

Machine Learning Engineer Nanodegree

Capstone Project

D. Chris Young

June 8, 2018

I. Definition

Project Overview

Consumer finance companies provide loan options to individuals to make every day purchases such as home appliances, personal computers or electronics. Credit risk is the risk that borrowers fail to make the required payments and the company recognizes a loss on the loan.¹ Credit policies and guidelines can establish minimum lending standards and models can be used to quantify the risk of default.

People often struggle to get loans due to insufficient or non-existent credit history but often are the very consumers that need financial assistance. Home Credit Group is a company that focuses on responsible lending primarily to people with little or no credit history. The company has posted the [Home Credit Default Risk](#) Kaggle competition to see if the community can help predict if a customer will have payment difficulties using their proprietary dataset.

The Home Credit Default Risk dataset for the competition can be found [here](#). The dataset is unique because it includes actual application, balance, point of sale purchase and payment statistics data for Home Credit customers. The training data is labeled with a binary variable TARGET and is the dependent variable in the machine learning task. The dataset also includes credit bureau application and balance statistics for loans customers have with other lenders.

The dataset relates to the problem because demographic information, payment history, purchase patterns and credit balance features should provide insights into the characteristics of customers that have defaulted on their loans. These customers have higher credit risk and the types of customers Home Credit should not offer new loans.

Problem Statement

The problem that needs to be solved is reducing the likelihood of Home Credit rejecting consumers that are capable of repaying their loans. The inputs are general applicant statistics for current and prior loans with Home Credit; loan, credit card balance and payment history data for prior loans with Home Credit as well as credit bureau and balance data for loans from other lenders.

¹ [Credit Risk Wikipedia](#)

This is a supervised learning task based on the binary classification on the TARGET variable. The output is the predicted probability that the consumer will have payment difficulty. The problem will be solved with 3 key steps:

1. Combine all tables in the data model into a final dataset with one row per applicant and perform all data preprocessing for modeling.
2. Train and evaluate different supervised learning algorithms that provide a probability of each example being of the TARGET class.
3. Make predictions on the test set and prepare a submission file for the Kaggle competition evaluation.

The key results that I'm looking for are the predicted probabilities, so this is important when I'm evaluating specific algorithms to solve this problem.

Metrics

Evaluation is based on the **Area Under the receiver operating characteristic (ROC) Curve** between the predicted probability and the observed target. AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. The higher the AUC the stronger the classification model is performing on the target variable.

The two key metrics used to calculate AUC is the True Positive Rate (TPR) and False Positive Rate (FPR) from a confusion matrix:

- True Negative (TN) is when a negative class is predicted, and the actual class is negative.
- False Negative (FN) is when a negative class is predicted, and the actual class is positive.
- False Positive (FP) is when a positive class is predicted, and the actual class is negative.
- True Positive (TP) is when a positive class is predicted, and the actual class is positive.

TPR is defined as $\frac{TP}{TP+FN}$ and FPR is defined as $\frac{FP}{FP+TN}$. The ROC curve is created by plotting the TPR against the FPR at various threshold settings.²

AUC is an appropriate evaluation metric for this project because the TARGET variable is an imbalanced class with only 8.1% of the training set being true. An evaluation metric such as Accuracy would not be sufficient because a model could simply predict false on the training data and be 91.9% accurate. AUC is insensitive to imbalanced classes so a model that always predicts false would have high accuracy but an AUC of .5 which is the equivalent of random guessing.³

² [Receiver Operating Characteristic Wikipedia](#)

³ [What you wanted to know about AUC](#)



Figure 1: TARGET variable Frequency Distribution and Count Plot

II. Analysis

Data Exploration

There are 219 columns across 7 files in the dataset. Below is a snapshot of the data model and information about each table.

