**PROJECT REPORT**

**ON**

**BANK LOAN DEFAULT CASE**

**By**

**TEJAS A C**

**Chapter 1: INTRODUCTION**

* **1.1 Problem Statement**
* **1.2 Data**

**Chapter 2: Data Preprocessing**

* **2.1 Missing Value Analysis**
* **2.2 Distribution of Variables**
* **2.3 Outlier Analysis**
* **2.4 Feature Selection**
* **2.5 Feature Scaling**

**Chapter 3: Modelling**

* **3.1 Logistic Regression**
* **3.2 Decision Trees**
* **3.3 Random Forest**
* **3.4 Gradient Boosting Classifier**

**Chapter 4: Model Evaluation**

* **4.1 Model Selection**

**Appendix**

**Bank Loan Default Case**

**Chapter 1: INTRODUCTION**

**1.1 Problem Statement**

The loan default dataset has 8 variables and 850 records, each record being loan default status for each customer. Each Applicant was rated as “Defaulted” or “Not-Defaulted”. New applicants for loan application can also be evaluated on these 8 predictor variables and classified as a default or non-default based on predictor variables.

Loans default will cause huge loss for the banks, so they pay much attention on this issue and apply various method to detect and predict default behaviors of their customers.

**What is Classification?**

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc.

**1.2 Data**

|  |  |  |  |
| --- | --- | --- | --- |
| Serial No. | Variable Name | Description | Variable Type |
| 1 | Age | Age of each customer | Numerical |
| 2 | Education | Education categories | Categorical |
| 3 | Employment | Employment status - Corresponds to job status and being converted to numeric format | Numerical |
| 4 | Address | Geographic area - Converted to numeric values | Numerical |
| 5 | Income | Gross Income of each customer | Numerical |
| 6 | debtinc | Individual’s debt payment to his or her gross income | Numerical |
| 7 | creddebt | debt-to-credit ratio is a measurement of how much you owe your creditors as a percentage of your available credit (credit limits) | Numerical |
| 8 | otherdebt | Any other debts | Numerical |

**Chapter 2: Data Preprocessing**

Data preprocessing is the major step before training the model because the Real-world data we obtained is incomplete and inconsistent there are lack of behaviors. In some cases, we need to extract the new features from the data we have, this can be done on a clear understanding of business problem and data.

**2.1 Missing Value Analysis:**

In missing value analysis, we find if there are any missing cells present in the data or not. If there is any data then we need to fill that using various techniques.

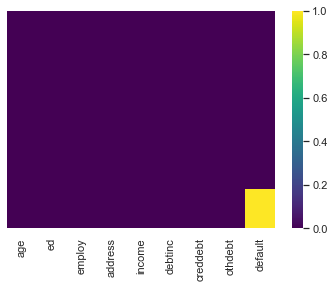
We find missing values in the data using

* credit\_data.isnull().sum()

**Missing Values with Percentage:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Variables | Counts | Percentage |
| 1 | default | 150 | 17.647059 |
| 2 | age | 0 | 0.0 |
| 3 | ed | 0 | 0.0 |
| 4 | employ | 0 | 0.0 |
| 5 | address | 0 | 0.0 |
| 6 | income | 0 | 0.0 |
| 7 | debtinc | 0 | 0.0 |
| 8 | creddebt | 0 | 0.0 |
| 9 | othdebt | 0 | 0.0 |

**Heat Map for Missing Values:**

****

Note: The dataset has 150 missing labeled values in the dependent variable column. Those 150 observations can be separated from the dataset and can be named as test data. After choosing the best model, these observations can be evaluated and classified as “default” or “not default”

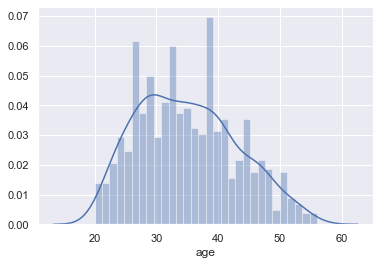
So here we have missing values present in one variable, we will impute those missing values by giving a unique value 2.

**2.2 Distribution of Variables**

Let us look few of the variable’s distribution

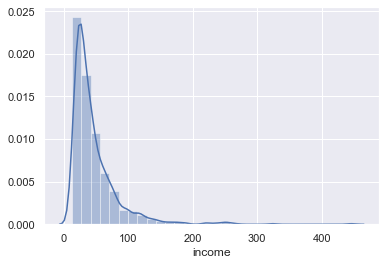
Let us see if we can find some interesting trend in the data distribution, especially on age, income, othdebt, creddebt, debtinc.

Data Distribution of age variable:

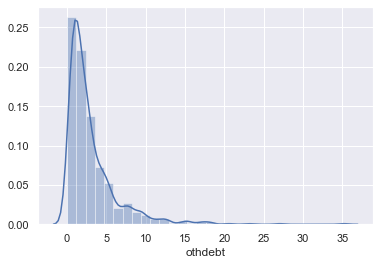


By seeing the above graph, we can say the age variable is normally distributed.

