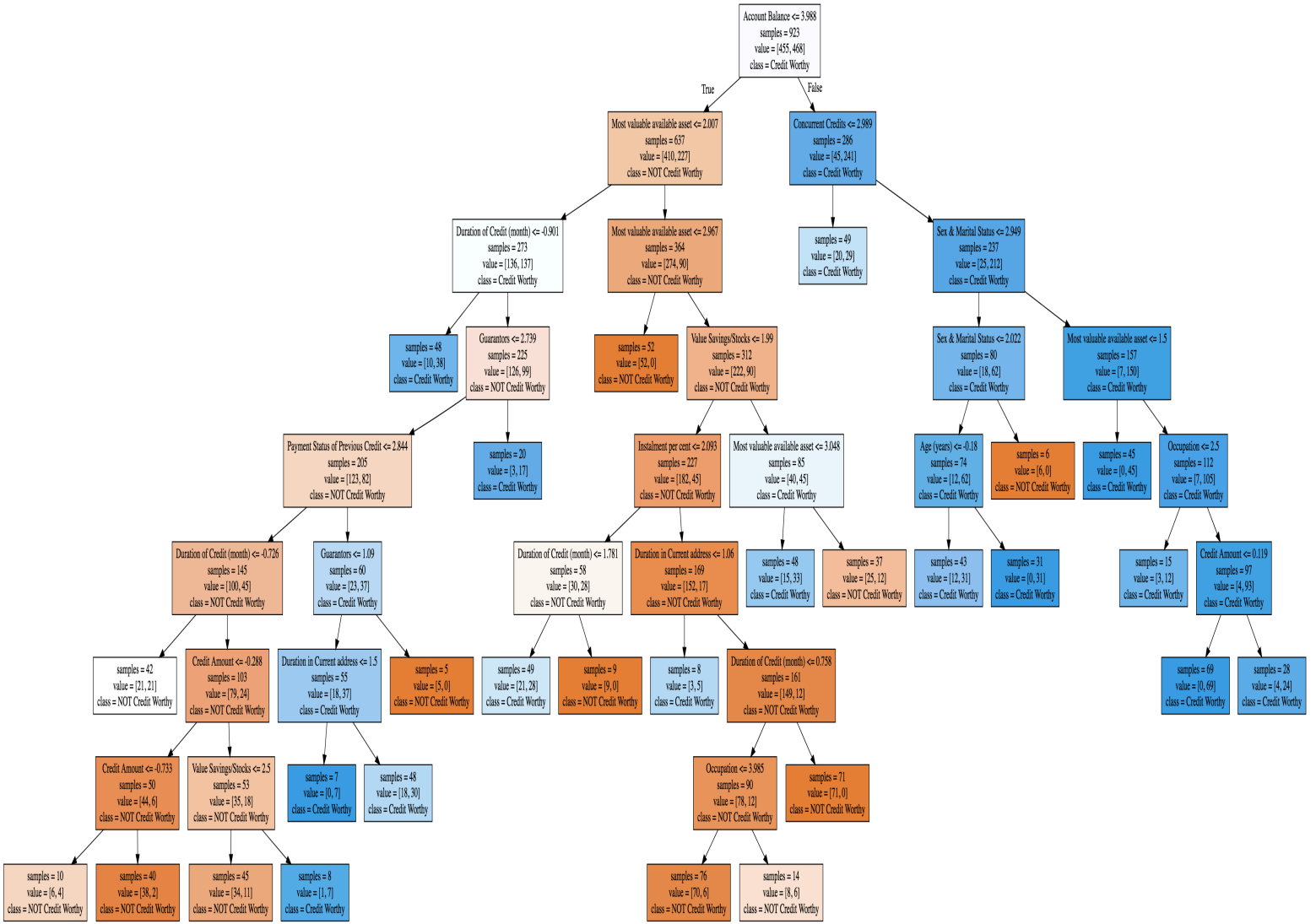

XGBoost Feature Importance
Credit Amount
Age (years)
Duration of Credit (month)
Purpose
Most valuable available asset
Length of current employment
Account Balance
Instalment per cent
Value Savings/Stocks
Sex & Marital Status
Random Forest Feature Importance
Account Balance
Duration of Credit (month)
Credit Amount
Age (years)
Payment Status of Previous Credit
Length of current employment
Value Savings/Stocks
Most valuable available asset
Instalment per cent
Purpose
AdaBoost Feature Importance
Credit Amount
Telephone
Occupation
Instalment per cent
Duration of Credit (month)
Type of apartment
Account Balance
No of Credits at this Bank
Length of current employment
Purpose
Gradient Boosting Feature Importance
Credit Amount
Age (years)
Duration of Credit (month)
Purpose
Account Balance
Instalment per cent
Sex & Marital Status
Most valuable available asset
Length of current employment
Duration in Current address

Fig 2-4


RandomForestClassifier, max_features='sqrt'
RandomForestClassifier, max_features='log2'
RandomForestClassifier, max_features=None
OOB error rate
0.14
0.12
0.10
0.08
0.06
200
400
600
800
1000
n_estimators


Account Balance <= 3.988
samples = 923
value = [455, 468]
class = Credit Worthy
True
False
Most valuable available asset <= 2.007
samples = 637
value = [410, 227]
class = NOT Credit Worthy
Concurrent Credits <= 2.989
samples = 286
value = [45, 241]
class = Credit Worthy
Duration of Credit (month) <= -0.901
samples = 273
value = [136, 137]
class = Credit Worthy
Most valuable available asset <= 2.967
samples = 364
value = [274, 90]
class = NOT Credit Worthy
samples = 49
value = [20, 29]
class = Credit Worthy
Sex & Marital Status <= 2.949
samples = 237
value = [25, 212]
class = Credit Worthy
samples = 48
value = [10, 38]
class = Credit Worthy
Guarantors <= 2.739
samples = 225
value = [126, 99]
class = NOT Credit Worthy
samples = 52
value = [52, 0]
class = NOT Credit Worthy
Value Savings/Stocks <= 1.99
samples = 312
value = [222, 90]
class = NOT Credit Worthy
Sex & Marital Status <= 2.022
samples = 80
value = [18, 62]
class = Credit Worthy
Most valuable available asset <= 1.5
samples = 157
value = [7, 150]
class = Credit Worthy
Payment Status of Previous Credit <= 2.844
samples = 205
value = [123, 82]
class = NOT Credit Worthy
samples = 20
value = [3, 17]
class = Credit Worthy
Instalment per cent <= 2.093
samples = 227
