Dev Joshi

Final Report : Credit Card Approval Prediction

Problem Statement

The personal information and data submitted by credit card applicants are used by the financial institutions to decide whether to issue a credit card to an applicant or not. Many of such applicants are rejected by the institutions for reasons such as high loan balances, low income levels, or too many inquiries on an individual's credit report, for example. Manual analysis of such applications is time-consuming and demanding. But, machine learning approaches can be applied to automate the task which many financial institutions are using these days. In this project, we build an automatic credit card approval predictor using machine learning techniques, just as the real banks do.

The dataset used in this project is the Credit Card Approval dataset from the kaggle website : https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction.

Data Wrangling:

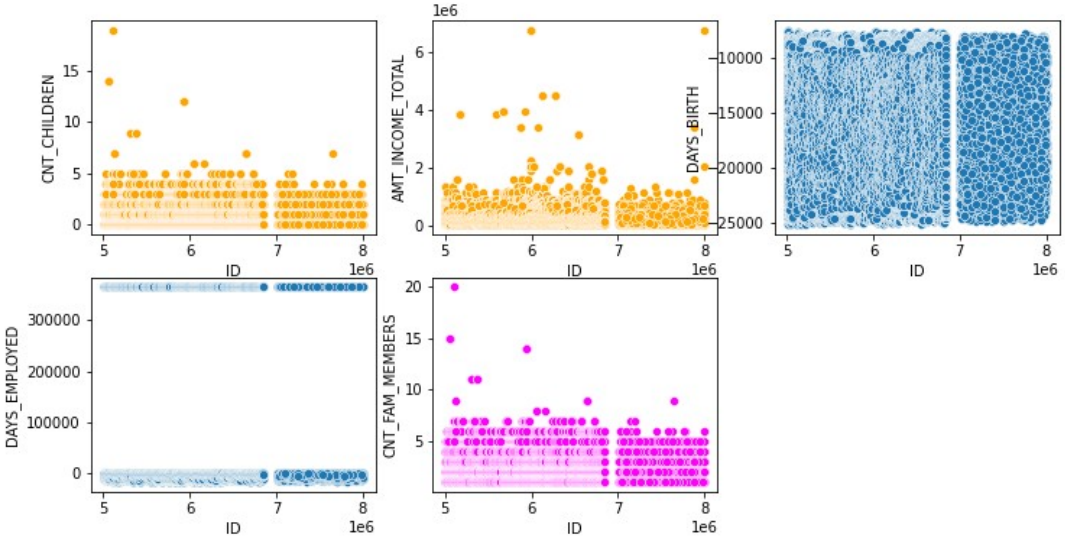
There are two datasets in the website – one with the application records and another with the credit-card records. We began by downloading both the datasets. The application record dataset has 438557 rows and 18 columns. The credit record dataset has 1048575 rows and 3 columns. We begin by looking at the column names of both the datasets and found they have a common column name ('ID') which could connect both the datasets. The length of intersection of two datasets is found to be 36457.

We explore the data and removed the duplicates in the application record dataset. In the same dataset, we also remove a column which had many missing data and few other columns which will be less useful in predicting a person's creditworthiness. The credit-card record dataset – with only three columns - has no missing values. We explore the data-types in the application record and convert the non-numeric data (object data-type) to numeric data using labelencoder. This will help in further analysis while applying the machine learning algorithms. Finally, we save the two datasets – with all the edits applied – in our folder in the local computer.