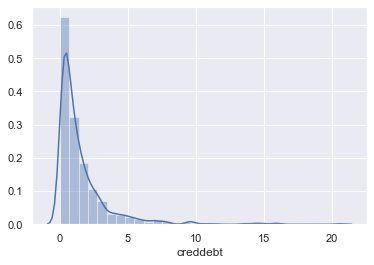
Data Distribution of income variable:



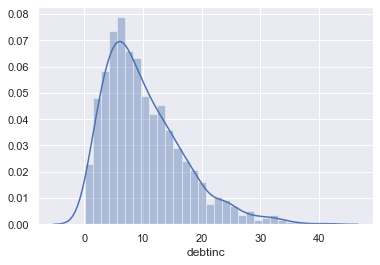
Data Distribution of othdebt variable:



Data Distribution of creddebt variable:



Data Distribution of debtinc variable:

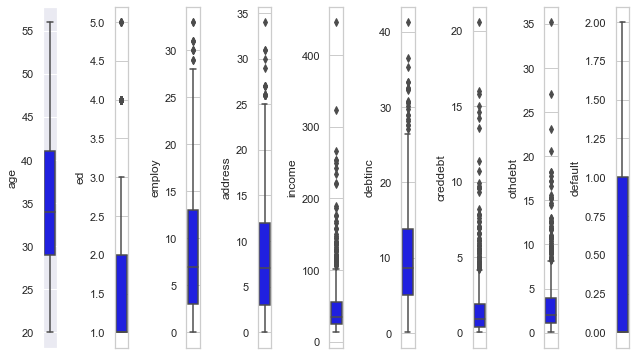


All the mentioned graphs above, are Right skewed or has a positive skew distribution.

**2.3 Outlier Analysis**

Outlier analysis is done to handle all inconsistent observations present in given dataset.

**Outlier Plot:**



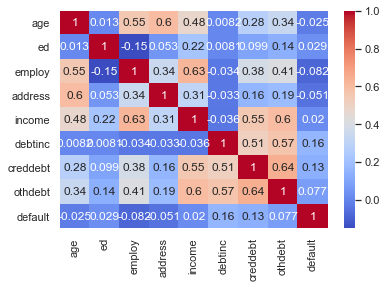
The above figure is a subplot which shows all the variables with outliers.

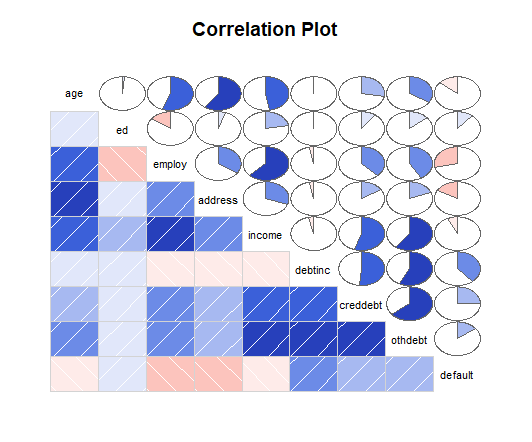
**Proposed course of action:** All rows will be considered for analysis.

**2.4 Feature Selection**

Feature selection is very important for modelling the dataset. Every dataset has good and unwanted features. The unwanted features would affect on the performance of the model, so we need to delete or keep those features on various techniques. We have to select best features by using ANOVA, Chi-Square test and Correlation Matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

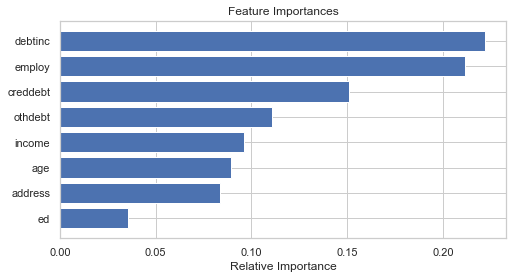
Plotting Correlation Heatmap:





* The above Correlation Matrix tells there are no two variables with High Correlation.
* Heat map shows that there is no multi-collinearity in data.

The below graph tells us the important features in order to build the model.



**2.5 Feature Scaling**

Feature scaling includes two functions normalization and standardization. It is done reduce unwanted variation either within or between variables and to bring all of the variables into proportion with one another.

**Confusion Matrix**

A confusion matrix is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.



Here,

• Class 1: Positive

• Class 2: Negative

**Definition of the Terms:**

• Positive (P): Observation is positive (for example: is an apple).

• Negative (N): Observation is not positive (for example: is not an apple).

• True Positive (TP): Observation is positive, and is predicted to be positive.

• False Negative (FN): Observation is positive, but is predicted negative.

• True Negative (TN): Observation is negative, and is predicted to be negative.

• False Positive (FP): Observation is negative, but is predicted positive.

**Classification Rate/Accuracy:**

Classification Rate or Accuracy is given by the relation:



However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

**ROC CURVES and AUC:**

* **Receiver operating characteristic (ROC) curve:** plots the true positive rate (TPR) versus the false positive rate (FPR) as a function of the model’s threshold for classifying a positive.
* **Area under the curve (AUC):** metric to calculate the overall performance of a classification model based on area under the ROC curve.

**Chapter 3: Modelling**

**Preparing dataset for Machine Learning Algorithm:**

In this case each Applicant was rated as “Defaulted” or “Not-Defaulted”. New applicants for loan application can also be evaluated on the 8 predictor variables and classified as a default or non-default based on predictor variables. Model having less error rate and more accuracy will be our final model. In these we have divided the dataset into train and test part using random sampling. For this model we have divided the dataset into train and test part using random sampling Where train contains 80% data of data set and test contains 20% data.

**3.1 Logistic Regression**

Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

**Null deviance** shows how well the response variable is predicted by a model that includes only the intercept (grand mean).

