Machine Learning Engineer Nanodegree

Capstone Project

Christian Nava

October 18, 2018

Using Supervised Learning to Predict Home Credit Default Risk

I. Definition

Project Overview

Home Credit Group is an international consumer finance provider with operations in 10 countries that focuses on responsible lending primarily to people with little or no credit history. Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Home Credit Group has partnered with Kaggle to help them unlock the full potential of their data and ensure that potential clients who are capable of repayment are not rejected for a loan.

Problem Statement

Predicting whether or not a client will repay, or have difficulty repaying, a loan is a critical business need. Creating a robust, effective, and automated risk score that can help with that prediction is still a major challenge for many banks. Application, demographic, and historical credit behavior data provided by Home Credit will be used to predict the future payment behavior of clients.

Solution Outline

As supervised learning problem techniques have been successful in predicting loan default probability, in this project, decision-tree based and ensemble classifiers will be used to predict whether or not a client will repay, or have difficulty repaying, a loan. The dataset will be preprocessed by joining data across several files and creating a set of features applicable to credit analysis and loan default. A logistic regression model will be used as a baseline or benchmark model and the results will then be compared with those from a random forest model and a LightGBM model.

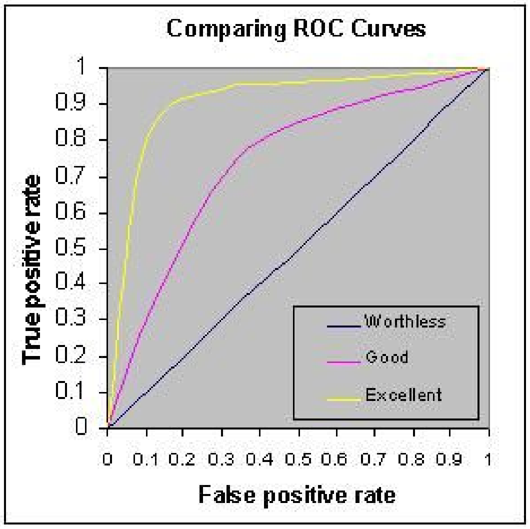
Gradient Boosting Decision Trees will be used as the main algorithm for a solution to the problem. LightGBM is a gradient-boosting framework that uses tree-based learning algorithms. It is a relatively new algorithm. The paper where LightGBM was introduced was published in December 2017 by Guolin Ke, et al and shows two ways for improving the usual gradient boosting algorithm where weak classifiers are decision trees, and the paper is oriented towards efficient implementation of the usual algorithm in order to speed up the learning of decision trees by taking into account previous computations and sparse data.

Using a tree-based model such as random forest or LightGBM appears to be a better fit than other models when undertaking a binary classification task as reported by Addo et al in their paper on predicting loan default probability.

Metrics

The metric used for this Kaggle competition is the **area under the receiver operating characteristic (ROC) curve**. The ROC curve graphs the performance of a classification model at all classification thresholds; it plots the true positive rate (TPR) against the false positive rate (FPR) as shown in Figure 1.

Figure 1. ROC Curve Comparison
Source: University of Nebraska Medical Center



Each curve represents the graph of a single model. Movement along the curve indicates changing the threshold used for classifying a positive instance. The area under the ROC curve measures the two-dimensional area under the ROC curve and is given by the integral of the curve, or the area under the curve (AUC):

$$AUC = \int_0^1 TPR(x)dx = \int_0^1 P(A > \tau(x))dx$$

where *x* is the FPR and *A* is the distribution of scores the model produces for data points that are actually in the positive class.

This metric is measured between 0 and 1. A higher score indicates a better model. Visually, a superior model will graph as a curve above and to the left of another curve. A straight diagonal line beginning at 0 on the bottom left and ending at 1 on the top right indicates a naive model (random guessing).

Other metrics such as accuracy are not the best metric for data with imbalanced classes.The Home Credit dataset is imbalanced: 92% of clients repaid the loan and 8% did not repay the loan. If we say “repaid” is the is the positive class and “did not repay” is the negative class, then high accuracy can be achieved with a model that classifies all loans as “did not repay”. Accuracy for this model would be close to 100%, however, the recall, the metric that identifies the proportion of actual positives that were identified correctly, would be zero since the model would never correctly predict if the loan was "repaid".

A model with a high ROC AUC will also have a high accuracy, therefore, the AUC score will be the metric employed in this analysis.

II. Analysis

Data Exploration

The data is provided by Home Credit, a service dedicated to providing lines of credit (loans) to the unbanked population.

There are 7 different sources of data:

