CO544-Machine Learning and Data Mining Project Report

Group 13
E/15/077- Dilhani K.P.W.A.K.K.
E/15/211- Maduwanthi S.A.I.
E15/279-Premathilaka L.S.W.S.

1. Introduction

In today's society electronic money is a new trend.Many people use electronic money like Credit cards,Debit cards,etc.Therefore,Commercial banks receive a number of applications asking for credit cards.So it is a difficult task to analyze those applications and approve them. Therefore,in this project, we were asked to experiment with a real world dataset, and to build an automatic credit card approval predictor using machine learning techniques.We were expected to gain experience using a common data-mining and machine learning library, Weka, and were expected to submit the predictions and a report about the dataset and the algorithms used.

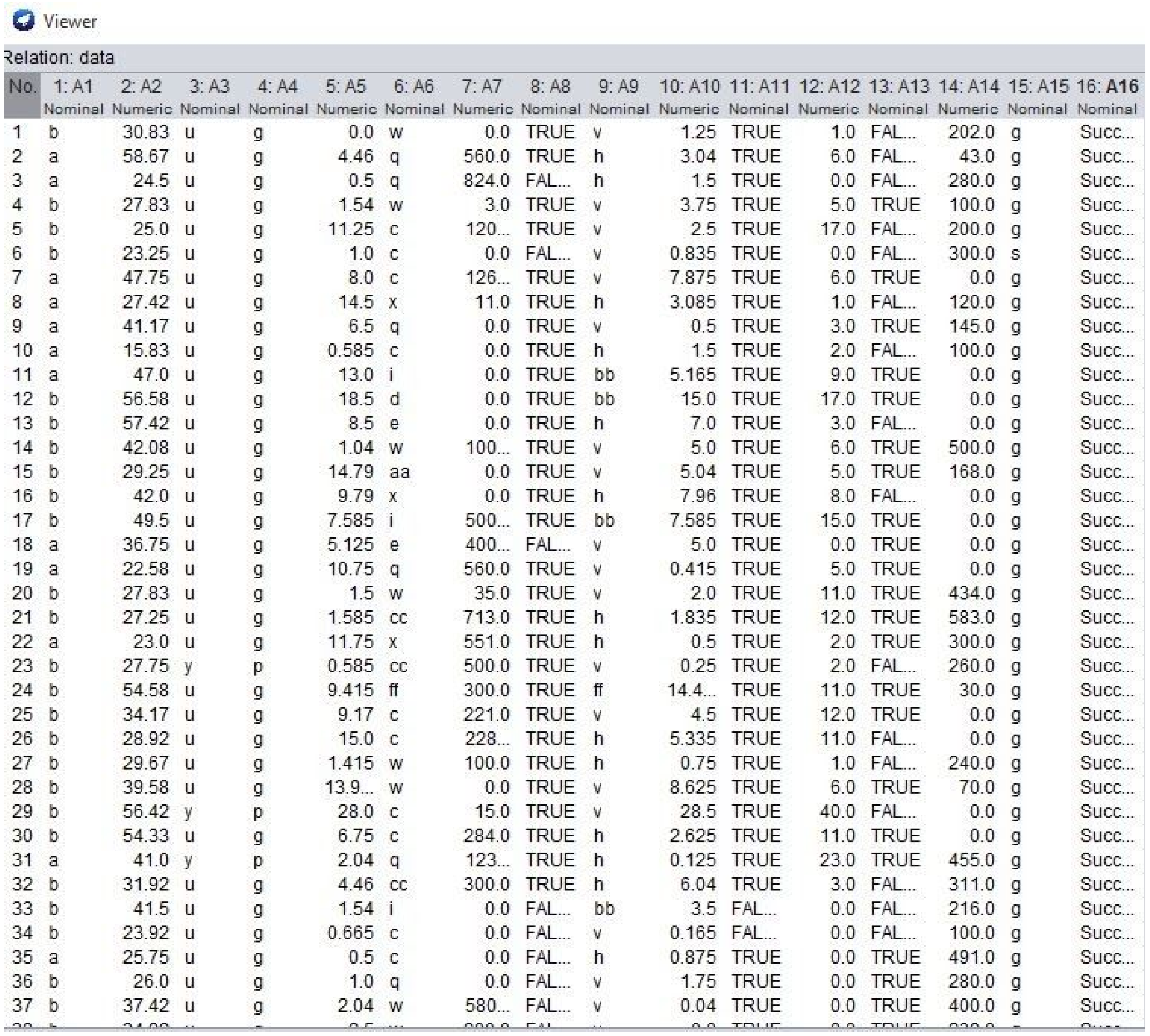
2. Task Definition

The well known fact is that data is never structured, it always has some anomalies like missing values, inconsistent, redundant data. And also there can be both numerical and nominal data columns. A good Machine Learning model can be built with a good data set. So before training the model make sure the data set is free from all these errors. When observing our data set, it has 16 columns in which the last column is the one that has to be predicted. A few details of the data set are,

Attribute Information:

A1: b, a
A2: continuous
A3: u, y, l
A4: g, p, gg
A5: continuous
A6: c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff
A7: continuous
A8: TRUE, FALSE
A9: v, h, bb, j, n, z, dd, ff, o
A10: continuous
A11: TRUE, FALSE
A12: continuous
A13: TRUE, FALSE
A14: continuous
A15: g, p, s
A16: Success,Failure(class attribute)

Figure 1 and Figure 2 show weka representation of the dataset and visualization of each attribute.