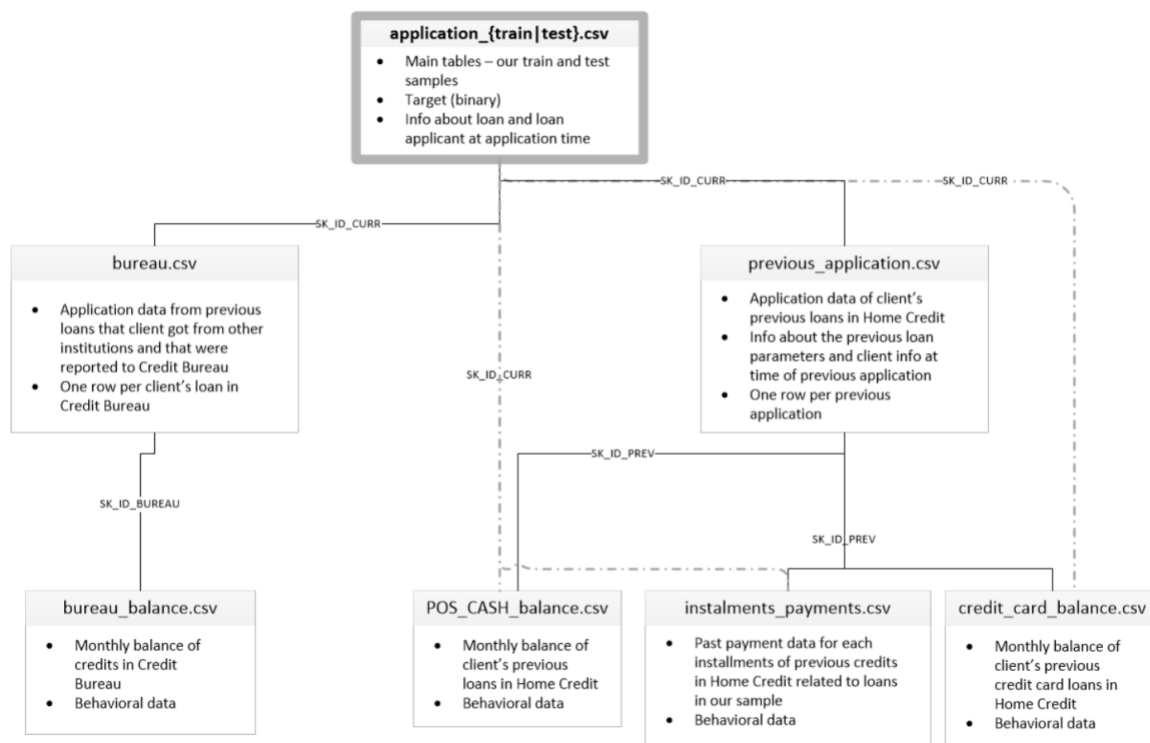


Figure 2: Home Credit Default Risk data model ⁴

⁴ [Home Credit Default Risk Data](#)

The dataset included a data dictionary (HomeCredit_columns_description) that includes the Table, Name, Description and Special comments about the column. Based on a review of the dictionary I observed multiple columns relating to the applicant's gender, family status/size, age, housing type and geographic housing area. Even if these had predictive power these could pose potential discriminatory issues in a credit risk decision model. For example, a particular region does show higher instances of payment issues, so loans won't be made to applicants that live in this region. That is an example of redlining.⁵ As a result all of these columns were dropped. In addition, I used some intuition to make some initial decisions to drop potentially irrelevant columns in an attempt to reduce the feature space and make it easier for exploratory analysis.

The next step was to review features with a large number of blank values. The table below represents a sample of columns in the application table with blank values greater than 1,000.

4	AMT_REQ_CREDIT_BUREAU_DAY	47568
5	AMT_REQ_CREDIT_BUREAU_HOUR	47568
6	AMT_REQ_CREDIT_BUREAU_MON	47568
7	AMT_REQ_CREDIT_BUREAU_QRT	47568
8	AMT_REQ_CREDIT_BUREAU_WEEK	47568
9	AMT_REQ_CREDIT_BUREAU_YEAR	47568
14	DEF_30_CNT_SOCIAL_CIRCLE	1050
15	DEF_60_CNT_SOCIAL_CIRCLE	1050
16	EXT_SOURCE_1	193910
18	EXT_SOURCE_3	69633
45	OBS_30_CNT_SOCIAL_CIRCLE	1050
46	OBS_60_CNT_SOCIAL_CIRCLE	1050
47	OCCUPATION_TYPE	111996
51	TARGET	48744

Figure 3: Column and Nan Counts

The AMT_REQ_ columns represents the number of credit inquiries by time period. The EXT_SOURCE columns represents normalized score data from an external data source. This seems like it could represent credit score type data. I observed that there were only blank values for some of these columns for each applicant. I'm assuming the score was only available from some of the external data sources. Credit scores are typically very important in a credit risk model so replacing the missing values with an accurate statistic should be pretty important.

The _SOCIAL_CIRCLE columns represents data for the applicant's social surroundings with observable days past due. This type of social data may not be available for all applicants but could be pretty useful in a model. OCCUPATION_TYPE is a categorical column so dummy

⁵ [Redlining Wikipedia](#)

columns will be created for the categories that have values. The TARGET column will not have values for the Test data that was combined with the Training data for data analysis and preprocessing purposes.

Descriptive statistics were reviewed for all categorical variables to determine which features will need to be transformed for modeling. In addition to dummy encoding there are FLAG fields with Y or N values so these will need to be converted to binary columns.

As shown in the Figure 2 the dataset includes several additional tables that will need to be aggregated by the SK_ID_CURR column and merged with the application data.

The Bureau table was used to get count of bureau records by status and type and the number of bureau records created within 30, 60, 120 and 121+ days from the application. I assumed if the applicant had a lot of new bureau records very close to the application that this would be a sign of credit shopping and potentially high risk. In addition, sums were aggregated for only active bureau records since an applicant's current credit accounts would seem to be more important.

