Vehicle Loan Default Prediction

Financial institutions incur significant loss when borrowers default on their loan. In the case of vehicles, each loan consists of many features about the car (i.e. car manufacturer), the borrower (i.e. income, loan to value, credit) and the branch information. ***Our objective is to identify the most important features attributed with faulty loans and build a classification model that predicts whether or not a borrower defaults.***

Data

Our data source is Kaggle, which provides us with the following csv files:

1. *train.csv* - information about the borrower, the loan and the bureau data and history. Our data shape has 233154 rows and 41 columns (features). We will be building and testing our model on this dataset.

2. *data.dictionary.csv* - file that provides a word description of each feature.

Objective

We will be predicting the *loan_default* feature that indicates whether or not the borrower made a timely first equated monthly installment (EMI) payment. The value 0 indicates the borrower made the EMI payment on time and 1 indicates he/she did not. A loanee institution makes money by correctly identifying both since we want to avoid rejecting borrowers who will make timely payments and reject those who will not. The f-1 score will be a key metric for this evaluation, which will be discussed in subsequent sections.

1. Data Cleaning

The first step in our analysis involves cleaning our dataset. Apart from the usual data observations like unique counts, the changes on the original data set were the following:

1. *Missing values*: only the *Employment.Type* contained missing values; more specifically, about 3.28% of the features were null. Since it is a small percentage, I decided to replace the null values with 'Other', giving us 'Salaried', 'Self employed' and 'Other' for the unique values.

2. *Dates:* For the features *Date.of.Birth* and *Disbursal.Date,* I built a short function fixYear() that transforms the year component (which was given as the last two digits) to its full year: for instance, 31-07-85 becomes 31-07-1985. This is to avoid confusion later on in the case that we might want to transform the date into a datetime object readily.

2. EDA

My goal in the exploratory data analysis was to identify the distinguishing features between the two *loan_default* classes. More specifically, I wanted to know if a feature distribution for a defaulter class did NOT match the feature distribution for the non-defaulter class. If the distributions did not match, then the feature is a distinguishing one and hence important in differentiating defaulters from non defaulters.

The steps in our EDA consisted of the following:

1. Plot the histograms for numerical features and bar plots for categorical features.
2. Separate dataset to non-defaulters *(loan_default = 0)* and defaulters *(loan_default = 1)*.
3. Perform the independent t-test (for numerical features) and the chi-square test (for categorical features) on the non-defaulter and defaulter datasets. This is key in identifying the important features.
    a. *Independent t-test*: This is a difference of means (for a feature) between two samples (defaulters and non defaulters). If the returned p-value < 0.05, we reject the null hypothesis that the two sample means are the same. We want to find numerical features that return a p-value < 0.05.
    b. *Chi-square test*: This tests whether the defaulter and non-defaulter distribution for a categorical feature matches. The higher the value, the less the two samples match. Hence we are looking for p-values that are significantly high. The closer the p-value is to 1, the more important the feature is for loan default prediction.
4. For the newfound important features, identify any trends (i.e. average feature value for non-defaulter and defaulter, threshold values etc.).
5. Plot a heatmap to show correlations.

Notable Features

***Numerical feature results*:** While there were many numerical features whose p-value < 0.05 for the difference of mean t-test, we look at three main features, *disbursed_amount, ltv* and *PERFORM_CNS.SCORE* whose p-values were essentially 0. For each of them, we show the average and standard deviation for both defaulter and non-defaulter classes, the t-test results, the histogram for the feature and violin plots for the feature partitioned by the *loan_default* classes. The plots are shown in Figure 1, Figure 2 and Figure 3.

Insight: <u>The vehicle loan creditor should look into setting higher credit score requirements for larger loan amounts.</u> We observe from the data defaulters have higher *disbursed_amount* and *ltv* values while having lower *PERFORM_CNS.SCORE* values. This means *that defaulters take larger loans and have higher loan to car value ratios BUT have lower credit scores*. The underwriting team should look into ensuring that a higher credit score checks for obtaining a larger loan.

***Categorical feature results**:* While there were many categorical features of importance based off of our chi-square test, we observe four major features, *manufacturer_id, branch_id,* and *State_ID* which all had a p-value extremely close to 1. We predict that these features will be extremely important for our model.