Viewer

Relation: data

No.	1: A1	2: A2	3: A3	4: A4	5: A5	6: A6	7: A7	8: A8	9: A9	10: A10	11: A11	12: A12	13: A13	14: A14	15: A15	16: **A16**
	Nominal	Numeric	Nominal	Nominal	Numeric	Nominal	Numeric	Nominal	Nominal	Numeric	Nominal	Numeric	Nominal	Numeric	Nominal	Nominal
1	b	30.83	u	g	0.0	w	0.0	TRUE	v	1.25	TRUE	1.0	FAL...	202.0	g	Succ...
2	a	58.67	u	g	4.46	q	560.0	TRUE	h	3.04	TRUE	6.0	FAL...	43.0	g	Succ...
3	a	24.5	u	g	0.5	q	824.0	FAL...	h	1.5	TRUE	0.0	FAL...	280.0	g	Succ...
4	b	27.83	u	g	1.54	w	3.0	TRUE	v	3.75	TRUE	5.0	TRUE	100.0	g	Succ...
5	b	25.0	u	g	11.25	c	120...	TRUE	v	2.5	TRUE	17.0	FAL...	200.0	g	Succ...
6	b	23.25	u	g	1.0	c	0.0	FAL...	v	0.835	TRUE	0.0	FAL...	300.0	s	Succ...
7	a	47.75	u	g	8.0	c	126...	TRUE	v	7.875	TRUE	6.0	TRUE	0.0	g	Succ...
8	a	27.42	u	g	14.5	x	11.0	TRUE	h	3.085	TRUE	1.0	FAL...	120.0	g	Succ...
9	a	41.17	u	g	6.5	q	0.0	TRUE	v	0.5	TRUE	3.0	TRUE	145.0	g	Succ...
10	a	15.83	u	g	0.585	c	0.0	TRUE	h	1.5	TRUE	2.0	FAL...	100.0	g	Succ...
11	a	47.0	u	g	13.0	i	0.0	TRUE	bb	5.165	TRUE	9.0	TRUE	0.0	g	Succ...
12	b	56.58	u	g	18.5	d	0.0	TRUE	bb	15.0	TRUE	17.0	TRUE	0.0	g	Succ...
13	b	57.42	u	g	8.5	e	0.0	TRUE	h	7.0	TRUE	3.0	FAL...	0.0	g	Succ...
14	b	42.08	u	g	1.04	w	100...	TRUE	v	5.0	TRUE	6.0	TRUE	500.0	g	Succ...
15	b	29.25	u	g	14.79	aa	0.0	TRUE	v	5.04	TRUE	5.0	TRUE	168.0	g	Succ...
16	b	42.0	u	g	9.79	x	0.0	TRUE	h	7.96	TRUE	8.0	FAL...	0.0	g	Succ...
17	b	49.5	u	g	7.585	i	500...	TRUE	bb	7.585	TRUE	15.0	TRUE	0.0	g	Succ...
18	a	36.75	u	g	5.125	e	400...	FAL...	v	5.0	TRUE	0.0	TRUE	0.0	g	Succ...
19	a	22.58	u	g	10.75	q	560.0	TRUE	v	0.415	TRUE	5.0	TRUE	0.0	g	Succ...
20	b	27.83	u	g	1.5	w	35.0	TRUE	v	2.0	TRUE	11.0	TRUE	434.0	g	Succ...
21	b	27.25	u	g	1.585	cc	713.0	TRUE	h	1.835	TRUE	12.0	TRUE	583.0	g	Succ...
22	a	23.0	u	g	11.75	x	551.0	TRUE	h	0.5	TRUE	2.0	TRUE	300.0	g	Succ...
23	b	27.75	y	p	0.585	cc	500.0	TRUE	v	0.25	TRUE	2.0	FAL...	260.0	g	Succ...
24	b	54.58	u	g	9.415	ff	300.0	TRUE	ff	14.4...	TRUE	11.0	TRUE	30.0	g	Succ...
25	b	34.17	u	g	9.17	c	221.0	TRUE	v	4.5	TRUE	12.0	TRUE	0.0	g	Succ...
26	b	28.92	u	g	15.0	c	228...	TRUE	h	5.335	TRUE	11.0	FAL...	0.0	g	Succ...
27	b	29.67	u	g	1.415	w	100.0	TRUE	h	0.75	TRUE	1.0	FAL...	240.0	g	Succ...
28	b	39.58	u	g	13.9...	w	0.0	TRUE	v	8.625	TRUE	6.0	TRUE	70.0	g	Succ...
29	b	56.42	y	p	28.0	c	15.0	TRUE	v	28.5	TRUE	40.0	FAL...	0.0	g	Succ...
30	b	54.33	u	g	6.75	c	284.0	TRUE	h	2.625	TRUE	11.0	TRUE	0.0	g	Succ...
31	a	41.0	y	p	2.04	q	123...	TRUE	h	0.125	TRUE	23.0	TRUE	455.0	g	Succ...
32	b	31.92	u	g	4.46	cc	300.0	TRUE	h	6.04	TRUE	3.0	FAL...	311.0	g	Succ...
33	b	41.5	u	g	1.54	i	0.0	FAL...	bb	3.5	FAL...	0.0	FAL...	216.0	g	Succ...
34	b	23.92	u	g	0.665	c	0.0	FAL...	v	0.165	FAL...	0.0	FAL...	100.0	g	Succ...
35	a	25.75	u	g	0.5	c	0.0	FAL...	h	0.875	TRUE	0.0	TRUE	491.0	g	Succ...
36	b	26.0	u	g	1.0	q	0.0	FAL...	v	1.75	TRUE	0.0	TRUE	280.0	g	Succ...
37	b	37.42	u	g	2.04	w	580...	FAL...	v	0.04	TRUE	0.0	TRUE	400.0	g	Succ...