The Bureau Balance table was used to get count by status for the latest balance record and sums of all historical records by status. The status code options are:

1. C means closed
2. X means status unknown
3. 0 means no DPD
4. 1 means DPD 1-30
5. 2 means DPD 31-60
6. 5 means DPD 120+ or sold or written off

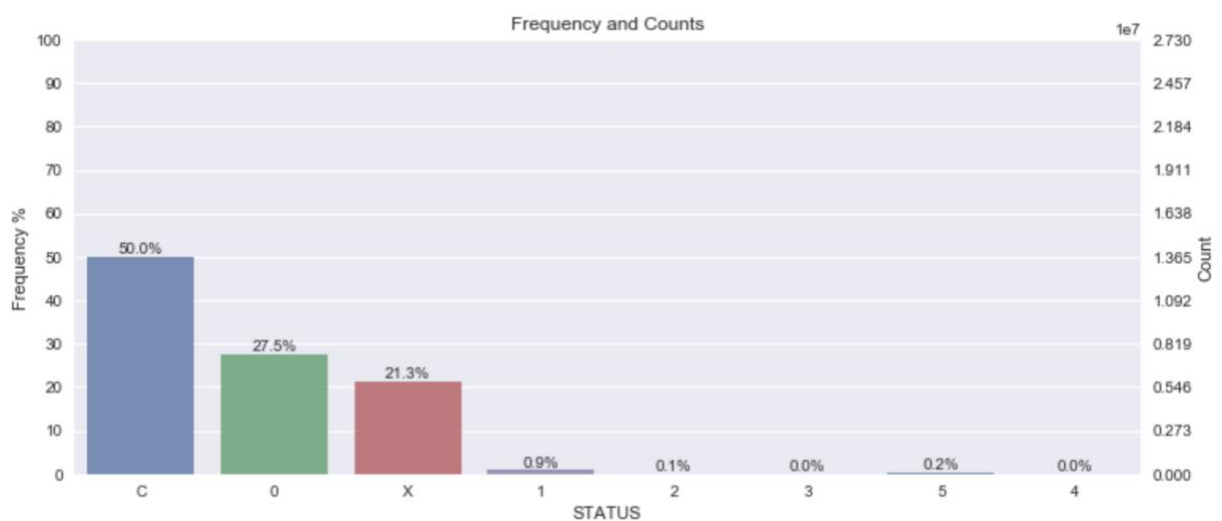


Figure 4: Bureau Balance STATUS frequency and counts

DPD (days past due) records should be important as it would show applicants that had payment difficulties in the past.

The POS CASH Balance table was used to get the sums of contract status types, POS installments left to pay and DPD. The Credit Card Balance table was used to get the sums of contract status types, DPD and active credit card statistics. The Installment Payments table was used to get the difference between installment payment amounts and days and calculate historical averages. I feel this will show how much and how timely applicants made their payments in the past. The Previous Application table was used to get sums and averages of prior application statistics. All 4 tables were then merged on SK_ID_CURR.

The final step was to merge application with previous application data and bureau data for the final combined dataset.

Exploratory Visualization

Histograms were plotted for 16 numeric features to review for data skewness.