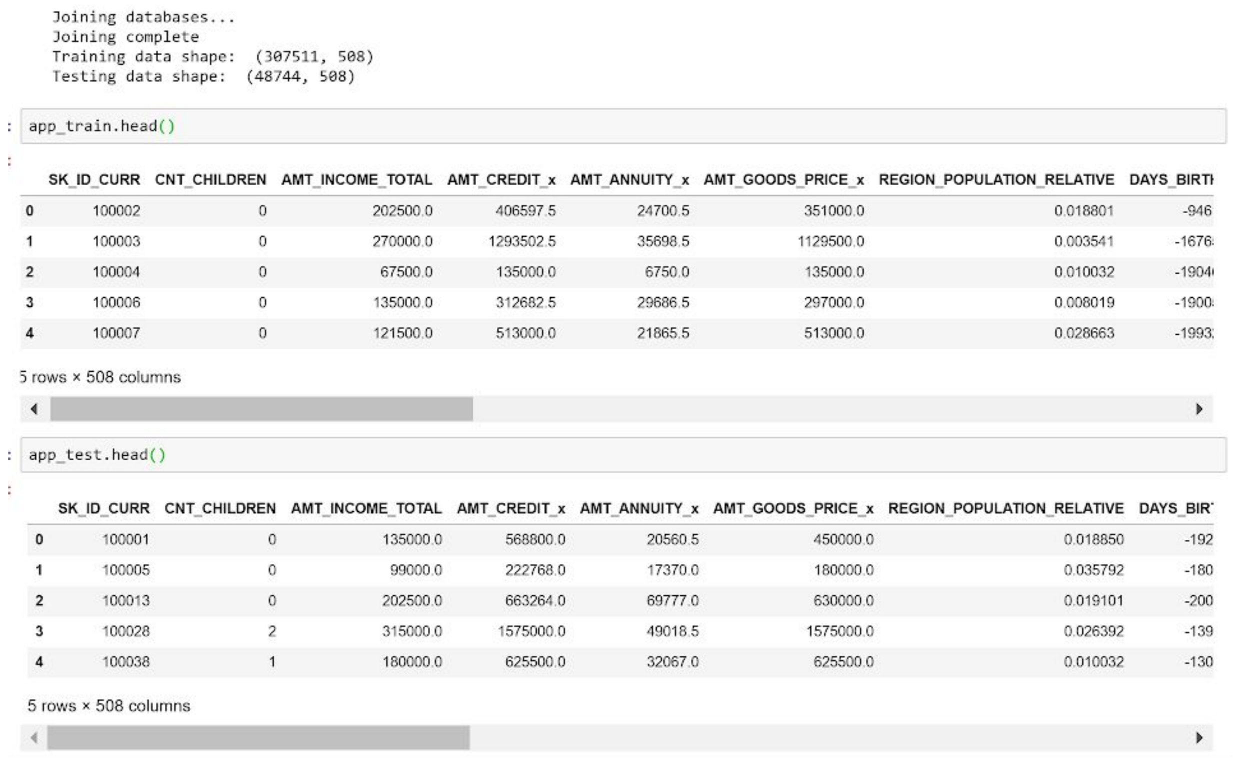
1. **application_train/application_test** (308k rows x 122 columns/48.7k rows x 121 columns): the main training and testing data with information about each loan application at Home Credit. It is broken into two files for Train (with TARGET) and Test (without TARGET).Every loan has its own row and is identified by the feature SK_ID_CURR.
2. **Bureau** (1.72m rows x 17 columns): data concerning client's previous credits from other financial institutions that were reported to Credit Bureau (for clients who have a loan in this sample). Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
3. **Bureau_balance** (27.3m rows x 3 columns): monthly data about the previous credits in 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. A single previous credit can have multiple rows, one for each month of the credit length.
4. **Previous_application** (1.6m rows x 37 columns): previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
5. **POS_CASH_BALANCE** (10.0m rows x 8 columns): monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one

month of a previous point of sale or cash loan, and a single previous loan can have many rows.

6. **Credit_card_balance** (3.84m rows x 23 columns): monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
7. **Installments_payment** (13.6m rows x 8 columns): payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

Once all datasets have been preprocessed and combined the training data set has 307,511 rows and 508 columns (features) and the testing data set has 48,744 rows and 508 columns.

Figure 2. Sample of Data



```
Joining databases...
Joining complete
Training data shape:  (307511, 508)
Testing data shape:  (48744, 508)
```

```
app_train.head()
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIRTI
0	100002	0	202500.0	406597.5	24700.5	351000.0	0.018801	-946
1	100003	0	270000.0	1293502.5	35698.5	1129500.0	0.003541	-1676
2	100004	0	67500.0	135000.0	6750.0	135000.0	0.010032	-1904
3	100006	0	135000.0	312682.5	29686.5	297000.0	0.008019	-1900
4	100007	0	121500.0	513000.0	21865.5	513000.0	0.028663	-1993

5 rows × 508 columns

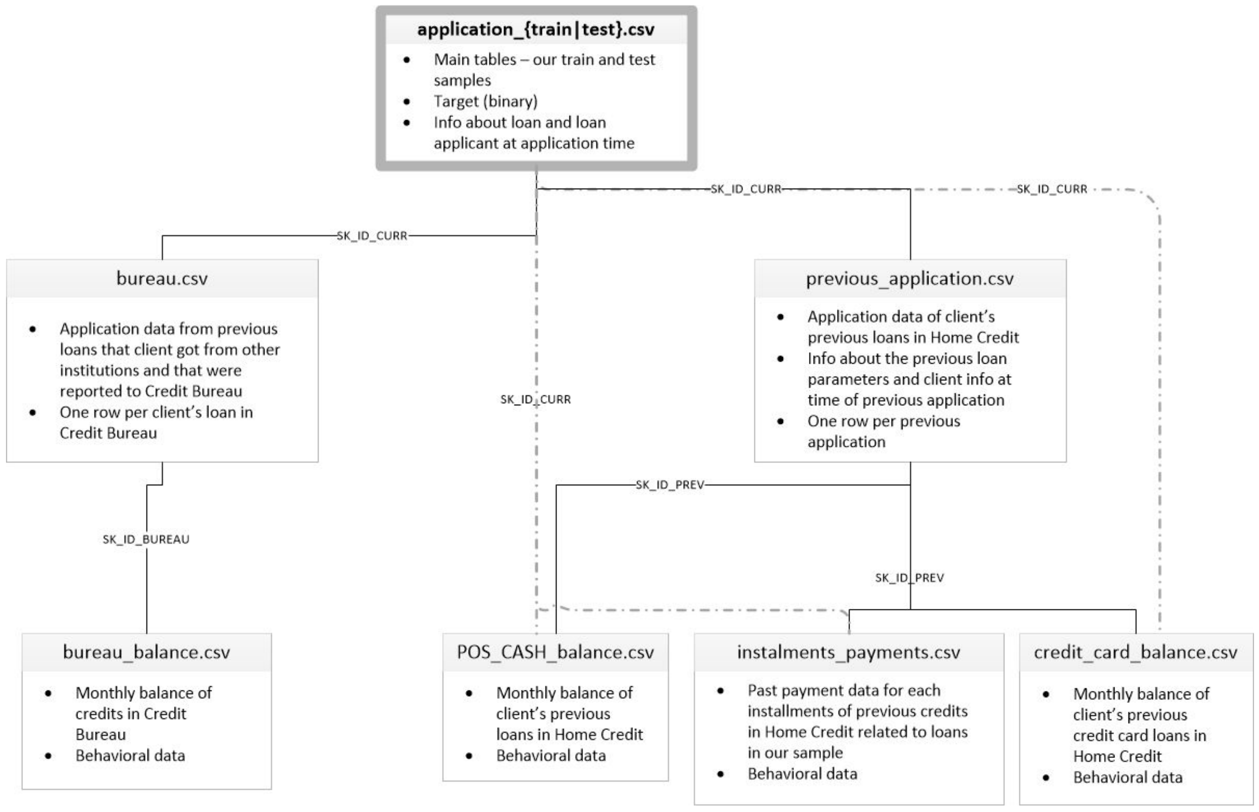
```
app_test.head()
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIVE	DAYS_BIR
0	100001	0	135000.0	568800.0	20560.5	450000.0	0.018850	-192
1	100005	0	99000.0	222768.0	17370.0	180000.0	0.035792	-180
2	100013	0	202500.0	663264.0	69777.0	630000.0	0.019101	-200
3	100028	2	315000.0	1575000.0	49018.5	1575000.0	0.026392	-139
4	100038	1	180000.0	625500.0	32067.0	625500.0	0.010032	-130