**Residual deviance** shows how well the response variable is predicted with the inclusion of independent variables.

**The Akaike information criterion (AIC)** is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.

Null deviance: 618.91 on 544 degrees of freedom

Residual deviance: 421.08 on 540 degrees of freedom

AIC: 431.08

**Code:**

from sklearn.linear\_model import LogisticRegression

clf\_logis = LogisticRegression()

clf\_logis.fit(Xtrain, ytrain)

y\_log\_predict = clf\_logis.predict(Xtest)

**Classification Report:**

Print(classification\_report(ytest, y\_preds))

precision recall f1-score support

0 0.88 0.94 0.91 132

1 0.76 0.60 0.68 43

accuracy 0.86 175

macro avg 0.82 0.77 0.79 175

weighted avg 0.85 0.86 0.85 175

**Defaulted:** 43

**Non-defaulted:** 132

**Confusion matrix:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 124 | 8 |
| 1 | 17 | 26 |

**Accuracy: 86%**

**3.2 Decision Trees**

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

**Code:**

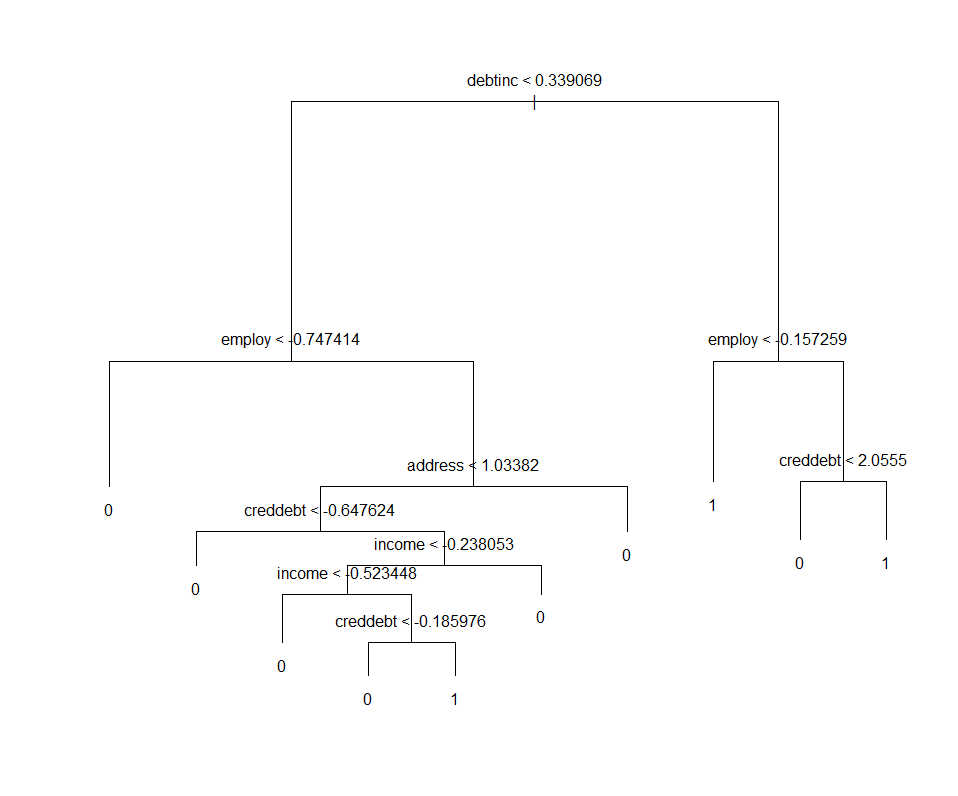
from sklearn.tree import DecisionTreeClassifier

clf\_gini = DecisionTreeClassifier(criterion = 'gini', random\_state = 10, max\_depth = 5, min\_samples\_leaf = 7)

clf\_gini.fit(Xtrain, ytrain)

dt\_predict = clf\_gini.predict(Xtest)

Looking at thebelow figure tree, here decision tree is using only one predictors variable to predict the model, which is not very impressive here the model is over fitted and biased towards only one predictor i.e. “Employ”.



**Confusion matrix:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 115 | 17 |
| 1 | 21 | 22 |

**Accuracy: 78%**

**3.3 Random Forest**

Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

Decision tress train over a single training set only. Decision trees take into account each and every variable and every observation in the training set. While decision trees are very fast (in terms of computational speed), they generally overfit the data and perform poorly on test sets.

**Code:**

from sklearn.ensemble import RandomForestClassifier

clf\_rf = RandomForestClassifier(random\_state = 42)

clf\_rf.fit(Xtrain, ytrain)

y\_predict\_rf = clf\_rf.predict(Xtest)

**Hyper Parameter Tuning:**

rf\_clf\_tuning = GridSearchCV(clf\_rf, param\_grid, cv = 5)

rf\_clf\_tuning.fit(Xtrain, ytrain)

best\_rf\_clf = RandomForestClassifier(criterion = 'gini', bootstrap = True, max\_features = 'log2', min\_samples\_leaf = 5, n\_estimators = 200)

best\_rf\_clf.fit(Xtrain, ytrain)

y\_best\_rf\_preds = best\_rf\_clf.predict(Xtest)

The hyper parameters that were tuned are:

**n\_estimtors:** This represents the maximum number of trees to build before aggregating all of these trees to form a single predictor. Generally, higher number of trees give better results.