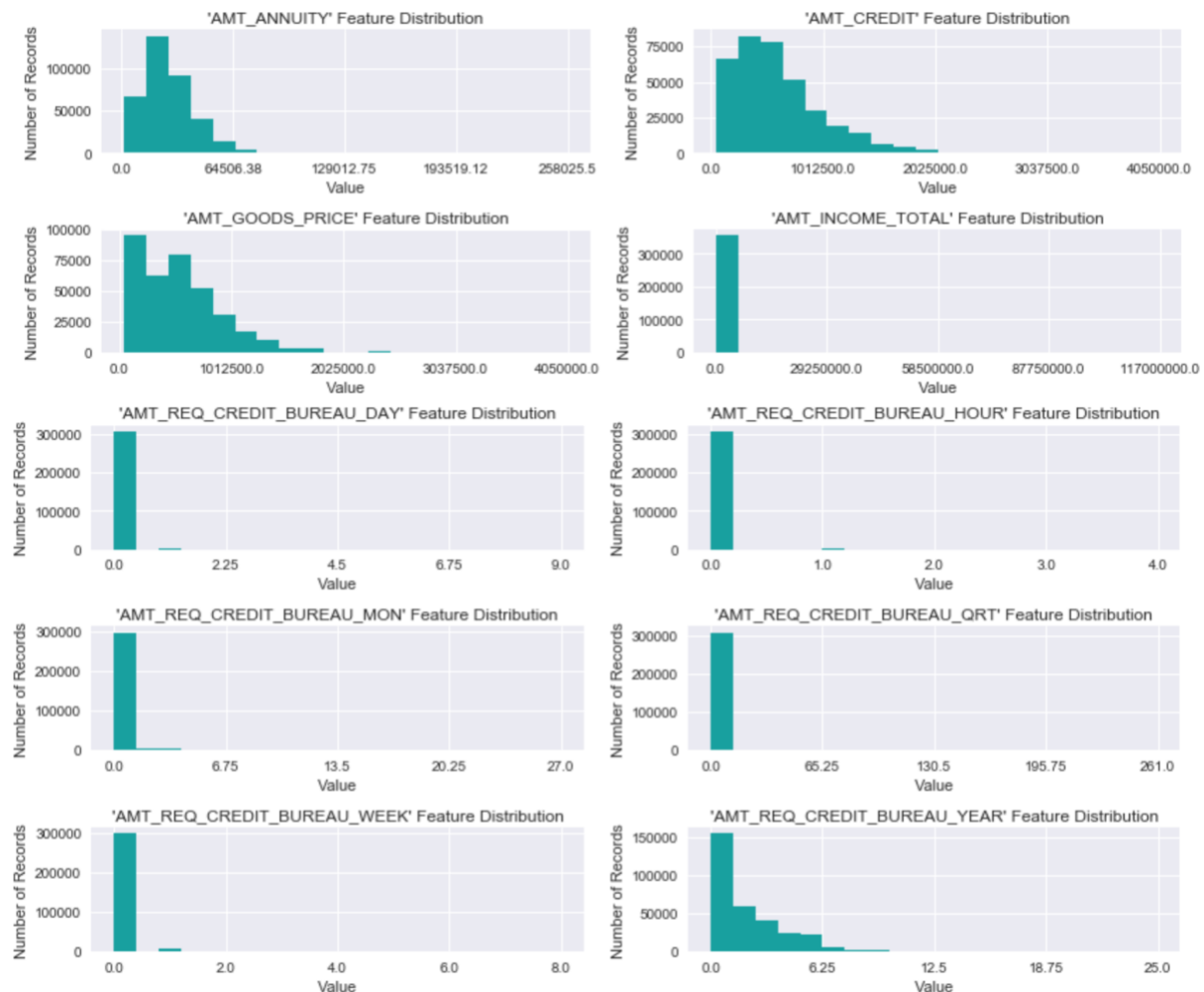




Figure 5: Numerical feature histograms

The AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE and AMT_INCOME_TOTAL features all display skewed right distributions.

The AMT_REQ_ features also display severe skewed right distributions with the majority of the values being close to zero except for the _YEAR feature. This makes sense the longer the time period the more applicants would have credit bureau inquiries.

The DAYS_EMPLOYED feature has an outlier with a positive value of 365243. The DAYS_ features are normally negative as they represent the time relationship in the past from the application month.

Algorithms and Techniques

The solution for this problem is a supervised learning task. The classifier must be probabilistic that outputs a probability distribution for the predicted class. Some suitable supervised learning models for this problem are:

- Logistic Regression
- Decision Trees
- Random Forest
- Gradient Boosting

The predict_proba method returns the probability estimates for the prediction task. This output can be used to measure model performance and to create the competition submission file.

The following chart compares performance for model training/predicting time and AUC scores on training/testing datasets for Logistic Regression, Random Forest Classifier and XGB (Extreme Gradient Boosting) Classifier models.



Figure 6: Comparison of Default Models

Random Forest Classifier severely overfits the training data but is quick to train and make predictions. Logistic Regression and XGB Classifier perform similarly on the training and testing sets but XGB requires more time to train and make predictions. The XGB algorithm has many parameters that can be tuned so it should have more upside from the default results shown above compared to Logistic Regression. I made the decision to tune the XGB Classifier for final modeling.

Benchmark

A random predictor would have an AUC score of .5. This is represented by the diagonal dashed line in the plot below and provides the benchmark if a model is useful. A model that produces an AUC score of 1 means the predictions are perfect.⁶

The orange solid line in the plot below is the AUC score of 61.8% for a Decision Tree Classifier with parameters `max_depth = 10` and `max_features = sqrt`. I used a Decision Tree Classifier

⁶ [What does AUC stand for and what is it](#)

because it could be run fairly quickly with minimal data preprocessing. Due to the nature of the dataset I wanted to create a quick benchmark. No feature engineering was performed, and only basic application features were included. This establishes the benchmark for a model that is slightly better than random guessing.