value = [182, 45]
class = NOT Credit Worthy
Most valuable available asset <= 3.048
samples = 85
value = [40, 45]
class = Credit Worthy
Age (years) <= -0.18
samples = 74
value = [12, 62]
class = Credit Worthy
samples = 6
value = [6, 0]
class = NOT Credit Worthy
samples = 45
value = [0, 45]
class = Credit Worthy
Occupation <= 2.5
samples = 112
value = [7, 105]
class = Credit Worthy
Duration of Credit (month) <= -0.726
samples = 145
value = [100, 45]
class = NOT Credit Worthy
Guarantors <= 1.09
samples = 60
value = [23, 37]
class = Credit Worthy
Duration of Credit (month) <= 1.781
samples = 58
value = [30, 28]
class = NOT Credit Worthy
Duration in Current address <= 1.06
samples = 169
value = [152, 17]
class = NOT Credit Worthy
samples = 48
value = [15, 33]
class = Credit Worthy
samples = 37
value = [25, 12]
class = NOT Credit Worthy
samples = 43
value = [12, 31]
class = Credit Worthy
samples = 31
value = [0, 31]
class = Credit Worthy
samples = 15
value = [3, 12]
class = Credit Worthy
Credit Amount <= 0.119
samples = 97
value = [4, 93]
class = Credit Worthy
samples = 42
value = [21, 21]
class = NOT Credit Worthy
Credit Amount <= -0.288
samples = 103
value = [79, 24]
class = NOT Credit Worthy
Duration in Current address <= 1.5
samples = 55
value = [18, 37]
class = Credit Worthy
samples = 5
value = [5, 0]
class = NOT Credit Worthy
samples = 49
value = [21, 28]
class = Credit Worthy
samples = 9
value = [9, 0]
class = NOT Credit Worthy
samples = 8
value = [3, 5]
class = Credit Worthy
Duration of Credit (month) <= 0.758
samples = 161
value = [149, 12]
class = NOT Credit Worthy
samples = 69
value = [0, 69]
class = Credit Worthy
samples = 28
value = [4, 24]
class = Credit Worthy
Credit Amount <= -0.733
samples = 50
value = [44, 6]
class = NOT Credit Worthy
Value Savings/Stocks <= 2.5
samples = 53
value = [35, 18]
class = NOT Credit Worthy
samples = 7
value = [0, 7]
class = Credit Worthy
samples = 48
value = [18, 30]
class = Credit Worthy
Occupation <= 3.985
samples = 90
value = [78, 12]
class = NOT Credit Worthy
samples = 71
value = [71, 0]
class = NOT Credit Worthy
samples = 10
value = [6, 4]
class = NOT Credit Worthy
samples = 40
value = [38, 2]
class = NOT Credit Worthy
samples = 45
value = [34, 11]
class = NOT Credit Worthy
samples = 8
value = [1, 7]
class = Credit Worthy
samples = 76
value = [70, 6]
class = NOT Credit Worthy
samples = 14
value = [8, 6]
class = NOT Credit Worthy