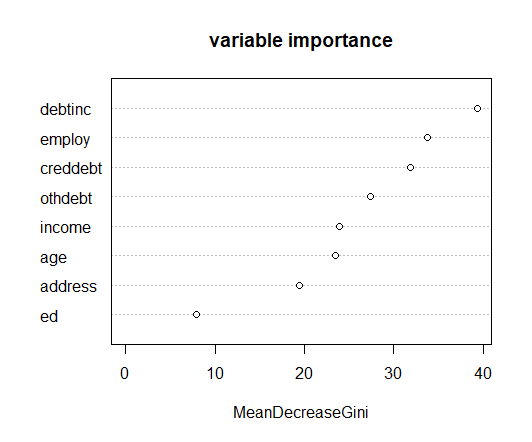
**max\_features:** Represents the maximum number of features to consider for a particular tree. Usually, the square root of the total number of features is chosen.

**criterion:** We choose between ‘gini’ and ‘entropy.

**bootstrap:** The default is true. If set to true (default), the random samples are generated by sampling with replacement.

**min\_sample-leaf:** A leaf represent the end node of a decision tree. Smaller leaves tend to produce ‘noisier’ models.

The below figure shows the important variables to build the Random Forest Model.



**Confusion matrix:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 122 | 10 |
| 1 | 23 | 20 |

**Accuracy: 81%**

**3.4 Gradient Boosting Classifier**

**Gradient boosting classifiers** are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing **gradient boosting**.

**Code:**

from sklearn.ensemble import GradientBoostingClassifier

clf\_xgb = GradientBoostingClassifier(n\_estimators=200)

clf\_xgb.fit(Xtrain, ytrain)

y\_predict\_xgb = clf\_xgb.predict(Xtest)

**Confusion matrix:**

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 118 | 14 |
| 1 | 24 | 19 |

**Accuracy: 78%**

**Model Evaluation**

**4.1 Model Selection**

Choosing the best Classifier:

As our dataset is imbalanced, only accuracy cannot judge the performance of the model. A plot of the precision vs. recall curves is illustrated below. Greater the area under the curve, the better the model. Logistic model has higher accuracy both in R and Python.

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds. Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds. ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

**Code:**

Fig, ax = plt.subplots(figsize=(8,8))

plt.plot(recall\_lg, precision\_lg)

plt.plot(recall\_dc, precision\_dc)

plt.plot(recall\_rf, precision\_rf)

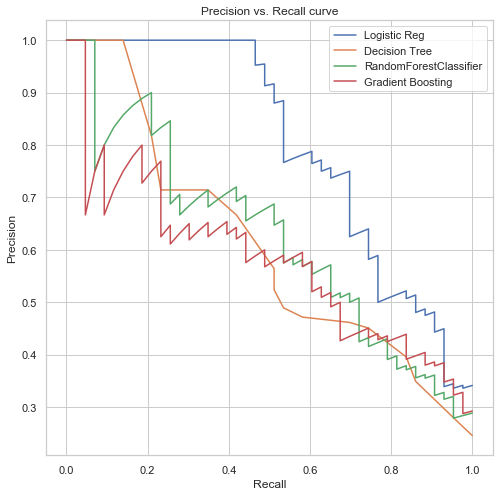
plt.plot(recall\_xgb, precision\_xgb)

plt.legend(('Logistic Reg', 'Decision Tree', 'RandomForestClassifier', 'Gradient Boosting'))

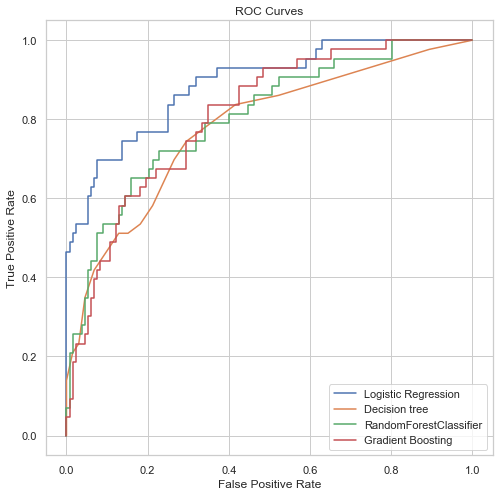
plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision vs. Recall curve')



Once again, the logistic regression classifier has the max. Area under the curve. Next, we will plot the ROC curves. These curves explain the tradeoff between the true positive rate and the false positive rate. Just as the Precision vs. Recall curves, greater areas under the curve represent better models. This plot has been illustrated below.



**Code:**

Areas\_ROC\_logistic = roc\_auc\_score(ytest, p\_clf\_logis\_ba[:, 1])

Areas\_ROC\_decision = roc\_auc\_score(ytest, dt\_predict1)

Areas\_ROC\_randomforest = roc\_auc\_score(ytest, y\_best\_proba)

Areas\_ROC\_xgb = roc\_auc\_score(ytest, y\_predict\_xgb1)

print(Areas\_ROC\_logistic)

print(Areas\_ROC\_decision)

print(Areas\_ROC\_randomforest)

print(Areas\_ROC\_xgb)

The logistic regression classifier has the maximum area under the Roc curve. The random forest and gradient boosting has almost same area under the curve.

Upon all considerations of evaluation matrix, Logistic regression performs better.

**CONCLUSION**

|  |  |  |
| --- | --- | --- |
|  | **Classification Algorithm** | **Accuracy** |
| **1** | **Logistic Regression** | **86** |
| **2** | **Decision Trees** | **78** |
| **3** | **Random Forest** | **81** |
| **4** | **Gradient Boosting** | **78** |

The conclusion is “the **Logistic Regression** is performing well for the given bank loan data case”.