5 rows × 508 columns

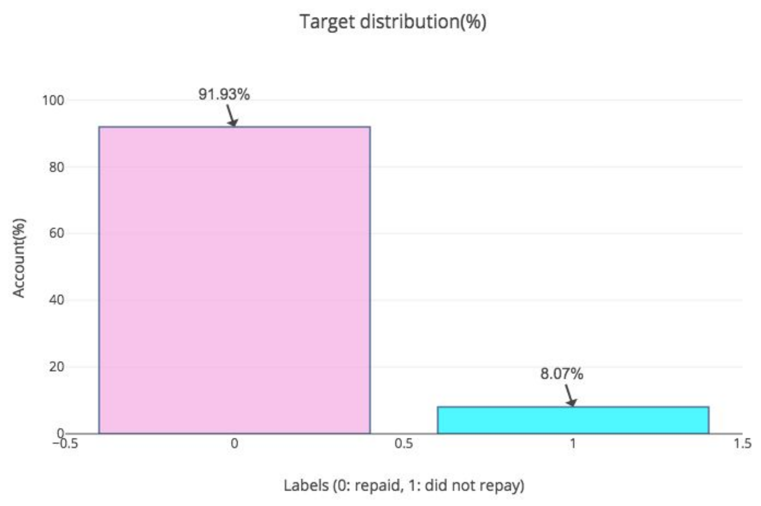
The Figure 3 illustrates the relationship of the data.

Figure 3. Relationship of 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_BUREAU
SK_ID_PREV
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

The training application data comes with the TARGET (response variable). A value of 0 indicates the loan was repaid, a value of 1 indicates the loan was not repaid. The distribution of the target class in the training file is imbalanced, that is, more loans were repaid on time than not. In an imbalanced classification task such as this, the area under the ROC curve is an appropriate metric. The classes can be weighted by their representation in the data to reflect the imbalance.

Figure 4. Imbalanced Target Distribution



Exploratory Data Analysis

In this section, a selection of the most important and representative features are visualized.

Effect of Age on Repayment

Per the documentation, DAYS_BIRTH is the age in days of the client at the time of the loan in negative days. The value of the feature is negative, however, the correlation is positive, which means that as clients get older they are less likely to have difficulty repaying the loan (i.e., the TARGET == 0).

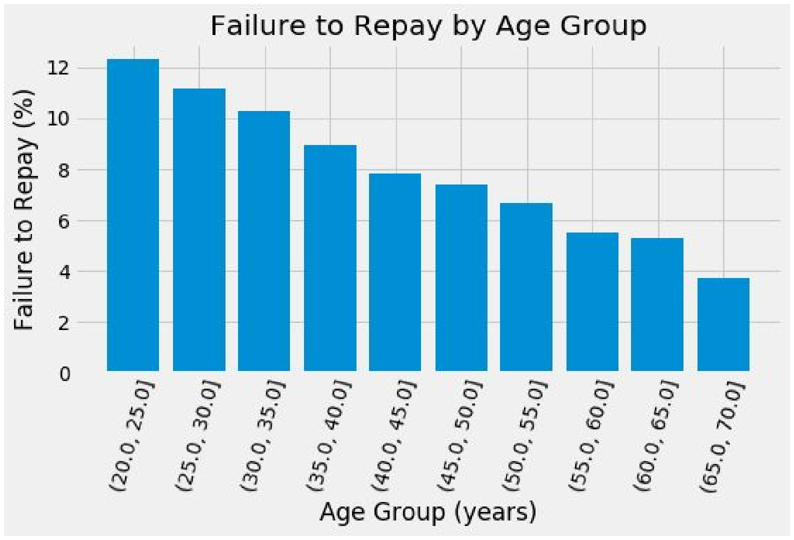
Taking the absolute value of the feature results in a negative correlation value of -0.0782393. As the client gets older, there is a negative linear relationship with the TARGET variable meaning that the client is less likely to have difficulty repaying the loan.

