Machine Learning Engineer Nanodegree

Capstone Project

Jaime Ignacio Rovirosa

July 16th, 2018

Home Credit Default Risk, Kaggle competition
Code available here: https://github.com/nachorovi/mlnd_capstone/blob/master/notebook.ipynb
Public Kaggle kernel: https://www.kaggle.com/nachorovi/homecreditdefaultrisk-mlnd-nachorovi

1. Definition

1.1 Project Overview

"If access to credit is directly instrumental to economic development, poverty reduction and the improved welfare of all citizens, then one can proclaim, as Nobel Prize Laureate M. Yunus has done, that it is a moral necessity to establish credit as a right."[1][2]

"Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders."[3]

The lending industry is a mature one with a long history, yet a significant part of the world population are left outside the financial and banking systems. Today most lenders just use Bureau data to determine the creditworthiness of an individual. In the US most lenders use a FICO-based model to determine who to lend to and at what interest rate. There are three main credit bureaus: Equifax, Experian and TransUnion. The FICO score was first introduced in 1989 and is made up of the following components[4]:

1. 35% for payment history
2. 30% debt burden
3. 15% length of credit history
4. 10% types of credit used

---

[1] M. Hudson, "*Should access to credit be a right?*", (ULB, 2007), citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.3757&rep=rep1&type=pdf
[2] Access to credit is not a right included in the United Nations Universal Declaration of Human Rights, www.un.org/en/universal-declaration-human-rights/.
[3] Kaggle, Home Credit Default Risk, Description, www.kaggle.com/c/home-credit-default-risk, (May 17, 2018).
[4] Wikipedia, Credit score in the United States, en.wikipedia.org/wiki/Credit_score_in_the_United_States#FICO_score.

5. 10% recent searches for credit or hard credit inquiries

Among the many shortfalls of the FICO model is that the FICO score may vary between the bureaus depending on the information they have collected. Another problem is that consumers without sufficient credit history and information have a hard time to secure a loan or do so at a very high interest rate, or even worse, they need to resort to terrible terms with predatory lenders.

In order to serve the unbanked underserved population, companies like Upstart[5] and Home Credit[6] apply Machine Learning and Artificial Intelligence tools in combination with a variety of alternative, non-conventional data (including telco and transactional information) to predict a person's ability to repay a loan.

1.2 Problem Statement

This project will focus on completing the Kaggle competition "*Home Credit Default Risk*"[7], whose main objective is to determine whether a person will be able to repay a loan. The data provided for the challenge goes beyond the traditional data collected by the bureau agencies, which can be incomplete, inaccurate or non-existent.

Lets decompose that statement. Among the datasets provided, the *Application Train* dataset includes a *targel label* called TARGET, which can take two values; 1 for clients having payment difficulties and 0 for all other cases. This puts us in the realm of *Supervised Learning*, where we can find either a classification or a regression problem. The fundamental difference is that classification is about predicting a label, while regression is about predicting a quantity[8].

Given that the target label can assume only two values, this a binary classification problem. Another subtlety to note, is that rather than predicting a discrete class label, we are trying to predict the probability of the target label.

Home Credit provides two benchmarks, the highest one is for a Random Forest Classifier. The objective when solving the competition will be to surpass this benchmark, first using the Random Forest Classifier and later with another model. The stretch goal will be to make a prediction that scores closer to the highest score for the competition than the Random Forest benchmark.

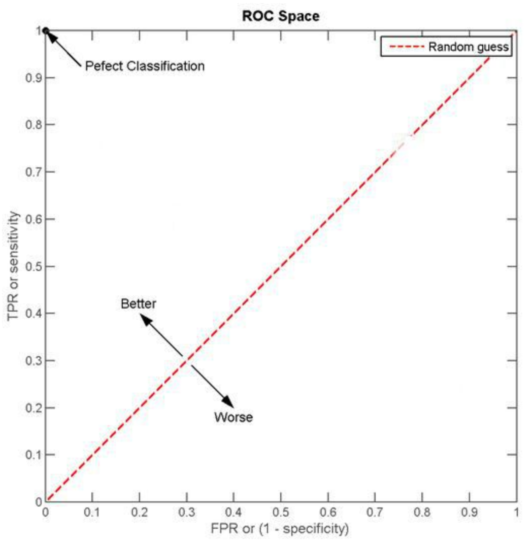
[5] Upstart, www.upstart.com/about#top.
[6] Home Credit, www.homecredit.net/about-us.aspx.
[7] Kaggle, Home Credit Default Risk, www.kaggle.com/c/home-credit-default-risk.
[8] Machine Learning Mastery, Jason Brownlee, Difference Between Classification and Regression in Machine Learning, machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

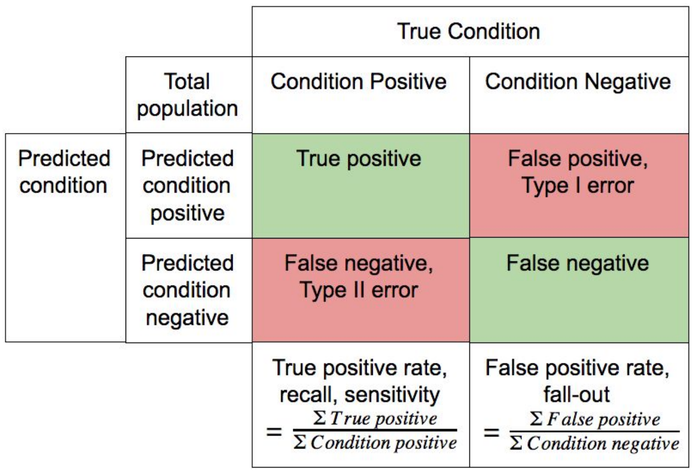
1.3 Metrics

The Kaggle competition evaluates the submissions on **area under the ROC curve** between the *predicted probability* and the *observed target*[9]. The solution that needs to be submitted for this kaggle competition is very simple, for every loan application in the *test dataset* a *predicted probability* between 0 and 1 inclusive needs to be determined.

A **receiver operating characteristic curve** or **ROC curve**[10] is a plot that shows the performance of a **binary classifier** system as its discrimination threshold is varied. It is created by plotting the **true positive rate** (**TPR**) against the **false positive rate** (**FPR**) at various threshold settings.