Based on the bar plots alone, the categorical features display a significant difference between the default and non-default class. Thank the Lord for the chi-square test to help in this regard.

```
Nondefaulter mean for disbursed_amount : 53826.47111091633 , std: 13140.699007454747
Defaulter mean for  disbursed_amount :  56270.47386931695 , std: 12150.255527172361

Ttest_indResult(statistic=39.32291321262725, pvalue=0.0)
```

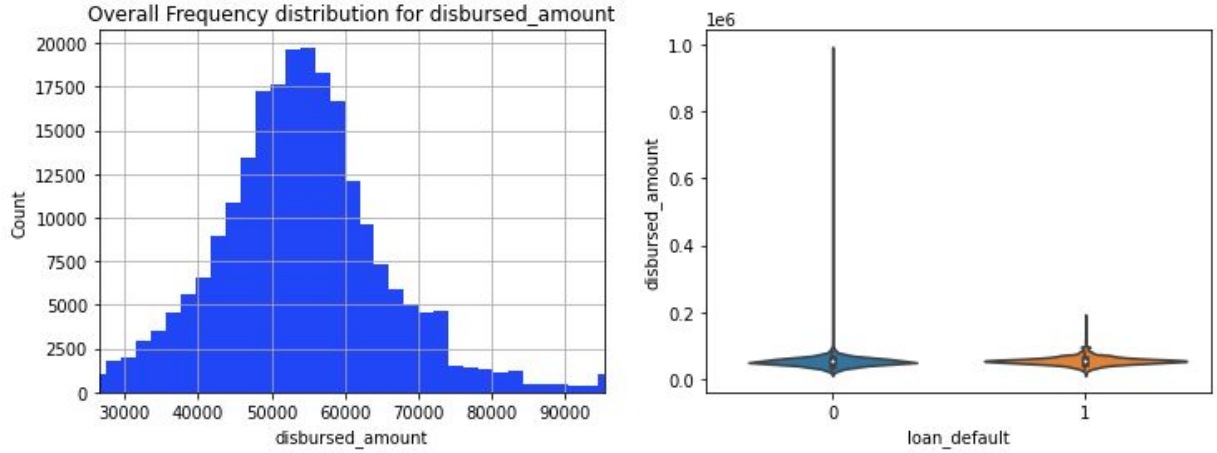


Figure 1. Plots and difference of mean t-test results for *'disbursed_amount'*. Histogram on the left. Violin plot on the right partitioned by our *'loan_default'* predictor classes. The low p-value indicates that there is a difference of means for *'disbursed_amount'* distributions between the defaulters and non-defaulters.

```
Nondefaulter mean for ltv : 74.15409333691578 , std: 11.681454560472389
Defaulter mean for  ltv :  76.88332180751246 , std: 10.327771446422924

Ttest_indResult(statistic=51.07804645618371, pvalue=0.0)
```

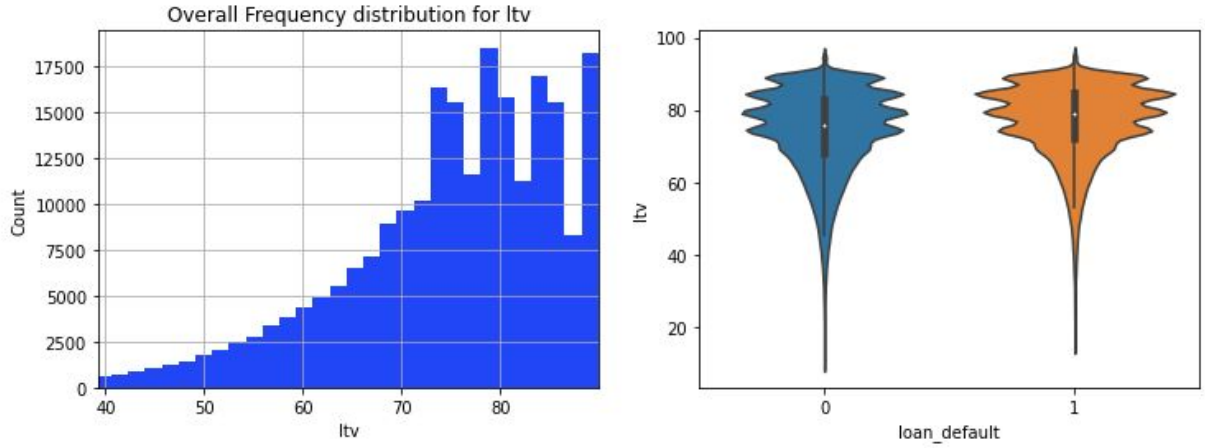


Figure 2. Plots and difference of mean t-test results for *'ltv'* or loan to value. Histogram on the left. Violin plot on the right partitioned by our *'loan_default'* predictor classes. The low p-value indicates that there is a difference of means for *'ltv'* distributions between the defaulters and non-defaulters.

```
Nondefaulter mean for PERFORM_CNS.SCORE : 590.6796912947271 , std: 244.5933876500854
Defaulter mean for  PERFORM_CNS.SCORE :  541.8708349250817 , std: 247.84156027494095

Ttest_indResult(statistic=-27.06155806056678, pvalue=1.1001651810636842e-159)
```

