CREDIT RISK MODELLING

ABHISHEK YADAV(19B090001) , ATUL VERMA(19B090004)
Dept of Mathematics, IIT BOMBAY

CREDIT RISK

Credit risk is the chance of a borrower defaulting on a debt by failing to make the required payments. It is the probability that the lender will not receive the principal and interest payments of a debt required to service the debt extended to a borrower.
Credit risk analysis is important because it helps in estimating the losses a firm might suffer in the event of a borrower's default. On the lender's side, credit risk causes disruption in cash flow and even increases the cash collection cost, since the lender may have to hire a debt collection agency to acquire the debt from the borrower.

PROJECT OBJECTIVE

Credit risk models look for behavioral patterns in factors ranging from payment history to current level of indebtedness to average length of credit history. In the current situation of technological advancement where we have thousands and millions of people coming to banks for loans, it is essential for the bank to be sure if the borrowers are able to repay the loan, otherwise, we have seen what had happened in the Great Recession of 2008, where banks kept on giving loans to people even with low credit scores & also against overvalued assets.
We will blend sophisticated models with robust data to build a model, which can predict if a client is going to default or not.

LIBRARIES USED

For the project following libraries are mainly used:
1) Pandas: This library is used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables. This library is also used to loading the datasets from different formats

2) Seaborn: Seaborn is an open-source Python library built on top of matplotlib. It is used for data visualization and exploratory data analysis. Seaborn works easily with dataframes and the Pandas library. The graphs created can also be customized easily.

3) Numpy: NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape

manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
4) Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

5) Scikit-learn (Sklearn): Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

6) Imblearn: Imbalanced-learn (imported as imblearn) is an open source, MIT-licensed library relying on scikit-learn (imported as sklearn) and provides tools when dealing with classification with imbalanced classes.

MODEL TRAINING

We are working with a data set containing 43 features for 45,000 clients. target_default is a True/False feature and is the target variable we are trying to predict. We'll explore all features searching for outliers, treating possible missing values, and making other necessary adjustments to improve the overall quality of the model.

All these features above have missing values that need to be treated. As we can see, they have skewed distribution, which is an indication that we should fill the missing values with the median value for each feature.

It's time to deal with the missing values from the remaining 32 columns. We are filling these values according to the particularities of each feature, as below:
- Categorical variables will be filled with the most recurrent value.
- Numerical variables will be filled with their median values.
- In the specific cases of last_amount_borrowed, last_borrowed_in_months and n_issues we'll fill the missing values with zero, as it is reasonable to believe that not every client would have values assigned to these variables.

After handling the missing values, case by case, we now have a data set free of null values.

We'll now preprocess the data, converting the categorical features into numerical values. LabelEncoder will be used for the binary variables while get_dummies will be used for the other categorical variables.

MODEL TESTING RESULTS:

1) XGBoost

Start by making some adjustments to the XGBoost estimator. XGBoost is known for being one of the most effective Machine Learning algorithms, due to its good performance on structured and tabular datasets on classification and regression predictive modeling problems. It is highly customizable and counts with a large range of hyperparameters to be tuned.

For the XGBoost model, we'll tune the following hyperparameters, according to the official documentation:

● n_estimators - The number of trees in the model
● max_depth - Maximum depth of a tree
● min_child_weight - Minimum sum of instance weight needed in a child
● gamma - Minimum loss reduction required to make a further partition on a leaf node of the tree
● learning_rate - Step size shrinkage used in the update to prevents overfitting

```
Best result: 0.6657327195142321 for {'n_estimators': 50}
```

```
Best     result:     0.6701307550047045     for     {'max_depth':     3,
'min_child_weight': 6}
```

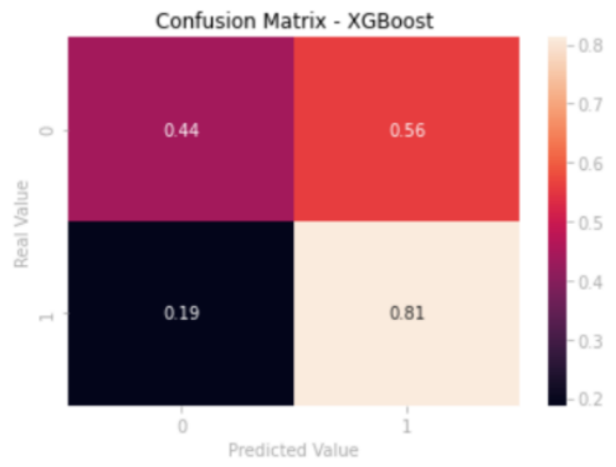
```
Best result: 0.6719385652158761 for {'gamma': 1}
```

```
Best result: 0.8170548699960465 for {'learning_rate': 0.0001}
```

```
              precision    recall  f1-score   support

           0       0.92      0.44      0.60      8771
           1       0.22      0.81      0.34      1665

    accuracy                           0.50     10436
   macro avg       0.57      0.63      0.47     10436
weighted avg       0.81      0.50      0.56     10436
```



2) LightGBM

Now, turning to the LightGBM model, another tree-based learning algorithm, we are going to tune the following hyperparameters, referring to the documentation:

● max_depth - Maximum depth of a tree
● learning_rate - Shrinkage rate
● num_leaves - Max number of leaves in one tree
● min_data_in_leaf - Minimal number of data in one leaf