"The **TPR** (also known as *recall* or *sensitivity*) defines how many correct positive results occur among all positive samples available during the test. **FPR** (also known as *fall-out* or (1 - *specificity*)), on the other hand, defines how many incorrect positive results occur among all negative samples available during the test."[11]





		True Condition	
	Total population	Condition Positive	Condition Negative
Predicted condition	Predicted condition positive	True positive	False positive, Type I error
	Predicted condition negative	False negative, Type II error	False negative
		True positive rate, recall, sensitivity $= \frac{\Sigma\, True\ positive}{\Sigma\, Condition\ positive}$	False positive rate, fall-out $= \frac{\Sigma\, False\ positive}{\Sigma\, Condition\ negative}$

Figure 1: ROC Space and Confusion matrix

The **area under the ROC curve (roc_auc_score)** metric available in the scikit learn library[12] is utilized to evaluate model performance.

[9] Kaggle, Home Credit Default Risk, Evaluation, www.kaggle.com/c/home-credit-default-risk#evaluation.
[10] Wikipedia, Receiver operating characteristic, en.wikipedia.org/wiki/Receiver_operating_characteristic.
[11] Wikipedia, Receiver operating characteristic, ROC space, en.wikipedia.org/wiki/Receiver_operating_characteristic#ROC_space.
[12] Scikit learn, metrics, roc_auc_score, scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.

2. Analysis

2.1 Data Exploration

All the information for this competition is proprietary and provided by *Home Credit* via *Kaggle*. Most of this information has been gathered by Home Credit directly and the rest comes from the credit bureaus. There is no ethical conflict with the customer data as the data is properly anonymized.

The data can be broken down into three main categories. The main one is the information about current loan applications with Home Credit. The second one is information regarding previous applications, repayment history, monthly balance snapshots of loans and credit cards applicants had with Home Credit. The final category contains information coming from credit bureaus regarding previous credits with other financial institutions and their monthly balances.

A brief description of the datasets provided by Home credit[13]:

- application_{train|test}.csv
    - This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
    - Static data for all applications. One row represents one loan in our data sample.
- bureau.csv
    - All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
    - For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.
- bureau_balance.csv
    - Monthly balances of previous credits in Credit Bureau.
    - This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.
- POS_CASH_balance.csv
    - Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.
- credit_card_balance.csv
    - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.

[13] Kaggle, Home Credit Default Risk, Data, www.kaggle.com/c/home-credit-default-risk/data.

    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.
- previous_application.csv
    - All previous applications for Home Credit loans of clients who have loans in our sample.
    - There is one row for each previous application related to loans in our data sample.
    - installments_payments.csv
    - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
    - There is a) one row for every payment that was made plus b) one row each for missed payment.
    - One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.
- HomeCredit_columns_description.csv
    - This file contains descriptions for the columns in the various data files.

An illustration of the relationship between the data:

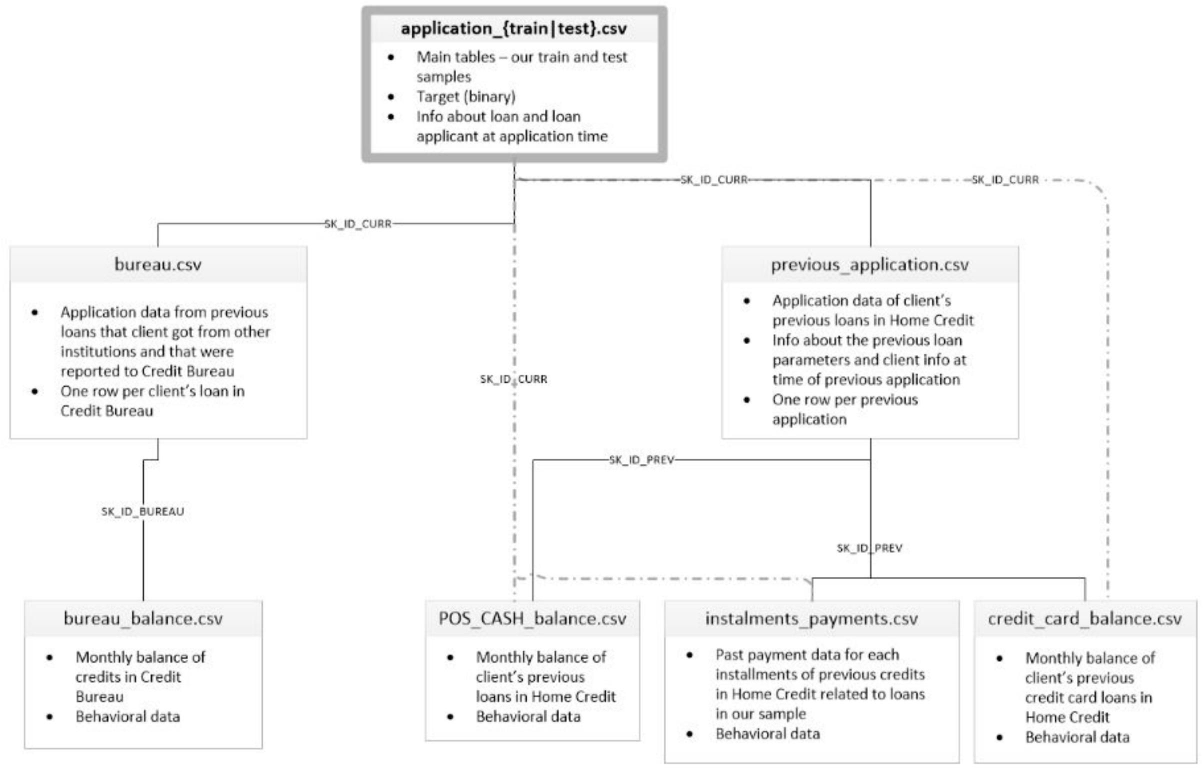
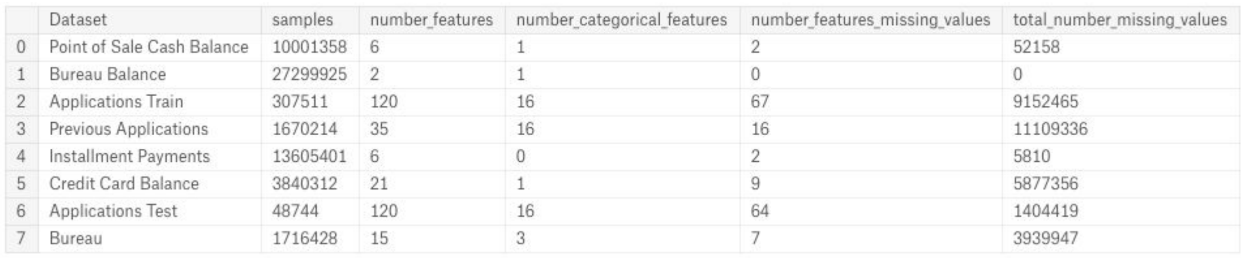

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
SK_ID_PREV
SK_ID_BUREAU
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Figure 2: Home Credit Default Risk datasets relationships[14]

[14] Kaggle, Home Credit Default Risk, Data, www.kaggle.com/c/home-credit-default-risk/data.



