LOAN ELIGIBILTY PREDICTION USING MACHINE LEARNING TECHNIQUES

CONTENTS

1.Introduction:

1.1Overview:

Loans are the core business of banks. The main profit comes directly from the loan's interest. The loan companies grant a loan after an intensive process of verification and validation. However, they still don't have assurance if the applicant is able to repay the loan with no difficulties.

Distribution of the loans is the core business part of almost every banks. The main portion the bank's assets is directly came from the profit earned from the loans distributed by the banks. The prime objective in banking environment is to invest their assets in safe hands where it is.

1.2 Purpose:

We can predict if the client can pay the loan amount taken by him/her from the bank or not based on past history and data of the clients. Result against particular Loan Id can be send to various department of banks so that they can take appropriate action on application.

This is a classification problem where we have to predict whether a loan will be approved or not. Specifically, it is a binary classification problem where we have to predict either one of the two classes given i.e. approved (Y) or not approved (N). Another way to frame the problem is to predict whether the loan will likely to approved or not based on data.

In this project, We suggest a solution to this problem by using algorithms in Machine Learning using Java Language.

2.Literature Survey:

2.1 Existing Problem:

The basic approach to predict is done by using Logistic Regression algorithm which depicts the output.But it takes lot of time to train the model and testing the model.

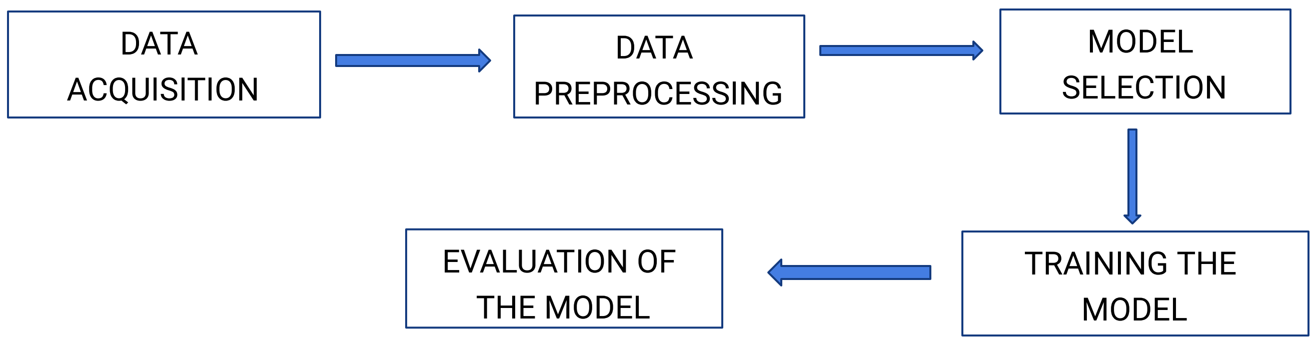
2.2 Proposed Solution:

We use Naive Bayes algorithm and Random Forest algorithm to build the model instead of using conventional Logistic Regression model.

The proposed model takes less time to train the model and yield better results.

3.Therotical Analysis:

3.1 Block Diagram:



3.2 Software designing:

- WEKA 3.8
- JAVA 16.0
- MAVEN
- ECLIPSE IDE
- VISUAL STUDIO CODE

4.Experimental Investigations:

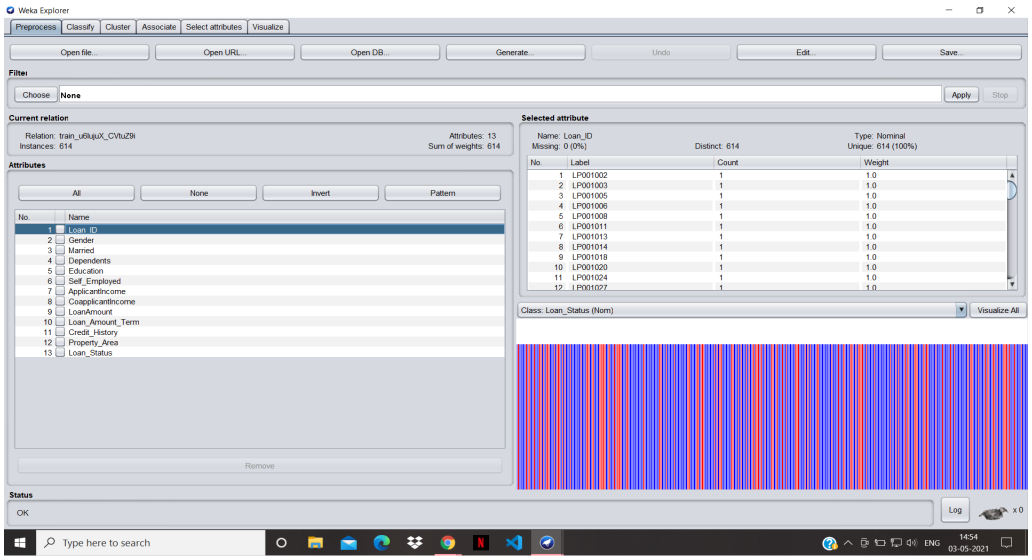
We have 12 independent variables and 1 target variable, i.e. Loan_Status in the training dataset.Loan_status depends on independent variables.The labels are:

1.Loan_ID
2.Gender
3.Married
4.Dependents
5.Education
6.Self_Employed
7.ApplicantIncome
8.CoapplicantIncome
9.LoanAmount

10.Loan_Amount_Term
11.Credit_History
12.Property_Area
13.Loan_Status

- Categorical features: These features have categories (Gender, Married, Self_Employed, Credit_History, Loan_Status)
- Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property_Area)
- Numerical features: These features have numerical values (ApplicantIncome, Co-applicantIncome, LoanAmount, Loan_Amount_Term)

There are total of 614 rows and 13 columns in the dataset.



