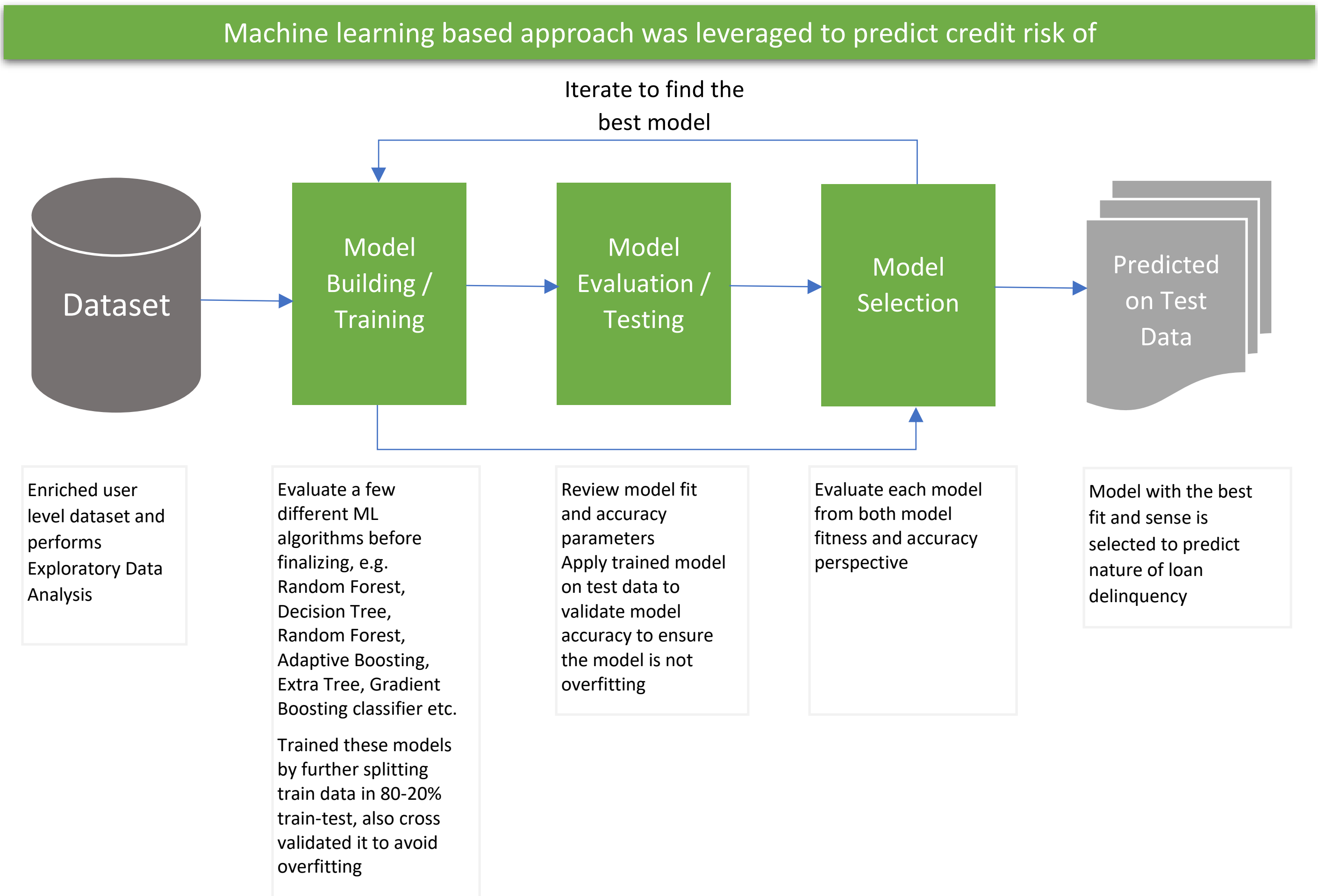
HackerEarth_HDFC_Bank_ML_Hiring_Approach_V1.00

This document covers below mentioned topics;

- Problem statement & Approach
- Data pre-processing
- Final model journey
- Key takeaways
- Appendix

1.) **Problem statement & Approach:**
  - The objective of this challenge is to predict the credit risk of loan delinquency based on user's liability account & their transactions. So, it's a **binary classification problem** (supervised learning).
  - Here, I started with the exploring the data, prepare the data and trained Random Forest model (with 10-fold cross validation and hyper-tuned it's parameters). High sparsity & imbalanced data labels problem were dealt with this ensemble technique (keeping "class_weight" parameter as "balanced"). Below scheme diagram will give you a nice overview of the approach that I followed.