	Dataset	samples	number_features	number_categorical_features	number_features_missing_values	total_number_missing_values
0	Point of Sale Cash Balance	10001358	6	1	2	52158
1	Bureau Balance	27299925	2	1	0	0
2	Applications Train	307511	120	16	67	9152465
3	Previous Applications	1670214	35	16	16	11109336
4	Installment Payments	13605401	6	0	2	5810
5	Credit Card Balance	3840312	21	1	9	5877356
6	Applications Test	48744	120	16	64	1404419
7	Bureau	1716428	15	3	7	3939947

Table 1: Datasets features and samples analysis

From the initial datasets analysis the following observations arise:

1. Most of the features in the eight datasets are numerical, the rest are categorical variables that need to be converted into numerical features.
2. Between a third and half of the features in every dataset has features with missing values.
3. Except for the Application Train and Test datasets, the other datasets have multiple samples for one loan application (*SK_ID_CURR*).

2.2 Exploratory Visualization

When visualizing the *target label* distribution for the *Application Train* dataset it stands out that only a small amount of the clients have difficulties repaying their loans. This imbalance of the target distribution could be a problem when training the model.

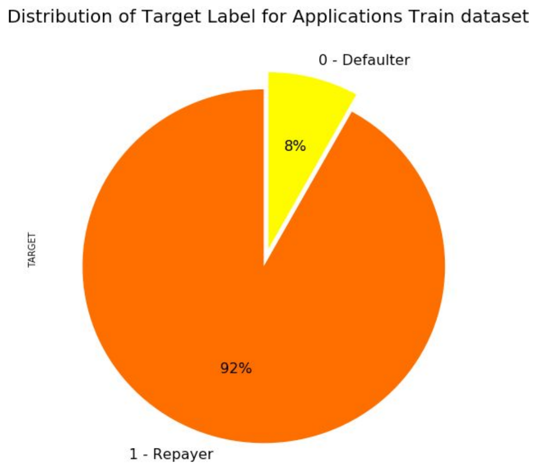


Figure 3: Applications Train dataset target label distribution

When looking at the *averages* for *total income*, *credit*, *annuity* and *goods price* amounts for repayers and defaulters, there is no significant difference. The most significant difference is that the *standard deviation* is about 4.5 times the average for defaulters but it is only 65% for repayers. Another value that stands out for defaulters is the maximum income total is 6.5 times higher than for repayers, but this is probably just an outlier.

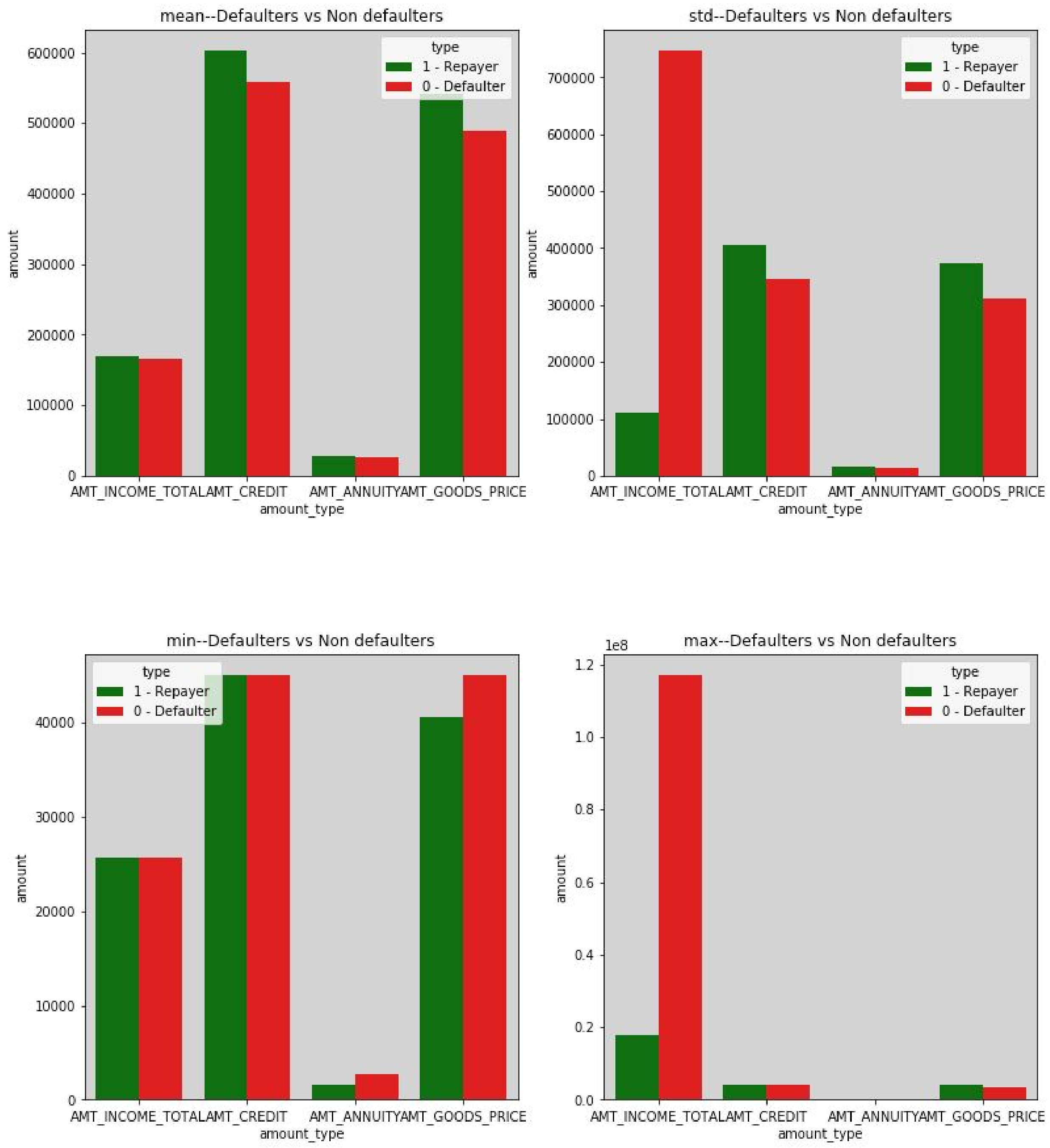


Figure 4: Applications Train statistical comparison of loan amounts by target label