By itself, the distribution of age does not tell us much other than that there are no outliers as all the ages are reasonable.

Average Failure to Repay Loans by Age Bracket

In the graph below, the age category was cut into bins of 5 years. For each bin, the average value of the target was calculated, which gives the ratio of loans that were not repaid in each age category. As can be seen below, younger applicants are more likely not to have difficulty repaying the loan. The rate of failure to repay is above 10% for the youngest three age groups and below 5% for the oldest age group.

Figure 5. Failure to Repay by Age Group

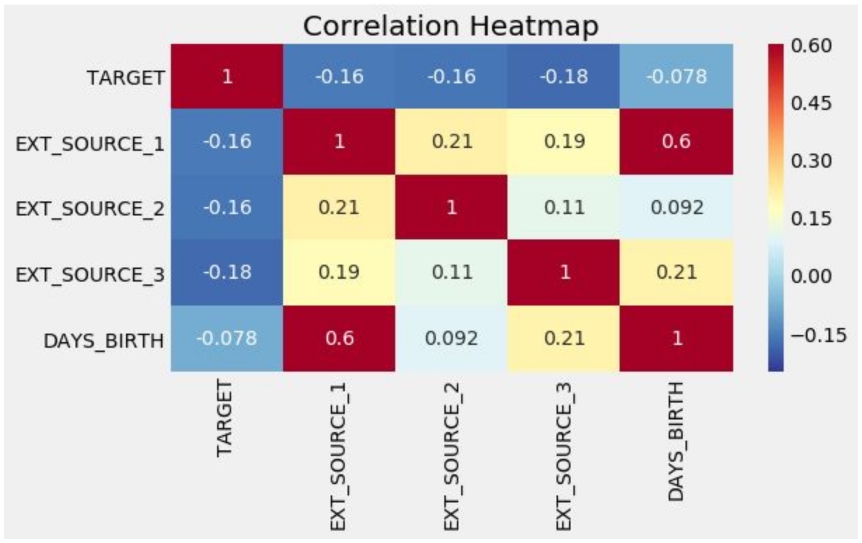


Exterior Sources

The following three variables have the strongest negative correlations with the target variable: EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3. Per the documentation, these features represent a "normalized score from external data source", which is unclear but may be an aggregated credit rating created from various sources of data.

Below is a heat map of the correlations of the EXT_SOURCE features with the target variable and with each other.

Figure 6. Correlation Heatmap



	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1	-0.16	-0.16	-0.18	-0.078
EXT_SOURCE_1	-0.16	1	0.21	0.19	0.6
EXT_SOURCE_2	-0.16	0.21	1	0.11	0.092
EXT_SOURCE_3	-0.18	0.19	0.11	1	0.21
DAYS_BIRTH	-0.078	0.6	0.092	0.21	1

All three EXT_SOURCE features have negative correlations with the target variable, indicating that as the value of the EXT_SOURCE increases, the client is more likely to repay the loan. We can also see that DAYS_BIRTH is positively correlated with EXT_SOURCE_1 indicating that maybe one of the factors in this score is the client's age.

EXT_SOURCE_3 displays the greatest difference between the values of the target. We can clearly see that this feature has some relationship to the likelihood of an applicant to repay a loan. The relationship is not very strong. In fact, they are all considered weak or very weak correlations:

- .00-.19 “very weak”
- .20-.39 “weak”
- .40-.59 “moderate”
- .60-.79 “strong”
- .80-1.0 “very strong”

For example, a correlation value of .46 would be a “moderate, positive correlation”. However weak these variables are, they will still be useful for a model to predict whether or not an applicant will repay a loan on time.

Algorithms and Techniques

Logistic Regression

Logistic regression is a method to estimate the probability of a binary response based on one or more independent variables. The logistic formulas are stated in terms of the probability that $Y = 1$, which is referred to as $\widehat{p}$. The probability that Y is 0 is $1 - \widehat{p}$. The logistic function can be written as

$$ln\left(\frac{\widehat{p}}{1-\widehat{p}}\right) = \beta_0 \ + \beta_1 x_1 + \ \beta_2 x_2 + \ldots + \beta_n x_n$$