Figure 1

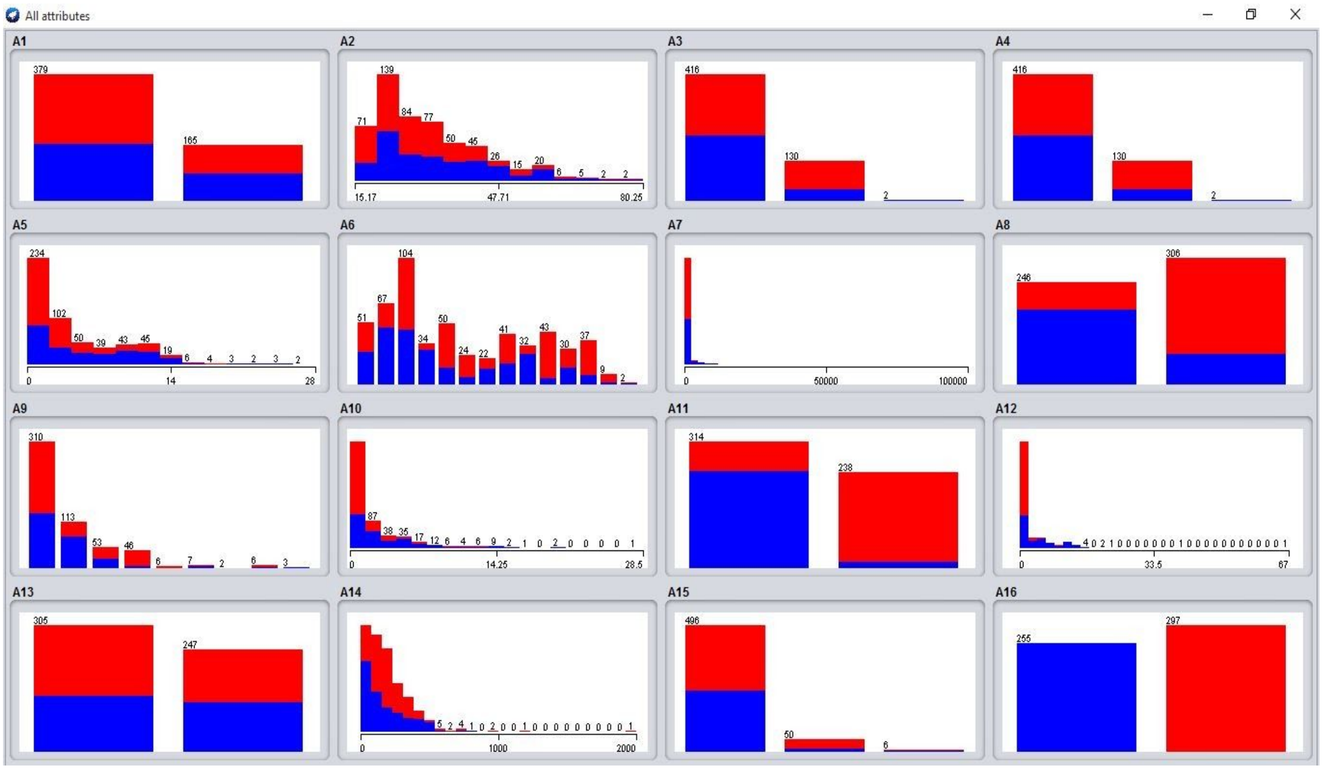


Figure 2:Weka attributes visualization

These are the few details that are available about the dataset.There are a total 552 instances in the training dataset. Our objective is getting high accuracy for predictions. Following are the project tasks that we met here.

- Handling missing values
- Preprocessing the data
- Splitting the data set into train and test sets
- Rescaling
- Building the model
- Predicting the output with measuring the accuracy
- Making predictions and evaluating performance

3. Methodology

First , we started off by loading and viewing the dataset. We can see that the dataset has a mixture of both numerical and non-numerical features, that it contains values from different ranges, plus that it contains a number of missing entries. We had to preprocess the dataset ,to ensure the machine learning model we choose can make good predictions. After our data is in good shape, we did some exploratory data analysis to build our intuitions. Finally, we built a machine learning model that can predict if an individual's application for a credit card will be accepted.

Libraries used:

- pandas
- numpy
- sklearn

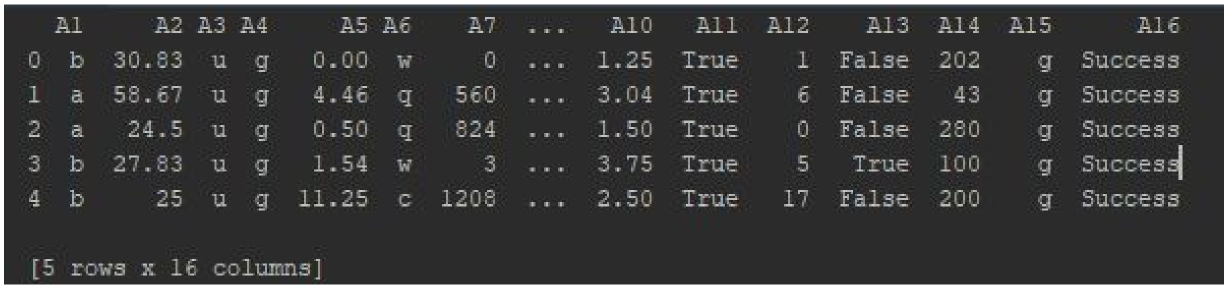
This is how we imported libraries using the import keyword in Python. Here we have used numpy as it contains a powerful N-dimensional array object and sophisticated functions. And pandas is for data manipulation and analysis. Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Scikit-learn (sklearn) is a free machine learning library for Python which features various algorithms like support vector machine(SVM), random forests, and k-neighbours.It will