2.3 Algorithms and Techniques

The objective of the competition is to determine whether an applicant will default on the loan. Given we classify the applicants into two groups, this is a *binary classification*[15] problem. Some of the methods commonly used for binary classification are:

[15] Wikipedia, Binary classification, en.wikipedia.org/wiki/Binary_classification.

1. Decision trees
2. Random forests
3. Bayesian networks
4. Support vector machines
5. Neural networks
6. Logistic regression
7. Probit model

The following models will be compared:

1. Random Forest[16]
2. LightGBM[17]

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks.[18]

Ensembles are a *divide-and-conquer approach* used to improve performance. The main principle behind ensemble methods is that a group of "*weak learners*" can come together to form a "*strong learner*". The random forest starts with a standard machine learning technique called a "*decision tree*" which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets.[19] The decision trees are then combined as an ensemble to produce the strong learner, the Random Forest.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples. After training, predictions of testing sets can be made by averaging the predictions from all the individual regression trees or by the taking the majority vote in the case of classification trees.[20]

The advantages of Random Forest is that it can be used for regression or classification, it has few and simple to understand hyperparameters, it is quick to train and it is not easily prone to overfitting if there are enough trees in the forest. On the other hand, among the disadvantages we have that it is slow to make predictions so it is generally inefficient for real-time predictions,

---

[16] Scikit Learn, RandomForrestClassifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
[17] LightGBM, lightgbm.readthedocs.io/en/latest/.
[18] Towards Data Science, Niklas Donges, The Random Forest Algorithm, towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.
[19] CitizenNet Blog, Dan Benyamin, A Gentle Introduction to Random Forests, Ensembles, and Performance Metrics in a Commercial System, blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics.
[20] Wikipedia, Random Forest, en.wikipedia.org/wiki/Random_forest.

and in order to achieve higher accuracy the model needs more trees resulting in a slower performance.

Among the most salient hyperparameters for the Random Forest model we have[21]:

1. n_estimators: The number of trees in the forest.
2. max_features: The number of features to consider (by each decision tree) when looking for the best split.
3. min_sample_leaf: The minimum number of samples required to split an internal node (can determine the depth of a tree if max_depth is not defined).
4. max_depth: The maximum depth of a tree.
5. random_state: If int, is the seed used by the random number generator (makes the model's output replicable).

Random Forest was chosen as the first model to implement. It is a common and popular model to handle binary classification problems, for its simplicity and decent accuracy. But it was primarily chosen because the benchmark to achieve was set for this model.

LightGBM is a *Gradient boosting decision tree* (*GBDT*) algorithm that has gained popularity because it of its speed and resource utilization efficiency. "LightGBM can significantly outperform *XGBoost* and *SGB* in terms of computational speed and memory consumption."[22] All of this while maintaining similar levels of accuracy but handling bigger datasets with more features. An "algorithm that contains two novel techniques: *Gradient-based One-Side Sampling* and *Exclusive Feature Bundling* to deal with large number of data instances and large number of features respectively."[23]

"Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree *leaf-wise* while other algorithms grows *level-wise*. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm."[24]

The advantages of LightGBM are that it is faster, it utilizes less memory, it can handle larger datasets with many features[25] and can use categorical features directly (without one-hot

---

[21] Scikit Learn, RandomForestClassifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[22] LightGBM: A Highly Efficient Gradient Boosting Decision Tree, Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu, 6 Conclusion, papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[23] Ibid.

[24] What is LightGBM, How to implement it? How to fine tune the parameters?, Pushkar Mandot, medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc.

[25] LightGBM: A Highly Efficient Gradient Boosting Decision Tree, Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu, 6 Conclusion, papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

encoding)[26]. Among its disadvantages, it is prone to overfitting for small datasets (because of its leaf-wise approach) and it is difficult to tune due to its large list of hyperparameters (over 100).

Among the most salient hyperparameters for the Random Forest model we have[27]:

1. num_leaves: This is the main parameter to control the complexity of the tree model.
2. min_data_in_leaf: This is a very important parameter to prevent over-fitting in a leaf-wise tree.
3. max_depth: Limit the max depth for tree model. Helps to deal with over-fitting for small datasets.

Hyperparameters tuning for better accuracy (speed is not a big concern in this case)[28]:

1. Use large max_bin (may be slower)
2. Use small learning_rate with large num_iterations
3. Use large num_leaves (may cause over-fitting)
4. Use bigger training data
5. Try dart

LightGBM was chosen as the second model to implement as its accuracy is comparable with other possible models like XGBoost, but it is faster and can handle well large datasets with a large number of features, which is our case.

2.4 Benchmark

As part of the Kaggle competition, the data set includes a sample submission. This submission has the same value of 0.5 assigned for all loan applications in the test dataset. This is equivalent to assigning a 50% chance to all the loans without considering any data and represents a good naive benchmark. When looking at the Leaderboard[29] this sample submission obtains a score of **0.5**.

Another higher benchmark is given for a Random Forest estimator, and it has a score of **0.688**. This is “the” benchmark chosen for result comparisons. Two weeks into the competition and the high score in the leaderboard is **0.800**. Seven week into the competition and the high score is **0.805** while a score of 0.8 is the 100th position. The stretch goal benchmark chosen is the midpoint the Random Forest estimator benchmark and the highest score, so **0.746**.