**Appendix**

**Python code:**

import os

os.chdir("G:/Tejas/Project/Project\_2")

os.getcwd()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

from matplotlib import pyplot

sns.set()

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make\_pipeline

from sklearn import svm

from sklearn.preprocessing import scale

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import auc

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

from sklearn.decomposition import PCA

from sklearn.ensemble import GradientBoostingClassifier

loan\_default = pd.read\_csv("bank\_loan.csv")

loan\_default.head()

loan\_default.info()

loan\_default.shape

loan\_default.describe()

#Find the total number of missing values in the dataframe

print("Missing values: ", loan\_default.isnull().sum().values.sum(), "\n")

#Printing total number of unique values in the dataframe

print("Unique Values:\n", loan\_default.nunique())

############ Univariate Analysis and Bivariate Analysis ##########################

# Analysis for single variable in the dataset and relation between 2 variables.

sns.distplot(loan\_default["age"], kde = True , bins = 30)

sns.distplot(loan\_default["income"], kde = True , bins = 30)

sns.distplot(loan\_default["othdebt"], kde = True , bins = 30)

sns.distplot(loan\_default["creddebt"], kde = True , bins = 30)

sns.distplot(loan\_default["debtinc"], kde = True , bins = 30)

sns.countplot( x = "default" , data = loan\_default)

sns.countplot( x = "ed" , data = loan\_default, hue = "default")

loan\_default.isnull().sum()

#Create dataframe with missing percentage

missing\_val = pd.DataFrame(loan\_default.isnull().sum())

#Reset index

missing\_val = missing\_val.reset\_index()

#Rename variable

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_Percentage'})

#Calculate percentage

missing\_val['Missing\_Percentage'] = (missing\_val['Missing\_Percentage']/len(loan\_default))\*100

#descending order

missing\_val = missing\_val.sort\_values('Missing\_Percentage', ascending = False).reset\_index(drop = True)

#save output results

missing\_val.to\_csv("Perc\_of\_MV.csv", index = False)

missing\_val.head(9)

default = loan\_default[loan\_default['default'].isnull()]

default.head()

default.tail()

sns.heatmap(loan\_default.isnull(), yticklabels = False, cmap = "viridis")

loan\_default.isnull().sum()

loan\_default.default = loan\_default.default.fillna(2)

loan\_default.default = loan\_default.default.astype(np.int64)

loan\_default.isnull().sum()

loan\_default.head()

loan\_default.tail()

l = loan\_default.columns.values

number\_of\_columns = 9

number\_of\_rows = len(l)-1/number\_of\_columns

plt.figure(figsize = (number\_of\_columns, 5\*number\_of\_rows))

for i in range(0, len(l)):

plt.subplot(number\_of\_rows + 1, number\_of\_columns, i+1)

sns.set\_style('whitegrid')

sns.boxplot(loan\_default[l[i]], color = 'blue', orient = 'v')

plt.tight\_layout()

loan\_default.info()

df1\_cor = loan\_default.corr()

sns.heatmap(df1\_cor, annot = True, cmap = "coolwarm")

sns.countplot(loan\_default.default)

sns.barplot(x = "default", y = "income", data = loan\_default)

#### selecting all missing values from dataset and we will predict those default case with best accurate

train = loan\_default.loc[loan\_default['default'] != 2]

print(train.head())

print(train.tail())

train.default.unique()

test = loan\_default.loc[loan\_default.default == 2]

test = test.iloc[:, 0:8]

print(test.head(2))

print(test.tail(2))

X = train[['age', 'ed', 'employ', 'address', 'income', 'debtinc', 'creddebt', 'othdebt']]

y = train['default']

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, random\_state = 42, test\_size = 0.25)

clf\_logis = LogisticRegression()

clf\_logis.fit(Xtrain, ytrain)

y\_log\_predict = clf\_logis.predict(Xtest)

accuracy\_score(ytest, y\_log\_predict)

C\_space = np.array([0.0001, 0.001, 0.1, 1])

param\_grid = {'C' : C\_space}

clf\_logis\_tuning = GridSearchCV(clf\_logis, param\_grid, cv=5)

clf\_logis\_tuning.fit(Xtrain, ytrain)

print("Tuned Logistic Regression Parameters: {}".format(clf\_logis\_tuning.best\_params\_))

print("Best score is {}".format(clf\_logis\_tuning.best\_score\_))

clf\_logis = LogisticRegression(C = 1.0)

clf\_logis.fit(Xtrain, ytrain)

y\_preds = clf\_logis.predict(Xtest)

p\_clf\_logis\_ba = clf\_logis.predict\_proba(Xtest)

accuracy\_score(ytest, y\_preds)

print(classification\_report(ytest, y\_preds))

precision\_lg, recall\_lg, thresholds\_lg = precision\_recall\_curve(ytest, p\_clf\_logis\_ba[:, 1])

fpr\_lg, tpr\_lg, thresholds\_lg = roc\_curve(ytest, p\_clf\_logis\_ba[:, 1])

#build confusion matrix

CM = pd.crosstab(ytest, y\_preds)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#accuracy\_score(ytest, y\_preds)\*100

print(((TP+TN)\*100)/(TP+TN+FP+FN))

#False Negative rate

print((FN\*100)/(FN+TP))

print("Defaulted", sum(y\_preds!=0))

print("Non-defaulted ", sum(y\_preds==0))

