REPORT

Date: 28 November 2021
Name: Abdul Ahmed

GOAL: The aim is to train a model which can classify *good* and *bad* customers.

SECTION 1: SOURCE CODE GUIDELINES

Below are the steps:

- Unzip the compressed file.
- Copy the folder to your current directory say (*Downloads*).
- *traning_transform.py, test_transform.py* are for transforming *training.csv, test.csv*
- You can use *test_transform.py* to transform *test.csv*. Excecuting this script will automatically save the transformed dataset as *xtest.csv* and *ytest.csv* in the *data* drive.
- Skip the training transformation and test transformation. You can find the transformed data as *(xtrain.csv, ytrain.csv)* and (*xtestcsv, ytest.csv)* in the *data* drive.
- Use *main.py* to load the transformed (*xtest.csv, ytest.csv)* and *(xtrain.csv, ytrain.csv)* from the *data* drive. Execute the script to get the results.
- The trained Model is saved in *saved_models* drive. You can re-train the model by simply uncommenting line no. 25 in the *main.py* script.
- Use *training_transform.py* for visualization only.
- There are 2 models *new_model1.h5* and *new_model2.h5* in *saved_models drive* ready for prediction.

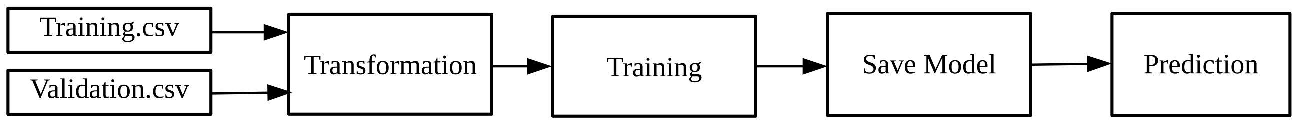
SECTION 2: ASSUMPTIONS & DESIGN CHOICE

- "**Unnamed:0"** column is an ID with high cardinality, so drop this column.
- Categorical feature: If the unique counts in feature<=25.
- Numerical feature: If the unique counts in feature>25.
- **"?"** is **NaN** only in case of Numerical feature.
- In case of Categorical feature **"?"** is considered as a category.
- **","** is converted to Decimals.
- **RARE** Categories: If percent count of (*'?', 't', 'a', 'bb', etc*.) are less than 5%.
- Categories **RARE** in one feature are **not RARE** in other.
- If **%NaN** is greater than 50% in a particular column, so drop it.
- Default threshold of **0.5** is used for binary prediction (hence giving equal importance to both classes) which can be changed according to busniess problem.

SECTION 3: TRAINING-DATA ANALYSIS

- Counts (good) = 2398
- Counts (bad) = 273
- No. of records in the training set = 2671
- No. of records in the validation set = 490
- Total no. of independent features = 21
- No. of Numeric features identified after feature engg.= 5
- No. of Categoric features identified after feature engg.= 12

SECTION 4: THE APPROACH



- Transform (feature engg./feature selection) both training and val/test data.
- Train the classifier.
- Save model as model.h5 file.
- Load model.h5 and do the final prediction on val/test data.