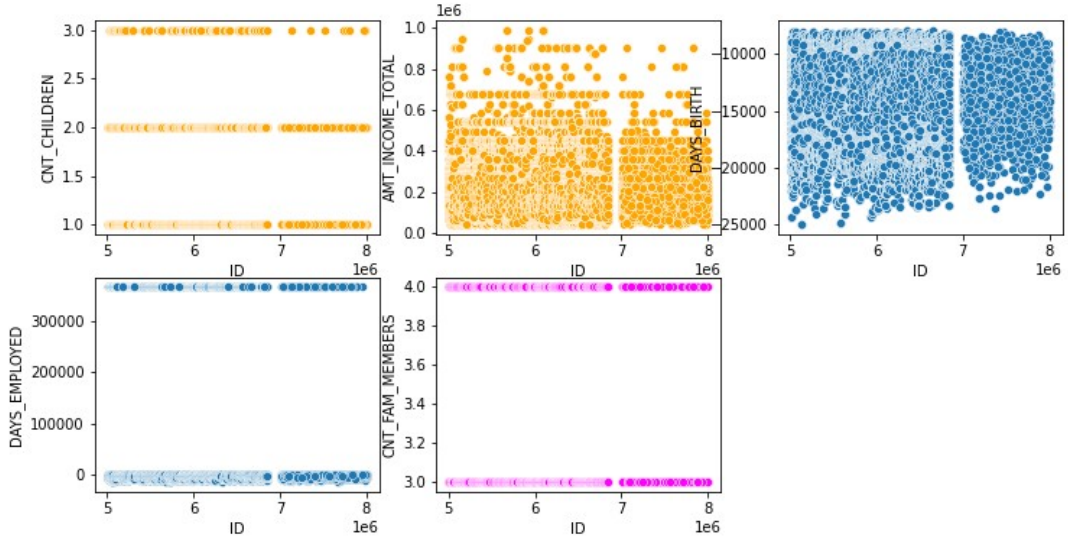
Exploratory Data Analysis:

We download the saved data in the previous section and explore the variables in the both data-sets in further detail.

We make a plot for few variables to check if there are outliers in the data. We observe there are outliers in ‘cnt_children’, ‘amt_income_total’, ‘cnt_fam_members’ as seen in the plot below.



We remove the outliers in these columns of the application record data-set and remake the above plot.



We check the credit-record dataset. The status column in this data-set is particularly interesting as it records the creditworthiness of a consumer into eight categories :

0: 1-29 days past due
1: 30-59 days past due
2: 60-89 days over due
3: 90-119 days overdue
4: 120-149 days overdue
5: Overdue or bad debts, write-offs for more than 150 days
C: paid off that month
X: No loan for the month.

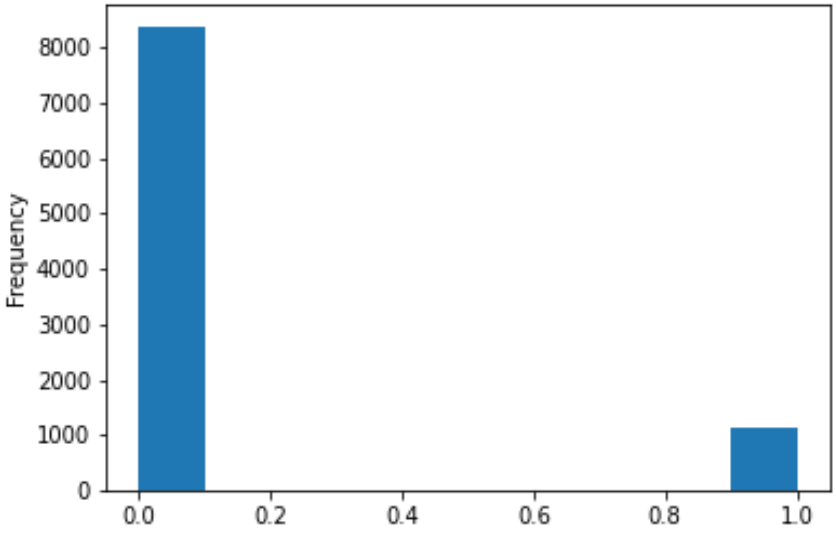
We regroup these categories in only two : 0 (creditworthiness – 0. C, X) and 1 (no credit-worthiness : 1, 2, 3, 4 , 5). We regroup the credit-record dataset loan applicant id so that it will help to link it with the application-record dataset. We make the 'ID' as the index of the credit-record dataset table.

We save these processed data-sets for further analysis in the next step.