(Python)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import auc, roc_curve, roc_auc_score
from sklearn.model_selection import learning_curve
from xgboost import XGBClassifier

from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

import warnings
warnings.filterwarnings('ignore')
import graphviz
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

1. Exploratory Analysis

1-1. Load the dataset

```
df = pd.read_csv('german_credit.csv')
df[df.dtypes[(df.dtypes == "float64")
         |(df.dtypes == "int64")
        ].index.values].hist(figsize=[16,18])
plt.show()
cont = ['Age (years)', 'Credit Amount', 'Duration of Credit (month)']
a = pd.DataFrame(df[cont].corr())
sns.heatmap(a, annot = True, linewidths = 0.2, cmap = "Blues")
fig = plt.gcf()
```

```
fig.set_size_inches(8,5)
plt.show()
df[cont] = preprocessing.scale(df[cont])
df[df.dtypes[(df.dtypes == "float64")
         |(df.dtypes == "int64")
        ].index.values].hist(figsize=[16,18])
plt.show()
credit_y = df['Creditability']
credit_x = df[df.columns[1:]]

dummy = df.drop(df[cont], axis =1)
dummy = dummy[dummy.columns[1:]]
dummy.head()

for i in dummy.columns:
    show = sns.factorplot(x=str(i), y = "Creditability", data = df, kind =
"bar")
    show.set(ylim = (0,1))
    show.despine(left = True)
```

```
from imblearn.over_sampling import SMOTE
x_smote, y_smote = SMOTE().fit_sample(credit_x, credit_y)
x_smote.shape, y_smote.shape
```

```
x_train, x_test, y_train, y_test = train_test_split(x_smote, y_smote,
test_size = 0.34)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

2. Predicting Models with single classifiers

We will predict if the customers have good credit or not using some classification algorithms such as,

1) Decision Tree

2) Logistic Regression

3) K-Nearest Neighbors

4) Support Vector Machines (linear)

5) Support Vector Machines (radial)

5) Random Forest

6) AdaBoost

7) Gradient Boosting

8) XGBoost

```
classifiers = ["Decision Tree","Logistic Regression",
                    "K-Nearest Neighbors",
                    "Support Vecotr Machines (linear)",
                    "Support Vecotr Machines (radial)",
                    "Random Forest",
                    "AdaBoost",
                    "Gradient Boosting",
                    'XGBoost']
n_estimators = 300
max_depth = 2
models = [DecisionTreeClassifier(max_depth = max_depth),
              LogisticRegression(),
              KNeighborsClassifier(),
              SVC(kernel='linear',C=0.1,gamma=0.1, probability = True),
              SVC(kernel='rbf',C=1,gamma=0.1,probability = True),
              RandomForestClassifier(n_estimators = n_estimators,
max_depth = max_depth),
              AdaBoostClassifier(n_estimators = n_estimators),
              GradientBoostingClassifier(n_estimators = n_estimators),
              XGBClassifier(n_estimators = n_estimators)]
```

2-1. Base Models