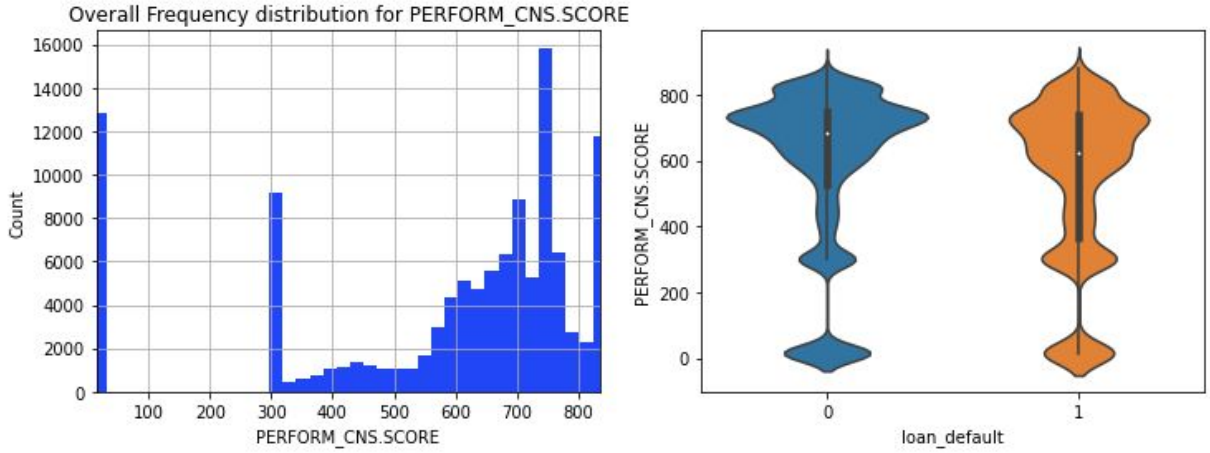


Figure 3. Plots and difference of mean t-test results for *'PERFORM_CNS.SCORE'* or the credit score. Histogram on the left. Violin plot on the right partitioned by our *'loan_default'* predictor classes. The low p-value indicates that there is a difference of means for the credit score distributions between the defaulters and non-defaulters.

3. Preprocessing

The preprocessing section consisted of four steps:

1. Convert non-numerical features into numerical which were the *Employment.Type, AVERAGE.ACCT.AGE* and *CREDIT.HISTORY.LENGTH* features. (ex. 'Other', 'Salaried', 'Self employed' -> 1,2,3)
2. Use the SMOTE package to balance our dataset by oversampling the minority class (*loan_default* = 1).
3. Split our dataset into training and test sets.
4. Apply StandardScaler() to our independent features X, to standardize our features to have mean equal to 0 and unit variances away from the mean

Note about dummy variables: Initially, I converted all the categorical variables to dummy features. This made our dataset too large for efficient model computation (some categorical features had close to a thousand unique values). Next, I converted features into dummies only if the unique value count was less than 10, which led to a less accurate model based off of the f-1 metric. I was not able to figure out this reason and due to a time constraint, I decided to convert all categorical features to numerical features which gave accurate results.

4. Models: Hyperparameter Tuning with GridSearchCV

The primary goals in this section were to build potential models through hyperparameter tuning. I built three decision tree models due to being able handle complex and large datasets. Each model consisted of grid search cross validation to obtain good parameters. Then we try to analyze if these are any overfitting

tendencies with parameter values. I will show the confusion matrix of the chosen model in section 5 for the sake of time.

Model 1: Single Decision Tree, Entropy vs Gini

For the single decision tree, I performed a grid search cross validation (CV) with 'gini' and 'entropy' for the criterion parameter and max_depth (tree depths) values from 5 to 25. Figure 4 represents the average CV test scores for all combinations of our parameters. The maximum average cv score occurs at max_depth = 16 and criterion = entropy. The criterion however is arbitrary as the two follow relatively the same pattern.

To avoid overplotting, we plot the training and test average CV scores for *max_depth* (decision tree depth) values of 5, 6, 7, …. 25. Figure 5 shows the results. We see that the overplotting trend starts for *max_depth* > 8. I decided to use a *max_depth* value of 10 as the overfitting isn't too significant and we obtain a CV test score closer to 0.75.

