Mario Bouzakhm

ID: 260954086

Deliverable 2 – MAIS 202 Project

Problem statement: In this project, I will be using data set from Kaggle to design a machine learning model that predicts whether a financial institution will accept or not a loan request of a client. The model used will be a binary logistic regression model.

Data Preprocessing:

The data set that will be used for this project is: https://www.kaggle.com/vikasukani/loan-eligibility-prediction-machine-learning/

Multiple changes were made to the dataset so that it can be used with the model. The changes were made to the dataset using the **pandas** and the **sklearn** library.

The dataset was first imported to pandas and converted to a DataFrame. From there I proceeded to map all string categorical columns into numbers to be used in the model. Categories with two values were converted to two values 0/1. This change covers the columns: 'Married', 'Gender', 'Education', 'Self_Employed' and 'Loan_Status' using label encoding. The column 'Property_Area' and 'Dependents' columns were also altered to only contain numbers. But label encoding was not used in this case because this method assumes that label sizes represents ordnality (meaning that the label of 2 is greater than a label of 1 which is not the case), hence I used one hot encoding to process these columns. Thus the total number of columns in the dataset was increased as a result of the one hot encoding process. LabelEncoding and OneHotEncoding was done using sklearn module.

I also dropped the column with ID information as this does not give us any unseful information for the model. Moreover, all the columns that contained missing data were dropped. All the missing data were hard to replace while keeping the model integrity: All columns with two data points cannot be filled with data. I also did not keep the columns were the loan amount/term was missing as I think that this is the most important part of the dataset and row without this is not really meaningfull.

Moreover, I performed normalization on the data that contains numbrs in a wide range. This allows us to give equal importance to all the features in the model. To perform this I used the sklearn.preprocessing package and specifically the MinMaxScale class.

Before Preprocessing:

```
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

After Preprocessing:

```
Int64Index: 480 entries, 1 to 613
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             480 non-null    int32
 1   Married            480 non-null    int32
 2   Education          480 non-null    int32
 3   Self_Employed      480 non-null    int32
 4   ApplicantIncome    480 non-null    float64
 5   CoapplicantIncome  480 non-null    float64
 6   LoanAmount         480 non-null    float64
 7   Loan_Amount_Term   480 non-null    float64
 8   Credit_History     480 non-null    float64
 9   D0                 480 non-null    float64
 10  D1                 480 non-null    float64
 11  D2                 480 non-null    float64
 12  D3                 480 non-null    float64
 13  P0                 480 non-null    float64
 14  P1                 480 non-null    float64
 15  P2                 480 non-null    float64
 16  Loan_Status        480 non-null    int32
dtypes: float64(12), int32(5)
memory usage: 58.1 KB
```

Machine Learning Model/Prelemenary Results:

As stated in the first deliverable and in the problem statement, I chose to use a logistic regression model for the prediction as I believe it is the most fit model to predict a result with only two possible answers Yes or No.

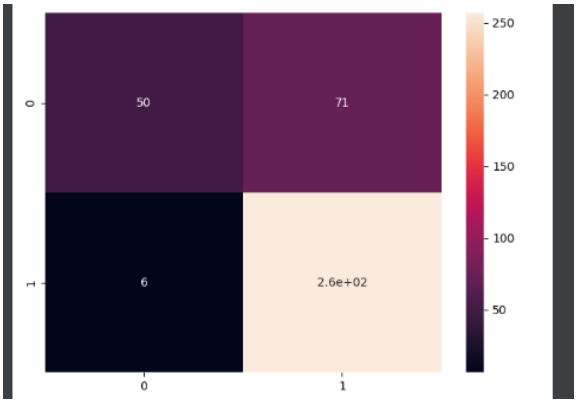
I chose to use sklearn module to implement my model, specificialy the LogisticRegression class. I set the number of iterations to 1000 as I found with testing that the accuracy of the model does not change for higher values and it makes training faster. I decided to use a 80/20 train-test split as the dataset does not contain any data and I'm trying to preserve as much data as possible to train on given that the

number of features is not low. (16 after preprocessing). Hence preserving a high data rows/features ratio should be high to get accurate results.

I tested my model on the trained data. I used classification report provided by the sklearn module to evaluate the model. It provides me with various metrics that are relevant to classification problems. Weights were saved using the pickle module to be used later.

Classification Report for Y_train/Y_Pred

	precision	recall	f1-score	support
0	0.89	0.41	0.56	121
1	0.78	0.98	0.87	263
accuracy			0.80	384
macro avg	0.84	0.70	0.72	384
weighted avg	0.82	0.80	0.77	384

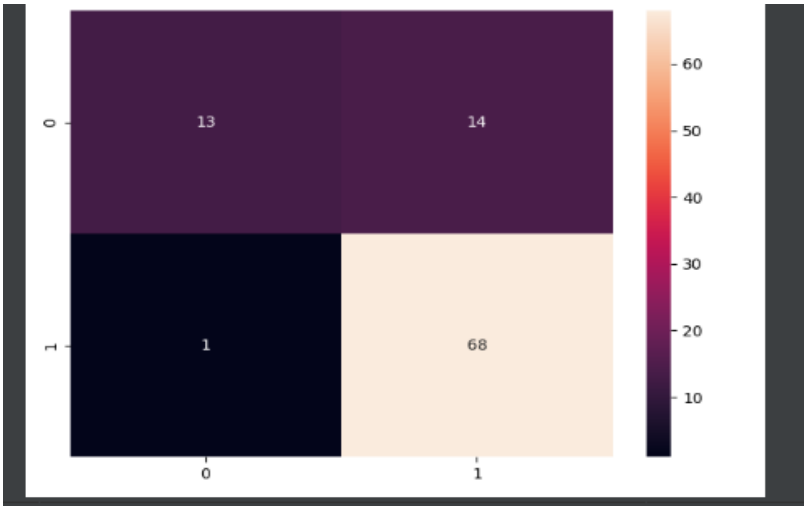


This indicates high accuracy meaning the model is accuratly predicting most of its inputs. The precision/recall values especiallt the recall value for 0 indicates that we can have some extra tuning of the model.

I proceeded to test the model on the test data that was collected using the train-test split.

Classification Report for Y_Test/Y_Test_Pred

	precision	recall	f1-score	support
0	0.93	0.48	0.63	27
1	0.83	0.99	0.90	69
accuracy			0.84	96
macro avg	0.88	0.73	0.77	96
weighted avg	0.86	0.84	0.83	96



I found the results surprising since I obtained even better results than with the train dataset. The model is not overfitting since we are seeing similar results on both the train and test data sets. From these preliminary results I can say that the model is feasible/effective on the problem statment. I observed a 15% error results which can be worked on to decrease during the next phase of the project.

Next Steps:

I developed the following ideas that could be implemented in the next deliveable to try and get better results:

- Try to add the median/mean value, and compare results for missing rows instead of deleting where applicable.
- Try to perform more data preprocessing, specifcally change min/max scaling to z score for Income since one of the values is really high and is offsetting the rest.
- Testing the correlation of the inputs to see if any of them can be removed/is irrelevant for our model in order to save testing time and reducing the number of features.