```
def baseModel(classifiers, models ,x_train, y_train, x_test, y_test):

    test_lst = []
    train_lst = []
    clfs = []
    roc = []

    for i, model in zip(classifiers,models):
        clf = model.fit(x_train, y_train)

        clfs.append(clf)
        train_lst.append(clf.score(x_train, y_train))
        test_lst.append(clf.score(x_test, y_test))

        prob = clf.predict_proba(x_test)[:,1]
        FP,TP,_ = roc_curve(y_test, prob)
        roc_auc = auc(FP,TP)
        roc.append(roc_auc)

    model_base = pd.DataFrame({"Train Accuracy": train_lst, "Test
Accuracy": test_lst,
                           "Area Under ROC Curve": roc}, index =
classifiers)
    return model_base
```

```
baseModel(classifiers,models,x_train, y_train, x_test, y_test)
```

- *### 2-2. Base Models with cross validate with k = 5*

```
def cvModel(baseModel, K):
    kfold = KFold(n_splits=K, random_state=22)

    cv_results = []
    for clf in models:
        cv_results.append(cross_val_score(clf, x_train, y_train,
scoring = "accuracy", cv = kfold, n_jobs = 3))

    cv_means = []
    cv_std = []
    for res in cv_results:
        cv_means.append(np.mean(res))
        cv_std.append(np.std(res))

    model_cv = pd.DataFrame({"Cross-validation Mean":cv_means, "Cross-
validation Error":cv_std}, index = classifiers)
    return model_cv

#cvModel(baseModel, 5)
```

2-5. Hyper-parameter Tuning for the base models
We can get a better model by tuning the parameters. Hyper-parameter Tuning is to tune to change the learning rate of the algorithm so that you can get a better model.

TUNE

```
dtc = DecisionTreeClassifier()
lrc = LogisticRegression()
svc_linear = SVC(kernel='linear', probability = True)
svc_radial = SVC(kernel='rbf',probability = True)
rfc = RandomForestClassifier()
adc = AdaBoostClassifier()
gbc = GradientBoostingClassifier()
xgb = XGBClassifier()

kfold = 5
```

- *#### DT tune*

```
# avoid overfitting
min_samples_split = list(range(50,500,50)) # restrict the size of sample
leaf
max_depth = list(range(2,100))
criterion = ['entropy']

param_grid = {'criterion' : criterion, 'min_samples_split':
min_samples_split,
```

```
              'max_depth': max_depth}
gd1 = GridSearchCV(estimator = dtc, param_grid = param_grid, cv = kfold)
gd1.fit(x_train,y_train)
lr = gd1.fit(x_train,y_train)

print(gd1.best_score_)
print(gd1.best_estimator_)

export_graphviz(gd1.best_estimator_, out_file = 'dt.dot', class_names =
["NOT Credit Worthy","Credit Worthy"],
                feature_names = credit_x.columns, impurity = False, filled
= True)
with open('dt.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

- *#### Logistic Regression tune*

```
param_grid = {"penalty" : ['l1', 'l2'], "C" : np.logspace(0, 4, 10)}
gd2 = GridSearchCV(estimator = lrc, param_grid = param_grid, cv = kfold)
gd2.fit(x_train,y_train)
print(gd2.best_score_)

print(gd2.best_estimator_)

lrclf = LogisticRegression(C=59.94842503189409, class_weight=None,
dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
lrmodel = lrclf.fit(x_train, y_train)
print('logit[win=1] = %.4f' %lrmodel.intercept_, end=' ')
for i in range(np.reshape(credit_x.columns.values, 20).shape[0]):
    print('+ (%.4f*%s)'% (np.reshape(lrmodel.coef_, 20)[i],
                          np.reshape(credit_x.columns.values, 20)[i]),
end=' ')
```

- ***SVM tune (linear)***

```
C = [0.001,0.05,0.1,0.5,1]
gamma = [0.1,0.25,0.5,0.75,1.0]
kernel = ['linear']

param_grid = {'kernel':kernel,'C':C,'gamma':gamma}
gd3 = GridSearchCV(estimator = svc_linear, param_grid = param_grid, cv =
kfold)
gd3.fit(x_train,y_train)
print(gd3.best_score_)
print(gd3.best_estimator_)
```