SECTION 5: DATA VISUALIZATION

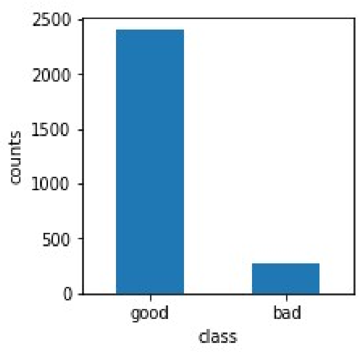


Figure 1: class distribution

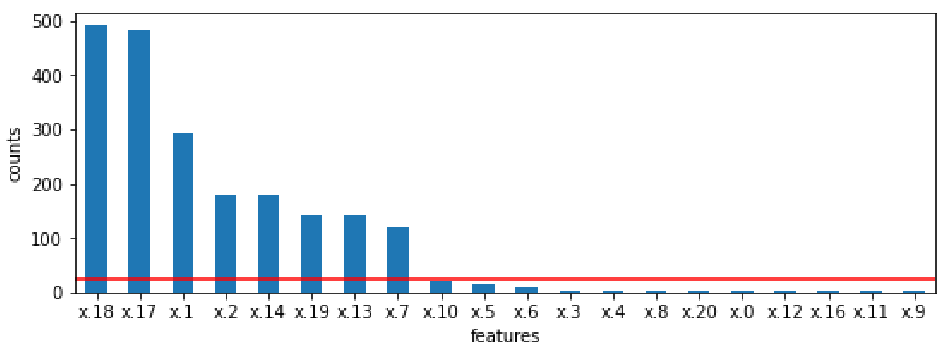


Figure 2: Unique counts in each feature

Figure 1 shows the class wise distribution of the target variable. The distribution is skewed. **Figure 2** shows the counts of unique values in each features. The **red** horizontal line is the threshold at count=25. The features having counts > 25 are Numerical and counts <= 25 are categorical.

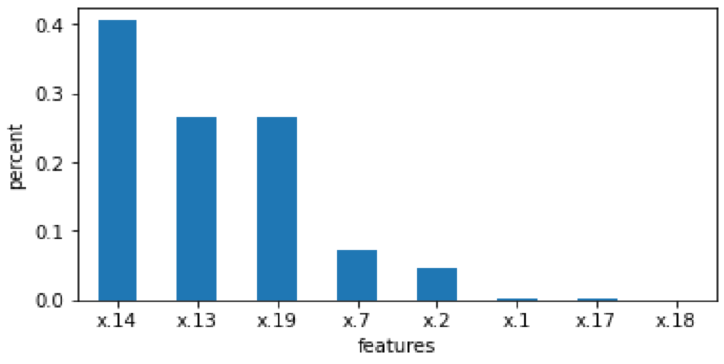


Figure 3: %NaN in Numerical feature

Figure 3 shows the **%NaNs** in each numeric feature column. NaNs for numeric features will be replaced by group mean of the respective target class. **Figure 4** shows the count of each categories in **x.6** feature. The **red** horizontal line is the 5% threshold. Rare categories by assumption are those categories whose %counts are < 5% of the total records. In case of **x.6** feature, rare categories are **{ff, z, ?, j, n, o, dd}**. Summarize these categories into one category ***'rare'*** as shown in **Figure 5.** Categories **{v, h, bb}** are important categories.

