Machine Learning Engineer Nanodegree

Capstone Project

Chirag Chadha
July 13, 2018

I. Definition

Project Overview

The Home-Credit-Default-Risk competition focuses on using alternative data on loan applicants, that wouldn't necessarily be considered by banks and other financial institutions, to predict their ability to repay said loan. In doing so, Home-Credit can correctly identify individuals who are able to repay their loans but are also unlikely to receive loans elsewhere – giving the company an advantage over their competitors. This project is a current Kaggle competition with a $70,000 prize amount for winners. The following data was supplied by Home-Credit for the purposes of the competition:

- application_train and application_test – these are the main tables containing static features such as gender, number of children, amount of loan, etc. for each applicant
- bureau – previous loans for the applicant provided by other financial institutions that were reported to the Credit Bureau
- bureau_balance – monthly balances of these previous loans
- POS_CASH_balance – monthly balance of previous point of sales and cash loans that applicants had with Home-Credit
- credit_card_balance – monthly balance of previous credit cards that applicants had with Home-Credit
- previous_applications – all previous applications for Home-Credit loans by applicants
- installments_payments – repayment history for previously disbursed loans in Home-Credit for applicants

Problem Statement

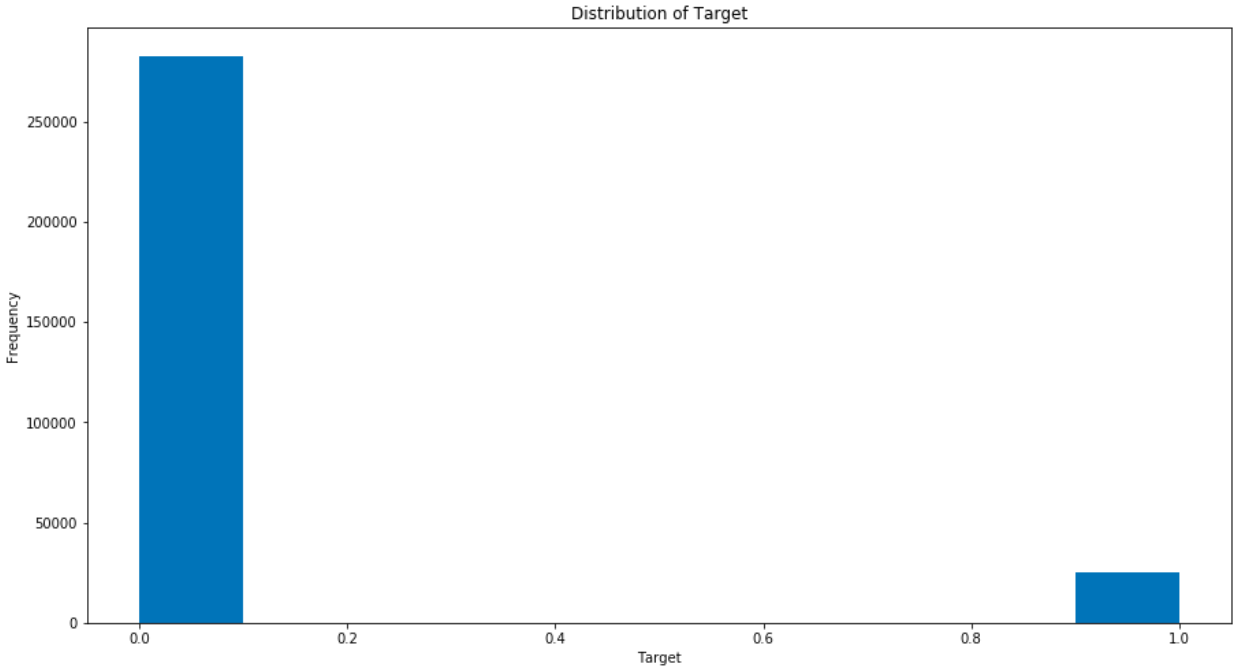
Home Credit Group are looking to determine whether an applicant will be capable or repaying a loan based on a number of alternative factors that banks and other financial institutions do not analyze. In addition to static information on applicants, they provide auxiliary tables relating to their credit history. As we are predicting whether an applicant is capable of repaying a loan, this is a binary classification problem. The process for finding a solution can be broken down as follows:

- Perform exploratory analysis on static data and pre-process
- Analyze auxiliary tables in terms of distribution of variables, missing values, anomalies, etc.
- Merge relevant information from auxiliary tables into main table by creating new relevant features, aggregating existing ones, and encoding categorical features
- Create functions from data pre-processing stage to streamline merging of tables into static data

- Fit LightGBM model to processed training data and predict on processed test data
- Make Kaggle submission to find area under ROC curve score
- Create validation set from training set on which the same LightGBM model achieves a similar score with a view to obtaining a smaller dataset on which the model can be quickly tweaked and re-assessed
- Refine model parameters to achieve a better area under ROC curve score

Metrics

A quick analysis of the distribution of the target in the training data shows a clear class imbalance:



We can see from above that most of the loan applicants in our sample were unable to repay their loans. As we previously learned in the Machine Learning Nanodegree, accuracy would be a bad metric in this scenario since we could build a "model" that classifies every individual as being unable to repay their loan and still get high accuracy.