CM

from sklearn.tree import DecisionTreeClassifier

clf\_gini = DecisionTreeClassifier(criterion = 'gini', random\_state = 10, max\_depth = 5, min\_samples\_leaf = 7)

clf\_gini.fit(Xtrain, ytrain)

dt\_predict = clf\_gini.predict(Xtest)

dt\_predict1 = clf\_gini.predict\_proba(Xtest)[:, 1]

accuracy\_score(ytest, dt\_predict)

print(classification\_report(ytest, dt\_predict))

precision\_dc, recall\_dc, thresholds\_dc = precision\_recall\_curve(ytest, dt\_predict1)

fpr\_dc, tpr\_dc, thresholds\_dc = roc\_curve(ytest, dt\_predict1)

#build confusion matrix

CM = pd.crosstab(ytest, dt\_predict)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#accuracy\_score(ytest, dt\_pred)\*100

print(((TP+TN)\*100)/(TP+TN+FP+FN))

#False Negative rate

print((FN\*100)/(FN+TP))

print("Defaulted", sum(dt\_predict!=0))

print("Non-defaulted ", sum(dt\_predict==0))

CM

clf\_rf = RandomForestClassifier(random\_state = 42)

clf\_rf.fit(Xtrain, ytrain)

y\_predict\_rf = clf\_rf.predict(Xtest)

cv\_score = cross\_val\_score(clf\_rf, Xtrain, ytrain, cv = 5)

print("Average 5-fold CV Score: {}".format(np.mean(cv\_score)))

cv\_score = cross\_val\_score(clf\_rf, Xtrain, ytrain, cv = 5, scoring = 'roc\_auc')

print("Average 5-fold CV Score using ROC\_AUC: {}".format(np.mean(cv\_score)))

accuracy\_score(ytest, y\_predict\_rf)

n\_space = np.array([5, 6, 10, 12, 15, 50, 100, 200, 500])

criterion\_vals = ['gini', 'entropy']

max\_features\_vals = ['auto', 'sqrt', 'log2']

min\_samples\_leaf\_sp = [1, 5, 10, 25, 50]

bootstrap\_sp = [True, False]

param\_grid = {'n\_estimators': n\_space, 'criterion' : criterion\_vals,

'max\_features':max\_features\_vals, 'min\_samples\_leaf': min\_samples\_leaf\_sp,

'bootstrap': bootstrap\_sp}

rf\_clf\_tuning = GridSearchCV(clf\_rf, param\_grid, cv = 5)

rf\_clf\_tuning.fit(Xtrain, ytrain)

print("Tuned RF Parameters: {}".format(rf\_clf\_tuning.best\_params\_))

print("Best score is {}".format(rf\_clf\_tuning.best\_score\_))

best\_rf\_clf = RandomForestClassifier(criterion = 'gini', bootstrap = True,

max\_features = 'log2', min\_samples\_leaf = 5, n\_estimators = 200)

best\_rf\_clf.fit(Xtrain, ytrain)

y\_best\_rf\_preds = best\_rf\_clf.predict(Xtest)

cv\_score = cross\_val\_score(best\_rf\_clf, Xtrain, ytrain, cv = 5)

print("Average 5-fold CV Score: {}".format(np.mean(cv\_score)))

accuracy\_score(ytest, y\_best\_rf\_preds)

y\_best\_proba = best\_rf\_clf.predict\_proba(Xtest)[:, 1]

print(classification\_report(ytest, (y\_best\_proba > 0.5).astype(int)))

#build confusion matrix

CM = pd.crosstab(ytest, y\_best\_rf\_preds)

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

print(((TP+TN)\*100)/(TP+TN+FP+FN))

#False Negative rate

print((FN\*100)/(FN+TP))

print("Defaulted", sum(y\_best\_rf\_preds!=0))

print("Non-defaulted ", sum(y\_best\_rf\_preds==0))

CM

fig, ax = plt.subplots(figsize=(8,4))

features = train.columns

importances = best\_rf\_clf.feature\_importances\_

indices = np.argsort(importances)

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color='b', align='center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()

y\_rf\_probs = clf\_rf.predict\_proba(Xtest)

precision\_rf, recall\_rf, thresholds\_rf = precision\_recall\_curve(ytest, y\_best\_proba)

fpr\_rf, tpr\_rf, thresholds\_rf = roc\_curve(ytest, y\_best\_proba)

from sklearn.ensemble import GradientBoostingClassifier

clf\_xgb = GradientBoostingClassifier(n\_estimators=200)

clf\_xgb.fit(Xtrain, ytrain)

y\_predict\_xgb = clf\_xgb.predict(Xtest)

accuracy\_score(ytest, y\_predict\_xgb)

y\_predict\_xgb1 = clf\_xgb.predict\_proba(Xtest)[:, 1]

print(classification\_report(ytest, y\_predict\_xgb))

precision\_xgb, recall\_xgb, thresholds\_xgb = precision\_recall\_curve(ytest, y\_predict\_xgb1)

fpr\_xgb, tpr\_xgb, thresholds\_xgb = roc\_curve(ytest, y\_predict\_xgb1)

#build confusion matrix

CM = pd.crosstab(ytest, y\_predict\_xgb)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#accuracy\_score(ytest, dt\_pred)\*100

print(((TP+TN)\*100)/(TP+TN+FP+FN))