Filter: None

Current relation
Relation: train_u6lujuX_CVtuZ9i
Instances: 614
Attributes: 13
Sum of weights: 614

Attributes

No.	Name
1	Loan_ID
2	Gender
3	Married
4	Dependents
5	Education
6	Self_Employed
7	ApplicantIncome
8	CoapplicantIncome
9	LoanAmount
10	Loan_Amount_Term
11	Credit_History
12	Property_Area
13	Loan_Status

Selected attribute
Name: Loan_ID
Missing: 0 (0%)
Distinct: 614
Type: Nominal
Unique: 614 (100%)

No.	Label	Count	Weight
1	LP001002	1	1.0
2	LP001003	1	1.0
3	LP001005	1	1.0
4	LP001006	1	1.0
5	LP001008	1	1.0
6	LP001011	1	1.0
7	LP001013	1	1.0
8	LP001014	1	1.0
9	LP001018	1	1.0
10	LP001020	1	1.0
11	LP001024	1	1.0
12	LP001027	1	1.0

Class: Loan_Status (Nom)

Status
OK

Test dataset have 367 rows and 12 columns.It has one column less than train data,i.e Loan_status.

Data Cleaning:

There are some missing values in the datset.We use WEKA explorer to fill the NaN values by using filter "RemoveMissingValue".As the NaN values or misiing values affect the data prediction there is a need to fill the values with appropriate values.We can consider these methods to fill the missing values:

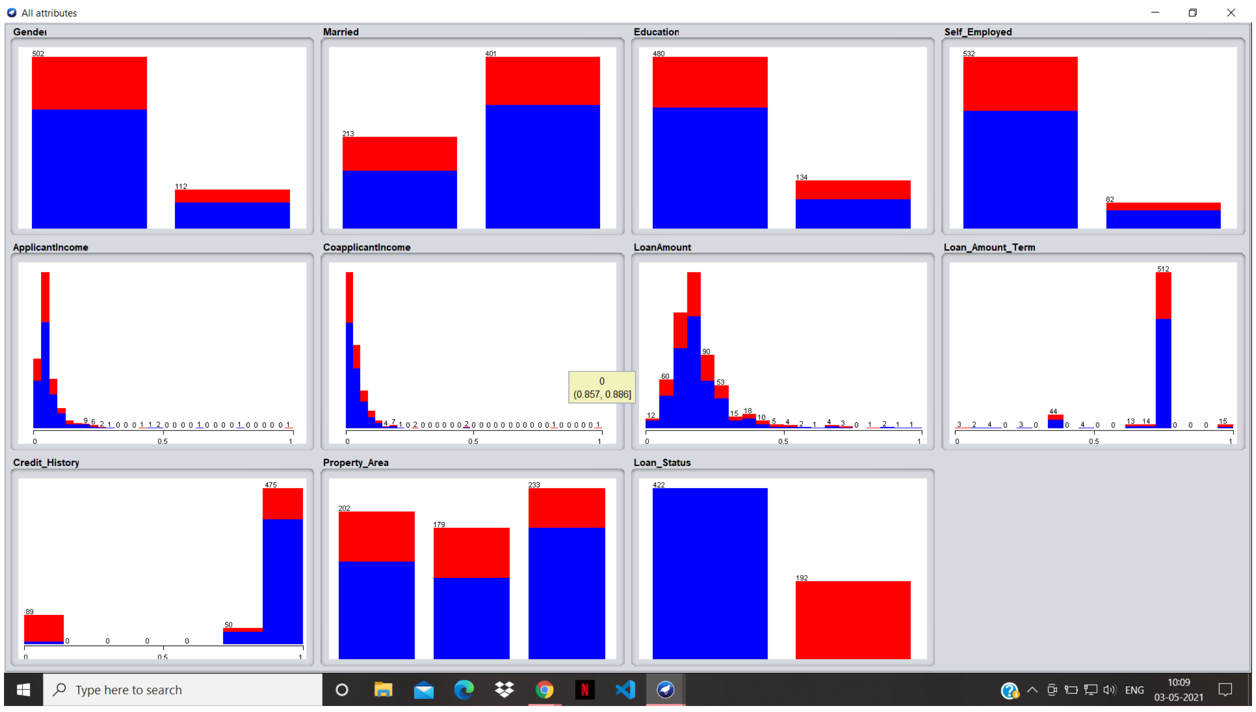
- For numerical variables: imputation using mean or median
- For categorical variables: imputation using mode