F-score could work well, especially with a beta value of less than one. Precision is more important than recall in this case since we don't want to give out loans to people that will be unable to repay them. However, the metric we will use to evaluate our model is the *area under receiver operating characteristic* curve (AUROC). The justification for choosing this metric is due to the evaluation process on Kaggle for this competition also using AUROC to score competitors' models. As a result, we can:

- easily compare our model to the baseline random forest model provided in the Kaggle competition
- create a local validation set that can be compared to the test data on Kaggle in terms of the AUROC score obtained by the same model on each

Area under receiver operating characteristic as a metric can be explained as follows:

1. The output from our binary classification model will be a prediction between 0 and 1 (with higher values meaning greater confidence in the individual being a positive classification – or somebody that will likely be able to repay their loan)
2. In binary classification, a discrimination threshold is set for each prediction, above which we consider that prediction to be a 1 or positive. This threshold can be set anywhere between 0 and 1
3. For a number of these thresholds, plot (true positives/all positives) and (false positives/all negatives) ratios on the x and y axes respectively and calculate the area under the curve that results

This area is the AUROC score.

II. Analysis

Data Exploration

The main application table with static features contains 122 columns including the target we are looking to predict. The test set contains the same columns but excludes the target. Their shapes are as follows:

```
application_train.csv shape: (307511, 122)
application_test.csv shape: (48744, 121)
```

Due to the sheer number of columns in these tables, it is difficult to present a sample of the data but the dataset can be viewed in-browser at the following URL: https://www.kaggle.com/c/home-credit-default-risk/data or downloaded at https://www.kaggle.com/c/home-credit-default-risk/download/application_train.csv

And an explanation of the columns can also be found here: https://www.kaggle.com/c/home-credit-default-risk/download/HomeCredit_columns_description.csv

A quick glance at the number of missing values in each column shows significant portions of the dataset are missing with some of the columns containing as much as 69% missing values. I assume the reason for the high number of missing values is that each individual filled out their applications themselves and many of the fields must have been left blank/not required. It is important to note however that the essential fields such as gender, type of loan, etc. did not contain any missing values.

Looking at the correlation between all of the features and the target shows the following highest negative correlating features:

```
EXT_SOURCE_3                -0.178919
EXT_SOURCE_2                -0.160472
EXT_SOURCE_1                -0.155317
```

And the following highest positive correlating features:

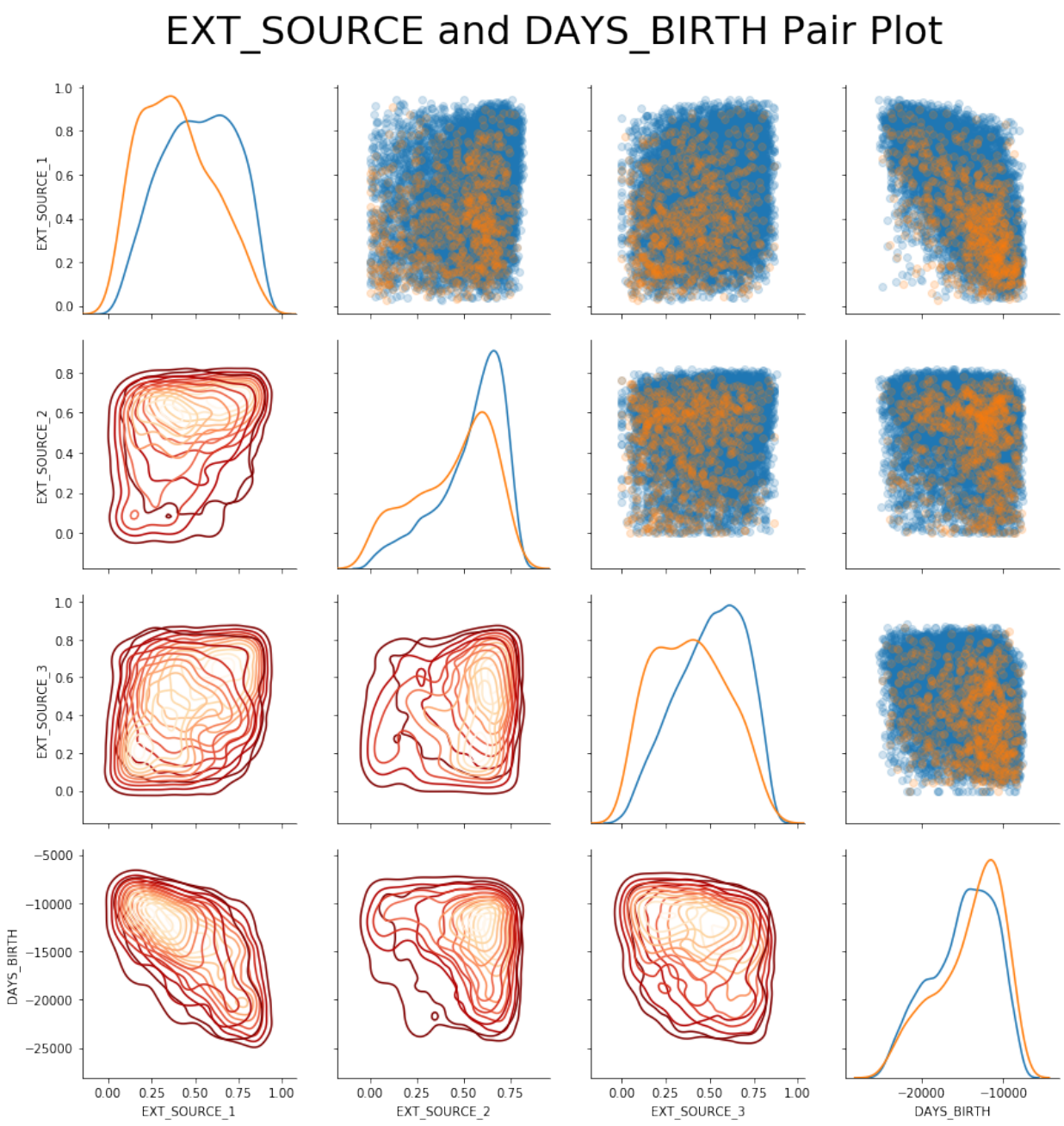
```
REGION_RATING_CLIENT           0.058899
REGION_RATING_CLIENT_W_CITY    0.060893
DAYS_BIRTH                     0.078239
```

EXT_SOURCE features are ratings for the applicant from other financial institutions, DAYS_BIRTH is the applicants age in days relative to the application date (negative value), and REGION_RATING_CLIENT and REGION_RATING_CLIENT_W_CITY is the rating of the region where the applicant resides.

Correlation between 0 and 0.2 can be considered weak but the three EXT_SOURCE features will still likely play an important role in prediction since they show the most significant correlation. Each of the three features also contain some amount of missing values with EXT_SOURCE_1 missing the greatest amount:

```
EXT_SOURCE_1 missing value counts: 173378, 0.56% of column missing
EXT_SOURCE_2 missing value counts: 660, 0.0022% of column missing
EXT_SOURCE_3 missing value counts: 60965, 0.20% of column missing
```

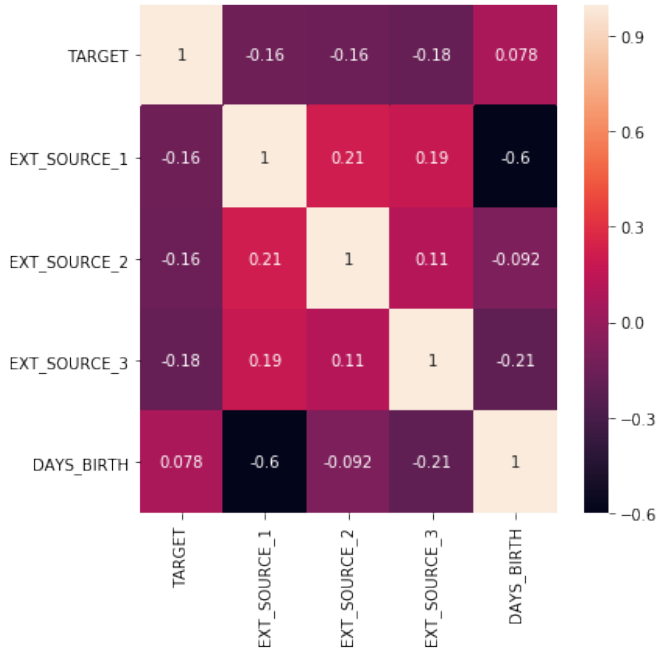
Looking at a pair plot of the three EXT_SOURCE features and DAYS_BIRTH shows the following:



We can see the concentration of orange points in each plot where applicants did repay their loan and the blue points where applicants did not repay their loan. There is a small correlation between DAYS_BIRTH and EXT_SOURCE_1. EXT_SOURCE_1 here appears to be a good predictor for ability to repay with a clear difference in KDE plots between those who could repay (orange hue) having a high density around 0.25 and those that couldn't repay (blue) having a high density around 0.5. However, as we saw above, EXT_SOURCE_1 is missing a lot of values so that may decrease its feature importance somewhat but this will likely still be a useful variable for us in prediction.