- ***SVM tune (radial)***

```
C = [0.001,0.05,0.1,0.5,1]
gamma = [0.1,0.25,0.5,0.75,1.0]
kernel = ['linear']

param_grid = {'kernel':kernel,'C':C,'gamma':gamma}
gd4 = GridSearchCV(estimator = svc_radial, param_grid = param_grid, cv
= kfold, verbose = True)
gd4.fit(x_train,y_train)
print(gd4.best_score_)
print(gd4.best_estimator_)
```

ENSEMBLES

ENSEMBLE : RANDOM FOREST

- ***Random Forest (DT)*** BEST

```
n_estimators = list(range(100,1000,100))
param_grid = {'n_estimators': n_estimators}

gd5 = GridSearchCV(estimator = rfc, param_grid = param_grid, cv =
kfold, verbose = True)
gd5.fit(x_train,y_train)
print(gd5.best_score_)
print(gd5.best_estimator_)
```

```
from sklearn.datasets import make_classification
from collections import OrderedDict

RANDOM_STATE = 123

# Generate a binary classification dataset.
X, y = make_classification(n_samples=500, n_features=25,
                           n_clusters_per_class=1, n_informative=15,
                           random_state=RANDOM_STATE)

# NOTE: Setting the `warm_start` construction parameter to `True` disables
# support for parallelized ensembles but is necessary for tracking the OOB
# error trajectory during training.
ensemble_clfs = [
    ("RandomForestClassifier, max_features='sqrt'",
        RandomForestClassifier(warm_start=True, oob_score=True,
                               max_features="sqrt",
                               random_state=RANDOM_STATE)),
    ("RandomForestClassifier, max_features='log2'",
        RandomForestClassifier(warm_start=True, max_features='log2',
                               oob_score=True,
                               random_state=RANDOM_STATE)),
    ("RandomForestClassifier, max_features=None",
        RandomForestClassifier(warm_start=True, max_features=None,
                               oob_score=True,
                               random_state=RANDOM_STATE))
```

```
]
# Map a classifier name to a list of (<n_estimators>, <error rate>) pairs.
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)
min_estimators = 15
max_estimators = 1000

for label, clf in ensemble_clfs:
    for i in range(min_estimators, max_estimators + 1):
        clf.set_params(n_estimators=i)
        clf.fit(X, y)

        # Record the OOB error for each `n_estimators=i` setting.
        oob_error = 1 - clf.oob_score_
        error_rate[label].append((i, oob_error))

# Generate the "OOB error rate" vs. "n_estimators" plot.
for label, clf_err in error_rate.items():
    xs, ys = zip(*clf_err)
    plt.plot(xs, ys, label=label)

plt.xlim(min_estimators, max_estimators)
plt.xlabel("n_estimators")
plt.ylabel("OOB error rate")
plt.legend(loc="upper right")
plt.show()
```

ENSEMBLE : AdaBoost

```
n_estimators = list(range(100,500,100))
learning_rate = [0.001,0.05,0.1,0.5,1]
#max_depth = list(range(1,20,2))
#min_samples_split = list(range(10,500,20))

param_grid = {'n_estimators':n_estimators, 'learning_rate':learning_rate}
gd6 = GridSearchCV(estimator = adc, param_grid = param_grid, cv = kfold,
verbose = True)
gd6.fit(x_train,y_train)
print(gd6.best_score_)
print(gd6.best_estimator_)
```

ENSEMBLE : GRADIENT BOOST¶

- *#### Gradient Boost Tune*

```
n_estimators = list(range(100,1000,100))
learning_rate = [0.001,0.05,0.1,0.5,1]
#max_depth = list(range(1,20,2))
#min_samples_split = list(range(10,500,20))
```

```
param_grid = {'n_estimators':n_estimators,
'learning_rate':learning_rate}
gd7 = GridSearchCV(estimator = gbc, param_grid = param_grid, cv =
kfold, verbose = True)
gd7.fit(x_train,y_train)
print(gd7.best_score_)
print(gd7.best_estimator_)
```