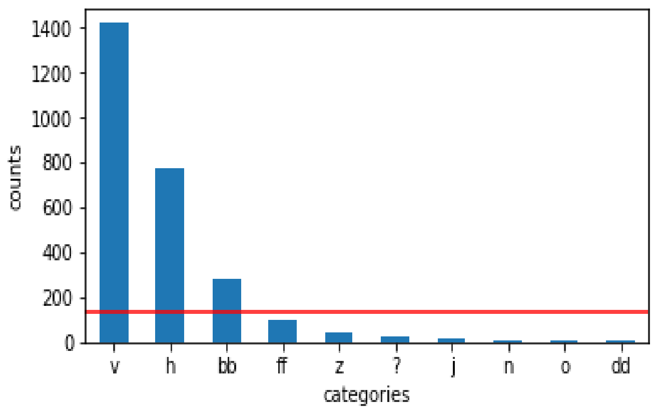


Figure 4: Categories Before 'rare' Summarization

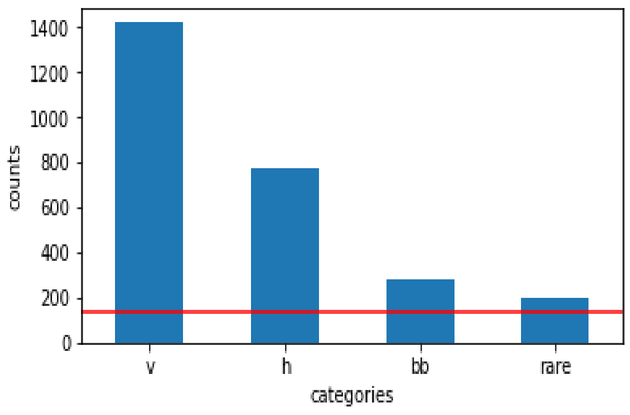


Figure 5: Categories After 'rare' Summarization

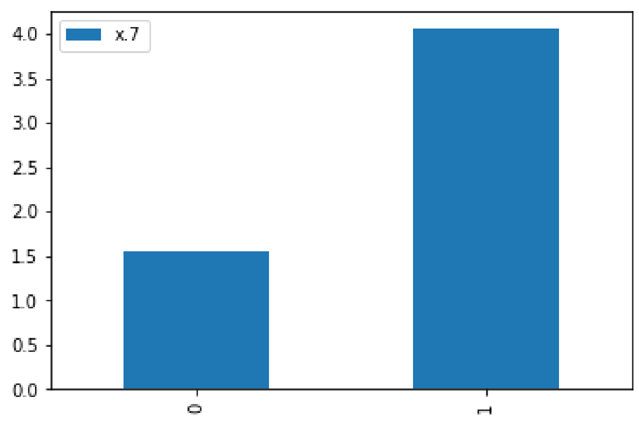


Figure 6: Groupby class mean for x.7

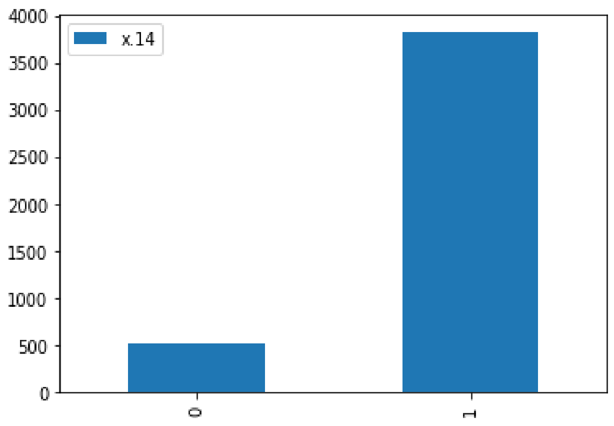
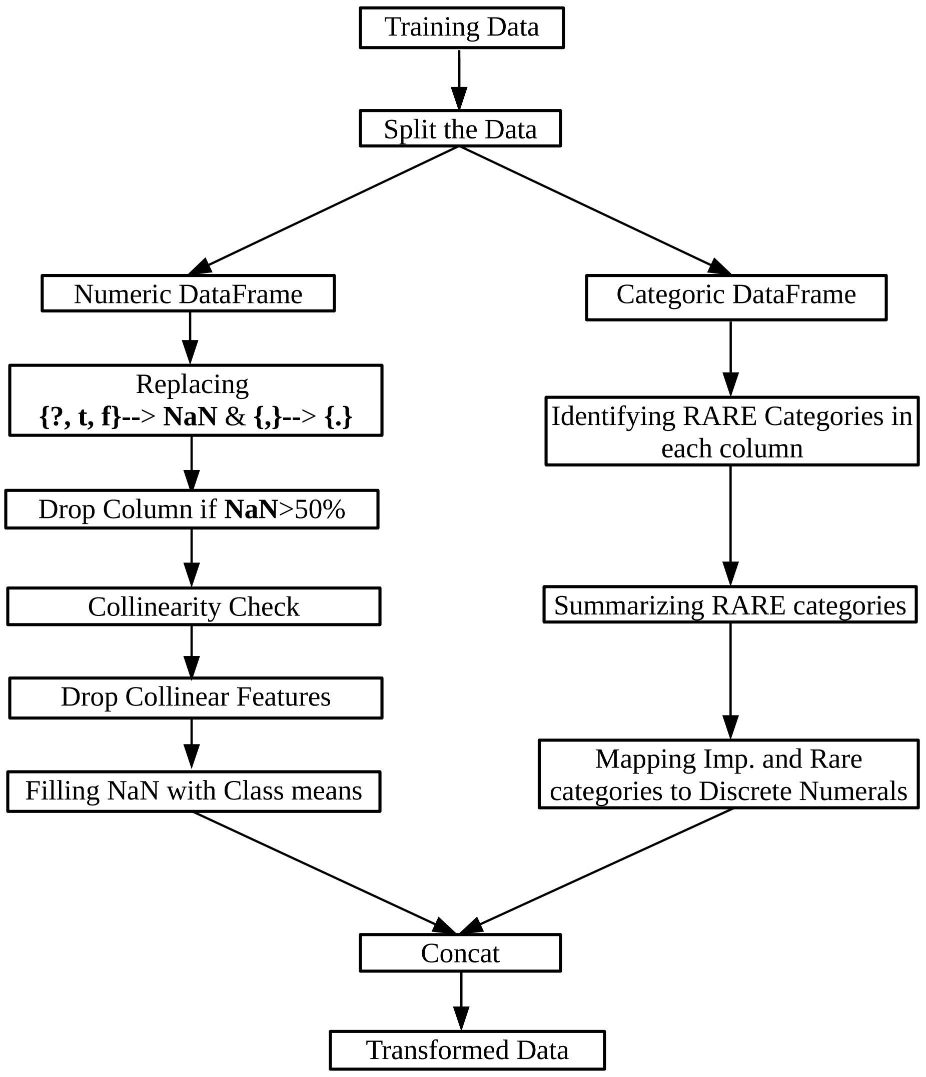