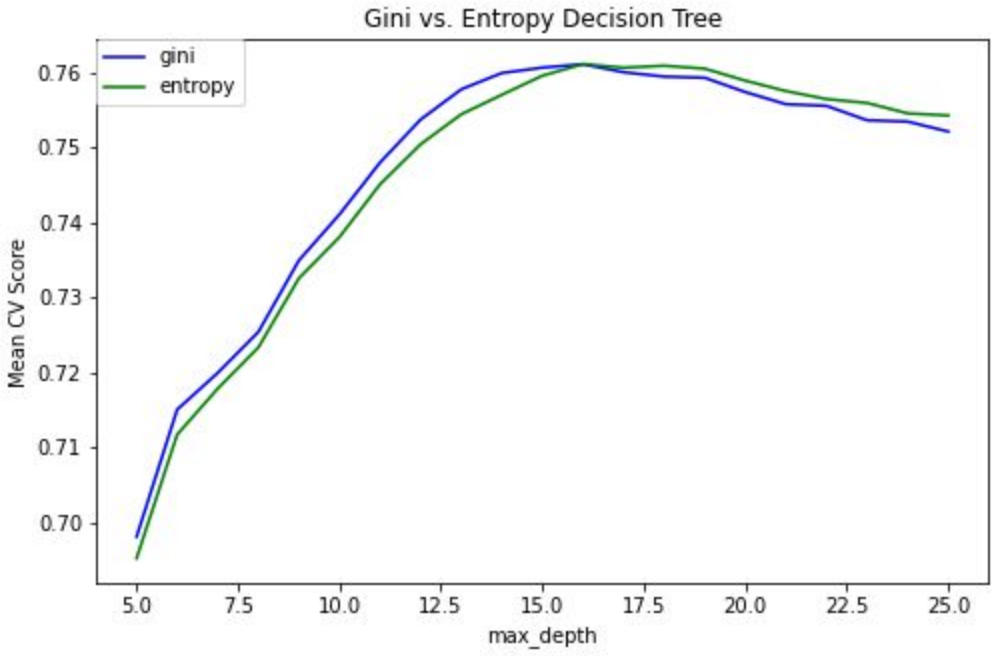


Figure 4. Grid search cross validation for our decision tree classifier with parameters *criterion* = ['gini', 'entropy'] and *max_depth* = [5, 6, 7 … 25]. The *'gini'* and *'entropy'* models are interchangeable as their mean cross validation (CV) test score is virtually the same. The parameters associated with highest CV score are *criterion* = *entropy* and *max_depth* = 16. We check on the plot in figure 5 to determine if a *max_depth* of 16 is overfitting.

Figure 5. Training and test CV score for various *max_depth* lengths. We see that our training starts to overfit around a *max_depth* value of 8. Therefore, we do not want to use the value 16, from the grid search. Any max_depth value from 8 to 10 seems reasonable.

Model 2: Random Forest Classifier

The random forest classifier aggregates many decision trees to obtain our accuracy metrics. We implemented a grid search cross validation with the following parameters: *bootstrap* = [True], max_depth = [5, 10, 15, 20], *n_estimators* = [35, 50, 100, 150]. We plot the mean cross validation score for each combination of these parameters. The results are shown in Figure 6. The parameter *max_depth* has a larger impact on mean CV test score than does *n_estimators*. The best estimators of the grid search return a *max_depth* value of 20 and *n_estimators* value of 150.

To avoid overfitting, we plotted the train vs test accuracy scores for various *max_depth* values of 5, 10, 15 ,20, 30, 50, and 100 to observe where the overfitting trend starts. Figure 7 shows the result. We examine the model starting to overfit after a *max_depth* value of 10. We conclude that the model should have *max_depth* of 10 and *n_estimators* of 150.

Note: We evaluated test vs training CV scores for various *n_estimators* and saw a constant CV score regardless of the number of decision trees. Therefore, we go with the best *n_estimator* parameter value of 150 obtained from the grid search CV.