It is important to note that the leaderboard is calculated with approximately 20% of the test data and the final results will be based on the other 80%. Also to remember that the submissions are

[26] LightGBM, July 14, 2018, Quick Start, 2.1.1 Caregorical Feature Support, media.readthedocs.org/pdf/lightgbm/latest/lightgbm.pdf.
[27] LightGBM, July 14, 2018, Parameters Tuning, 7.1 Tune Parameters for the Leaf-wise (Best-first) Tree, media.readthedocs.org/pdf/lightgbm/latest/lightgbm.pdf.
[28] Ibid., 7.3 For Better Accuracy, media.readthedocs.org/pdf/lightgbm/latest/lightgbm.pdf.
[29] Kaggle, Home Credit Default Risk, Leaderboard, www.kaggle.com/c/home-credit-default-risk/leaderboard.

evaluated on area under the ROC curve between the predicted probability and the observed target.

Estimator	Comments	Competition AUC score
Naive estimator	All 50%	0.5
Random Forest	Benchmark	0.688
Top 1	Leaderboard 6/15/18	0.805
Top 100	Leaderboard 6/15/18	0.801
Top 500	Leaderboard 6/15/18	0.799
Top 1,000	Leaderboard 6/15/18	0.797

Table 2: Benchmark comparison

3. Methodology

3.1. Data Preprocessing

Based on the work done during the Exploratory Data Analysis three main issues had to be addressed: handling *missing data*, *encoding categorical variables*, and *flattening the datasets* to leverage all the data available. Other common pre-processing tasks that were left for future iterations deemed to have lower impact on the outcome or too time consuming for an initial pass: *scaling data*, *skewed data*, *removing outliers* and *feature engineering*.

The one-hot encoding scheme (the most common method for dealing with categorical data) was used to turn categorical variables into numerical features. One-hot encoding creates a *"dummy"* variable for each possible category of each non-numeric feature. The number of features in the datasets grew on average **166%** (from 185 to 493). Some models, like LightGBM, can use categorical variables directly without the need to encode them first. But to be able to support different models with the same preprocessed data the encoding was performed.

Initially, the one-hot encoding scheme was performed on the fully merged dataset. But the merging process applied was dropping the categorical variables, which is undesirable as there is loss of information. Unexpectedly, fitting the estimator with the resulting merged dataset yield a lower-scoring prediction than fitting the same estimator with just the *Application Train* dataset. The solution was to perform the one-hot encoding before merging the datasets.

Out of the eight datasets provided, one is the *Applications Test*, which has the same features as the *Applications Train* dataset. From the remaining 6 datasets, *Bureau Balance* is the only one that doesn't have a column *SK_ID_CURR* to map the data to the current applications. The Bureau Balance dataset needs to first be merged with the *Bureau* dataset.

This, now 5 datasets, have multiple samples per each unique loan application *SK_ID_CURR*. What this means is that all this samples need to be condensed into one before it can be merged with the main *Applications Train* dataset. The mean of these multiple samples was used to condense them into one. Then the datasets were merged together based on the common column *SK_ID_CURR*.

The resulting merged dataset has **307,511** samples (same as the *Applications Train* dataset) and **493** features instead of the original **185** (the sum of the each dataset number of features). This includes any new feature introduced as part of condensing multiple samples with the same *SK_ID_CURR* into one. This is part of the **Feature Engineering** effort which is covered later.

The *Applications Test* dataset has **48,744** samples (the number of samples doesn't change with the data preprocessing) and the number of features is the same as the *Application Train* dataset. If the number of features in the test dataset would be different, the model predict method would throw an error.

Common ways to deal with missing data include[30,31]:

1. Dropping the whole column if there are too many missing values (over 70%, this percentage varies depending on the problem at hand.)
2. Imputing missing values, usually using the *mean* or *median* for numeric features and *mode* with categorical features.

For the first iteration, to avoid losing data, missing values were imputed using the mean for the column.

Initially the merged and Test datasets were preprocessed separately to handle categorical features and missing data. The result was that the datasets ended up with a different amount of features because the Test dataset categorical features did not include samples for all of the possible values. To solve this, the Test dataset was merged with the rest of the datasets for preprocessing and later separated before splitting into Train and Validation datasets. Another advantage of doing this is when imputing missing values with the mean of the column, more samples are used (in this case 4 times more), resulting in more meaningful replacements.

3.2. Implementation

The following steps were taken to solve the *Home Credit Default Risk* Kaggle competition:

1. The first step focused on doing **Exploratory Data Analysis** or **EDA**.
2. The second step was **Data Preprocessing**, including *cleaning* invalid or missing data, *normalization* of numerical features, *transforming* skewed features, *converting*

[30] Data preprocessing in Machine Learning, Abhijit Annaldas, abhijitannaldas.com/data-preprocessing-in-machine-learning.html.

[31] Analytics India, 5 Ways to Handle Missing Values in Machine Learning Datasets, Kishan Maladkar, analyticsindiamag.com/5-ways-handle-missing-values-machine-learning-datasets.

categorical variables into numerical features, *merging* all the datasets into one and doing **Feature Engineering**.
3. The third step was shuffling and **splitting the data** into training and validation sets.
4. The fourth step was to determine and codify the **evaluation metrics** to measure the model performance.
5. The fifth step involved comparing multiple models before doing a **model selection**. For each model the process included:
    a. **Fit** the model with the train data.
    b. Use the trained model to perform **predictions** on the validation data.
    c. Use the *evaluation metrics* to measure the performance.
    d. **Tune** the model *hyperparameters*.
    e. Determine *feature importance*.
    f. Submit the prediction results and assess the results versus the naive and benchmark predictors and the other submissions.

Steps 1 and 2 were already covered in section 3.1 Data Preprocessing above. The rest of the steps are covered in this section.

A common practice when solving Machine Learning problems is to take all the available data and split it into two datasets, *Train* and *Test*. The percentages may vary but a regular split might be 70 or 80 % for Train set and the rest for Test, depending on the problem at hand, the dataset characteristics and model. The *Test* Dataset is used to provide an unbiased evaluation of a final model fit on the training dataset. The *Validation* Dataset is used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. Finally the *Train* Dataset is used to fit the model.

When splitting the dataset we want to ensure that the resulting Train and Test datasets have approximately the same percentage of samples of each target class as the complete set[32]. If the dataset has a large amount of each class this shouldn't be a problem, but if the dataset is imbalanced (92% of the Application Train dataset are repayers, or target label 0) then we might consider using *Stratification*[33]. Stratified sampling will be covered in section 4.1 Model Evaluation and Validation.