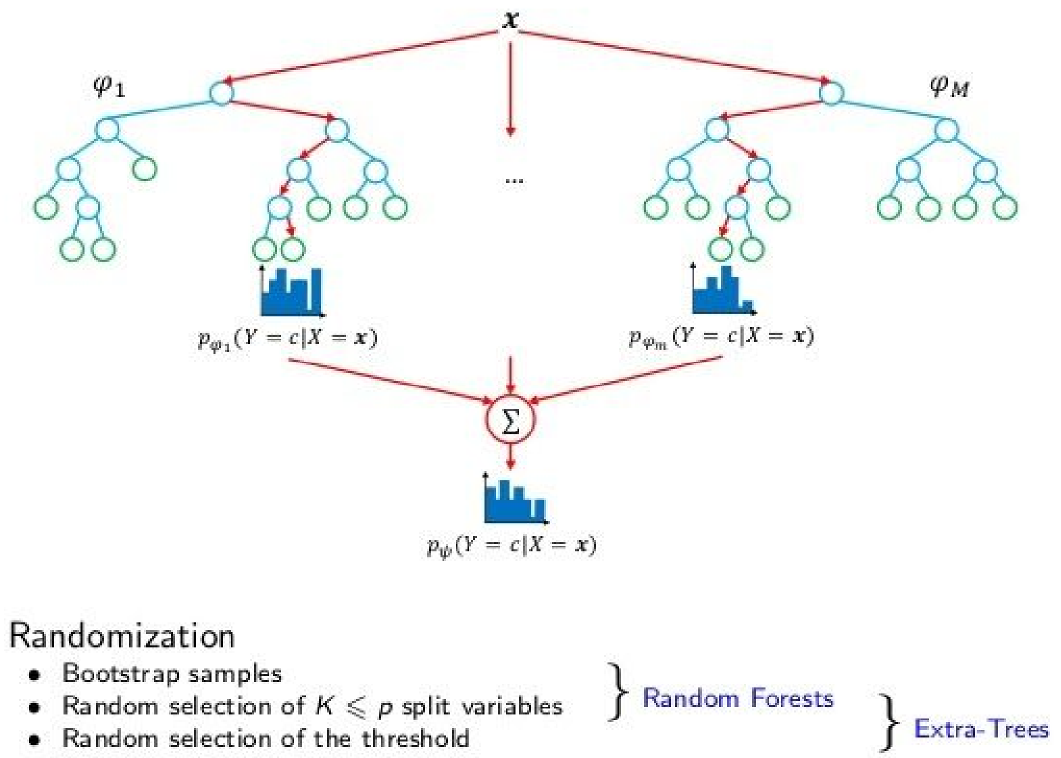
where $\hat{p}$ is the probability of the response variable to be predicted, β_0 is the intercept from the linear regression equation, β_n is the regression coefficient, and x_n is a predictor variable (feature).

LogisticRegression from the Scikit-Learn library will be used as a benchmark model. The data will be preprocessed by filling in the missing values (imputation) and normalizing the range of the features (feature scaling).

Random Forest

The random forest model is an ensemble method of learning for classification and is more powerful than logistic regression. The random forest model groups a set of decision trees and then it predicts based on the majority of votes (when used for classification) or aggregation (when used for regression) from each of the decision trees made.

Figure 7. Random Forest Diagram
Source: KDNuggets


x
φ_1
φ_M
...
$p_{\varphi_1}(Y = c|X = x)$
$p_{\varphi_m}(Y = c|X = x)$
Σ
$p_\psi(Y = c|X = x)$
Randomization
Bootstrap samples
Random selection of $K \leqslant p$ split variables
Random selection of the threshold
Random Forests
Extra-Trees

This ensemble method is based on randomization where each tree in the forest is built randomly with the same distribution. Each tree is trained in isolation. The basic procedure for the algorithm involves the following steps illustrated above.

1. Select φ_M bootstrap samples from the training data, χ.
2. For each of the bootstrap samples φ_M, grow an unpruned classification or regression tree to the largest extent possible (i.e., split the node into two daughter nodes until the minimum node size is reached).
3. Predict new data by aggregating the predictions of the φ_M trees. These predictions can be assembled into a probability distribution, whose mode (i.e., majority votes for classification) or average (i.e., aggregation for regression) yields the random forest's proposed rating $p_\psi(Y = c|X = \chi)$.

Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data, which is the case with the Home Credit dataset as shown in the Data Exploration section.

Light Gradient Boosting Machine

The Gradient Boosting Machine algorithm is currently the leading model for learning on structured datasets. Gradient Boosting Machine will be implemented using the LightGBM library. LightGBM is a gradient-boosting framework that uses tree-based learning algorithms, gradient boosted decision trees (GBDT).

Gradient Boosted Decision Trees is a machine learning algorithm that is built over time by adding a new tree into the model that minimizes the error by previous learners. This is done by fitting the new tree on the residuals of the previous trees. The algorithm consists of splitting a node and then, using the best split, the sample space is partitioned by growing n nodes and updating the residuals.

It was introduced by Jerome Friedman in 1999. Friedman begins, by stating that "given a 'training' sample $\{y_i, x_i\}_1^N$ of known (y, x) values, the goal is to find a function $F^*(x)$ that maps x to y, such that over the joint distribution of all (y, x) values, the expected value of some specified loss function $\Psi(y, F(x))$ is minimized." The algorithm looks like this:

F is initialized with a constant value.

$$F_0(x) = argmin_\gamma \sum_{i=1}^{N} \Psi(y_i, \gamma)$$

For m = 1 to M do:

Compute pseudo residuals at each estimator m (iteration), which gives the best steepest-descent step direction in the N-dimensional data space at $F_{m-1}(x)$.

$$\tilde{y}_{im} = -\left[\frac{\partial\Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \qquad , i = 1, N$$

Fit the base learner to pseudo residuals.

$$\left\{R_{lm}\right\}_1^L = L - terminalnodetree\left(\left\{\tilde{y}_{im}, x_i\right\}_1^N\right)$$

Compute a step magnitude multiplier γ_{lm}. In the case of tree models, compute a different γ_{lm} for every leaf.

$$\gamma_{lm} = argmin_\gamma \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma)$$

Update.

$$F_m(x) = F_{m-1}(x) + \nu \cdot \sum_{i=1}^{L} \gamma_{lm} 1(x \in R_{lm})$$

end

Where $\{y_i, x_i\}$ is the dataset,

Ψ is the loss function we want to minimize,

γ is the node score (step magnitude multiplier),

M is the number of trees,

N is the size of the training set,

$F_m(x)$ is the m^{th} tree,

$\tilde{y}_{im}$ are the residuals, and

ν is the shrinkage parameter, which controls the learning rate.