Step 1 : Import the data-set

```
import pandas as pd
df = pd.read_csv('data.csv')
```

Here df(data frame) represents the training dataset. By using Pandas we imported the data-set and the file format used here is .csv. After importing the dataset, we used head function, to test that each object has the right type of data in it.Here we tested only 5 rows.

```
print(df1.head(5))
```

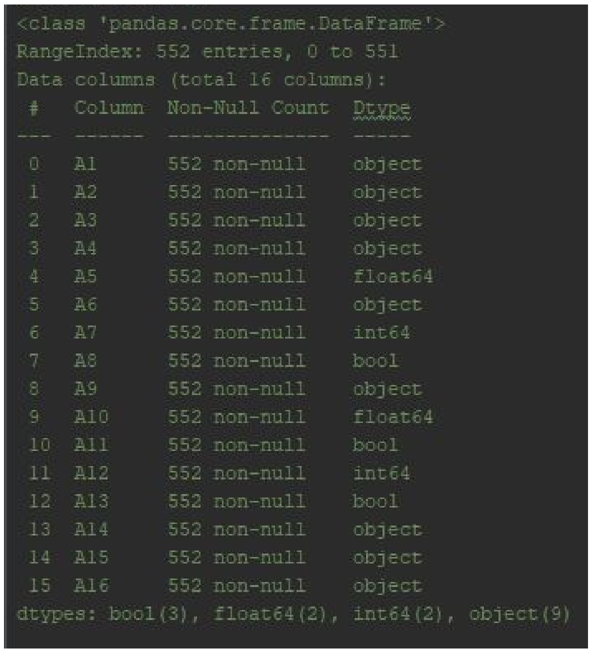


```
  A1     A2 A3 A4     A5 A6    A7  ...   A10   A11  A12    A13  A14  A15      A16
0  b  30.83  u  g   0.00  w     0  ...  1.25  True    1  False  202    g  Success
1  a  58.67  u  g   4.46  q   560  ...  3.04  True    6  False   43    g  Success
2  a   24.5  u  g   0.50  q   824  ...  1.50  True    0  False  280    g  Success
3  b  27.83  u  g   1.54  w     3  ...  3.75  True    5   True  100    g  Success
4  b     25  u  g  11.25  c  1208  ...  2.50  True   17  False  200    g  Success

[5 rows x 16 columns]
```

Figure 3

It is important to know the data type of each column.By using the following command we can see the data types.

```
print(df1.info())
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 552 entries, 0 to 551
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      552 non-null    object
 1   A2      552 non-null    object
 2   A3      552 non-null    object
 3   A4      552 non-null    object
 4   A5      552 non-null    float64
 5   A6      552 non-null    object
 6   A7      552 non-null    int64
 7   A8      552 non-null    bool
 8   A9      552 non-null    object
 9   A10     552 non-null    float64
 10  A11     552 non-null    bool
 11  A12     552 non-null    int64
 12  A13     552 non-null    bool
 13  A14     552 non-null    object
 14  A15     552 non-null    object
 15  A16     552 non-null    object
dtypes: bool(3), float64(2), int64(2), object(9)
```

Figure 4

Our dataset contains both numerical and non-numerical data (float64, int64, bool and object types). Specifically, the features A5, A7, A10 and A12 contain numeric values (of types float64,int64, float64 and int64 respectively) and all the other features contain non-numeric values.

Step 2.Check the Class Distribution

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. Imagine a case where 99% of the data belongs to one class, this can cause the classification model to ignore the remaining class and indeed it would get very good accuracy. But a small difference often does not matter and this is the case in our dataset.