After this, the *Train* dataset is again split into two sets, actual *Train* and *Validation*. This split might algo look like 80% Train and the rest for Validation. The model is then iteratively trained and validated on these different sets. Another way to split the *Train* dataset, which is getting more popular is known as *Cross-Validation*. Basically the training set is used to generate multiple splits of Train and Validation sets. This helps avoid overfitting. K-fold Cross-validation is

[32] StackExchange, Franck Dernoncourt, Benefits of stratified vs random sampling for generating training data in classification, stats.stackexchange.com/questions/250273/benefits-of-stratified-vs-random-sampling-for-generating-training-data-in-classi.
[33] DataMiningBlog.com, Sandro Saitta, Stratification for data mining, www.dataminingblog.com/stratification-for-data-mining/.

the most popular method of cross-validation[34]. Explorations with K-fold cross-validation will be covered in section 4.1 Model Evaluation and Validation.

An *Application Test* dataset is provided as part of the competition, so we don't need to generate that. In order to evaluate model performance, the merged dataset is split into *training* and *validation* sets (both *features* and *label*). The model is fitted with the **training** set and its performance is evaluated using the **validation** set. The *validation* set is used to fine-tune the model hyperparameters. The intent of the validation set is to provide an unbiased estimate of the model skill to predict unseen data and make sure the model is not overfitting[35].

Before actually splitting the preprocessed, merged dataset, the *Target Label* (the column TARGET in the *Application Train* dataset) is removed and becomes its own pandas Series.

To split the merged dataset into *Train* and *Validation* datasets the **train_test_split[36]** method was used. It is important to note that the merged dataset, with 307,511 samples, is large enough to provide for meaningful *Train* and *Validation* datasets. The percentage used for split was 80% Train and 20% Validation, fairly standard. This percentage varies depending on the number of samples available and on the number of hyperparameters to fine-tune.

After the split, the *Training* Dataset has 246,008 samples and 493 features and the *Validation* Dataset has 61,503 samples and 493 features. For comparison the Test *Dataset* has 48,744 samples and 493 features.

For the initial pass, the **RandomForestClassifier** model was used to try and reach the Leaderboard benchmark score of 0.688. Later other models were used to compare performance. The LightGBM model is used for the final pass.

For all the models in question the procedure is the same:

1. The *Training* Dataset is used to **fit** the model.
2. Grid search optimization is performed for the model by tuning multiple hyperparameters using **GridSearchCV**[37].
3. The *Validation* Dataset is used to **predict** the model performance. The predicted outcome is compared with the target label (known outcome) using the chosen metric (area under the ROC curve).

After the model has been fitted and its hyperparameters have been tuned, we use the pre-processed *Application Test* dataset to predict the outcome to loan default. Given the

[34] Towards Data Science, Tarang Shah, About Train, Validation and Test Sets in Machine Learning, towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.
[35] Machine Learning Mastery, Jason Brownlee, What is the Difference Between Test and Validation Datasets?, machinelearningmastery.com/difference-test-validation-datasets/.
[36] Scikit Learn, train_test_split, scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.
[37] Scikit Learn, GridSearchCV, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

Application Test dataset has no *target label*, we submit the prediction to the competition and obtain a prediction score.

To optimize the resources utilization and obtain a faster execution, the datasets were optimized after loading from the csv files. For each numerical value the smallest primitive type was used (np.int8, np.int16, np.int32, np.int64, np.float16, np.float32, np.float64) resulting in a 71% decrease in memory utilization. Similarly these original datasets were deleted from memory once they were preprocessed and merged into one final dataset.

3.3. Refinement

The initial approach was to reach the Random Forest benchmark AUC score of 0.688. Hence the RandomForestClassifer model was selected. The first pass included the model with its default hyperparameters and leveraging the *Application Train* dataset only handling missing values and categorical features encoding. This yield a score of 0.591. Next the model's hyperparameters were fine-tuned, and still using the *Application Train* dataset obtained a score of 7.22, surpassing "the" benchmark.

The next iteration represented a setback give the obtained score was 0.71. The only changed was the dataset used. Instead of just using the *Application Train* dataset, an initial merge of the full datasets was used. The problem was that this merge was dropping the categorical features. After this was corrected (the categorical features were encoded before merging the datasets) and more preprocessing performed, the model's score was 0.744 (almost surpassing the benchmark stretched goal). This concluded the initial pass.

After running the same process for different models and measuring their performance, the LightGBM was chosen for the final submission. The same comparisons were made and the results can be found on Table 3.

Estimator	Comments	Calculated AUC	Competition AUC score
Naive estimator	Leaderboard Benchmark	N/A	0.500
Random Forest	Leaderboard Benchmark	N/A	0.688
LightGBM	Tuned Hyperparameters Data: merged datasets (one-hot encoding before flattening) Sampling: KFold CV	0.782816	0.779 (top 49% on 6/15/18)
LightGBM	Tuned Hyperparameters Data: merged datasets (one-hot encoding before flattening)	0.782326	0.775 (top 56% on 6/4/18)

LightGBM	Tuned Hyperparameters Data: preprocessed app_train dataset	0.762501	0.744
LightGBM	Default Hyperparameters Data: preprocessed app_train dataset	0.758563	0.739
RandomForestClassifier	Tuned Hyperparameters Data: merged datasets (one-hot encoding before flattening)	0.759890	0.744
RandomForestClassifier	Tuned Hyperparameters Data: merged datasets (one-hot encoding after flattening)	0.754273	0.710
RandomForestClassifier	Tuned Hyperparameters Data: preprocessed app_train dataset	0.749339	0.733
RandomForestClassifier	Tuned Hyperparameters Data: unprocessed app_train dataset	0.748494	0.722
RandomForestClassifier	Default Hyperparameters Data: preprocessed app_train dataset	0.659334	0.650
RandomForestClassifier	Default Hyperparameters Data: unprocessed app_train dataset	N/A	0.591

Table 3: Models output comparison

4. Results

4.1. Model Evaluation and Validation

In order to arrive at the final model and to understand the importance and impact of each change to the datasets and the model selection and tuning, each change was separate and the final prediction submitted to the competition for scoring. This facilitated identifying changes that lead to improvements and other that did not.