Figure 7: Groupby class mean for x.14

Figure 6 and **Figure 7** shows the Groupby means wrt Target Class of features **x.7** and **x.14** numeric feature. From the plots it is evident that it is better to replace **NaNs** in feature columns by **Group Means** instead of **Column Mean**.

SECTION 6: FEATURE ENGINEERING & FEATURE SELECTION



Training Data
Split the Data
Numeric DataFrame
Categoric DataFrame
Replacing {?, t, f}--> NaN & {,}--> {.}
Drop Column if NaN>50%
Collinearity Check
Drop Collinear Features
Filling NaN with Class means
Identifying RARE Categories in each column
Summarizing RARE categories
Mapping Imp. and Rare categories to Discrete Numerals
Concat
Transformed Data

The chart shows the flow of training data during transformation

One of the key findings of this work was multicollinearity in the numeric Dataframe. I found that some features are almost similar **x.1 ≈ x.17, x.2 ≈ x.18** and **x.13 ≈ x.19. Table 1** shows the correlation coefficients from the correlation matrix. So, I dropped these 3 features (**x.17, x.18** and **x.19**) from our training dataset.

Table 1: Multicollinearity in the Dataset

Feature 1	Feature 2	Corr. Coeff.
x.1	x.17	0.999
x.2	x.18	0.999
x.13	x.19	1.000

Table 2: Dropped features from the Dataset

Features	Reason of Dropping
Unnamed:0	Very high Cardinality (ID column)
x.17, x.18, x.19	Multicollinear