LightGBM can handle large datasets (>10,000 rows) and does not require a lot of memory to run. LightGBM grows a decision tree in a leaf-wise method (i.e., it will select the leaf with the largest delta loss, which can reduce loss more than a level-wise algorithm).

Benchmark Model

Naive random guessing (luck) will produce a benchmark model of 0.5 area under the ROC curve. Anything under a 0.5 area under the ROC curve would indicate mistaken labeling of classifier targets or a bad training algorithm.

Random guessing is rather simplistic and a better approach would be to use logistic regression as a benchmark model.

Baseline benchmark results from the logistic regression model yielded an AUC score of 0.760940.

III. Methodology

Data Preprocessing

Feature Engineering

Feature engineering refers to a general process and can involve adding new features from the existing data (feature construction) and choosing only the most important features or other methods of dimensionality reduction.

The following features have been created:

1. **CREDIT_INCOME_PERCENT**: the percentage of the credit amount relative to a client's income
2. **ANNUITY_INCOME_PERCENT**: the percentage of the loan annuity relative to a client's income
3. **CREDIT_TERM**: the length of the payment in months (since the annuity is the monthly amount due)
4. **DAYS_EMPLOYED_PERCENT**: the percentage of the days employed relative to the client's age

Label Encoding and One-hot Encoding Categorical Variables

Models such as LightGBM can handle categorical variable without much preprocessing. However, logistic regression and random forest models cannot handle categorical variables and we must encode, or transform, the categorical variables into numbers that can be handled by the models. Label encoding assigns an integer to each unique category in a categorical variable, and it does not create any new columns. Label encoding gives the categories an arbitrary ordering. If we only have two unique values for a categorical variable (such as Male/Female), then label encoding is fine, but for more than 2 unique categories, one-hot encoding is the safer option.

One-hot encoding creates a new column for each unique category in a categorical variable. Each observation receives a 1 in the column for its corresponding category and a 0 in all other new columns. The only downside to one-hot encoding is that the number of features (dimensions of the data) can explode with categorical variables with many categories.

Label encoding will be used for any categorical variables with only 2 categories and one-hot encoding for any categorical variables with more than 2 categories. For label encoding, we use the Scikit-Learn LabelEncoder and for one-hot encoding the pandas get_dummies (df) function.

Missing Values

Missing values will be filled in with the median value of the column via imputation and the range of the features will be normalized (feature scaling). Since the LightGBM model does not need missing values to be imputed, we can directly fit on the training data for that model. Early stopping will be used to determine the optimal number of iterations and run the model twice, averaging the feature importance to try and avoid overfitting to a certain set of features.

Implementation

Logistic Regression and Random Forest Implementation

Data were split into a training and testing set using train_test_split, where 30% of the data were reserved for testing and 70% for training for the logistic regression and random forest implementations.

The three methods discussed and justified in the Algorithms and Techniques section were used here to perform the binary classification task. For the LightGBM implementation, k-fold cross-validation was used. Default parameters for these three methods are used and the AUC Scores received for the default parameters were as follows:

Figure 8. Comparison of AUC Scores with Default Parameters

Model (default parameters)	AUC Score
Logistic Regression	0.760940
Random Forest	0.629973
LightGBM	0.774867

Refinement

Logistic Regression

No modifications were made to the logistic regression benchmark model.

Random Forest

It was surprising to see the random forest model perform worse than the baseline model. Hyperparameters were changed manually. Initially the AUC score with the default parameters was 0.721359. Changing the number of estimators to 400, the min_sample_leaf value was changed from 2 to 50. This gave the score a bump to 0.723440; however, it also slowed the model a bit, which makes sense given the increase in the number of estimators.

Even after tuning hyperparameters, the random forest model did not outperform the benchmark model and only achieved an AUC score of 0.725734, which suggests that linear combinations of features may have more predictive power for this dataset.

LightGBM

For the LightGBM model, the parameters were obtained by Bayesian optimization from the kernel created by the Kaggle user Tilii7. After spending days trying to figure out how to implement random and grid search cross-validation and manually tuning hyperparameters, I found that these hyperparameters produced the best results. Per the LightGBM documentation, the learning_rate, num_leaves, max_depth, reg_alpha, and reg_lambda seemed to be some of the most important parameters and I manually adjusted those as a starting point, but soon realized there are too many parameters to tune manually that it would take a very long time of trial and error to get a significant improvement in the AUC score.

A comparison between the default and the tuned/optimized parameters along with the corresponding AUC score is show below in Figure 9.

Figure 9. LGBM Default and Bayesian Optimized Parameters with AUC Score

Parameter	Default	Tuned/Optimized
learning_rate	0.1	0.03
n_estimators	100	10000
max_depth	-1	8
num_leaves	31	34
min_child_weight	1e-3	39.3259775
reg_alpha	0.0	0.041545473
reg_lambda	0.0	0.0735294
subsample	1.0	0.8715623
colsample_bytree	1.0	0.9497036
AUC Score:	**0.774867**	**0.781326**

Without the default parameters, the LightGBM model achieved an AUC score of 0.774867. Using the Bayesian optimized parameters the model achieved an AUC Score of 0.781326. For the Bayesian optimization, 15 initialization and 25 optimization iterations were run over the ranges listed in Figure 10 and the GaussianProcessRegressor from the sklearn library was employed.