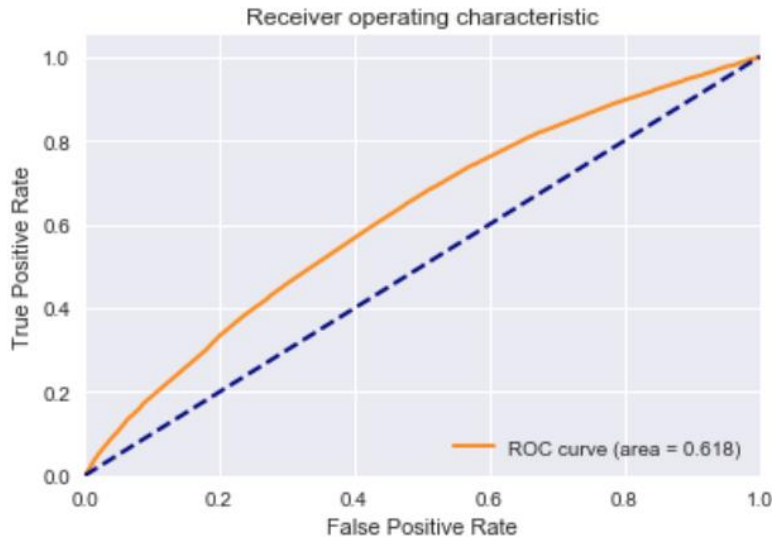


Figure 7: Receiver operating characteristic (ROC) curve for benchmark model

III. Methodology

Data Preprocessing

The dataset had missing values for 16 raw columns from the dataset and all missing values were initially replaced with the median.

The DAYS_ features have negative values so these were updated with the absolute value. The outlier for DAYS_EMPLOYED $\geq +365,243$ was set to 0 since a positive value is not consistent with the data format.

All of the right skewed features discussed in the exploratory visualization section were log transformed. All features were also scaled using a min max scaler. Tree based classifiers are insensitive to non-normalized features, but Logistic Regression was included in the model comparison so normalizing the features allowed for consistency in the dataset for all models.

Implementation

A significant portion of the project was spent on data analysis, feature engineering and data aggregation to prepare a tidy dataset with one row per applicant. The benchmark model was run with only basic application data and the AUC score was pretty low at .618. I knew from the start engineering features from the Bureau, Bureau Balance, Previous Application, POS Cash Balance, Installment Payments and Credit Card Balance tables was going to be very important.

This required extensive analysis on the structure of each table and how it related back to the application table. Fortunately, a pretty extensive data dictionary was provided so this helped with the education process. I spent more than 50% of the project time on feature engineering.

I used my domain knowledge in banking and financial services to make a lot of decisions to create various aggregates statistics from the auxiliary tables. For example, the Credit Card Balance table had data for all account statuses including completed so I isolated the active accounts to perform aggregates. Although all prior accounts would be useful for statistics such as delinquency only active accounts should be useful for balance data. The Installments table had data about when and how much payments were made so it seemed logical that the delta between the time and amounts could provide insight into historical payment consistency. I used intuition when deciding if the aggregate should be a sum vs. mean. The data dictionary also had clues on the previous application table that some records may be duplicates so these were excluded.

As discussed in the data exploration section many features were removed without even exploring them further. In a business setting it would be difficult to go through model governance and justify making credit decisions on a protected class not to mention the ethical and legal considerations. Although this may impact model performance it was a reasonable tradeoff.

Custom functions were coded to help with the implementation process and to separate these from the notebook. These can be found in helpers.py and visuals.py. These include the following:

- helpers.py
 - convert_flag – Convert FLAG_ fields to a binary column
 - drop_cols – Drop columns if exists
 - create_dummy – Get dummy columns and drop original categorical columns
 - rename_cols_df – Clean column names for consistency and add suffix if specified
 - perform_aggregate – Custom solution to perform pandas groupby
 - merge_df – Custom solution to perform multiple pandas merges in one step
 - predict_missing_ext – Custom solution to predict missing values for EXT_SOURCE_ columns
 - train_predict – Run simulations on different models
 - cv_result_analysis – Custom solution to run XGB cross-validation analysis
 - create_submission – Create dataframe for Kaggle submission
 - create_output – Create Kaggle submission csv file
- visuals.py
 - plot_histograms – Plot multiple histograms
 - plot_counts – Plot counts and distribution counts
 - plot_roc_auc_curve – Plot ROC Curve and naïve guessing line
 - evaluate – Plot charts from train_predict results

The out of the box model for XGB had an AUC score of .764 on the held-out testing set and a Kaggle public leaderboard AUC score of .766. This was a pretty good result and well above the benchmark model.

