# Random Forest Method II

## Learn How Ensemble Learning Can be Used for Predictive Modeling

# Contents

# Case Study – Predicting Loan Defaulters

## Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

## Objective

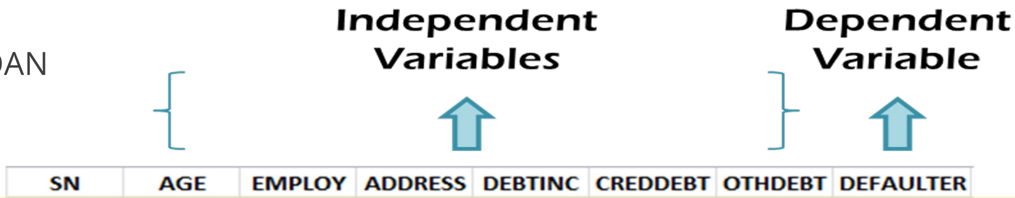
- To predict whether the customer applying for the loan will be a defaulter

## Available Information

- Sample size is 700
- Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts are the independent variables
- Defaulter (=1 if defaulter, 0 otherwise) is the dependent variable

# Data Snapshot

BANK LOAN




| Column | Description | Type | Measurement | Possible Values |
|---|---|---|---|---|
| SN | Serial Number | Integer | - | - |
| AGE | Age Groups | Integer | 1(<28 years), 2(28-40 years), 3(>40 years) | 3 |
| EMPLOY | Number of years customer working at current employer | Integer | - | Positive value |
| ADDRESS | Number of years customer staying at current address | Integer | - | Positive value |
| DEBTINC | Debt to Income Ratio | Continuous | - | Positive value |
| CREDDEBT | Credit to Debit Ratio | Continuous | - | Positive value |
| OTHDEBT | Other Debt | Continuous | - | Positive value |
| DEFAULTER | Whether customer defaulted on loan | Integer | 1(Defaulter), 0(Non-Defaulter) | 2 |

# Random Forest in Python

```
# Import required Libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,precision_score,
recall_score, accuracy_score,roc_curve, roc_auc_score
```

```
# Importing and Readying the Data
```

```
bankloan = pd.read_csv("BANK LOAN.csv")
bankloan1 = bankloan.drop(['SN'], axis = 1)
```

# Random Forest in Python

```
# Importing and Readying the Data
bankloan1['AGE'] = bankloan1['AGE'].astype('category')
bankloan1.dtypes
```

- Since it's a classification problem, dependent variable is assigned classes by converting to categorical using **as.type('category').**

```
# Output:
AGE          category
EMPLOY          int64
ADDRESS         int64
DEBTINC       float64
CREDDEBT      float64
OTHDEBT       float64
DEFAULTER       int64
```

```
bankloan2 = pd.get_dummies(bankloan1)
bankloan2.head()
```

- Create dummies **using pd.get_dummies** to convert categorical variable into dummy/indicator variables.

```
# Output:
```

| | EMPLOY | ADDRESS | DEBTINC | CREDDEBT | OTHDEBT | DEFAULTER | AGE_1 | AGE_2 | AGE_3 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 17 | 12 | 9.3 | 11.36 | 5.01 | 1 | 0 | 0 | 1 |
| 1 | 10 | 6 | 17.3 | 1.36 | 4.00 | 0 | 1 | 0 | 0 |
| 2 | 15 | 14 | 5.5 | 0.86 | 2.17 | 0 | 0 | 1 | 0 |
| 3 | 15 | 14 | 2.9 | 2.66 | 0.82 | 0 | 0 | 0 | 1 |
| 4 | 2 | 0 | 17.3 | 1.79 | 3.06 | 1 | 1 | 0 | 0 |

# Random Forest in Python

```
# Creating Train and Test Data Sets
```

```
X = bankloan2.loc[:,bankloan2.columns != 'DEFAULTER']
y = bankloan2.loc[:, 'DEFAULTER']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state = 999)
```

```
# Build Random Forest model
```

```
rf = RandomForestClassifier(random_state=999, n_estimators=100,
oob_score=True, max_features='sqrt')
rf.fit(X_train, y_train)
```