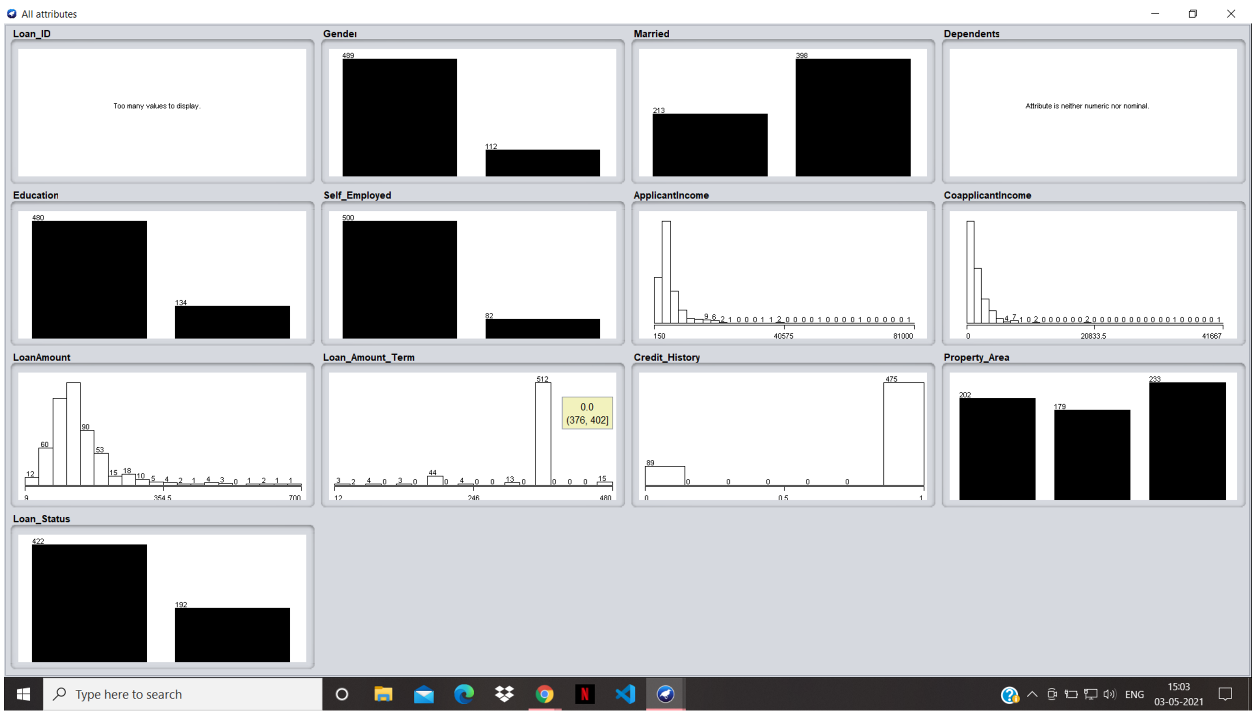
There are very few missing values in Gender, Married, Dependents, Credit_History, and Self_Employed features so we can fill them using the mode of the features.For numerical variables like Loan_amount, we use median to fill out the missing values.We will predict the Loan_Status using the model built using the train data.

Data Visualization and Pre-processing:

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format.Visualizing the data is most crucial part in analyzing the model.We use WEKA explorer to visualize in the form of Histogram and Bar plots.By viewing the data we can see that there are some outliers in the data.In order to build a good model we need to clear out the outliers and do feature engineering.We can normalize the data by using the log function.

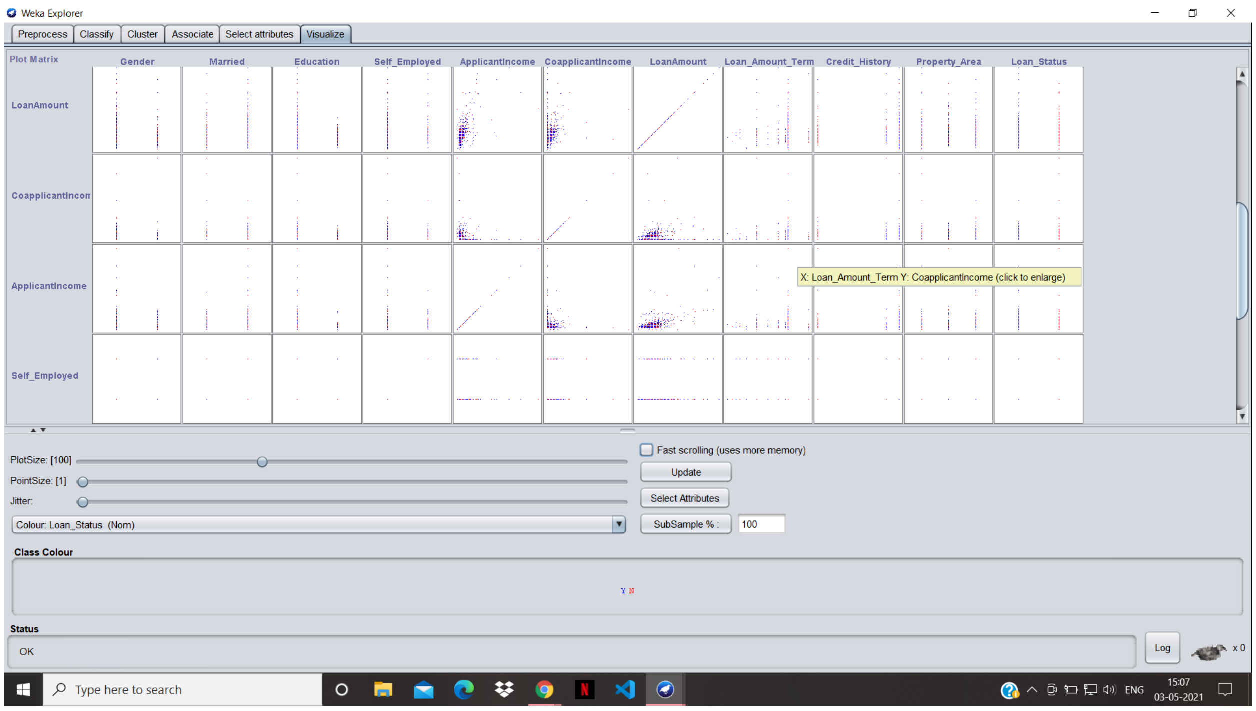
Due to these outliers bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skewness. One way to remove the skewness is by doing the log transformation. As we take the log transformation, it does not affect the smaller values much but reduces the larger values.We can remove these outliers in WEKA explorer by using InterQuartile filter which removes outliers and helps the model in achieving good results





Hence,this is a cleaned dataset with no outliers and misiing values.