Note: Above flow diagram would help in understanding interrelated information of all sequential steps that I took in order to build banking behavioural scorecard model for Internal Liability customers

2.) **Data pre-processing:** It's a state-of-art (very important for a model)

- 'Col1': unique key (dropped it from data)
- Target variable: 11% of users labelled as delinquent cases, which causes imbalanced in data

- **Categorical variables**: There were 14 categorical columns ('Col1', 'Col702', 'Col733', 'Col742', 'Col747', 'Col754', 'Col763', 'Col791', 'Col813', 'Col822', 'Col831', 'Col836', 'Col843', 'Col852');
    i. Col1: Unique ID column
    ii. Other columns were loaded as categorical variables because these columns had hyphen ("-") as datapoint. I treated these values as missing values. Also, >90% of these datapoints were missing in these columns

- **Numerical Variables:**
    i. All variables were numerical (int/float/bool)
    ii. Two major problems with some of these variables were missing values & outliers, which were treated as;
        - Missing values: Variables (222) which had more than 50% of missing values were dropped & rest of the variables were imputed by median, because most of them were having high skewness (mostly right) & kurtosis (mostly leptokurtic)
        - Outliers: Did Min-Max Scaling to get all variables in the range of 0-1
    iii. Over 2k+ variables: Most of the variables were just creating noise & had no significant impact over the target variable. So, I tried dimensionality reduction technique & with classification algorithm

3.) **Final model Journey:**

- **Model selection & measure:** After data preparation, I split the train data itself into train-test (80-20% rule) & tried simple classification models with their by-default parameters. All these models were giving more than 90% accuracy on both train & test data. The reason being imbalanced data, as these models are accurately measuring non-delinquency in data which resulted in high accuracy. So, I looked at f1-socre, area under the curve (AUC), receiver operating characteristic (ROC) curve, confusion-matrix, classification report (for both classes) & decide decision/threshold value

- **Hyper-tuning:** Logistic Regression & Random Forest were giving good f1-score. Later, I hyper-tuned these parameters of these 2 classification models with grid search method & turned out to be that Logistic Regression & Random Forest were giving f1-score of 85.73 & 86.54 respectively. Next, I looked out for overfitting for my best classifier (**Random Forest**) model by 10-fold cross validation, which gave me consistent results for both train & test data. It concludes that my model generalizes well

- **Final model output:** So, finally I choose Random Forest model to predict the credit score probability at the decision boundary of 0.61, where 999 users predicted as labelled 1 out of 20,442. So, parameters of this model were shown in below table;

Paramter	Value
bootstrap	TRUE
class_weight	'balanced'
criterion	'gini'
max_depth	10
max_features	'auto'
max_leaf_nodes	None
min_impurity_decrease	0
min_impurity_split	None
min_samples_leaf	20
min_samples_split	10
min_weight_fraction_leaf	0
n_estimators	200
n_jobs	None
oob_score	FALSE
random_state	29
verbose	0
warm_start	False)

4.) **Key takeaways:**

- Don't fall for sensitivity/specificity of a model in this type of imbalance data, instead look out of other metrics also, here for example f1-score, confusion-matrix, classification reports etc.
- Don't use algorithms which are memory intensive or taking a lot of time in training the model
- Here train data had only 17.5k records, so I didn't use highly complex model like extreme gradient boosting or light gradient boosting machine, as these models generalize well on data with millions of records.
- Here the data had anonymized features, so it was little difficult to understand the data well in order to get insights & importance of variables
- Data preparation & feature engineering is the most important part of building a model, so understand/explore the data first instead of using any algorithms blindly
- Use either ensemble methods or resampling techniques or modify the loss function to deal with imbalance data issue
- It's important to hyper-tune parameters of the model and even more important to set the classification/decision threshold value
- In trying your best, don't add complexity in the model, which eventually will lead to the overfitting problem

5.) **Appendix:** Few more valuable information regarding code file (python language) & libraries that I used in order to make this model. Packages used & their version:

- Pandas: 0.24.1
- Numpy: 1.16.2
- Matplotlib: 2.1.1
- Seaborn: 0.9.0
- Statsmodels: 0.10.1
- Sklearn: 0.21.3
- Imblearn: 0.5.0

Below mentioned functions & algorithms I used from these packages;

- import pandas as pd
- import numpy as np
- import matplotlib.pyplot as plt
- import seaborn as sns
- %matplotlib inline
- from statsmodels.stats.outliers_influence import variance_inflation_factor
- from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler
- from sklearn.preprocessing import Imputer
- from sklearn.metrics import f1_score
- from sklearn.model_selection import train_test_split
- from sklearn.linear_model import LogisticRegression
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier, ExtraTreesClassifier,GradientBoostingClassifier
- from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
- from sklearn.model_selection import cross_val_score, KFold
- from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
- from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
- import datetime,time
- import warnings
- warnings.filterwarnings("ignore")