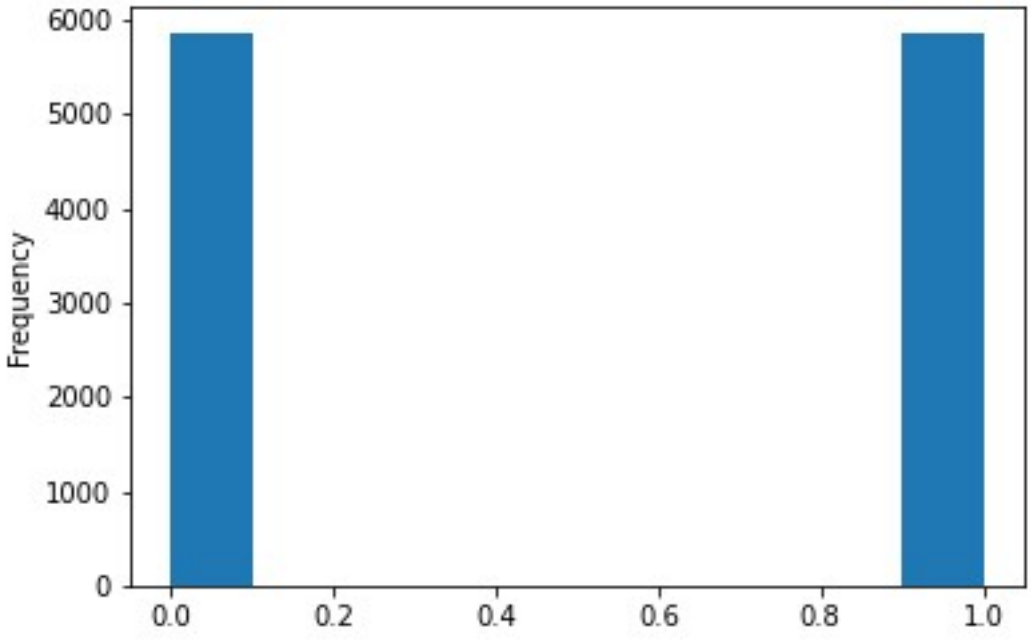
Preprocessing and Training:

We download the data-files saved in the previous step and join the application record and credit record datasets with ID as the common column. They have 9516 common 'elements'. The joined data-set has 9516 rows and 15 columns. The 'status' – which is our target 'variable' – has 87.97 % as 0 (creditworthiness) and 12.03 % as 1 (no credit-worthiness). The creditworthy category has larger population than the non-creditworthy. So, it suffers oversampling which could be corrected either by decreasing the first category or by increasing the second category. Decreasing the first category isn't good as it will lead to further loss of data. So, we choose to increase the second type of data using SMOTE (Synthetic Minority Over-sampling Technique). This technique helps to "generate" new data and avoid overfitting the model. We divide the data into those that will be used to train the model and those that will be used to predict the approval : 70 % for training and 30 % for testing. After this division, we apply the SMOTE algorithm to both train and test data which creates equal number of 'creditworthiness' and 'non-creditworthiness' data-points.

The creditworthiness (0 or 1 in the x-axis) is plotted below before the application of the 'SMOTE' algorithm.



The creditworthiness for the test data – after splitting for testing and training – after the application of 'SMOTE' algorithm is seen below. It shows the algorithm has helped to address the lack of data.