We can conclude from the following graphs is that around 80% of the applicants are Graduate.Most of the applicants are from the Semiurban area.



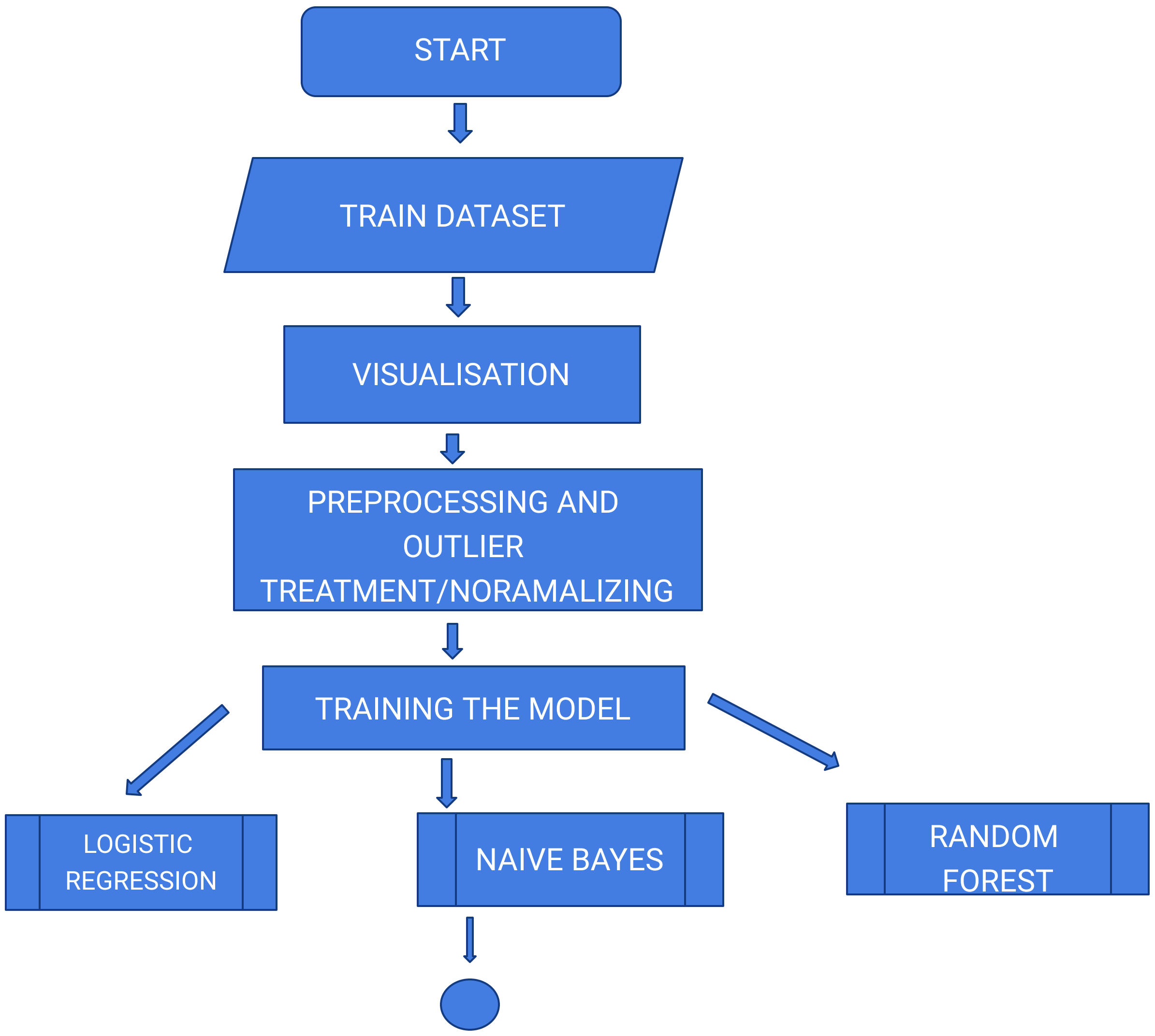
From the scatter plot, we can see that Proportion of loans getting approved for applicants having low Total_Income is very less compared to that of applicants with Average, High & Very High Income.We can see the correlation of the variables.By viewing the scatter plots we can see the correlation between the attributes,how one affects others.

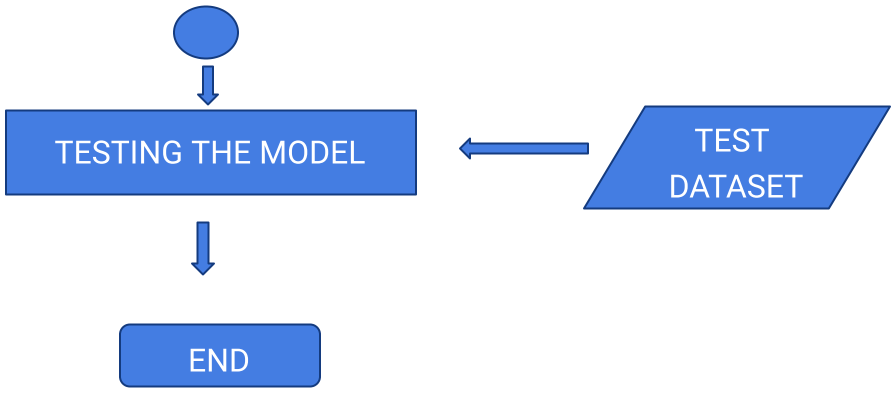
Now we will train the model on the training dataset and make predictions for the test dataset.We are using the following algorithms or predicting the model.