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x="A16", data=df, palette="Greens_d")
plt.title("Class Distribution")
plt.show()
```

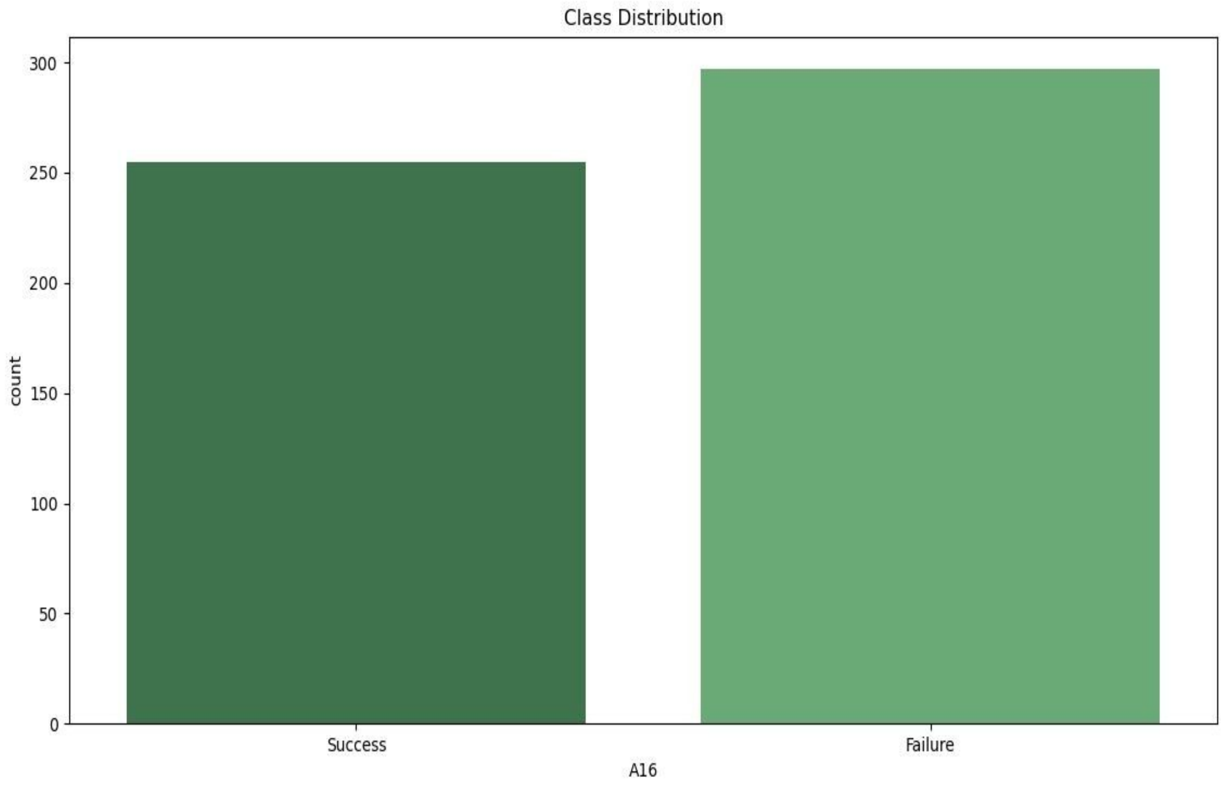
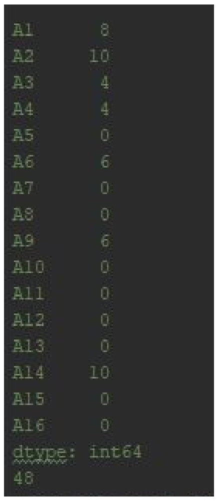


Figure 5: Class Distribution

Step 3 : Handling missing values

First we checked out whether there are missing values.

```
print(df.isnull().sum())
```



```
A1      8
A2     10
A3      4
A4      4
A5      0
A6      6
A7      0
A8      0
A9      6
A10     0
A11     0
A12     0
A13     0
A14    10
A15     0
A16     0
dtype: int64
48
```

Figure 6

There are a total 552 instances in the dataset and out of them, 7 attributes have missing values and, in total there are 48 missing values.We had to take care about these things in this section.

First we identified that in this data set the missing values are represented as ‘?’. So we checked for the rows which have missing values and replaced all the question marks with ‘NaN’s(Not a Number -they are represented as None value). This helped us in the next missing value treatment that we performed.

```
import numpy as np
df1 = df1.replace(['?'],np.NaN)
```

Here we had two options when handling the missing values.

1.Ignore the missing values.

2.Handle the missing values usefully.(mean imputation and replacing)

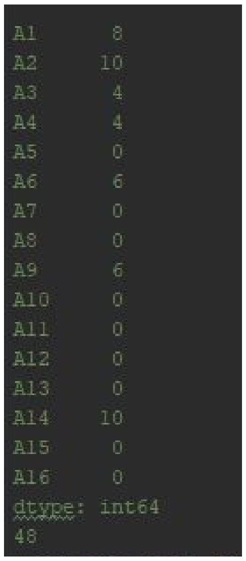
Ignoring missing values can affect the performance of a machine learning model heavily. While ignoring the missing values, machine learning models may miss out on information about the dataset that may be useful for its training. Then, there are many models which cannot handle missing values implicitly such as LDA.

So, to avoid this problem, we imputed the missing values with a strategy called **mean imputation**.
Mean imputation method simply calculates the mean of the observed values for that variable for all individuals who are non-missing. It has the advantage of keeping the same mean and the same sample size.

```
df1.fillna(df1.mean(), inplace=True)
```

By this command,we imputed the missing values with mean imputation. But fillna() function only fills up the numeric columns. So again the number of remaining Nans were counted to verify whether there were non-numeric missing values in the particular dataset.

```
print(df.isnull().sum())
```



```
A1      8
A2     10
A3      4
A4      4
A5      0
A6      6
A7      0
A8      0
A9      6
A10     0
A11     0
A12     0
A13     0
A14    10
A15     0
A16     0
dtype: int64
48
```

Figure 7

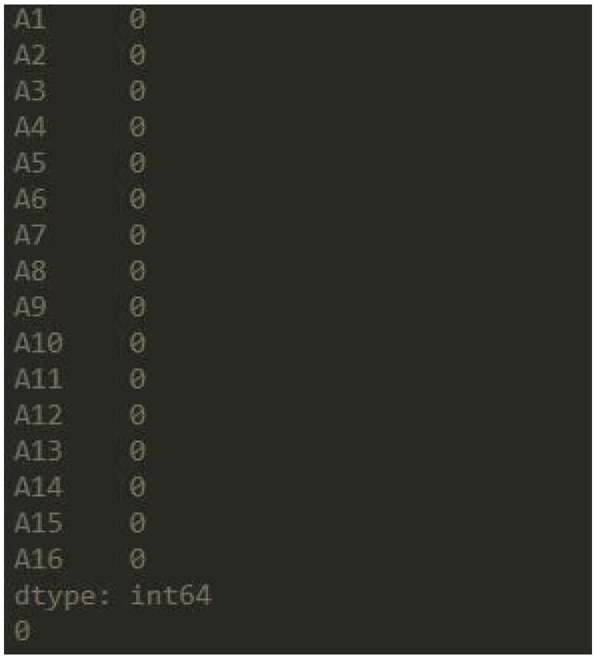
We successfully took care of the missing values presented in the numeric columns(A5,A7,A10,A12).But there are still missing values to be imputed for columns A1, A2, A3, A4, A6, A9 and A14. All of these columns contain non-numeric data (object and bool types) and this is why the mean imputation strategy would not work here. This needs a different treatment.

We imputed these missing values with the most frequent values as present in the respective columns. This is good, when it comes to imputing missing values for categorical data in general.