- **RandomForestClassifier()** performs Random Forest Algorithm
- **random_state=** sets the seed for random sampling
- **n_estimators=** defines the number of trees in the forest.
- **oob_score=** defines whether to use out-of-bag samples to estimate the generalization accuracy.
- **max_features=** defines the number of features to consider when looking for the best split: If "auto", then max_features=sqrt(n_features). If "sqrt", then max_features=sqrt(n_features) (same as "auto"). If "log2", then max_features=log2(n_features). If None, then max_features=n_features.

* Note : Since the samples are generated randomly, the outputs will vary slightly for different devices.

# Random Forest in Python – Prediction

```
# Calculating Predictions for the model
```

```
y_pred = rf.predict(X_test)
y_pred_probs = rf.predict_proba(X_test)

cutoff = 0.3
pred_test = np.where(y_pred_probs[:,1] > cutoff, 1, 0)
pred_test
```

```
# Output
```

```
array([0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1])
```

# Random Forest in Python – Confusion Matrix

```
# Confusion Matrix
confusion_matrix(y_test, pred_test, labels=[0, 1])

array([[127,  30],
       [ 17,  36]], dtype=int64)
accuracy_score(y_test, pred_test)
0.7761904761904762
precision_score(y_test, pred_test)
0.5454545454545454
recall_score(y_test, pred_test)
0.6792452830188679
```

- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as ‘Sensitivity’ = true positives / (true positives + false negatives)

```
# Area Under ROC Curve
auc = roc_auc_score(y_test, y_pred_probs[:,1])
print('AUC: %.3f' % auc)
AUC: 0.852
```

# Random Forest in Python – ROC Curve

- **oob_score_** gives out of bag accuracy
- **feature_importances_** gives the feature importances

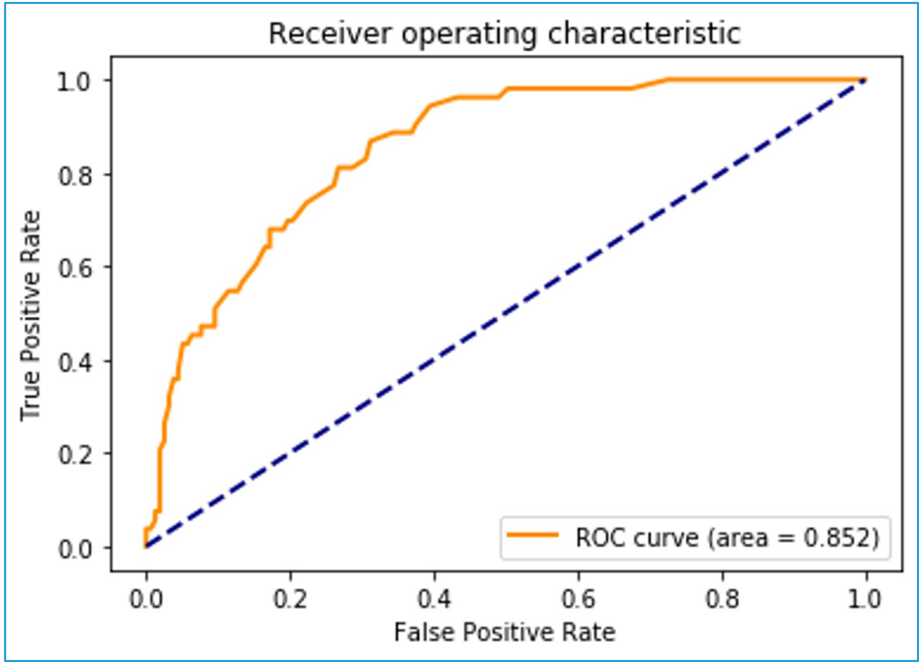
```
# OOB Score
rf.oob_score_
0.753061224489796
rf.feature_importances_
array([0.18827389, 0.14472019, 0.23581877, 0.20153387, 0.18166248,
       0.0233408 , 0.01439653, 0.01025348])
```

```
# ROC Curve
RFfpr, RFtpr, thresholds = roc_curve(y_test, y_pred_probs[:,1])

# plot the roc curve for the model
plt.figure()
lw = 2
plt.plot(RFfpr, RFtpr, color='darkorange',lw=lw, label='ROC curve
(area = %0.3f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.axis('tight')
plt.xlabel('False Positive Rate');plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

# Random Forest in Python – ROC Curve