- Logistic Regression
- Random Forest
- Naive Bayes algorithm

Visualization is done with WEKA explorer and modelling is done with both Java Code in Eclipse IDE and WEKA.

5.Flowchart:

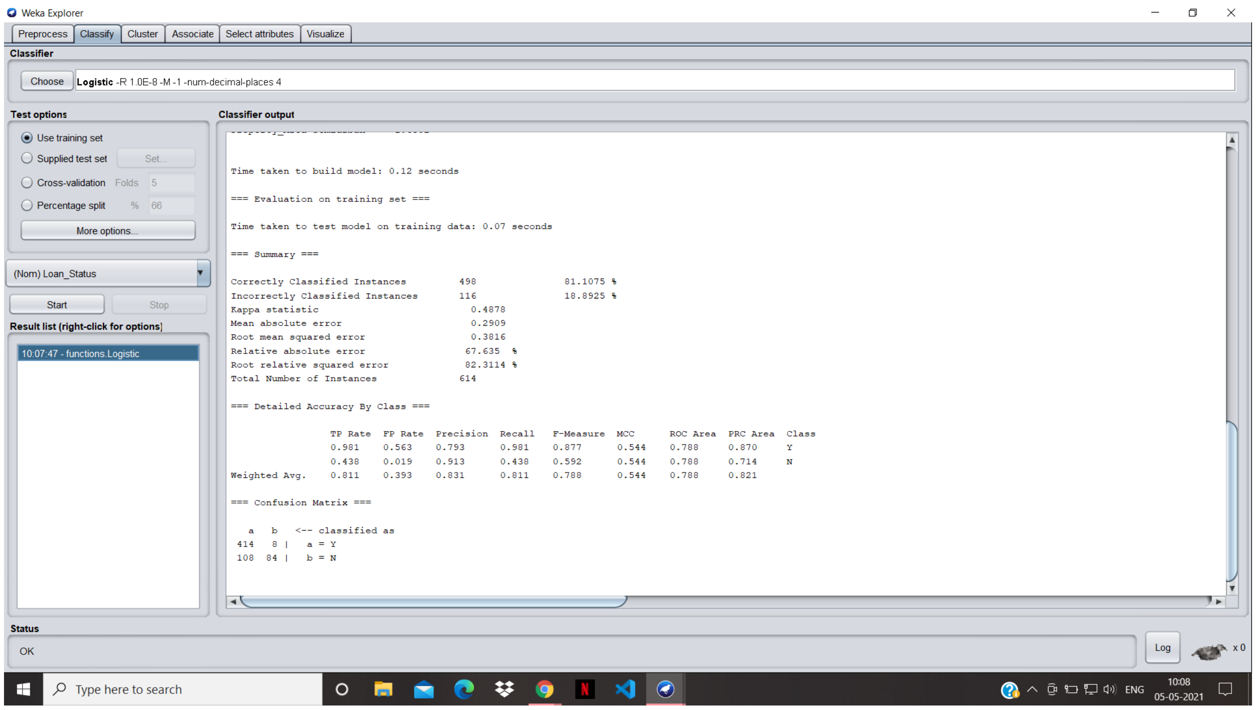




6.Result:

We trained the model by suing the training dataset by splitting into two parts in the ratio 80% and 20%.Logistic regression ,Random forest, Naive Bayes algorithms are used and viewed in WEKA explorer.

- Accuracy by using Logistic Regression is 81.1% with Stratified_K-folds of 5.



Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

(●) Use training set
Cross-validation Folds 5
Percentage split % 66

(Nom) Loan_Status

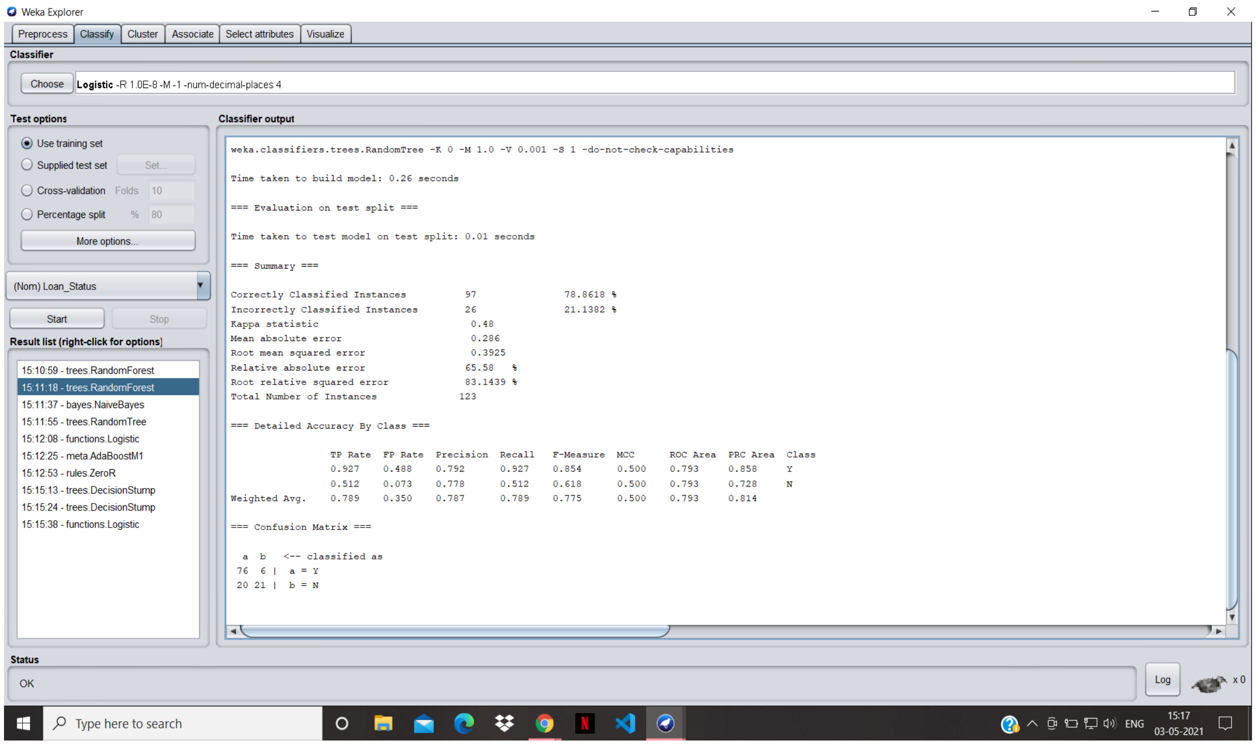
10:07:47 - functions.Logistic

```
Time taken to build model: 0.12 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.07 seconds

=== Summary ===

Correctly Classified Instances         498               81.1075 %
Incorrectly Classified Instances       116               18.8925 %
Kappa statistic                          0.4878
Mean absolute error                      0.2909
Root mean squared error                  0.3816
Relative absolute error                 67.635  %
Root relative squared error             82.3114 %
Total Number of Instances              614

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.981    0.563    0.793      0.981    0.877      0.544    0.788     0.870     Y
                 0.438    0.019    0.913      0.438    0.592      0.544    0.788     0.714     N
Weighted Avg.    0.811    0.393    0.831      0.811    0.788      0.544    0.788     0.821

=== Confusion Matrix ===

   a   b   <-- classified as
 414   8 |   a = Y
 108  84 |   b = N
```

OK

10:08
05-05-2021

- Accuracy by using Random Foresr is 78.8% with Stratified_K-folds of 5.