```
  # Iterate over each column of df
for col in df:
# Check if the column is of object type
  if df[col].dtypes == 'object':
 # Impute with the most frequent value
    df = df.fillna(df[col].value_counts().index[0])
```

Now we can check whether there are any more missing values.

```
print(df.isnull().sum())
print(df.isnull().values.sum())
```



```
A1     0
A2     0
A3     0
A4     0
A5     0
A6     0
A7     0
A8     0
A9     0
A10    0
A11    0
A12    0
A13    0
A14    0
A15    0
A16    0
dtype: int64
0
```

Figure 8:

The missing values are now successfully handled.

Data preprocessing

There was still some minor but essential data preprocessing needed before we proceed towards building our machine learning model. We divided those remaining preprocessing steps into three main tasks:

1. Convert the non-numeric data into numeric.
2. Split the data into train and test sets.
3. Scale the feature values to a uniform range.

Step 4 : See the Categorical Values

Since the machine learning models are based on mathematical equations, we needed to encode categorical variables. To convert Categorical variables into Numerical data, we used LabelEncoder() function from sklearn library.

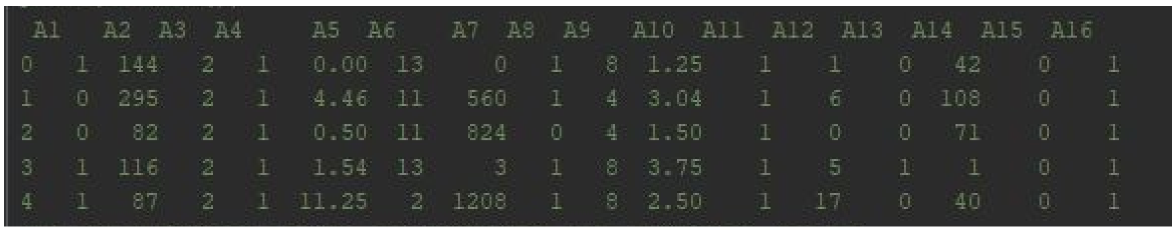
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder() #Instantiate LabelEncoder
```

"le" is the object which we used in transferring Categorical data into Numerical data. Then it was iterated over the values of each column and extracted data types and converted them into numerical values like below.

```
for col in df:
   # Compare if the dtype is object
   if df[col].dtypes =='object' or df[col].dtypes =='bool':
   # Use LabelEncoder to do the numeric transformation
       df[col]=le.fit_transform(df[col])
```

Then, we checked out to verify whether all values are numerical or not.

```
print(df.head())
```



```
 A1   A2  A3  A4     A5  A6    A7  A8  A9   A10  A11  A12  A13  A14  A15  A16
0  1  144   2   1   0.00  13     0   1   8  1.25    1    1    0   42    0    1
1  0  295   2   1   4.46  11   560   1   4  3.04    1    6    0  108    0    1
2  0   82   2   1   0.50  11   824   0   4  1.50    1    0    0   71    0    1
3  1  116   2   1   1.54  13     3   1   8  3.75    1    5    1    1    0    1
4  1   87   2   1  11.25   2  1208   1   8  2.50    1   17    0   40    0    1
```

Figure 9

All non-numerical values are successfully converted into numerical values.

Step 5 : Splitting the data-set into Training and Test Set

Then, we split our data into a train set and a test set to prepare our data for two different phases of machine learning modeling: training and testing. Ideally, no information from the test data should be used to scale the training data or should be used to direct the training process of a machine learning model. Hence, we first split the data and then apply the scaling.

For that , dataFrame should be a numpy array. So first, convert the dataFrame into a numpy array.

```
      df = df.values
```

Then Segregate features and labels into separate variables.

```
      X,y = df1[:,0:15] , df1[:,15]
```

Then we split the dataset into the train set and the test set like below.

```
      from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33,
random_state=15)
```

Generally we split the data-set into 70:30 ratio or 80:20 what does it mean, 70 percent data take in train and 30 percent data take in test. Here we used 2:1 . That means test_size = 1 / 3 . However, this Splitting can vary according to the data-set shape and size.

X_train is the training part of the matrix of features. X_test is the test part of the matrix of features. y_train is the training part of the dependent variable that is associated with X_train here. y_test is the test part of the dependent variable that is associated with X_train here.