Figure 10. Parameter Ranges for Bayesian Optimization

```
'max_depth': (3, 8.99),
'min_data_in_leaf': (20, 100),
'num_leaves': (20, 100),
'min_child_weight': (1, 100),
'reg_alpha': (0.00001, 0.2),
'reg_lambda': (0.00001, 0.2),
'min_split_gain': (0.00001, 0.1),
'subsample': (0.2, 1.0),
'colsample_bytree' :(0.2, 1.0)
```

With both the Random Forest and the LightGBM models, tuning the hyperparameters achieved an increase in the overall AUC Score. However, I found that adding the engineered features accounted for more of a change in the the AUC Score than tuning the hyperparameters. Once the engineered features were added the models achieved the AUC scores shown in Figure 11.

Figure 11. Comparison of AUC Scores with Tuned Parameters

Model (tuned hyperparameters & engineered features)	AUC Score
Logistic Regression	0.760940
Random Forest	0.723440
LightGBM	0.781326

Coding Complications

I learned that Python does not necessarily release the memory back to the Operating System. Instead, it has a specialized object allocator for small objects (smaller or equal to 512 bytes), which keeps some chunks of already allocated memory for further use in future. I ran into a memory issue that took me a while to figure out. I kept getting a memory error when joining the various datasets together. I had to employ the *gc module* for memory management which seemed to fix the issue.

IV. Results

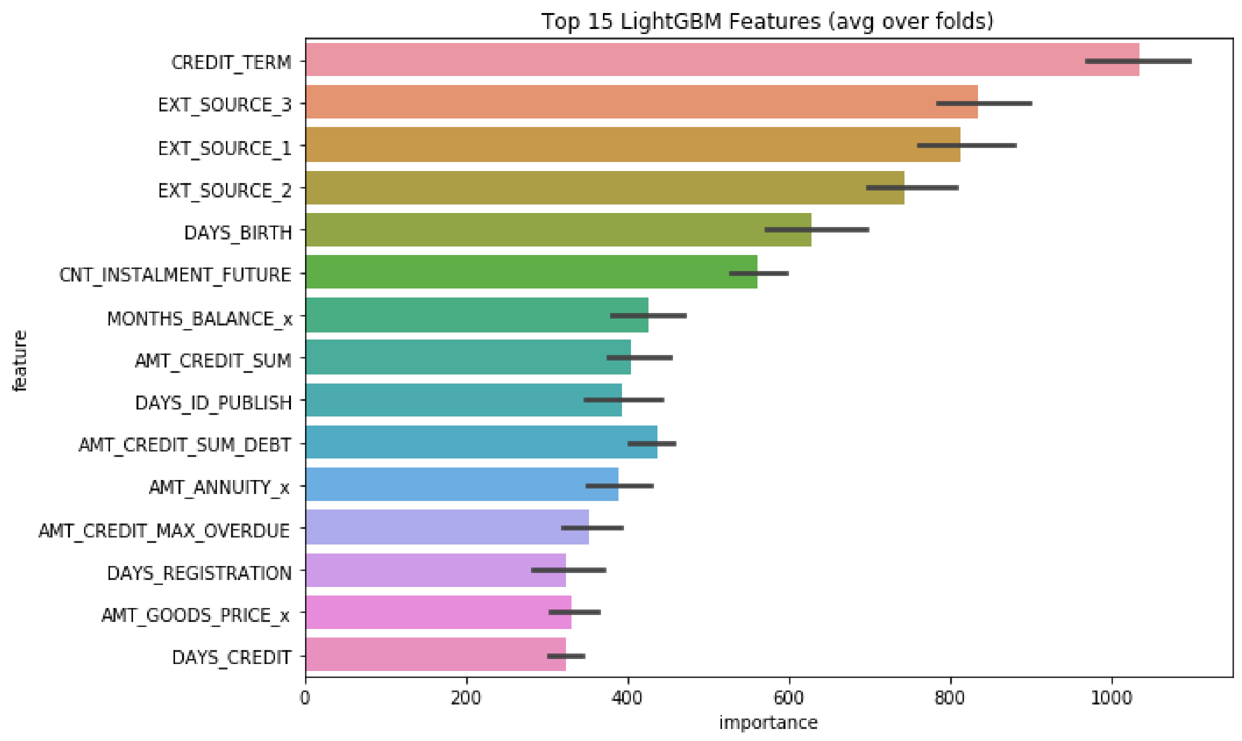
Model Evaluation and Validation

The LightGBM model outperformed the logistic regression and random forest models by 0.020386 AUC score units and 0.057886 AUC score units, respectively.

Feature Importance

As can be seen in the graph below, the five most important features are CREDIT_TERM (an engineered feature), those having to do with exterior sources(EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3), and the client’s age.

Figure 12. Top 15 Features Ranked by Importance



Justification

The logistic regression baseline established earlier (AUC Score: 0.760940), as well as the naive model of random guessing (AUC Score: 0.50), did not perform better than the LightGBM model (AUC Score: 0.781326). One of the reasons for the improved performance may be the k-fold cross validation implemented in the LightGBM model.