EXT_SOURCE_2 has the lowest number of missing values by far (660) but the distribution of values between those who could and could not repay their loans is largely similar. The use of all three of these scores will likely still be important in prediction despite them appearing to be weak factors, since there is some correlation between them and the target

Looking at a correlation map between the external sources and the age of the applicant in days shows the following:



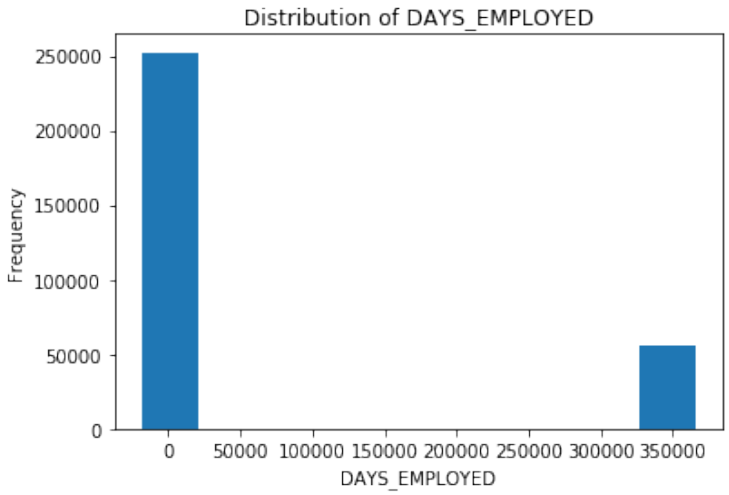
EXT_SOURCE_1 and DAYS_BIRTH show a correlation meaning that age is likely a factor in the creation of the EXT_SOURCE_1 rating for applicants.

Exploratory Visualization

Looking elsewhere in the dataset, DAYS_EMPLOYED (the number of days the applicant has been employed prior to application – this is a negative value) shows an extremely high max so there's likely an error in recording values here. The following are some statistics on the column:

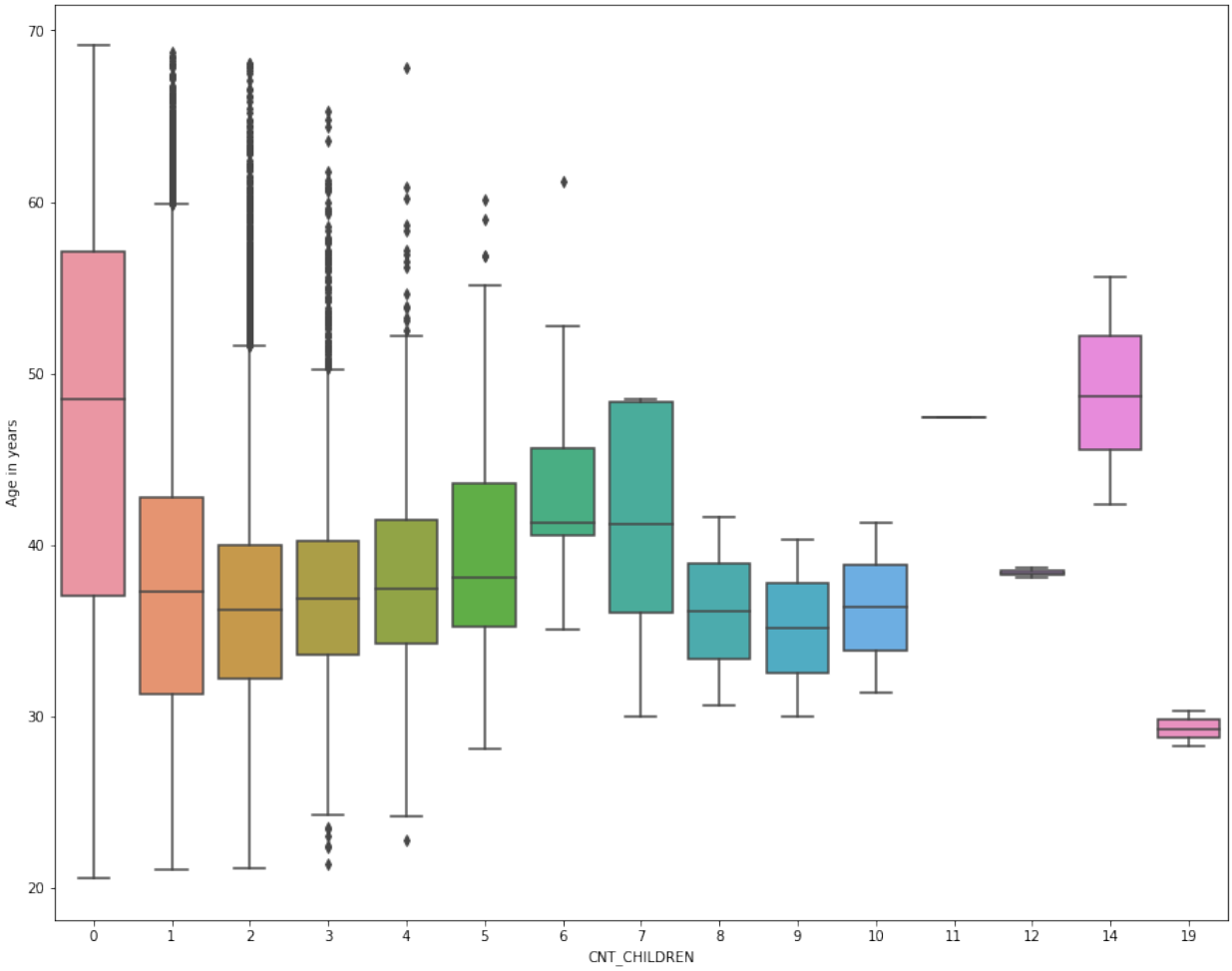
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