ENSEMBLE : XGBOOST

- *#### XGBoost*

```
n_estimators = list(range(100,1000,100))
learning_rate = [0.001,0.05,0.1,0.5,1]

param_grid =
{'n_estimators':n_estimators,'learning_rate':learning_rate}
gd8 = GridSearchCV(estimator = xgb, param_grid = param_grid, cv = 5,
verbose = True)
gd8.fit(x_train,y_train)
print(gd8.best_score_)
print(gd8.best_estimator_)
```

BEST TUNED MODEL

```
classifiers = ["Decision Tree","Logistic Regression",
               "Support Vecotr Machines (linear)",
               "Support Vecotr Machines (radial)",
               "Random Forest", "AdaBoost",
               "Gradient Boosting", "XGBoost"]

models = [gd1.best_estimator_, gd2.best_estimator_,
          gd3.best_estimator_, gd4.best_estimator_,
          gd5.best_estimator_, gd6.best_estimator_,
          gd7.best_estimator_, gd8.best_estimator_]

acclst = []
stdlst = []
roclst = []
loslst = []
prelst = []
reclst = []
senlst = []
spelst = []

for model in models:
    # Accuracy
    acc = cross_val_score(model, x_train, y_train, scoring = "accuracy",
                          cv = kfold,  n_jobs = 3)
    acclst.append(np.mean(acc))
    # Standard deviation
    stdlst.append(np.mean(acc.std()))
    # ROC-AUC score
```

```
    roc = cross_val_score(model, x_train, y_train, scoring = "roc_auc",
                          cv = kfold,  n_jobs = 3)
    roclst.append(np.mean(roc))
    # Log-loss score
    los = cross_val_score(model, x_train, y_train, scoring =
"neg_log_loss",
                          cv = kfold,  n_jobs = 3)
    loslst.append(np.mean(los))
    # Precision
    pre = cross_val_score(model, x_train, y_train, scoring = "precision",
                          cv = kfold,  n_jobs = 3)
    prelst.append(np.mean(pre))
    # Recall
    rec = cross_val_score(model, x_train, y_train, scoring = "recall",
                          cv = kfold,  n_jobs = 3)
    reclst.append(np.mean(rec))

    # confusion matrix
    y_pred = cross_val_predict(model, x_test, y_test, cv = 5)
    conf_mat = confusion_matrix(y_test, y_pred)
    TP = conf_mat[1, 1]
    TN = conf_mat[0, 0]
    FP = conf_mat[0, 1]
    FN = conf_mat[1, 0]
    Sen = TP/(TP+FN)
    Spe = TN/(TN+FP)
    senlst.append(Sen)
    spelst.append(Spe)


model_tuned = pd.DataFrame({"Accuracy": acclst,
                            "Acc-std.dev": stdlst,
                            "Roc-Auc": roclst,
                            "Log Loss": loslst,
                            "Precision": prelst,
                            "Recall": reclst,
                            "Sensitivity":senlst,
                            "Specificity":spelst}, index = classifiers)

model_tuned
```

ROC

```
for i, model in zip(classifiers,models):
    clf = model.fit(x_train, y_train)

    prob = clf.predict_proba(x_test)[:,1]
    FP,TP,_ = roc_curve(y_test, prob)
    roc_auc = auc(FP,TP)

    plt.figure()
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.plot(FP, TP, label = str(i)+ '(area = %0.2f)' % roc_auc)
```

```
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="best")
    plt.show()
```

7. Feature Importance

```
sub_clf = ["Decision Tree", "Random Forest", "AdaBoost",
           "Gradient Boosting", "XGBoost"]

sub_model = [gd1.best_estimator_,
             gd5.best_estimator_, gd6.best_estimator_,
             gd7.best_estimator_, gd8.best_estimator_]
for i, model in zip(sub_clf,sub_model):
    imp = pd.Series(model.feature_importances_, index = credit_x.columns)
    imp = imp.nlargest(10).sort_values(ascending = True)
    imp.plot(kind = 'barh', title = str(i)+' Feature Importance')
    plt.show()
```