Step 6 : Feature Scaling

The data is now split into two separate sets - train and test sets respectively. We are only left with one final preprocessing step of scaling before we can fit a machine learning model to the data. It helps the algorithm to converge much faster. We used `MinMaxScaler()` function from sklearn library.

```
      from sklearn.preprocessing import MinMaxScaler
```

Instantiate MinMaxScaler and use it to rescale X_train and X_test.

```
      scaler = MinMaxScaler(feature_range=(0, 1))

      rescaledX_train = scaler.fit_transform(X_train)
      rescaledX_test = scaler.fit_transform(X_test)
      rescaled_testdata = scaler.fit_transform(df2)
```

Step 7: Building the model

This step must be followed by the cross validation which evaluates accuracy of different algorithms and selects the best among them. First we have to identify which type of problem it is, It is clearly a classification problem since we have to predict the output in which class it will belong (Success or Failure). A good machine learning model should be able to accurately predict the status of the applications with respect to these statistics. Evaluating the performance of classification algorithms like logistic regression, SVM, Decision Tree, Random Forest, Kth Nearest Neighbour.Here we have used the **Random Forest model** as our final classification model.

```
from sklearn.ensemble import RandomForestClassifier
```

Import RandomForest library from skitch-learn. Then create an instance for RandomForest classifier ("model" is our instance) with default parameter values and fit it to trani set using fit() function.

```
model = RandomForestClassifier(random_state=600)
model.fit(rescaledX_train, y_train)
```

Step 8: Predicting the result

This is the final step. Now model building is completed. To predict off the y_test values and evaluate our model, predict the class of not.fully.paid for the X_test data.

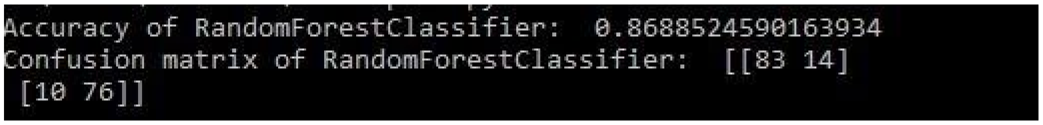
```
y_pred = model.predict(rescaledX_test)
```

It's time to predict the results and check for the accuracy of the model on the testing dataset.

```
from sklearn.metrics import accuracy_score
print("Accuracy of RandomForestClassifier:" ,accuracy_score (y_test,y_pred))
```

Finally evaluating model performance using a confusion matrix.

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix


print("Confusion matrix of RandomForestClassifier: ", confusion_matrix(y_test, y_pred))
```



```
Accuracy of RandomForestClassifier:  0.8688524590163934
Confusion matrix of RandomForestClassifier:  [[83 14]
 [10 76]]
```

Figure 10

For the confusion matrix, the first element of the first row of the confusion matrix denotes the true negatives meaning the number of negative instances (denied applications) predicted by the model correctly. And the last element of the second row of the confusion matrix denotes the true positives meaning the number of positive instances (approved applications) predicted by the model correctly.

4. Results

Accuracy and confusion matrix like below.

Accuracy of RandomForestClassifier: 0.8688524590163934