As we can see, this max value does not make any sense so is either an error in recording or the value has been placed into the data in place of missing data. The plot of the distribution of the column shows the following:

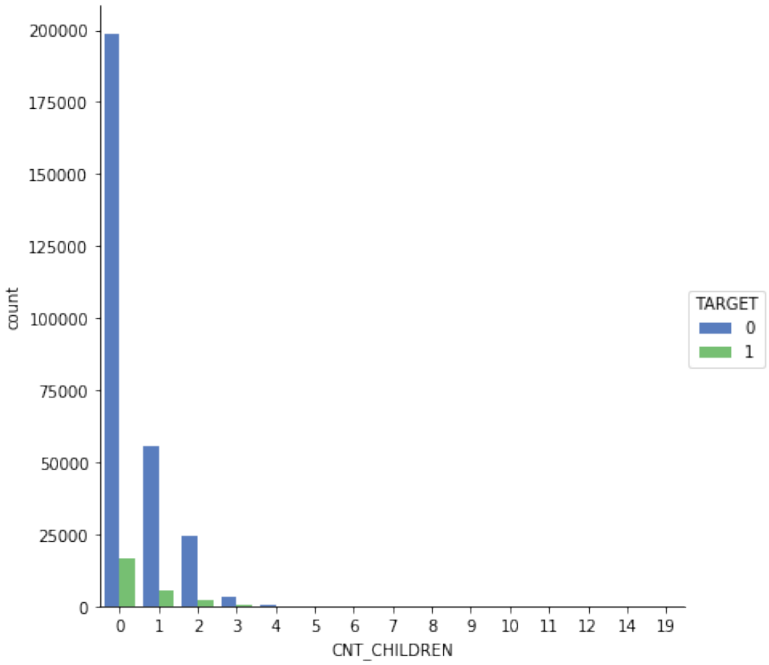


Meaning 365243 was likely recorded instead of a missing value so we replace this with NaN values in our data but create a new column which records that these values were altered through a binary flag.

Also an interesting observation between CNT_CHILDREN and DAYS_BIRTH shows that those without children, on average, are older than those with children. These two features showed a negative correlation of -0.33 with each other.



Those with children tend to be, on average, younger than those without children. Additionally, those without children and unable to pay back their loans make up the majority of the population:



However, we already know that there is a class imbalance in the data between those who could/could not repay their loans.

Algorithms and Techniques

The algorithm I intend to use to solve this problem is Microsoft's LightGBM. This is a gradient boosting machine model that uses decision trees but unlike other GBMs, it uses a small amount of memory while still providing highly accurate results. As such, it has recently been widely used to win several Kaggle competitions. My reasoning for choosing this model is due to: the ease with which tree-based models handle missing values, the fact that they do not require any data normalization/scaling, and LightGBM can be trained locally on large datasets without requiring a significant amount of time to run. Further, as referenced above, this particular model is highly popular in the data science world at the moment so I wanted to use this capstone project to get some hands-on time with it and see how effective it is.