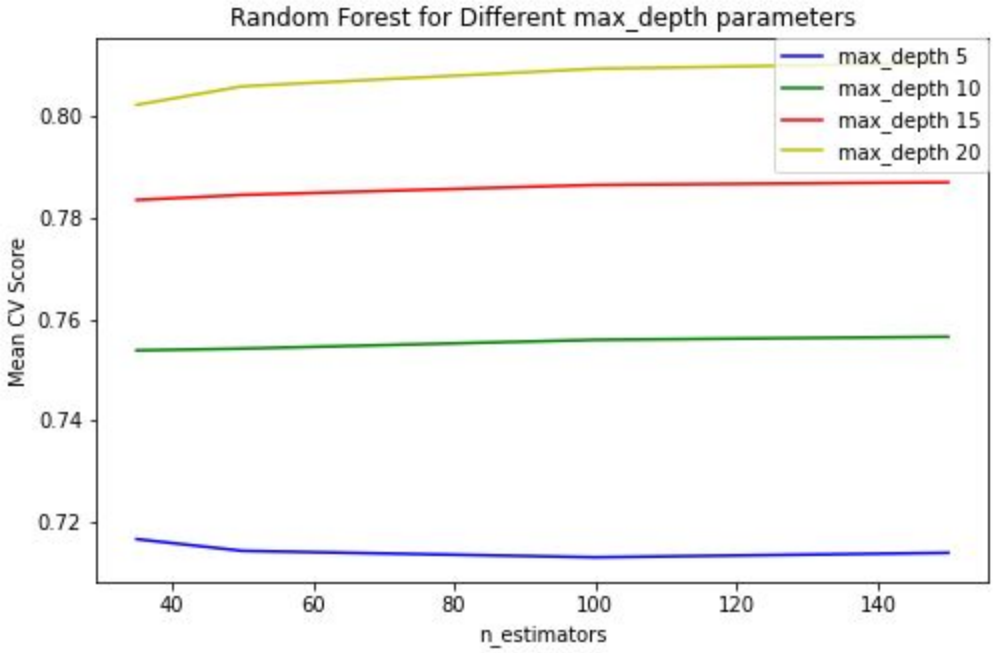


Figure 6: Grid search cross validation for our Random Forest classifier with parameters *max_depth* = [5,10,15,20], *n_estimators* = [35,50,100,150]. The parameters associated with the best CV score are *max_depth* = 20 and *n_estimators* = 150. We want to verify that the *max_depth* of 20 is not overfitting. Check Figure 7.

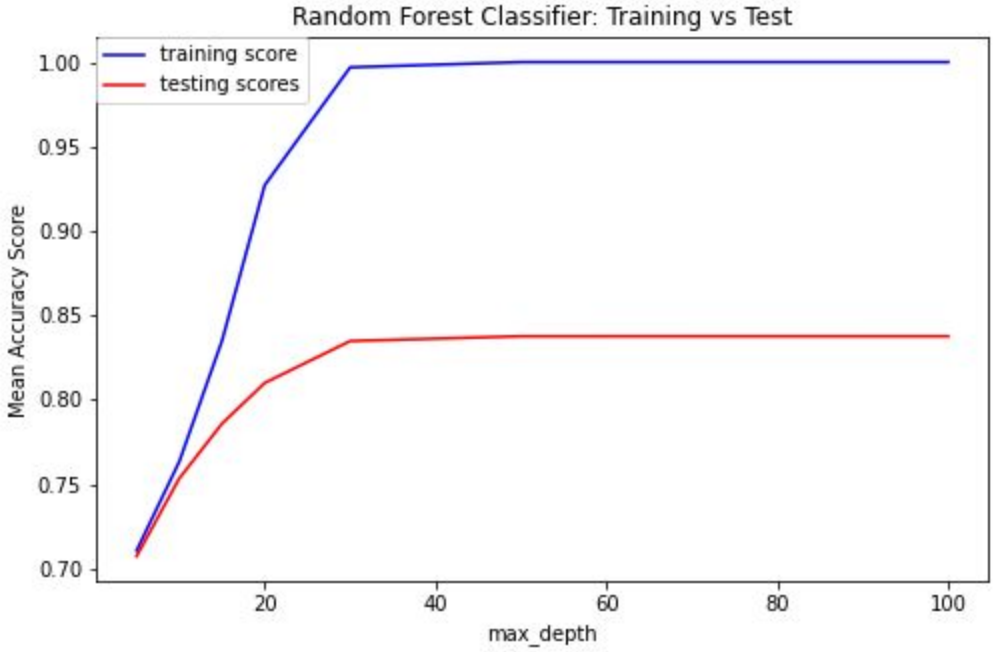


Figure 7: We see that the Random Forest Classifier model starts to overfit around a *max_depth* value of 10. The n_estimators have a small effect on changing the CV scores. We will choose *max_depth* = 10 and *n_estimators* = 150 as parameters for our best Random Forest model.