Table 3: Selected features

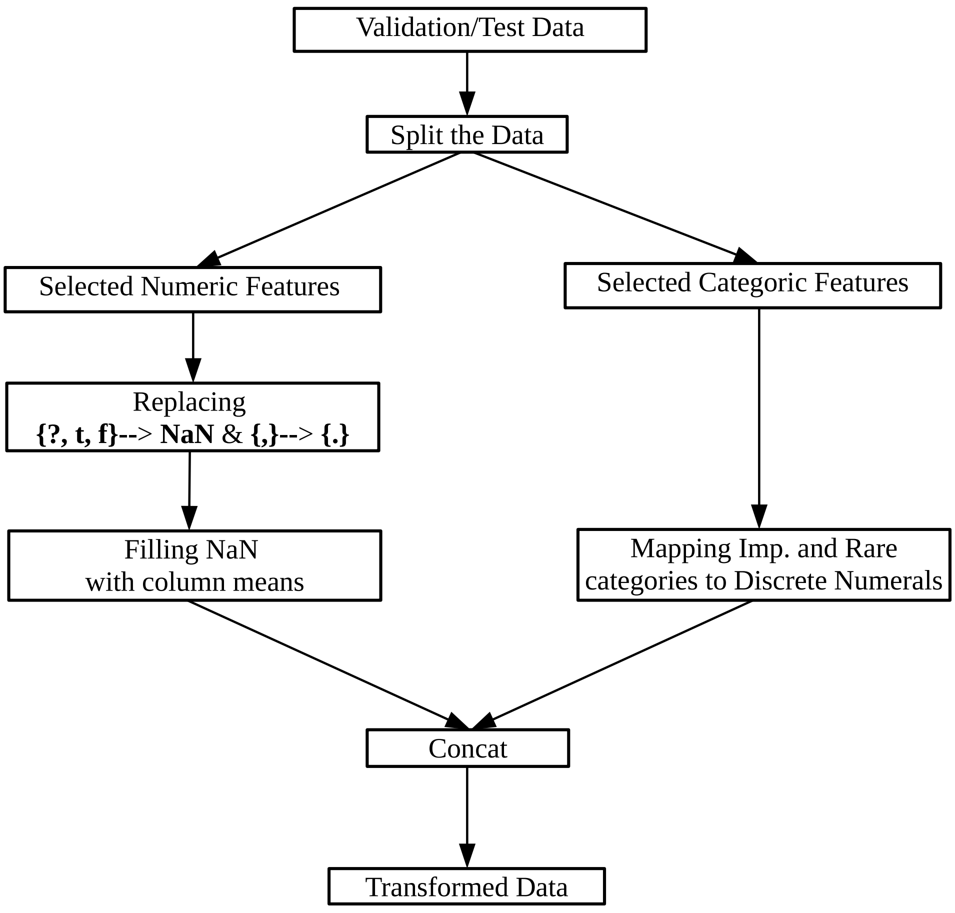
Type	Features
Numeric	{x.1, x.2, x.7, x.13, x.14}
Categoric	{x.0, x.3, x.4, x.5, x.6, x.8, x.9, x.10, x.12, x.20, x.16}

Table 2 and **Table 3** shows the dropped and selected features after feature engineering and data visualization.

Table 4: List of important and rare categories

Categorical Features	Important Categories	Rare/Summarized Categories
x.0	{a, b}	{?}
x.3	{u, y}	{?, l}
x.4	{g, p}	{?, gg}
x.5	{q, c, cc, w, x, i, aa, m, k}	{e, ff, d, ?, j, r}
x.6	{v, h, bb}	{ff, z, ?, j, n, o, dd}
x.8	{t, a, c}	{b}
x.9	{t, f}	{}
x.10	{t, f, 10, 2, 3, 11, 7, 5, 12}	{9, 14, 8, 15, 4, 10, 17, 40, 67, 23, 13, 20, 16}
x.12	{g, s}	{}
x.20	{t, f, ?}	{}
x.16	{t, f, ?}	{p}

Table 4 lists the important and rare categories of each categorical feature after applying the summarization (5% threshold). The Rare Categories column in the above Table will be categorised as ***'rare'*** category for each categorical features.



The chart shows the flow of val/test data during transformation.