#False Negative rate

print((FN\*100)/(FN+TP))

print("Defaulted", sum(y\_predict\_xgb!=0))

print("Non-defaulted ", sum(y\_predict\_xgb==0))

CM

fig, ax = plt.subplots(figsize=(8,8))

plt.plot(recall\_lg, precision\_lg)

plt.plot(recall\_dc, precision\_dc)

plt.plot(recall\_rf, precision\_rf)

plt.plot(recall\_xgb, precision\_xgb)

plt.legend(('Logistic Reg', 'Decision Tree', 'RandomForestClassifier', 'Gradient Boosting'))

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision vs. Recall curve')

area\_log\_reg = auc(recall\_lg, precision\_lg)

print(area\_log\_reg)

area\_dc = auc(recall\_dc, precision\_dc)

print(area\_dc)

area\_rf = auc(recall\_rf, precision\_rf)

print(area\_rf)

area\_xgb = auc(recall\_xgb, precision\_xgb)

print(area\_xgb)

fig, ax = plt.subplots(figsize=(8,8))

plt.plot(fpr\_lg, tpr\_lg)

plt.plot(fpr\_dc, tpr\_dc)

plt.plot(fpr\_rf, tpr\_rf)

plt.plot(fpr\_xgb, tpr\_xgb)

plt.legend(('Logistic Regression' , 'Decision tree', 'RandomForestClassifier', 'Gradient Boosting'))

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves')

Areas\_ROC\_logistic = roc\_auc\_score(ytest, p\_clf\_logis\_ba[:, 1])

Areas\_ROC\_decision = roc\_auc\_score(ytest, dt\_predict1)

Areas\_ROC\_randomforest = roc\_auc\_score(ytest, y\_best\_proba)

Areas\_ROC\_xgb = roc\_auc\_score(ytest, y\_predict\_xgb1)

print(Areas\_ROC\_logistic)

print(Areas\_ROC\_decision)

print(Areas\_ROC\_randomforest)

print(Areas\_ROC\_xgb)

**R code:**

rm(list = ls())

setwd("G:Tejas/R\_practice/Project2\_prac")

getwd()

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees','fastDummies', 'psych')

#install.packages(x)

lapply(x, require, character.only = TRUE)

#load packages

library("dplyr")

library("plyr")

library("ggplot2")

library("data.table")

library("GGally")

library(tidyr)

#load data into r

loan\_default = read.csv("bank\_loan.csv", header = T, na.strings = c(" ","","NA"))

View(loan\_default)

# Summarizing data

# dim helps to see no of observations and variables in the dataset.

# dataset contains 850 obs. of 9 variables

dim(loan\_default)

### TAble helps us to see how many for default and non default

table(loan\_default$default)

### Unique Values ###

unique(loan\_default$default)

head(loan\_default)

tail(loan\_default)

colnames(loan\_default)

str(loan\_default)

loan\_default$default = as.factor(loan\_default$default)

str(loan\_default)

describe(loan\_default)

############## Checking the distribution of the age ###

#### age is normally distributed.

plot(density(loan\_default$age))

hist(loan\_default$age, main = " age histogram " , xlab = 'age', ylab = "freq")

#####

ggplot(loan\_default) +

geom\_bar(aes(x=ed),fill="grey")

################ Missing Value Analysis ###################

sum(is.na(loan\_default$default))

missing\_val = data.frame(apply(loan\_default, 2, function(x){sum(is.na(x))}))

missing\_val

View(missing\_val)

missing\_val$columns = row.names(missing\_val)

row.names(missing\_val) = NULL

names(missing\_val)[1] = "Count"

names(missing\_val)[2] = "Variables"

missing\_val$Missing\_Percentage = (missing\_val$Count/nrow(loan\_default))\*100

missing\_val = missing\_val[order(-missing\_val$Missing\_Percentage),]

missing\_val = missing\_val[, c(2, 1)]

missing\_val$Missing\_Percentage = (missing\_val$Count/nrow(loan\_default))\*100

sum(is.na(loan\_default))

#Write output result into disk

write.csv(missing\_val, "Missing\_perc.csv", row.names = F)

##################### outlier check #####################

##### All the given variables has outliers. I assume that the income and related

# debt values are dependent on the observer. more the education more the values

### might be.

boxplot(loan\_default$age, main = " outlier check of age variable", ylab = "age", col = 5)

boxplot(loan\_default$income, main = " outlier check of income variable", ylab = "income", col = 5)

boxplot(loan\_default$creddebt, main = " outlier check of creddebt variable", ylab = "creddebt", col = 5)

boxplot(loan\_default$othdebt, main = " outlier check of othdebt variable", ylab = "othdebt", col = 5)

## Correlation Plot

numeric\_index = sapply(loan\_default, is.numeric)

corrgram(loan\_default[,numeric\_index], order = F,

upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

ggcorr(loan\_default, label = T, label\_size = 3, label\_round = 2, hjust = 1, size = 3, color = "royalblue", layout.exp = 5, low = "dodgerblue", mid = "gray95", high = "red2", name = "Correlation Heatmap")

###### standardisation###########

loan\_default\_scaled = loan\_default

cnames = colnames(loan\_default\_scaled)

for (i in cnames) {

print(i)

loan\_default\_scaled[,i] = (loan\_default\_scaled[,i]-mean(loan\_default\_scaled[,i]))/sd(loan\_default\_scaled[,i])

}

###################### separating the labeled and not labeled observations.