```
# Output:
```



# Random Forest in Python – Variable Importance

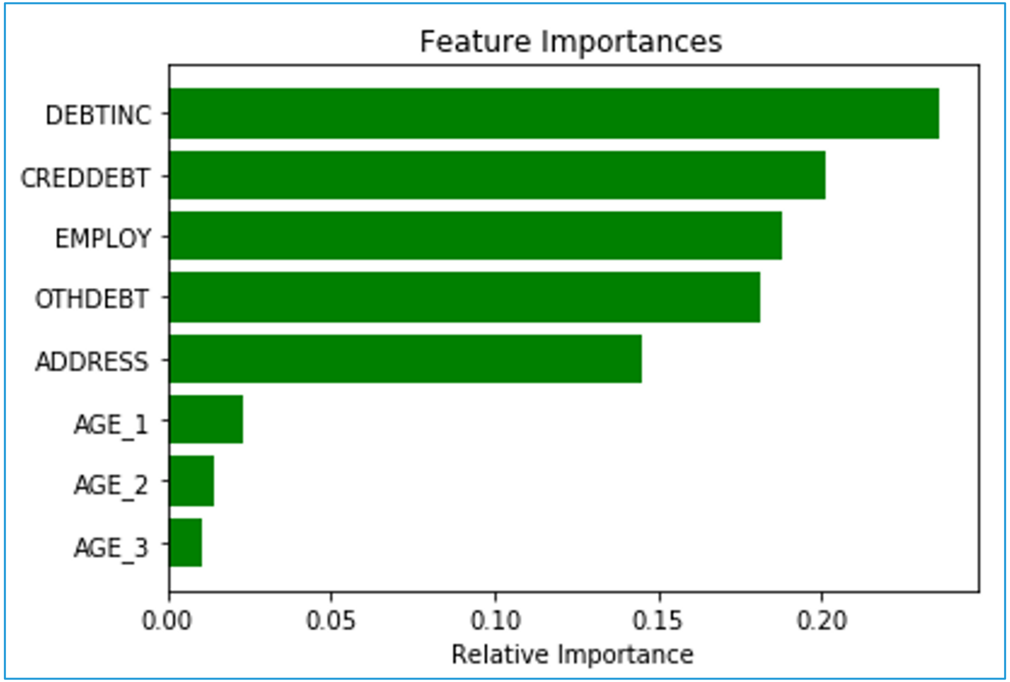
```
# Importance Matrix
features = list(X.columns)
importances = rf.feature_importances_
indices = np.argsort(importances)


plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='g',
align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show();
```

- **argsort()** is used to sort along the given axis, here it being '**importances**'

# Random Forest in Python – Variable Importance

```
# Output
```



# Quick Recap

**Bootstrapping**

- Method for estimating the sampling distribution of an estimator by resampling with replacement from the original sample

**Bagging**

- “Bagging” stands for “Bootstrap Aggregating”
- It is an ensemble method: a method of combining results from multiple resamples

**Random Forest Method**

- Its an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees
- Random forests also work for regression problems
- The method combines Breiman's “Bagging" idea and the random selection of features

**Random Forest in Python**

- `RandomForestClassifier()` in library “**sklearn**” runs random forest analysis
- The output can even generate variable importance and can be used for predictions.