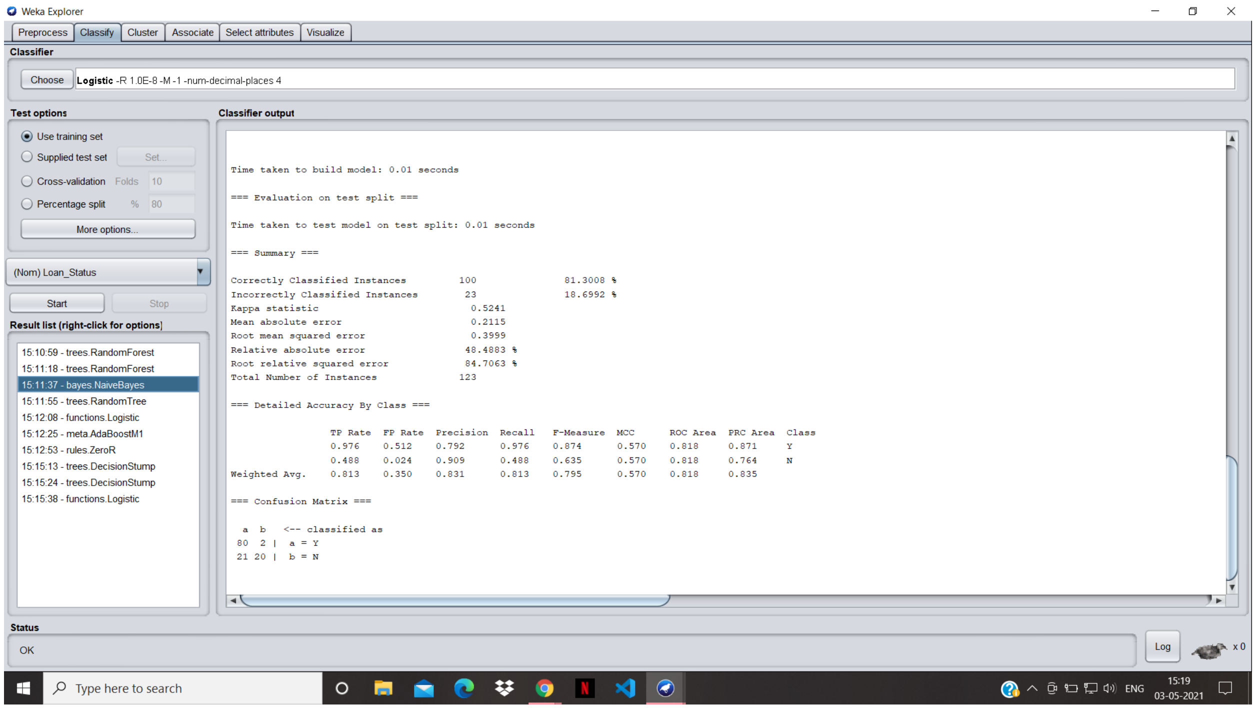


Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

Result list:
- 15:10:59 - trees.RandomForest
- 15:11:18 - trees.RandomForest
- 15:11:37 - bayes.NaiveBayes
- 15:11:55 - trees.RandomTree
- 15:12:08 - functions.Logistic
- 15:12:25 - meta.AdaBoostM1
- 15:12:53 - rules.ZeroR
- 15:15:13 - trees.DecisionStump
- 15:15:24 - trees.DecisionStump
- 15:15:38 - functions.Logistic

```
weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.26 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          97               78.8618 %
Incorrectly Classified Instances        26               21.1382 %
Kappa statistic                          0.48
Mean absolute error                      0.286
Root mean squared error                  0.3925
Relative absolute error                 65.58   %
Root relative squared error             83.1439 %
Total Number of Instances              123

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.927    0.488    0.792      0.927    0.854      0.500    0.793     0.858     Y
                 0.512    0.073    0.778      0.512    0.618      0.500    0.793     0.728     N
Weighted Avg.    0.789    0.350    0.787      0.789    0.775      0.500    0.793     0.814

=== Confusion Matrix ===

  a  b   <-- classified as
 76  6 |  a = Y
 20 21 |  b = N
```

- Accuracy by using Naive Bayes algorithm is 81.3% with better parameters.



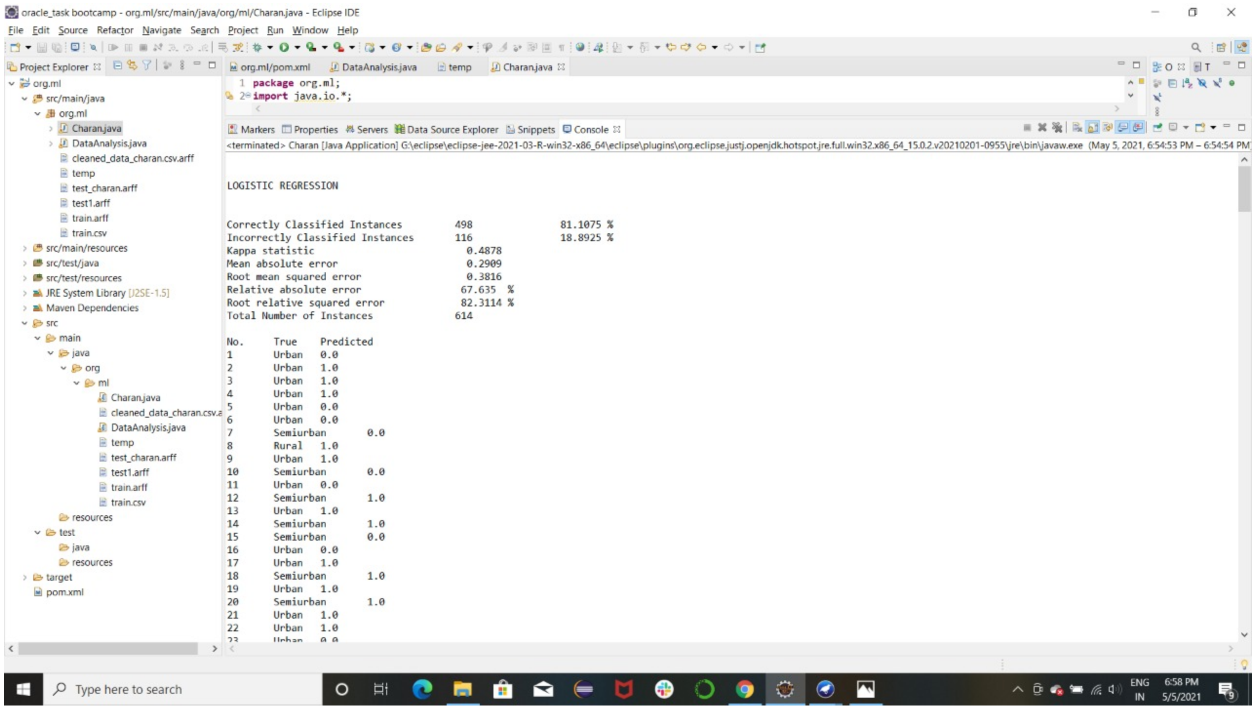
Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

Result list:
- 15:10:59 - trees.RandomForest
- 15:11:18 - trees.RandomForest
- 15:11:37 - bayes.NaiveBayes
- 15:11:55 - trees.RandomTree
- 15:12:08 - functions.Logistic
- 15:12:25 - meta.AdaBoostM1
- 15:12:53 - rules.ZeroR
- 15:15:13 - trees.DecisionStump
- 15:15:24 - trees.DecisionStump
- 15:15:38 - functions.Logistic

```
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances         100               81.3008 %
Incorrectly Classified Instances        23               18.6992 %
Kappa statistic                          0.5241
Mean absolute error                      0.2115
Root mean squared error                  0.3999
Relative absolute error                 48.4883 %
Root relative squared error             84.7063 %
Total Number of Instances              123

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.976    0.512    0.792      0.976    0.874      0.570    0.818     0.871     Y
                 0.488    0.024    0.909      0.488    0.635      0.570    0.818     0.764     N
Weighted Avg.    0.813    0.350    0.831      0.813    0.795      0.570    0.818     0.835

=== Confusion Matrix ===

  a  b   <-- classified as
 80  2 |  a = Y
 21 20 |  b = N
```