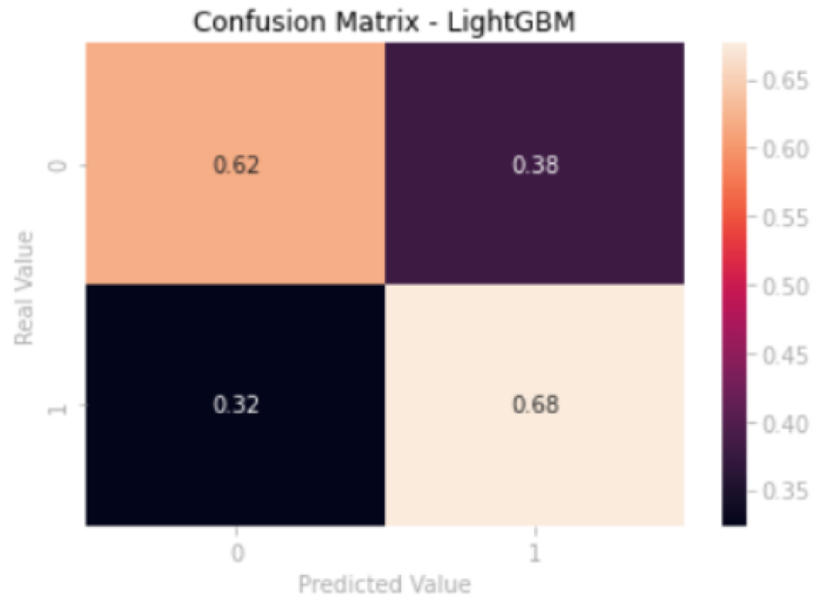
```
Best   result:   0.6883483723819858   for   {'learning_rate':   0.01,
'max_depth': 5, 'num_leaves': 70}
```

```
Best result: 0.6961582230489793 for {'min_data_in_leaf': 400}
```

```
              precision    recall  f1-score   support

           0       0.91      0.62      0.74      8771
           1       0.25      0.68      0.37      1665

    accuracy                           0.63     10436
   macro avg       0.58      0.65      0.55     10436
weighted avg       0.81      0.63      0.68     10436
```



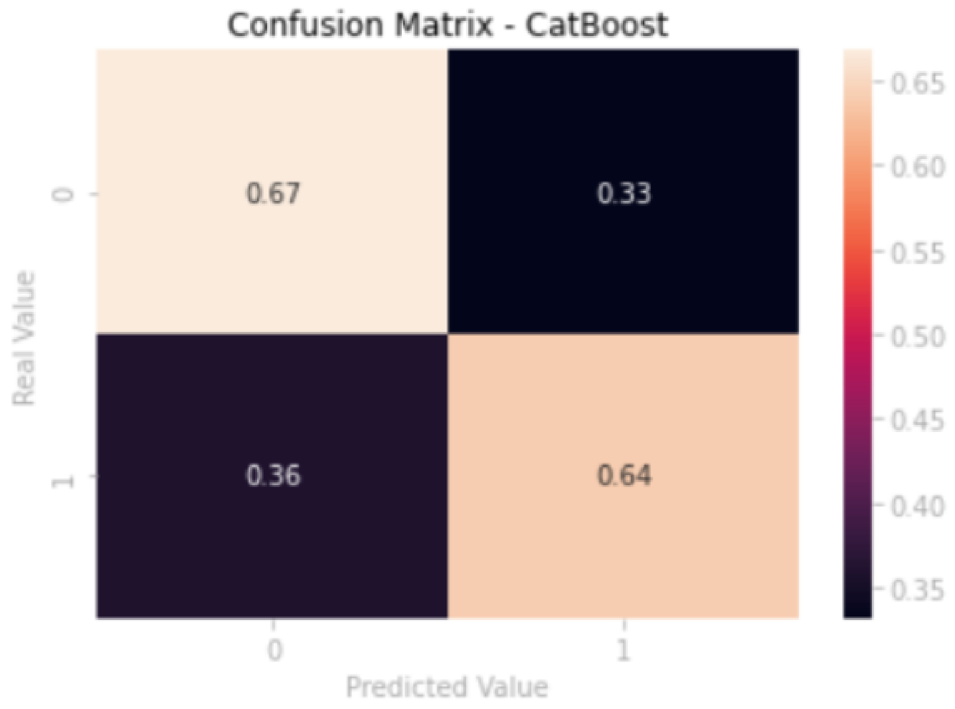
3) CatBoost
Lastly, we're going to search over hyperparameter values for CatBoost, our third gradient boosting algorithm. The following hyperparameters will be tuned, according to the documentation:

- depth - Depth of the tree
- learning_rate - As we already know, the learning rate
- l2_leaf_reg - Coefficient at the L2 regularization term of the cost function

```
              precision    recall  f1-score   support

           0       0.91      0.67      0.77      8771
           1       0.27      0.64      0.38      1665

    accuracy                           0.66     10436
   macro avg       0.59      0.65      0.57     10436
weighted avg       0.81      0.66      0.71     10436
```



CONCLUSION

The main objective was to build a machine learning algorithm that would be able to identify potential defaulters and therefore reduce company loss. The best model possible would be the one that could minimize false negatives, identifying all defaulters among the client base, while also minimizing false positives, preventing clients to be wrongly classified as defaulters.

Meeting these requirements can be quite tricky as there is a tradeoff between precision and recall, meaning that increasing the value of one of these metrics often decreases the value of the other. Considering the importance of minimizing company loss, we decided to give more emphasis on reducing false positives, searching for the best hyperparameters that could increase the recall rate.

Among the three Gradient Boosting Algorithms tested, XGBoost yielded the best results, with a recall rate of 81%, although it delivered an undesired 56% of false positives. On the other hand, LightGBM and CatBoost delivered a better count of false positives, with 38% and 33% respectively, but their false negatives were substantially higher than that of XGBoost, resulting in a weaker recall rate.