Model 3: Gradient Boosting Classifier

The final model is the Gradient Boost Classifier Decision Tree model. We tune the n_estimators and learning rate parameters in the grid search and plot their mean CV scores. Our grid search cross validation consisted of the following parameters:'learning_rate' = [0.1, 0.25, 0.5, 0.75, 1], 'n_estimators' =[20, 35, 50, 75]. To save on computation time, I capped the n_estimators to 75. Figure 8 shows the results of the plot. There seems to be an overall increasing trend for all learning rates. The best estimators with the highest CV score was a *learning rate* of 1 and an *n_estimator* value of 75.

Again to avoid overfitting, I plot the training and test mean CV scores for various learning rates of 0.1, 0.25, 0.5, 0.75, and 1. While a *learning_rate* of 1 was a good result, I wanted to verify if there were any overfitting trends. Figure 9 shows the training and testing scores. While the plots don't display a significant overfitting trend, we want to be aware that a higher learning rate can result in a convergence to a suboptimal CV value. Based off of the figure, a *learning_rate* of 0.25 seemed like a solid parameter for our model. We build our gradient boosting model with *learning_rate* of 0.25 and an *n_estimators* of 150 (same number of trees as the Random Forest).

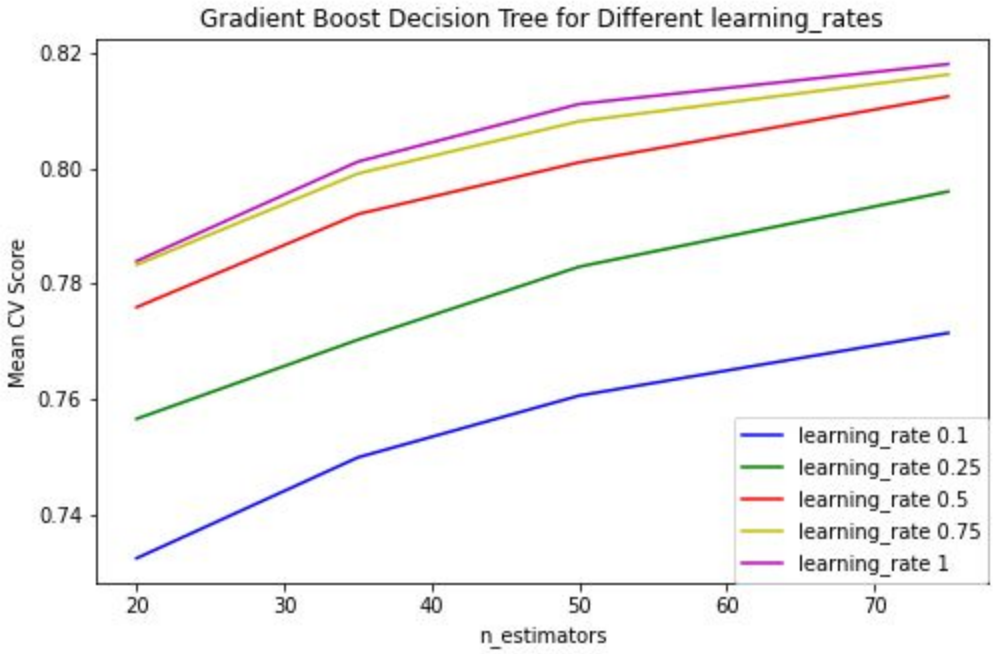


Figure 8: Grid search cross validation for our Gradient Boosting classifier with parameters *learning_rate* = [0.1, 0.25, 0.5, 0.75, 1], *n_estimators* = [20, 35, 50, 75]. The parameters associated with the best CV score are *learning_rate* = 1 and *n_estimators* = 70. Since we want to avoid a learning rate that is too high, we will check for overfitting. Check Figure 9.