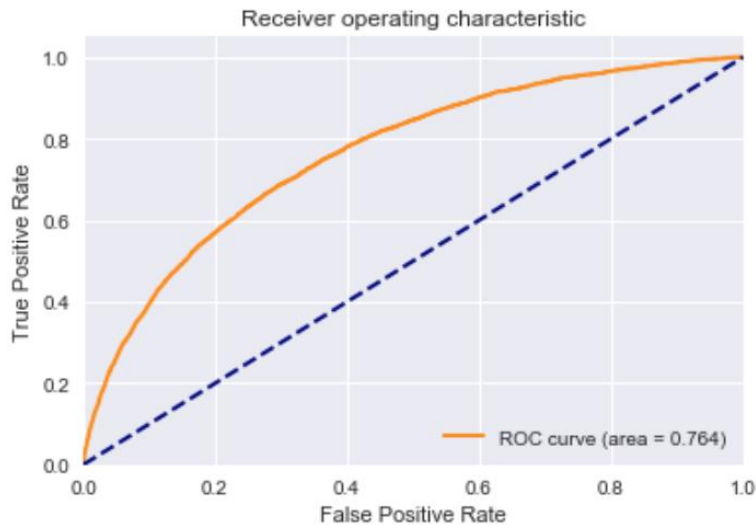


Figure 8: Receiver operating characteristic (ROC) curve for out of the box XGB model

Refinement

I reviewed the feature importance scores for the first XGB model and it was observed the EXT_SOURCE_ features were the top 3. This validated some of my initial assumptions about these features since the data dictionary was pretty vague. However, this raised some concerns since the number of missing values for EXT_SOURCE_1 and EXT_SOURCE_3 was pretty large. Did the method used to replace the missing values with the median misrepresent the data?

My first thought was instead of replacing the missing values to use the MIN, MAX, MEAN and MEDIAN for the available values across all 3 and then drop the original features. However, this actually had an adverse effect on performance dropping performance down to .759.

My next thought was to replace the missing values with a prediction. This required re-work in the data engineering phase of the project and I ended up having to move this down to the data preprocessing area. The missing values for these were predicted using a Random Forest Regressor. The MIN, MAX, MEAN and MEDIAN values were still calculated but after the missing values were replaced. The figure below shows the results for some sample data.

	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	EXT_SOURCE_MIN	EXT_SOURCE_MAX	EXT_SOURCE_AVG	EXT_SOURCE_MID
0	0.083037	0.262949	0.139376	0.083037	0.262949	0.161787	0.139376
1	0.311267	0.622246	0.542313	0.311267	0.622246	0.491942	0.542313
2	0.415475	0.555912	0.729567	0.415475	0.729567	0.566985	0.555912
3	0.449913	0.650442	0.657622	0.449913	0.657622	0.585992	0.650442
4	0.463555	0.322738	0.650230	0.322738	0.650230	0.478841	0.463555

Figure 9: EXT_SOURCE_ example data after missing value predictions and calculated columns

The number of parameters available in the XGB model is pretty extensive. The XGB [documentation](#) for parameter tuning has a lot of good information on understanding the bias-variance tradeoff and controlling overfitting. As shown in figure 6 the out of box model actually does pretty good not overfitting the training set. However, once you start to build a more complex model this can be an issue. The XGB library has a really useful cross-validation class (cv) and this was used to narrow down parameter values for 3 of the key parameters max_depth, min_child_weight and gamma. These 3 parameters were selected initially because they are outlined in the documentation as key for controlling overfitting.

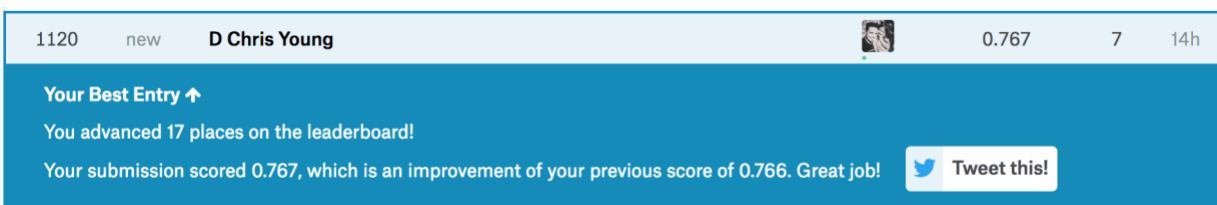
A grid search was run to identify the optimal features for these 3 parameters and I manually tuned the remaining tree and regularization parameters. The grid search required a lot of time due to training, so it was necessary to find a good balance between automatic and manual tuning.

The XGB train class has a really great feature of setting a high number of boosting rounds but add in an evaluator watchlist on the held-out testing set. This really maximizes the power of the boosting process without having to manually set the number of rounds. The final model for XGB had an AUC score of .778 on the held-out testing set and the Kaggle public leaderboard score was .767.

```
[0]      test-auc:0.721429
Will train until test-auc hasn't improved in 50 rounds.
[50]      test-auc:0.76107
[100]     test-auc:0.771371
[150]     test-auc:0.773917
[200]     test-auc:0.775062
[250]     test-auc:0.77583
[300]     test-auc:0.77634
[350]     test-auc:0.777036
[400]     test-auc:0.777426
[450]     test-auc:0.777623
Stopping. Best iteration:
[437]     test-auc:0.777735
```

Figure 10: Final model train and test evaluation results

This was a pretty good improvement on the held-out testing set but only a marginal improvement on the public leaderboard. It is noted the public leaderboard is based on 20% of the test set or 9,749 samples. The held-out testing set is 61,503 samples. The cross-validation results will be trusted and the full Kaggle leaderboard will be available when the competition closes on August 29, 2018.