SECTION 7: MODELS USED

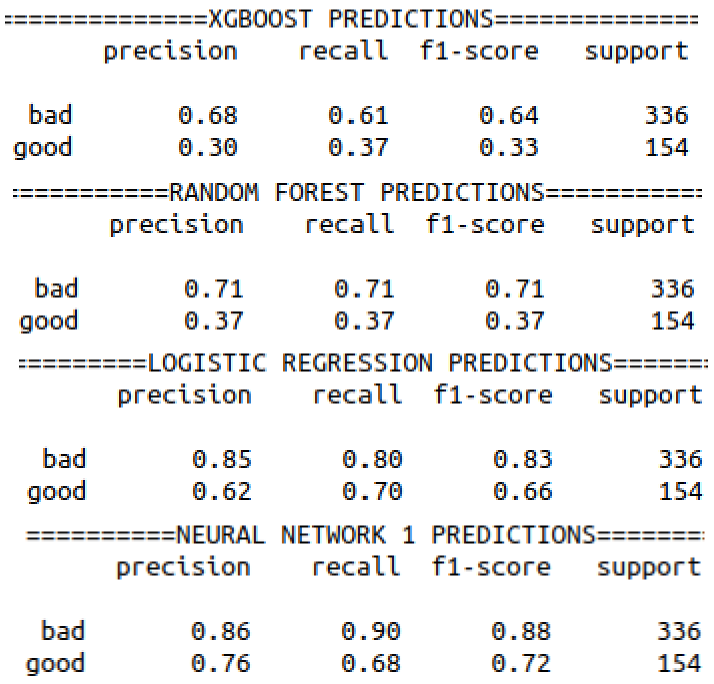
- **XGBoost:** Boosting is an ensemble method. It uses multiple models chained together. Every trees within boosting scheme is going to boost attributes that led to misclassification from previous trees. It basically built multiple trees on top of each other to correct the errors of the previous tree before it. It uses L1 and L2 regularisation to prevent overfitting. Unlike Decision trees where it stops branching once it stops seeing the benefit of it, XGBoost takes a different approach by actually going very deep by default then tries to prune that tree backwards. This results in deeper trees but highly optimised.

- **Random Forest:** It is a bagging algorithm which randomly selects subsets of features and build many decision trees. Since the each DT are not trained on all the features (or informations) and records random forest helps avoids overfitting. The majority votes is the decision taken by Random Forest after getting the decision of each DTs.

- **Logistic Regression:** It is similar to linear regression which predicts True or False instead of predicting something continuous. It uses logit function.

- **Neural Network:** Neural networks are used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions like (ReLU, sigmoid, tanh, etc). A neural network has 3 components: Input layer (no of features), hidden layers (no of computation layers) and the output layer. It is usually trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neuron’s coefficients or weights are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

SECTION 8: RESULTS



```
==============XGBOOST PREDICTIONS==============
       precision    recall  f1-score   support

 bad        0.68      0.61      0.64       336
good        0.30      0.37      0.33       154
===========RANDOM FOREST PREDICTIONS===========
       precision    recall  f1-score   support

 bad        0.71      0.71      0.71       336
good        0.37      0.37      0.37       154
=========LOGISTIC REGRESSION PREDICTIONS=======
       precision    recall  f1-score   support

 bad        0.85      0.80      0.83       336
good        0.62      0.70      0.66       154
==========NEURAL NETWORK 1 PREDICTIONS=======
       precision    recall  f1-score   support

 bad        0.86      0.90      0.88       336
good        0.76      0.68      0.72       154
```

Table 5: Comparison of Models

Classifier	Avg. F Score	AUC Score
XGBoost	0.49	0.71
Random Forest	0.54	0.72
Logistic Regression	0.77	0.85
Neural Network-1	0.80	0.87