The initial pass included no datasets merge and basic preprocessing (missing values imputing and categorical features encoding) with a RandomForestClassifier with default hyperparameters. The final pass included a merged dataset with more robust preprocessing with a LightGBM model with tuned hyperparameters. The most significant improvements in overall competition scoring performance were:

1. Hyperparameter tuning
2. Dataset merging

The two main points were expected as they are the bulk of the work, so no surprise there. Steps that did not yield a significant improvement or actually decreased the score were:

1. Dataset merging without encoding categorical features first
2. Feature engineering (decreased the score)
3. Data normalization (no change to the score)

These results on the other hand were unexpected, except for point 1. The interpretation derived is that they were not implemented correctly. They are definitely in the list for future improvements.

In order to assess the robustness of the final model, **KFold**[38] cross-validation and stratified sampling (**StratifiedKFold**[39]) were implemented. These two were originally left for future improvements. When doing cross-validation the validation set is no longer needed. The Training set is split into k smaller sets and the model is trained using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data and helps with overfitting.[40]

After fitting the model with 5 folds (without stratification) the average AUC calculated score obtained was 0.782816 with a standard deviation of 0.002451, and a competition AUC score of 0.779. Later the model was fitted with 5 folds (with stratification using StratifiedKFold) the average AUC calculated score obtained was 0.782215 with a standard deviation of 0.003404, and a competition AUC score of 0.778.

4.2 Justification

The current best prediction[41] submitted for the competition received a score of **0.779** and is among the top **49%** of the overall submissions (position 2,260 out of 4,625). This score is **13.2%** higher than the Random Forest Benchmark (which was "the" chosen benchmark), **4.4%** higher than the stretch benchmark goal and **3.2%** lower than the highest score. This result is significant enough to consider it solves the Home Credit Default Risk Kaggle competition. The competition ends on August 29, 2018 at 11:59PM UTC, which will allow for the implementation of some of the improvements listed in section 5.3 after this report is submitted.

The final model used was the LightGBM, with the following hyperparameters[42]:

```
nthread=4,
n_estimators=10000,
learning_rate=0.02,
```

[38] Scikit Learn, KFold, scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.
[39] Scikit Learn, StratifiedKFold, scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html.
[40] Scikit Learn, Cross-validation: evaluating estimator performance, scikit-learn.org/stable/modules/cross_validation.html.
[41] As of July 15, 2018, Kaggle profile, www.kaggle.com/nachorovi.
[42] LightGBM, Parameters, lightgbm.readthedocs.io/en/latest/Parameters.html.

```
num_leaves=34,
colsample_bytree=0.9497036,
subsample=0.8715623,
max_depth=8,
reg_alpha=0.041545473,
reg_lambda=0.0735294,
min_split_gain=0.0222415,
min_child_weight=39.3259775,
```

The *auc* score obtained on the validation dataset was 0.782816 and the model fit time was on average 296 seconds per fold, very quick for a training dataset with 307,511 samples and 493 features.

Estimator	Comments	Competition AUC score
Random Forest	Leaderboard Benchmark	0.688
Top 1	Leaderboard 6/15/18	0.805
Top 100	Leaderboard 6/15/18	0.801
Top 500	Leaderboard 6/15/18	0.799
Top 1,000	Leaderboard 6/15/18	0.797
LightGBM	tuned, merged data, KFold	0.779

Table 4: Final submission score comparison

5. Conclusion

5.1. Free-Form Visualization

One key step in the iterative process which can help increase the model's performance is to perform *feature importance analysis*. A model's feature importance reveals which features provide the most *predictive power*, that can explain the relationship between the crucial features and the target label.

Thus using the feature importance scores we can prune the feature set and train the model again. Another way to leverage this information is to focus on preprocessing (normalization, unskewing, removing outliers, etc.) first the features with the highest scores. When looking for possible improvements, the focus was first placed on the features with the highest importance in the final model. This also helps reduce the model training time and resources utilized (mostly memory) as the feature set is reduced.