1120 new **D Chris Young** 0.767 7 14h

Your Best Entry ↑

You advanced 17 places on the leaderboard!

Your submission scored 0.767, which is an improvement of your previous score of 0.766. Great job!

[Tweet this!](#)

Figure 11: Final model Kaggle public leaderboard score

IV. Results

Model Evaluation and Validation

Multiple models were trained and tested on the final dataset. The training data was split 80% for training and 20% for cross-validation testing. The initial improved XGB Classifier had good results but was overfitting the training data with an AUC training score of .85. The regularization parameters were manually tuned to reduce variance. The subsample ratio of the training instance was reduced from 1 to .75. This added randomness to make training robust to noise. The reg_alpha L1 regularization term on weights was increased from 0 to 5. The final improved model had an AUC training score of .815 and testing score of .778. Based on these results I'm confident the model is strong and will perform well on unseen data.

Classifier	Decision Tree Classifier (Benchmark)	Logistic Regression	Random Forest Classifier	XGB Classifier	XGB Classifier Improved
CV AUC Score	.618	.763	.666	.764	.778
Public Leaderboard				.766	.767

Figure 12: Model comparison

Justification

The final results are much better than the benchmark model. The improvement from .618 to .778 shows how important it was to spend a lot of time engineering features for the final combined dataset. At this point it is good to revisit the problem statement:

The problem that needs to be solved is reducing the likelihood of Home Credit rejecting consumers that are capable of repaying their loans.

AUC is the probability of ranking a random positive example over a random negative example. Home Credit wants to reduce the chances of rejecting an applicant for a new loan when they were in fact a good applicant that would make all of their payments. Let's assume we randomly selected an applicant that had payment difficulties and another applicant that did not. My model is saying that 78% of the time it would correctly classify the random applicants in its respective class.

It really depends on the context if this performance is considered good. In the medical field when attempting to predict if a patient has a rare disease this result would be quite poor. However, this problem is related to consumer behavior which is challenging to predict. I would say based on this context this result is pretty good but there is room for improvement.

V. Conclusion

Free-Form Visualization

Data analysis and feature engineering was very time-consuming part of this project. I was anxious to plot the top 20 features by importance to see if all that work actually was beneficial in the final model.