I plan to create a single dataset using table joins between the static data table (application_train and application_test) and the other tables provided by Home-Credit and to use that data as input into the LightGBM after some feature engineering. A tree based method is highly convenient in this scenario where we are faced with hundreds of features since it doesn't require us to go through each column and try to handle missing values manually. The parameters of the model used for the purposes of this report are found by Bayesian optimization in the following kernel: https://www.kaggle.com/tilii7/olivier-lightgbm-parameters-by-bayesian-opt/code

Benchmark

The benchmark used is a random forest model provided in the Kaggle challenge that has an AUROC score of 0.688. This is a good benchmark for LightGBM since both are ensemble tree based methods and comparing the two will give us somewhat of an insight into the performance of LightGBM (although we do not know how much of the data the random forest model used for training and prediction and the parameters of the model).

III. Methodology

Data Preprocessing

Six functions were built out to preprocess the data before the LightGBM is trained and tested on it. Each of these functions processes separate table(s) and joins them to the application_train and application_test tables. The following preprocessing steps were taken for each table:

1. application_train and application_test
    a. Both tables were merged into one dataframe for ease of processing
    b. 365243 values for DAYS_EMPLOYED replaced with NaN and new feature with NaN flags created
    c. Binary features label encoded
    d. Categorical features one-hot encoded
2. bureau and bureau_balance
    a. Number of previous loans counted
    b. Number of active and closed loans counted
    c. Categorical features one-hot encoded
    d. Numerical features aggregated in terms of min, max, mean, and count (since this is time-series data, there are monthly snapshots for each separate loan that the current applicant had previously taken out, so we need to aggregate those snapshots for each individual)
    e. Categorical features aggregated into mean values after one-hot encoding (value closer to 1 implies that an individual had a lot of positive values in that particular feature)
    f. Merge into application_train and test
3. credit_card_balance
    a. Number of previous credit loans counted
    b. Number of active and completed loans counted
    c. Categorical featues one-hot encoded
    d. Numerical features aggregated in terms of min, max, mean, sum, and var
    e. Merged into application_train and test
4. installments_payments
    a. Calculated difference between when installment was due and when it was paid and averaged for each individual
    b. Calculated difference between amount due in installment and how much was paid and averaged for each individual
    c. Merged into application_train and test

5. POS_CASH_balance
    a. Counted number of previous POS loans
    b. Counted number of previous completed POS loans
    c. Counted number of previous active POS loans
    d. Aggregated numerical features by min, max, and mean
    e. Merged into application_train and test
6. previous_applications
    a. Replaced 365243 max values in a number of DAYS columns with NaN
    b. One-hot encoded categorical features
    c. Aggregated numerical features in terms of min, max, mean, size, sum, and median
        i. Approved loans aggregated
        ii. Refused loans aggregated
    d. Aggregated categorical features (after one-hot encoding as in bureau table) in terms of mean
    e. Merged into application_train and test

Implementation

A LightGBM implementation with K-fold cross-validation was used. The following steps were taken in implementation:

1. Created function with parameters for input dataframe, number of folds, and whether stratified K-fold CV is required
2. Processed dataframe split into training and testing sets based on where target contained null values (test set)
3. Two arrays were created to store the results from training and testing
4. A dataframe is initialized to store the feature importance values
5. Training dataframe is split into folds and indices are assigned for training rows and validation rows
6. LightGBM classifier with Bayesian optimized parameters initialized
7. Classifier fit to training data and given evaluation sets of training data and validation data
8. Predictions on validation set are stored in array
9. Predictions on test set are stored in another array
10. Dataframe for fold importance initialized and data on features, feature importances, and fold number are stored therein
11. Fold importance dataframe concatenated with feature importance dataframe initialized earlier
12. Classifier, training data, and validation data are deleted from memory
13. Target column in test set replaced with predictions on test set
14. SK_ID_CURR (ID of loan applicant) and test set predictions saved to csv file
15. Feature importances are displayed in a seaborn plot

Some complications occurred in terms of how to store and display feature importances since the K-fold CV method only provides feature importances for each feature over the separate folds. As such, the average of all the feature importances for each feature over every fold was found and

plotted. This method calculating and plotting feature importances was then placed into a function that can easily be called at the end of the LightGBM run.