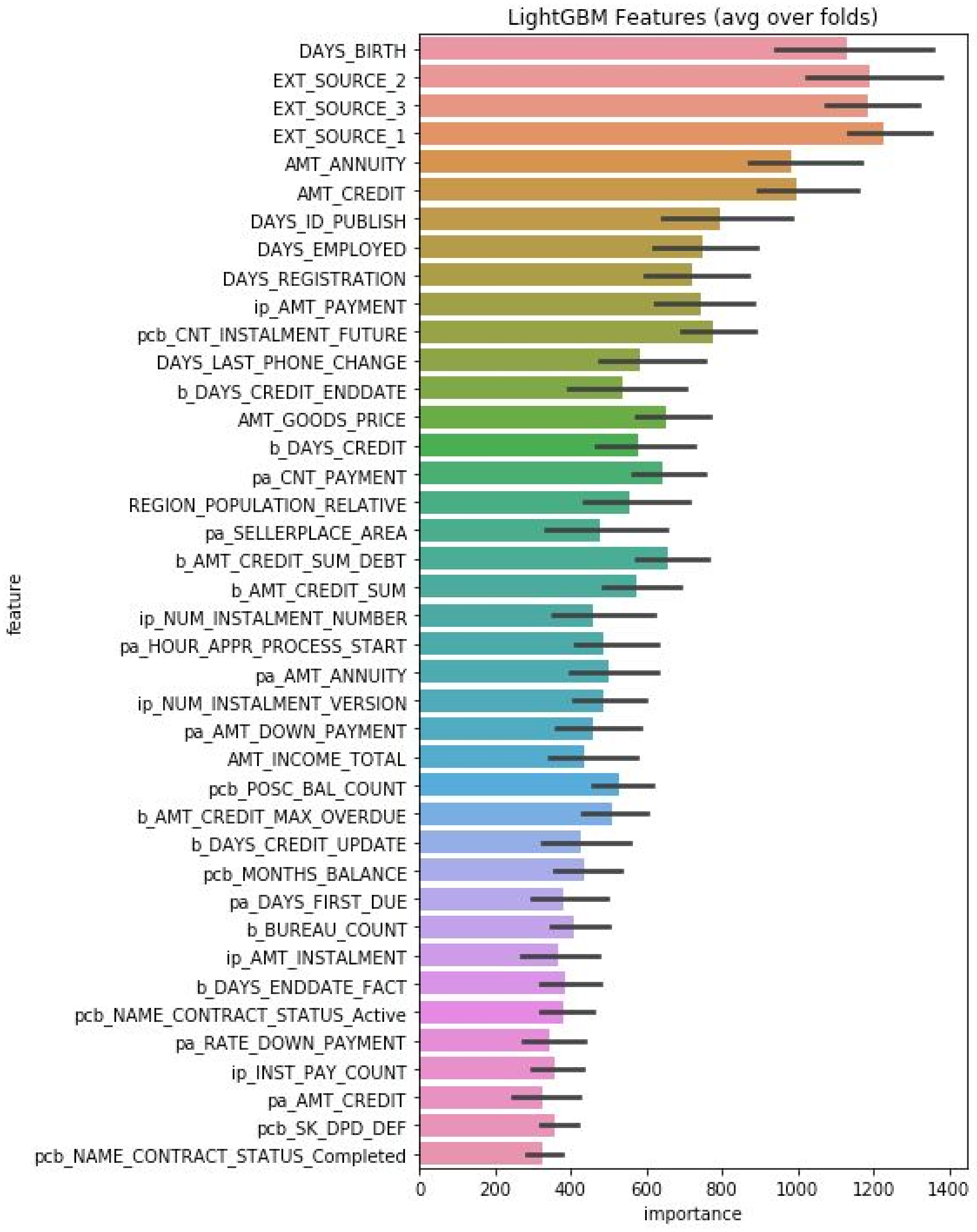


Figure 5: Feature importance

5.2. Reflection

The initial goal was to make a better prediction than the naive estimator (random chance). Afterwards, the goal was to beat the Random Forest benchmark. Finally, the ultimate goal was to obtain the highest score possible. To achieve this, I needed to be able to iterate quickly and be able to understand the impact each change had on the score.

The initial iteration performed basic pre-processing (imputing missing values and encoding categorical variables) and using the Random Forest Classifier (used for the second benchmark) with its default settings. Next the problem was broken into main areas: Data preprocessing and Model selection and tuning.

Iterations for Data preprocessing were faster and three main areas stood out: Missing values, merging datasets and Feature Engineering. The learning when dealing with missing values were that most models don't handle it well, it can be appropriate to drop columns when too many samples are missing a value and that for practical purposes imputing missing, in its different forms, can yield in better model performance.

Merging dataset was a new challenge that took multiple tries and dissecting the datasets before and after merging to understand it was dropping categorical features and to eventually find a satisfactory implementation. But the most interesting learning was Feature Engineering. To think and understand that combining the original datasets into new features can generate a more meaningful dataset and lead to higher model predictive accuracy was inspiring. After several failed attempts to come up with useful new features it quickly became clear why it is more art than science and it is acquired with experience.

Iterations for model selection and hyperparameter tuning were simpler but more time consuming and required more resources. The fact that different models have different data preprocessing requirements was put aside and the datasets were preprocessed to work with all models alike. Testing the models with default hyperparameters is simple. Doing basic hyperparameters tuning for models with just a few ones doesn't take very long. But tuning the LightGBM model with the amounts of parameters it has was a challenge. Multiple times the Kaggle kernel used to run the code was killed after 6 hours of running. So tuning this model took multiple attempts and hours at a time.

On a personal note, I wanted to work on a relevant, real challenge and I find the consumer lending space to be one. So mission accomplished. A second objective was to complete an online competition (clear requirements, datasets and measurable outcome) to become familiar with process and have the expertise to complete others in the future. Again, mission accomplished. Which made the Capstone Project a very rewarding learning experience.

5.3. Improvement

Every aspect covered can be further improved to increase the competition score. Since the first successful submission, every improvement was done in small iterative steps, making sure a new submission was ready to be send if considered appropriate. Making it quick and simple to understand the positive or negative effect of the change introduced.

Next is a list future steps for possible improvements:

1. Data Preprocessing:
    a. Applications Train dataset target distribution imbalance.
    b. Handle missing values:
        i. Drop columns missing more than 70% of the samples
        ii. Use the median instead of mean for numeric features
        iii. Use mode for categorical features before encoding categorical features
    c. Normalization - Was tried with no effect. Try again.
    d. Skewed data - Identify skewed features and fix it them. Was tried and failed.
    e. Removing outliers - Remove outliers for main features
    f. Dimensionality reduction using PCA, 493 are too many features.
        i. Feature Selection[43]
2. Feature engineering:
    a. Flattening datasets:
        i. New feature introduced was counting the number of samples per SK_ID_CURR.
        ii. *Mean* was used when merging multiple samples for the same SK_ID_CURR. Add the *min*, *max*, *mode* and *median* values as well.
        iii. Consider other ways to extract information.
    b. Within a dataset:
        i. Ratios between features. Ex. for application_train.csv:
            1. AMT_CREDIT / AMT_INCOME_TOTAL
            2. AMT_GOODS_PRICE / AMT_CREDIT
        ii. Sums or differences between features. Ex. for application_train.csv:
            1. SUM of FLAG_DOCUMENT_X
3. Model selection
    a. Try different models
        i. Logistic Regression[44]
        ii. Random Forest[45]

[43] Machine Learning Mastery, Jason Brownlee, An Introduction to Feature Selection, machinelearningmastery.com/an-introduction-to-feature-selection/.
[44] Scikit Learn, LogisticRegression, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.
[45] Scikit Learn, RandomForrestClassifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

iii. Decision Tree[46]
iv. Gaussian Naive Bayes[47]
v. Extreme Gradient Boosting (XGBoost)[48]
vi. Gradient Boosting[49]
vii. AdaBoostClassifier[50]
viii. SupportVectorMachine[51]
ix. Bagging Classifier[52]
x. Ensemble Methods[53]

[46] Scikit Learn, DecisionTreeClassifier, scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.
[47] Scikit Learn, GaussianNB, scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.
[48] XGBoost, xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier.
[49] Scikit Learn, GradientBoostingClassifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.
[50] Scikit Learn, AdaBoost Classifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier.
[51] Scikit Learn, Support Vector Classification, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.
[52] Scikit Learn, BaggingClassifier, scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html.
[53]Scikit Learn, Ensemble Methods, scikit-learn.org/stable/modules/ensemble.html.