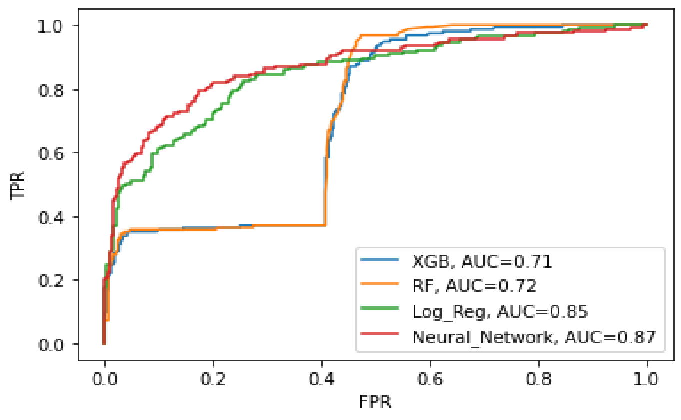


Figure 8: ROC curve

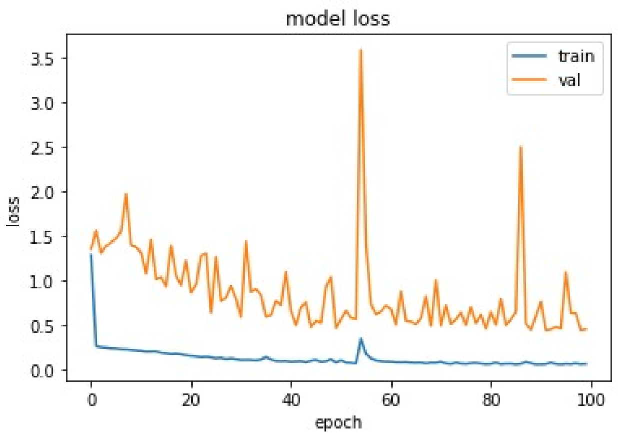


Figure 9: Training and Validation Loss

4 popular models were used for completing the binary task**.** From **Table 5** and **Figure 8** it is evident that Neural Network has outperformed the other Machine Learning models. ***0.80*** and ***0.87*** were the maximum ***Avg. Fscore*** and ***AUC score*** obtained respectively. The parameters used in the Neural Network are shown in **Table 6**. **Figure 9** shows the training and val loss of the model.

Table 6: List of parameters

Parameters	**Neural Network**
input dimensions	17
no. of hidden layers	3
activation function (Hidden Layers)	ReLU
activation function (Ouput Layer)	sigmoid
batch_size	16
epochs	100
optimizer	adam
loss	binary_crossentropy

SECTION 9: CONCLUSION

- Models like Random Forest and XGBoost cannot extrapolate. This means that they can only make a good prediction for situations already encountered in the training history. Hence both models overfitted.
- Linear Models like logistic regression have very few parameters compared to Random Forests. It means that Random Forests will overfit more easily than a Logistic Regression.
- Neural networks allowed to discover features and add nonlinearity unlike Logistic Regression. It can learn complexity in the dataset easily by updating its weights hence outperforming other ML Models.

SECTION 10: THREATS TO VALIDITY

- New/unseen categories in the test_data say (‘abc’, ‘xyz’, etc.) may affect the performance of the model.
- Training the Neural Network with epoch > 150 will overfit the model.
- Any voilation of Assumption will affect the performance of the model.
- Full hyperparameter tuning of XGBoost were not done.
- Categorical features independence tests using Chi square were not done.

SECTION 11: EXPERIMENTS WHICH DIDN’T WORKED

- Reducing the dimensions using PCA.
- Standardizing/Normalizing numeric features using standard scaling/log transform.
- Removing Outlier i.e., records which are above 99 percentile and below 1 percentile in the quantile range.

SECTION 12: EXPERIMENTS WORTH TRYING

- Since feature **x.1** has a comma between numbers. One can split it into 2 Numeric features.
- Feature **x.10** is a mix of numeric and categoric variables. This feature can be splitted into 2 separate ones.

**Contact 2ahmedabdullah@gmail.com for any queries*