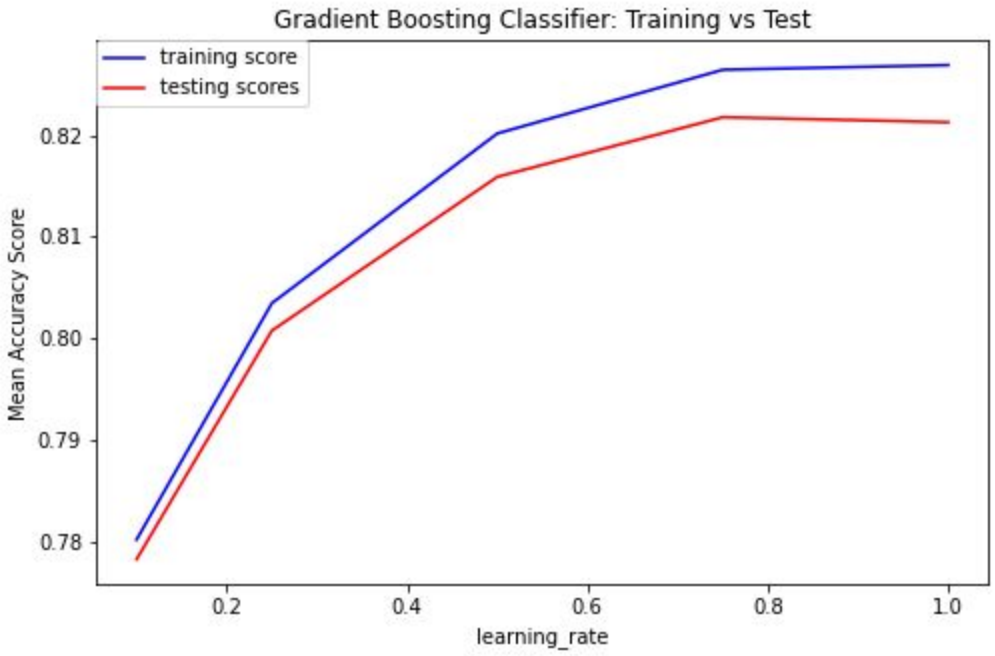


Figure 9: Training and test CV score for various *learning_rate* values. While there isn't a significantly large case of overfitting, a *learning_rate* = 0.25 shows similar CV scores for the training and test sets. Therefore, our Gradient Boosting Classifier Model will be built with the parameters *learning_rate* = 0.25 and *max_depth* = 150 (same max_depth as our Random Forest).

5. Final Model

Best Model and Confusion Matrix

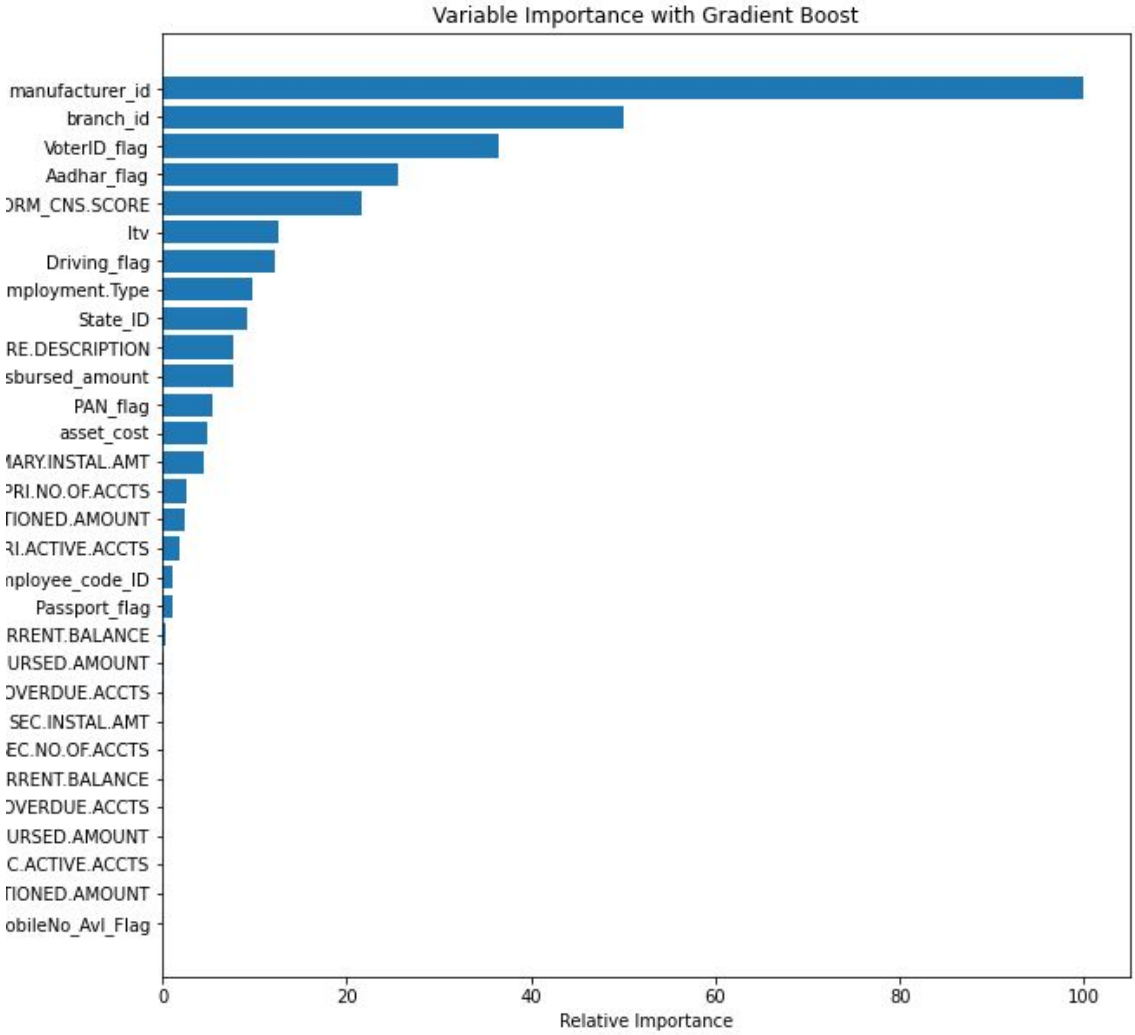
The best model is measured through the f-1 score statistic which takes a weighted average of the precision and recall scores for predicting whether or not a borrower defaults. The model with the highest f-1 scores was model 3 or the Gradient Boosting Classifier. The chart below shows the model accuracy, f-1 score and the confusion matrix for our Gradient Boost Classifier with a *learning_rate* = 0.25, *n_estimators* = 150 and *max_depth* with the default value of 3.

```
Decision Tree: Accuracy=0.811
Decision Tree: f1-score=0.809
              precision    recall  f1-score   support

           0       0.75      0.92      0.83     36561
           1       0.90      0.70      0.79     36457

    accuracy                           0.81     73018
   macro avg       0.83      0.81      0.81     73018
weighted avg       0.83      0.81      0.81     73018
```