MODEL	ACCURACY
Logistic Regression	81.1%
Random Forest	78.8%
Naive Bayes	81.3%

Implementation of Logistic regression in Eclipse IDE using Java Language obtained 81.1% of accuracy with minimum error.



```
package org.ml;
import java.io.*;
```

```
<terminated> Charan [Java Application] G:\eclipse\eclipse-jee-2021-03-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (May 5, 2021, 6:54:53 PM – 6:54:54 PM


LOGISTIC REGRESSION


Correctly Classified Instances         498               81.1075 %
Incorrectly Classified Instances       116               18.8925 %
Kappa statistic                          0.4878
Mean absolute error                      0.2909
Root mean squared error                  0.3816
Relative absolute error                 67.635  %
Root relative squared error             82.3114 %
Total Number of Instances              614

No.     True    Predicted
1       Urban   0.0
2       Urban   1.0
3       Urban   1.0
4       Urban   1.0
5       Urban   0.0
6       Urban   0.0
7       Semiurban       0.0
8       Rural   1.0
9       Urban   1.0
10      Semiurban       0.0
11      Urban   0.0
12      Semiurban       1.0
13      Urban   1.0
14      Semiurban       1.0
15      Semiurban       0.0
16      Urban   0.0
17      Urban   1.0
18      Semiurban       1.0
19      Urban   1.0
20      Semiurban       1.0
21      Urban   1.0
22      Urban   1.0
```

7.Advantages & Disadvantages:

Advantages:

- ➤ Able to predict the outcome above 80% of accuracy
- ➤ Robust in nature
- ➤ Feature Engineering is not needed
- ➤ Robust

Disadvantages:

- ➤ Accuracy is not high
- ➤ Overfitting model
- ➤ More time to train the model

8.Application:

Model can be used deployed in following sectors:

- Banking Sector
- Financial Firms
- Private Loan Lending Firms

9.Conclusion:

After trying and testing 3 different algorithms, the best accuracy is achieved by Logistic Regression (0.811) and Naive Bayes Algorithm(0.813) while Random forest produced underwhelming result(0.788).While new features can be created via feature engineering may help in predicting the target variable. On the whole, a logistic regression classifier provides the best result in terms of accuracy for the given dataset, without any feature engineering needed. Because of its simplicity and the fact that it can be implemented relatively easy and quick, Logistic Regression is often a good baseline that data scientists can use to measure the performance of other more complex algorithms. In this case, however, a basic Logistic Regression has already outperformed other more complex algorithms like Random Forest for the given dataset.

10.Future Scope:

The Designed Model can be enhanced by using newly advanced algorithms like Gradient boosting algorithm and AdaBoosting algorithm etc.These algorithms help in better performance and take less time in training the model.

By Hyper-Parameter tuning , in-depth feature engineering and using neural networks could be a key factor in enchancing the model.

11.Bibliography:

- https://www.analyticsvidhya.com/blog/2015/10/basics-logistic-regression/

- https://towardsdatascience.com/data-types-in-statistics-347e152e8bee
- https://machinelearning-blog.com/2018/04/23/logistic-regression-101/
- https://jtablesaw.github.io/tablesaw/gettingstarted.html
- https://www.javadoc.io/doc/tech.tablesaw/tablesaw-core/latest/overview-summary.html
- https://www.codingame.com/playgrounds/4821/machine-learning-with-java---part-2-logistic-regression
- https://www.baeldung.com/java-logistic-regression

12.Appendix:

WEB.xml:

```
<projectxmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https
://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com</groupId>

<artifactId>org.ml</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>nz.ac.waikato.cms.weka</groupId>
<artifactId>weka-stable</artifactId>
<version>3.8.0</version>
</dependency>
<dependency>
<groupId>tech.tablesaw</groupId>
<artifactId>tablesaw-core</artifactId>
<version>0.38.1</version>
</dependency><dependency>
<groupId>tech.tablesaw</groupId>
<artifactId>tablesaw-jsplot</artifactId>
<version>0.38.1</version>
</dependency><!-- Thanks for using https:/jar-download.com --></dependencies>
<properties>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

```
</project>
```

MAIN.JAVA:

```
package org.ml;
import java.io.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
public class Charan {

        private static String getCurrentDirectoryPath() {
                String path=System.getProperty("user.dir")+"\\src\\main\\java\\org\\ml";
                //System.out.println(path);
                return path.replace("\\","\\\\");
        }
        public static String trainDataPath="";
        public static String testDataPath="";
        public static void main(String args[]) throws Exception{
                String trainFileName="\\\\cleaned_data_charan.csv.arff";
                String testFileName="\\\\test_charan.arff";
                trainDataPath=getCurrentDirectoryPath()+trainFileName;
                //System.out.println(trainDataPath);
                testDataPath=getCurrentDirectoryPath()+testFileName;
                BufferedReader br=new BufferedReader(new FileReader(trainDataPath));
                Instances trainData=new Instances(br);
                BufferedReader brTest=new BufferedReader(new FileReader(testDataPath));
                Instances testData=new Instances(brTest);
                testData.setClassIndex(testData.numAttributes()-1);
                trainData.setClassIndex(trainData.numAttributes()-1);
                br.close();
                logisticRegression(testData,trainData);
        }
```

```
	private static void logisticRegression(Instances testData,Instances trainData) throws
Exception{
		System.out.println("\n\nLOGISTIC REGRESSION\n");
		Classifier classifier = new weka.classifiers.functions.Logistic();
		classifier.buildClassifier(trainData);
		Evaluation eval = new Evaluation(trainData);
		eval.evaluateModel(classifier,trainData);
		System.out.println(eval.toSummaryString());
		System.out.println("No.\tTrue\tPredicted");
		for(int i=0;i<testData.numInstances();i++) {
			String trueLabel;
			trueLabel=testData.instance(i).toString(testData.classIndex());
			double predIndex=classifier.classifyInstance(testData.instance(i));
			String predLabel;
			System.out.println((i+1)+"\t"+trueLabel+"\t"+predIndex);
		}
	}

}
```