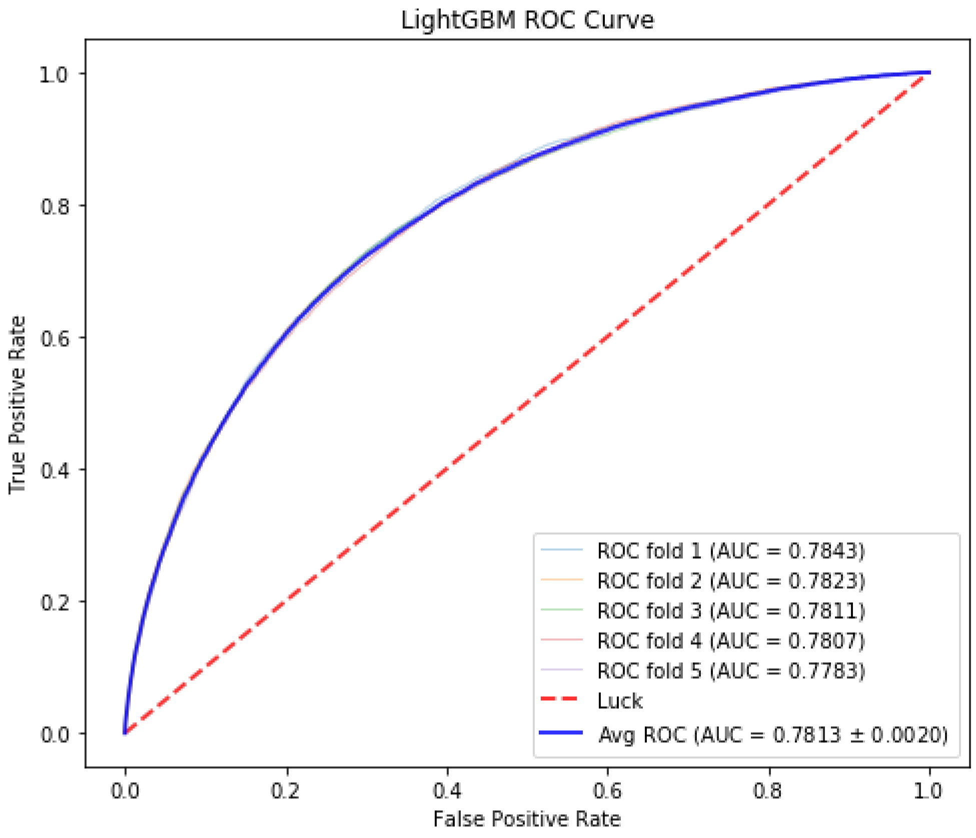
Top leaderboard scores in the competition are near 0.805 AUC, which is fairly close to the result obtained in this project.

V. Conclusion

Free-Form Visualization

Below is a graph of the ROC curves for each fold and the average of the folds for the final LightGBM model. The x-axis for the ROC curves represents the false positive rate and the y-axis represents the true positive rate. The plots show that the model's predictions are better than naive guessing (luck), which is the diagonal line running from (0, 0) to (1, 1). The different lines in the plot correspond to the different folds in the cross-validation process.

Figure 13. ROC Curve for LightGBM



Reflection

Using LightGBM for a binary classification problem was the right choice given that the LightGBM algorithm does well with missing and categorical values.

I completely underestimated how much time and effort this would require particularly in finding the appropriate features and learning how to code new formulas and analytical techniques and understand how they work.

Overall I found the process to be challenging and found the following interesting along each step of the process:

1. Exploratory Data Analysis - I am continually impressed by the knowledge and skill level of Kagglers out there that have shared EDA kernels. These have been incredibly useful not only in understanding the data but also in teaching me new coding and analysis skills.
2. Preprocessing - this component took me the most time trying to figure out how to do it properly and helped me further develop my data wrangling skills. I found using other people's examples helpful, however, sometimes I came across errors that took me much too long to debug and then frustratingly found out I was using a different version of numpy or pandas than the example I was following.
3. Feature engineering - feature engineering matters much more than I expected, and getting a better score is all a matter of engineering the right features. Domain knowledge helps, but I also saw other Kagglers come up with features that were beyond the scope of domain knowledge but seemed to give them a score boost. I found that quite intriguing.
4. Joining datasets - the final dataset had over 500 columns, which slowed down the performance of the model. I chose not to remove any of the columns since I din not know beforehand if the feature would be important even if it was missing a lot of data. In hindsight, I could have done a better job of dimensionality reduction.
5. Implementing different models, parameter tuning, and comparing results - I'm glad I used logistic regression as a baseline model for benchmarking. Had I just used a naive assumption of 0.5 AUC I would have been more impressed with the performance for the random forest classifier. However, I was quite surprised that the random forest model did not perform better than the logistic regression model.

Improvement

Further improvements include better feature engineering and selection. I would implement the use of automated feature selection using featuretools and implement better feature selection and dimensionality reduction techniques such as PCA.

I wasn't really sure how to implement neural networks with this problem, but I think it would be a possibility. Also, if I used my final solution as the new benchmark I could use stacking and blending diverse models of LightGBM, with different data representations & parameters to improve the performance of the original algorithm.

References

Addo, Peter M.; Guegan, Dominique; and Hassani, Bertrand. “Credit Risk Analysis Using Machine and Deep Learning Models”, *Risks* 6, no. 2: 38. April 16, 2018.

Bacham, Dinesh, Zhao, Janet. “Machine Learning: Challenges, Lessons, and Opportunities in Credit Risk Modeling.” Moody’s Analytics. July 2017.

Breiman, L. “Random Forests.” Machine Learning 45, 5–32 (2001)

Breiman, L.; Cutler, J. Random Forests https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#intro

Friedman, J. “Stochastic Gradient Boosting.” *Journal Computational Statistics & Data Analysis*, vol. 38 issue 4, February 28, 2002.

Home Credit Default Risk. Kaggle. May 17, 2018.

Jia, Hongri. “Bank Loan Default Prediction with Machine Learning.” Medium. April 10, 2018.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree.” Advances in Neural Information Processing Systems. pp. 3149–3157 (2017)

Tsai, Chia-Cheng; Lu, Mi-Cheng; and Wei, Chih-Chiang. “Decision Tree–Based Classifier Combined with Neural-Based Predictor for Water-Stage Forecasts in a River Basin During Typhoons: A Case Study in Taiwan”. *Environmental Engineering Science* 29.2 (2012): 108–116. PMC.