We obtain a f-1 score of 0.809. Predictions of class 1 (defaulters) have a high precision of 0.90 and a sufficient recall score of 0.70. While we can obtain better f-1 scores, this comes at the cost of overfitting and not generalizable to new data. Since we optimized the parameters through cross validation and comparing training vs test CV scores, I am confident in this model being able to bring about accurate results to new datasets.

Most Important Features

In the exploratory data analysis (EDA) section, we identified both numerical and categorical features that seemed important for distinguishing between the defaulter and non-defaulter class. The Gradient Boost model provides us with a table for verifying which features were important. The table below shows the results.



In the EDA section, we had specifically identified '*manufacturer_id', 'branch_id', 'State_ID', 'PERFORMANCE_CNS.SCORE'*, '*disbursed_amount',* and *'ltv'* as important features due to its p-value significance from the independent t-test and Chi-square test for numerical and categorical features respectively. The table verifies that they are indeed important features, namely '*manufacturer_id'* and *'branch_id'*.

How the Vehicle Loan Company Can Utilize this Model

There are three practical ways the company can use the results of this model.

1. *Important Features Identification:* The chart from Section 5 shows which features were important in distinguishing between defaulters and non-defaulters. The vehicle loan company can use this information to revise their underwriting process.
2. *Default Prediction:* The company can use our model as a final measure when they are not entirely sure if an applicant is risky. When all the underwriting procedures have been conducted and a final decision can't be made, the model can be used to predict if the applicant will default.
3. *Credit Score Discrepancy:* In the exploratory data analysis section, I pointed out that defaulters on average had lower credit scores but larger approved loan amounts than the non-defaulters. While there are many factors that go into a loan applicant, I would expect that on average you would need a higher credit score to achieve larger loans. I would look into why this is the case.

Future Improvements and Suggestions

Future improvements on this project involve the following.

Data Visualization for Underwriting Improvements

We had inferred before that defaulters obtain larger loans even though they have lower credit scores than non-defaulters. It was difficult to identify the reasons for why this was occurring as visualizations like scatter plots and linear regression didn't provide us with any real insight. Important features like *'manufacturer_id'* and *'branch_id'* display similar bar plots for the two predictor classes. I'd like to explore data visualization of how all these important features interact together in distinguishing between our classes which will guide us to look into underwriting improvements.

Feature Engineering and Sampling

We had an extremely large dataset which did not allow us to create dummy variables for our model. My next step in improving this project is to look into principal component analysis that would allow us to represent most of the data with a subset of the features. I would also look at training increasing samples of our training data set and observe a convergent value for the test scores.

Sparse Matrix

We can also create dummy variables and store them in a sparse matrix which can save storage space and computation time. Having dummy variables could potentially improve our model accuracy while not overfitting.

Credit

I'd like to thank my Data Science mentor Shmuel Naaman for his patience and his practical domain knowledge especially in the Exploratory Analysis and Hyperparameter Tuning sections. I'd also like to thank SpringBoard for providing the relevant analytics tools and exercises that were of tremendous help in the whole capstone project progress.