Refinement

A separate notebook was developed that used a portion of the training dataset as a validation set, which was itself split into a training and testing set. This validation set was designed to give the same or similar result from the LightGBM model after training and predicting as the full training and testing sets. Reading in a third of the training set and skipping every second row seemed to work well as the model obtained an AUROC score on the validation set that had only a 0.01 difference between the two datasets. Using this validation set, another notebook called model_refinement was created that reads in the training and testing validation sets along with the labels from the test set that are set to null within the test set itself. This notebook does not carry out any processing since the data is read in pre-processed (speeding up the time taken to run significantly from 560 seconds to 230 seconds). The model is trained on the training set and predicts on the test set that has had labels removed. The predictions are then compared to the labels and an AUROC score is given. This notebook was specifically designed to allow tweaking of model parameters.

Initially, parameters such as max_depth and num_leaves were tweaked across a number of runs and provided good results that largely matched those given on the full dataset. However, I discovered a kernel that used Bayesian optimization to find parameter values that appear to be optimal (https://www.kaggle.com/tilii7/olivier-lightgbm-parameters-by-bayesian-opt/code). In future, I plan to amend parts of the data preprocessing pipeline to hopefully better improve the model and as such, a new validation set will need to be created. Additionally, the model as it is set up at the moment takes the training data and splits the folds created into training and validation rows – it may be worthwhile instead to train the model on all of the available training data and simply test on the test set without holding out training data for validation.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

The final model was chosen after significant feature engineering work (shown in the data_analysis.ipynb notebook) that led to a single dataset containing as much information as possible from all of the data provided in the Kaggle competition. The LightGBM was chosen for this data due to its low memory usage, excellent results in prior competitions, and handling of missing and categorical values. As discussed above, a validation set was created from the training data to tweak model parameters and obtain results quickly and a hold- out set is used in training on the full dataset that allows us to see if the model is overfitting. Both of these solutions were used to validate the model.

The model obtains an AUROC score of 0.779 on the full dataset. A similar score of 0.78 is obtained on the validation set. The score on the full dataset's hold-out set is also approximately 0.78. The

current leader in the competition obtains a score of 0.805 using the same metric. The random forest benchmark, on the other hand, obtains a score of 0.688. As such, the LightGBM model sits in the top 49% of the leaderboard. This is aligned with my expectations as most competitors are using some form of a LightGBM model and the parameters I have used are also being used by many others. It should be noted that testing the validation set on another public LightGBM kernel provided a similar public score to the one that their model obtained on the full data. As such, the results from the model can be trusted since the validation set was itself validated using a different model and the AUROC score for the actual predictions was validated through Kaggle's submission process.

Justification

The model obtains a score of 0.779 while the benchmark random forest model scores 0.688. An AUROC score of 0.5 is random, in other words, it would assign a classification of 0 or 1 to the target with equal probability. Therefore, the created model is significantly better than both the random model and the random forest benchmark model.

It is interesting also to compare the two models in terms of them both being ensemble tree based methods. Clearly, LightGBM far outperforms the random forest model even though both are made up of weak decision trees. This can likely be attributed to the following points:

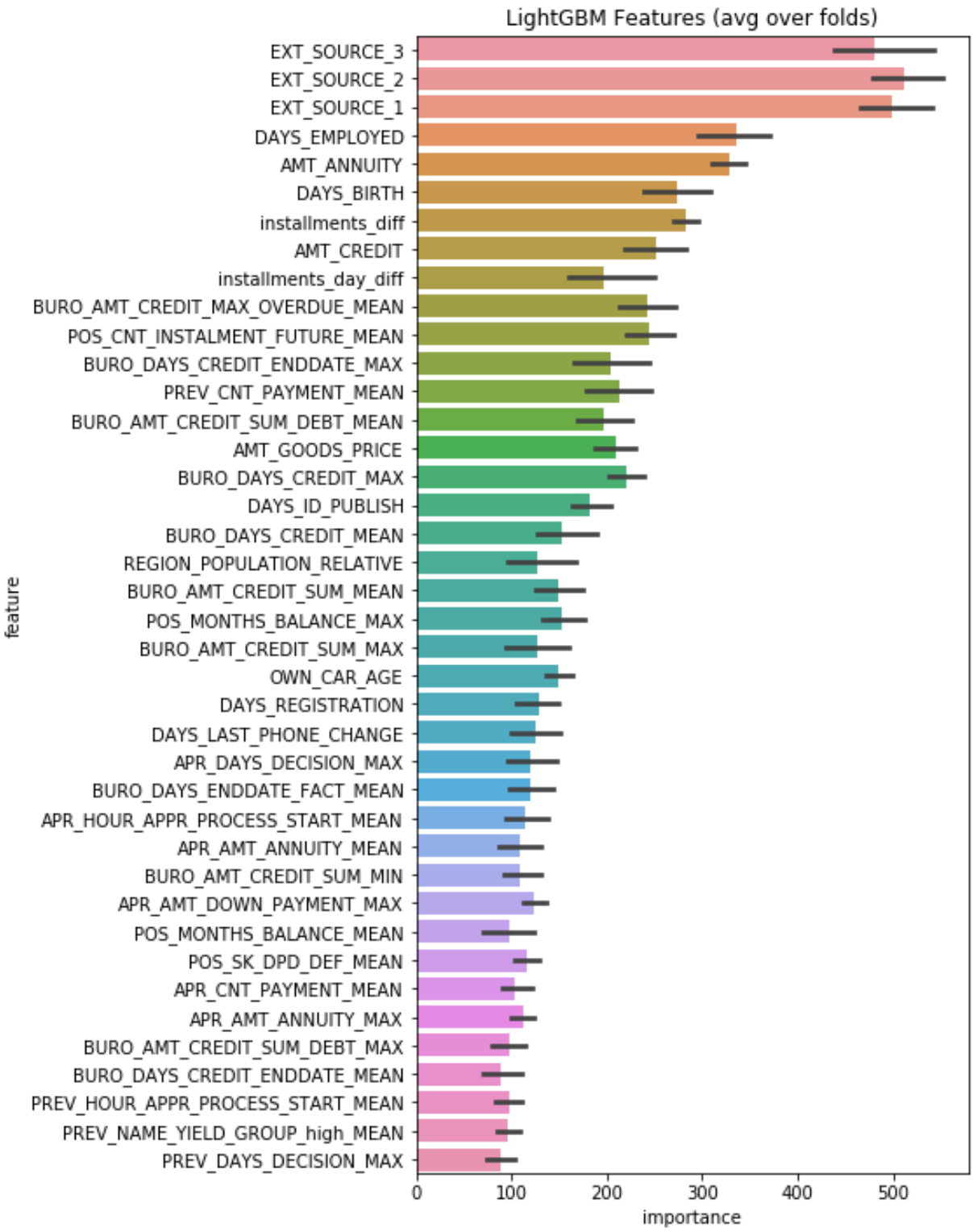
- LightGBM uses a leaf-wise growth method for trees instead of the level-wise method used in random forest trees. This means that trees will grow from any node based on the greatest delta loss possible, resulting in less time taken to find an optimal solution.
- Our LightGBM model used all of the data available in training and prediction. It is unclear whether the random forest model used all of the data or if it only used the static features in application_train and application_test.

This final solution obtains an AUROC score that is far enough above the benchmark and random that it solves the problem effectively. The difference between the score obtained and the top of the leaderboard is a further 0.026 AUROC score which is not hugely significant in the grand scheme of things and this model could effectively be used to predict how capable each applicant is of repaying a loan.

V. Conclusion

Free-Form Visualization

An important visualization is the feature importance graph shown below:



This plot justifies our early analysis that showed EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3, and DAYS_BIRTH would be important in prediction. Also interesting to note is the number of aggregated features from the auxiliary tables that show up in the feature

importance plot, justifying our usage of the data. The importance of feature engineering is greatly emphasized in the number of engineered features that have shown up herein. The importance values on the x-axis are calculated through LightGBM's feature_importances_ method which calculates the importance of features based on the number of times that the feature is used in the model – the number of splits on that feature in this case.

Reflection

The end-to-end problem solution can thus be summarized as feature engineering a collection of tables containing relevant information on loan applicants in order to better predict their ability to repay a loan using a LightGBM model. The most difficult portion of the project was the feature engineering and figuring out how to merge the separate tables to maximise the amount of data available. Prior to beginning, I had some awareness that this would be a feature engineering heavy challenge but I didn't realise exactly how much work it would require. I found it interesting and rewarding however, that a lot of the features that I engineered ended up being quite important for the model – meaning that I'm following the correct process. A score of 0.779 is also better than I was expecting and I certainly feel as though I have the means to improve on my position on the leaderboard. In my opinion, the final model and solution do fit my expectations for the problem and could be used in a general setting for these sorts of problems since it handles missing values well, requires a low amount of memory to run, and is highly accurate.

Improvement

In order to improve the model further, even more work on feature engineering is required and further investigation into parameters (perhaps using random search) is needed. Combining the LightGBM model with some other model in an ensemble (maybe a neural network) is also a possible avenue. I was considering adding some form of unsupervised learning to group the applicants in the data into a cohort and to then run the LightGBM model on top of the grouped data to better identify those able to repay their loans. The three EXT_SOURCE features ended up being quite important to the model so it could be interesting to see if feature interactions between these features are useful also.

The creation of a separate validation set using the training set provided by Kaggle means that the model can be fully trained on the training data without the need for a hold-out set. As such, I would add the hold-out set back into the training data and re-train the model on all of the available data – validating only using the separate, smaller, validation set.