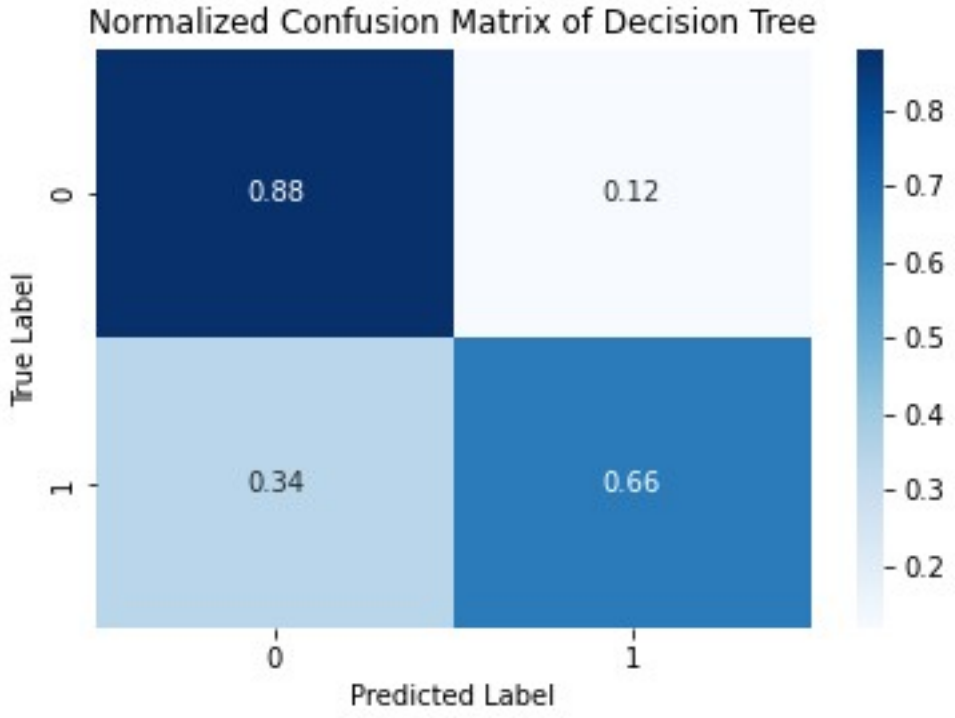


Finally, in this step, we save the splitted and SMOTE algorithm applied train and test data.

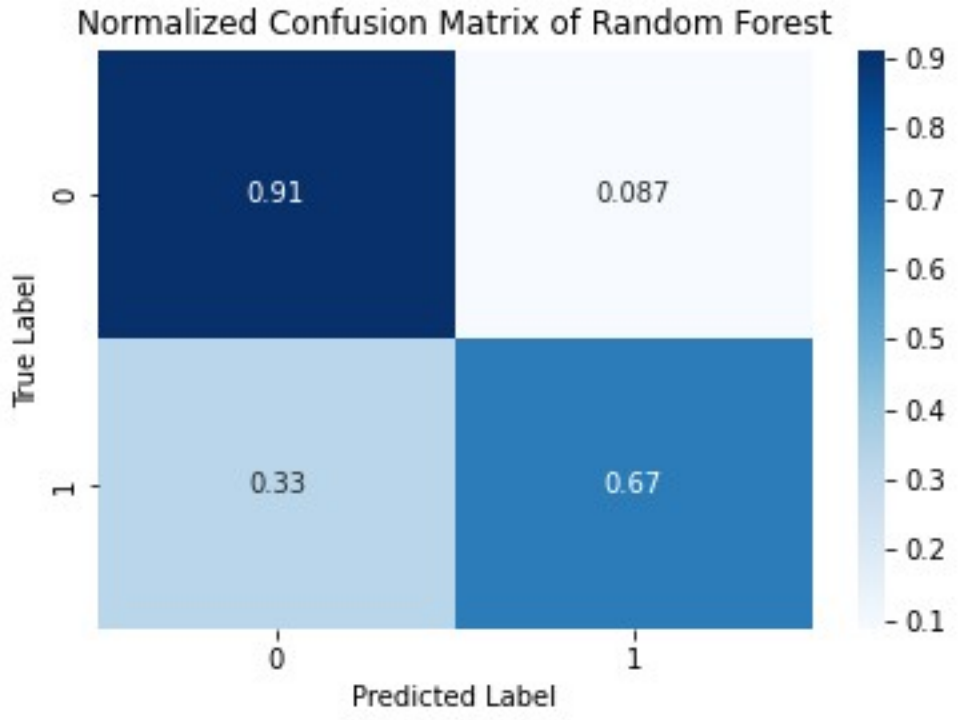
Modeling:

We download the data saved in the previous step and apply various machine-learning models to the data.

We begin with the Decision Tree Classifier and plot the confusion matrix. As is known, the confusion matrix summarizes the performance of a machine learning model on a set of test data. The plot displays True Negatives, False Positives (upper row) and false negatives , True Positives (lower row). The accuracy score from this confusion matrix is 0.77. There are other metrics such as Precision , Recall and F1-score which could be calculated from the confusion matrix.