##### last 150 observations are not labled. Those observations to be

### predicted after the model building. we will predict those values after

### chossing best model.

loan\_train = loan\_default\_scaled[1:700,1:9]

dim(loan\_train)

sum(is.na(loan\_train))

sum(is.na(loan\_default$default))

loan\_test = loan\_default\_scaled[701:850, 1:8]

dim(loan\_test)

############# Model Building##############

library(caTools)

#Splitting into training and testing data

set.seed(123)

sample = sample.split(loan\_train, SplitRatio = 0.8)

sample

training = subset(loan\_train, sample==TRUE)

str(training)

testing = subset(loan\_train, sample==FALSE)

str(testing)

######################logistic regression #########################

model = glm(default~.,training, family = "binomial")

summary(model)

#########################model with high important variables############

model6 = glm(default~creddebt+debtinc+address+employ, training, family = "binomial")

summary(model6)

res = predict(model6, testing, type = "response")

range(res)

confusion\_matrix = table(Actualvalue=testing$default, predictedvalue=res>0.5)

print(confusion\_matrix)

accuracy = (104 + 20)/(104 + 20 + 24 + 7)

print(accuracy)

####### Threshold Evaluation ################

### ROC CURVE ##########

######AUC####

library(ROCR)

pred\_log = prediction(res, testing$default)

acc = performance(pred\_log, "acc")

plot(acc)

roc\_curve = performance(pred\_log, "tpr" , "fpr")

plot(roc\_curve)

plot(roc\_curve , colorize = T, print.cutoffs.at=seq(0.1,by=0.1))

###### using threshold value of 0.4 we can incraese the true positive rate

confusion\_matrix = table(Actualvalue = testing$default, predictedvalue = res>0.4)

print(confusion\_matrix)

accuracy = (93 + 25)/(93 + 25 + 19 + 18)

print(accuracy)

auc = performance(pred\_log, "auc")

auc

#accuracy = 76%

#AUC = 0.82

#############Precision recall curve ##############

library(PRROC)

PRC\_curve = performance(pred\_log, "prec" , "rec")

plot(PRC\_curve, colorize = T)

############################## DEcision tree###############

library(tree)

deci\_model = tree(default~., data = training)

summary(deci\_model)

### plotting

plot(deci\_model)

text(deci\_model,pretty = 0)

#### prediction

deci\_pred = predict(deci\_model, testing, type = "class")

confusion\_matrix = table(Actualvalue = testing$default, predictedvalue = deci\_pred)

print(confusion\_matrix)

accuracy = (96 + 19)/(96 + 19 + 25 + 15)

accuracy

#### cross validation

cv.deci\_model = cv.tree(deci\_model, FUN = prune.misclass)

cv.deci\_model

plot(cv.deci\_model)

####pruning

prune.deci\_model = prune.misclass(deci\_model, best = 10)

plot(prune.deci\_model)

text(prune.deci\_model)

#### prediction of values again

deci\_predict\_1 = predict(prune.deci\_model, testing, type = "class")

Confusion\_matrix\_1 = table(testing$default, deci\_predict\_1)

print(Confusion\_matrix\_1)

# accuracy = 0.74 # precision = 0.54 # recall = 0.43

accuracy = (96 + 19)/(96 + 19 + 25 + 15)

accuracy

##############################Random Forest#################

#####random forest 1

library(randomForest)

rf = randomForest(default~., data = training)

print(rf)

## prediction

rf\_predict = predict(rf,testing)

confusion\_matrix = table(Actualvalue = testing$default, predictedvalue = rf\_predict)

print(confusion\_matrix)

accuracy = (101 + 17)/(101 + 17 + 27 + 10)

print(accuracy)

## tune mtry

tuneRF(training[,-9], training[,9],stepfactor = 0.5,

plot = TRUE , ntreeTry = 1000,

trace = TRUE ,

improve = 0.05)

rf1 = randomForest(default~.,data = training, ntree = 1000, mtry = 2)

rf1

# predict

rf\_predict1 = predict(rf1,testing)

confusion\_matrix1 = table(Actualvalue=testing$default, predictedvalue=rf\_predict1)

print(confusion\_matrix1)

accuracy = (101 + 18)/(101 + 18 + 26 + 10)

print(accuracy)

# no. of nodes for the trees

hist(treesize(rf1),main = " no. of nodes for the trees", col = "green")

# variable importance

varImpPlot(rf1,

sort = T,

main = "variable importance")

importance(rf1)

varUsed(rf1)

############## we will build random forest by taking only max meandecreaseGini

### considering debtinc, employ, creddebt, othdeb, income.

### build model

rf\_final = randomForest(default~debtinc+employ+creddebt+othdebt+income ,

data = training,

ntree = 1000, mtry = 2)

rf\_final

# prediction

rf\_predict\_final = predict(rf\_final,testing)

confusion\_matrix\_f = table(Actualvalue=testing$default, predictedvalue=rf\_predict\_final)

print(confusion\_matrix\_f)

accuracy = (100 + 14)/(100 + 14 + 30 + 11)

print(accuracy)

# accuracy = 73.5

# precision = 0.55

# recall = 0.340

#### we cannot decide the perfomance of model only based on the accuracy

# we need to have a good tradeoff between precision and recall.

# logistic model has 80.0 % accuracy with good trade-off b/t prec and recall.

##### Conclusion ======= logistic model is the best suited model on this dataset.

#### predicting for the test data.

res = predict(model6, loan\_test, type = "response")

range(res)