```
Confusion matrix of RandomForestClassifier:  [[83 14]
                                              [10 76]]
```

5. Discussion

5.1 Choose the most suitable Classification Model

When considering models, there are various methods that can be used for data prediction. Such like,

- ❏ Logistic Regression model
- ❏ Decision Tree
- ❏ Random Forest model
- ❏ Kth Nearest Neighbour
- ❏ SVC
- ❏ GaussianNB

We tried those models and the accuracy scores that we got are included in the following table.

Classifier	**Accuracy Score(as a percentage)**
KNeighborsClassifier	65.0273%
SVC	53.0055%
DecisionTreeClassifier	82.5137%
RandomForestClassifier	86.8852%
GaussianNB	78.6885%
LogisticRegression	81.9672%

Table 1:Classification of accuracy for each model

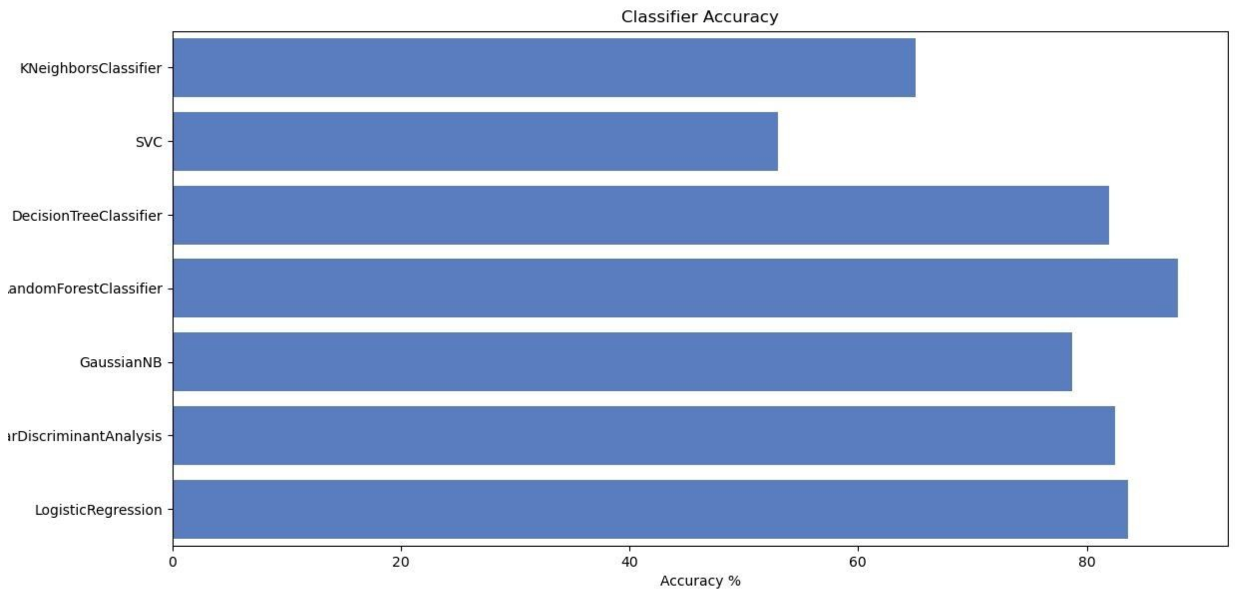


Figure 13:Classifier Accuracy

5.2 The Feature importance

Feature importance is one of the key aspects of a machine learning model. Understanding which variable is contributing the most to a model is critical to interpreting the results. This is what data scientists strive for when building models that need to be explained to non-technical stakeholders.

Our dataset has multiple features and it is often difficult to understand which feature is dominant. This is where the feature importance function of random forest is so helpful. Let’s look at the importance of features for our current model (including visualizing them by their importance):

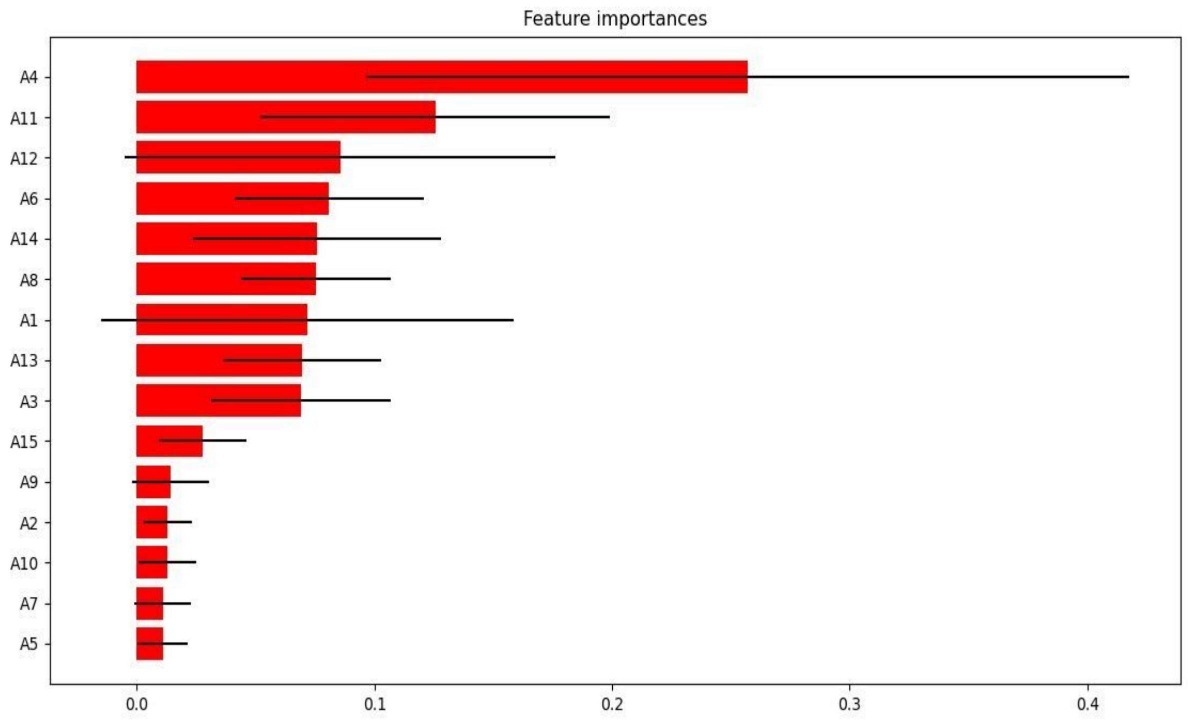


Figure 12: Feature importance

According to Figure12,we can see the importance of the features and every feature has contributed to a certain level.Therefore we didn't remove any column in this process.

5.2 Effect of the handling missing values

In this project, our accuracy for prediction is 0.8743 . Accuracy depends on the way we handle missing values and the type of the model that we have used to predict. Here, we have used mean imputation for numeric missing values and for non-numeric missing values we have replaced them with the most frequent instance of each column. And also we have used the Random Forest model to predict data.

As the above in the report we have mentioned that, there are several ways that can be used to handle missing values. Such like,

- ➢ Delete the rows which are having the missing values.

By this, delete a particular row if it has a null value for a particular feature and a particular column if it has more than 75% of missing values. This method is advised only when there are enough samples in the data set and not a good method for a small number of datasets. It may cause a reduction in accuracy. One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss of information which will not give the expected results while predicting the output.

- ➢ If the variables are continuous replace all the missing values with the mean or median of the attribute.

We can calculate the mean, median or mode of the feature and replace it with the missing values. This is an approximation which can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns. Replacing with the above three approximations is a statistical approach of handling the numerical missing values **.**

- ➢ If the variable in categorical, replace with the most common value of that variable

If there are only non-numerical attributes there cannot be taken as a mean value. So choose the most frequent value in each attribute column and replace it with missing values.

Therefore in our project there are only 552 instances with both numerical and categorical instances. So ignoring missing values may cause weak accuracy. So we have used mean imputation.

6. Conclusion

According to the above table,we can clearly see the Random Forest Classifier has the best accuracy score.Therefore, we conclude that, among all the models, Random forest is the most effective classifier.However, we still have an error rate larger than 10%. It may be due to the limited number of instances.