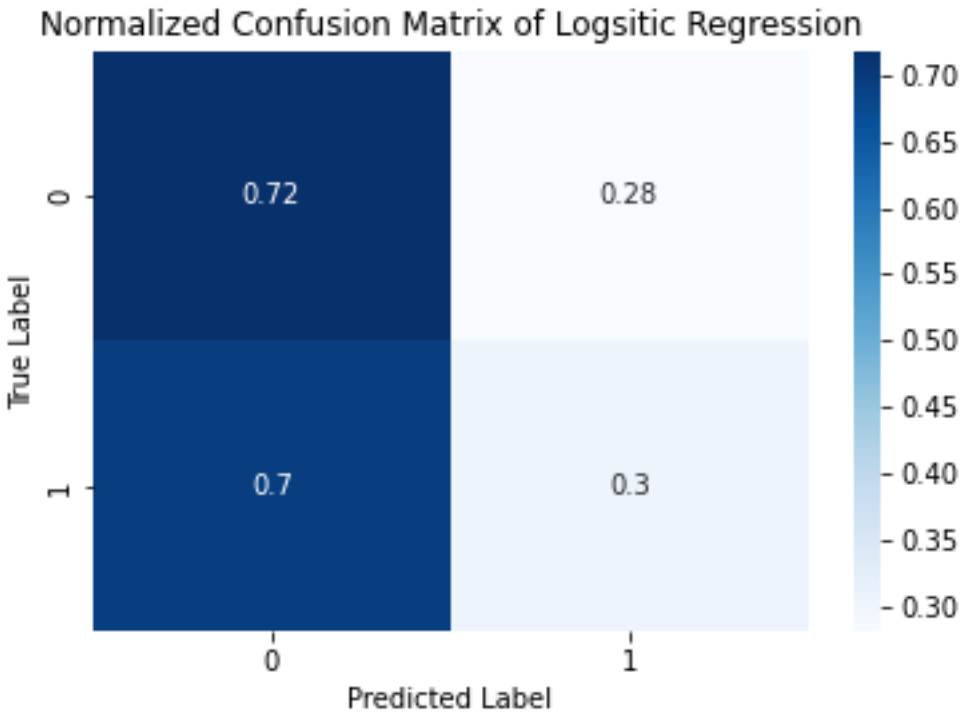


We then moved to Random Forest modeling and repeated the same process as above to plot confusion matrix for this model. We fit the model on the training set of the data and use the model to make prediction using the test data. We use this predicted data and true test data for the target variable to build the confusion matrix. The accuracy for the Random Forest model is ~ 0.79.



The Random Forest model was built with three variables : n_estimators = 150, max_depth = 16, min_samples_leaf = 12. We employed GridSearchCV to get the optimum values of these parameters and found n_estimators = 300, max_depth = 20 and min_samples_leaf = 9 . We refit the model using these parameters and get the accuracy to be 0.80 which is close to accuracy we got before employing GridSearchCV. It means there wasn't any significant improvement with GridSearchCV and the model was initially run with the optimal paramets.

We then repeat the process of plotting the confusion matrix with the Logistic Regression model. The confusion matrix is seen below. We get the accuracy of this model to be 0.511. .



Based on the accuracy values , the Random Forest model is the most accurate.

We can look at other metrics as well such as Precision , Recall and F1-score to make a detailed comparison between the models. In that case we ought to compare between precision and recall. If the precision of a model is high, it suggests that the classifier is returning accurate results (high true positives), but at the cost of missing a number of actual positives (low recall). In another case, if the recall is high, it means the classifier is returning most of the positive results, but with a number of false positives. The trade-off between precision and recall will also depend upon the threshold chosen for the classification in the model. The precision-recall trade-off is a very challenging problem for data scientists to solve when working with imbalanced data and in some cases, precision should be prioritized while in other cases recall should be prioritized, there is no universal right or wrong answer. This can be future work in this investigation.