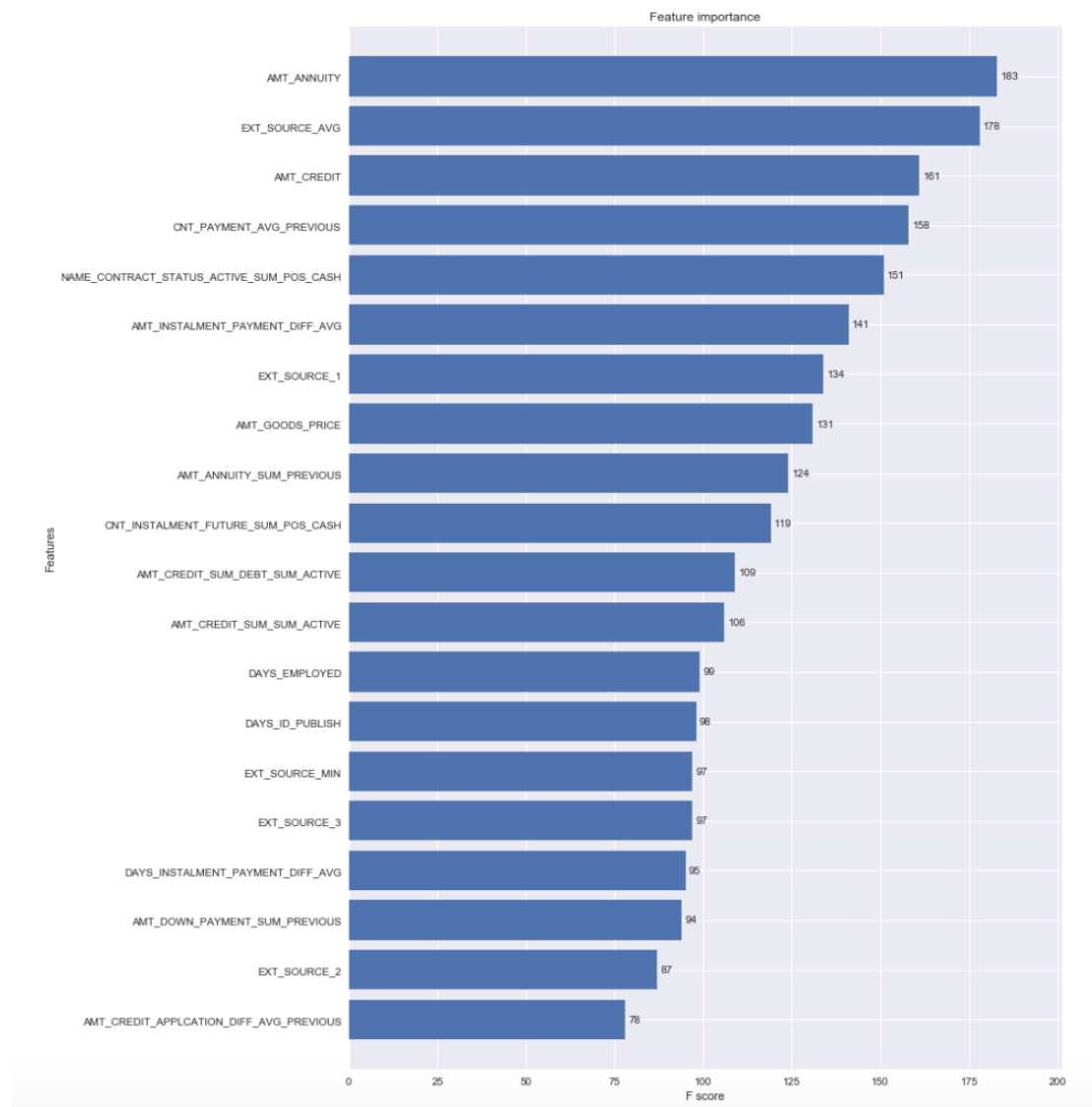


Figure 13: Final model feature imporances

As shown above 12 out of the top 20 features were engineered from the auxiliary tables. The features are easily identifiable because of the suffixes I added when the features were created. Therefore, it is reasonable to conclude the feature engineering work had a positive impact on the success of the model.

Reflection

Below are the 7 detailed steps I followed for this project:

1. Extract Data
2. Data Exploration
3. Exploratory Visualization
4. Data Preprocessing
5. Train and Evaluate Multiple Models
6. Implementation and Tune Final Model
7. Evaluation and Make Predictions

In this reflection section I will focus on 2. Data Exploration and 6. Implementation and Tune Final Model.

This project was truly an end-to-end machine learning project. Most projects require exploratory data analysis, data cleaning and transformations but before I could even get to those steps I had to transform the data model into a flat dataset with one row per applicant. This was actually the most difficult but fun aspect of the project. This type of work very closely relates to my current work environment, but I typically do most of my analysis and transformations in an SQL relational database environment. I spent a lot of time researching various methods in pandas to transform data. I even wrote my own custom solutions to perform different aggregates and dataframe merges built on top of the pandas methods to make my solution more elegant.

Implementation of the model was actually quite easy once I got a tidy dataset. It was a good experience because in the real-world data will almost never be perfect. My experience with the XGB algorithm was pretty limited prior to this project. It was really interesting to utilize the various classes within the library to perform cross-validation to control overfitting. The parameters to control the number of boosting rounds and to simultaneously evaluate your model as it is training on a hold-out testing set was quite useful. However, the size of the dataset required a lot of training time, so I spent a lot of time tuning parameters both automatically with a grid search and manually trying different options and validating the result on the training and testing sets.

The final model meets my expectations for the amount of work that I spent on the project. I feel the solution is appropriate for the problem and is a viable solution.

Improvement

One improvement would be to engineer more complicated features. Even though I engineered a lot of new features the majority of these were simple aggregate statistics or column derivations. For example, 2 of the features in the top 20 were:

AMT_INSTALLMENT_PAYMENT_DIFF_AVG
DAYS_INSTALLMENT_PAYMENT_DIFF_AVG

These were new features that measured the time difference between the amount and days when an applicant's installment payments were made. I had good intuition that these features would have good predictive power and it turns out I was right. However, I could have spent more time theorizing on more features such as these. Due to time constraints I had to move along with the project, but this is an improvement I can continue to work on before the competition closes.

The other improvement is related to training time for XGB. I had to really take advantage of the boosting rounds in order to achieve the desired result. The final model was trained for 437 rounds and this was very time-consuming process and also made it difficult to tune the model parameters. I noticed on the competition kernels page a lot of people posted solutions using the Light GBM algorithm. Based on some research Light GBM has some advantages over XGB specifically faster training speed.

I'm going to use my final solution for this project as the new benchmark and see if I can engineer more complex features and train a Light GBM model to come up with an even better solution.

References

1. [Pandas](#)
2. [XGBoost Read the Docs](#)
3. [Complete guide parameter tuning xgboost with codes python](#)
4. [Why is AUC Area under ROC insensitive to class distribution changes](#)
5. [Is auc the probability of correctly classifying a randomly selected instance](#)
6. [Which algorithm takes the crown light gbm vs xgboost](#)