Jeromie Clark

Capstone Report

Project Overview

Kaggle, a data science competition site, is currently hosting a competition on Mortgage Default Risk[1] targeting profiles of individuals who may little to no credit history or were victims of "untrustworthy lenders". Instead of relying exclusively on traditional markers like credit score and payment history, the company uses "alternative data, like telco and transactional information" to determine their customers' creditworthiness.

The contest sponsor has provided a comprehensive, anonymized dataset that includes any previous credit bureau history and open loan balances, details on any outstanding commercial credit and cash loans, credit card balances, previous applications for loans from the same individual, and a history of all payments made on the loans issued. The application dataset is comprised of several hundred features, and hundreds of thousands of records, comprised of a mix of continuous, binary and categorical data.

It's also worth noting that the contest provides both a labeled training set and unlabeled test set. We'll use the training set exclusively for model selection and evaluation using k-fold cross evaluations for benchmarking, and submit predictions to Kaggle based on the provided test set to retrieve actual contest scores.

Problem Statement

Using the dataset provided, which includes both metadata about each borrower as well as the payment history of each loan, the goal is to successfully create a model that provides an accurate prediction of whether or not a borrower will repay a given loan.

Metrics

Our contest sponsor cares primarily about avoiding borrowers that are likely to have performance problems in the future. To that end, we are using accuracy as the primary performance metric for this project. Accuracy is defined as the ratio between the number of correct predictions to the total number of predictions.

As a secondary concern, erroneously identifying borrowers as potentially problematic is detrimental to our lender's business, as it unnecessarily reduces their pool of potential customers. The fewer loans our lender makes, the less opportunity they have to profit from issuing a loan.

[1] https://www.kaggle.com/kailex/tidy-xgb-0-777/data

Initially, I used Accuracy and F-Beta Scores as metrics; however, because this is an imbalanced problem (only 8% of borrowers actually default), accuracy in and of itself is fairly useless. F-Beta scores give us a good sense of performance against both accuracy and recall; however, the Kaggle competition uses another metric, Receiver Operating Characteristic, Area under Curve (ROC AUC) for scoring. For consistency, I ultimately adopted ROC AUC as the performance metric for this project.

The Receiver Operating Characteric Curve is a graphical illustration that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied[2]. More simply put, it's the plot of the True Positive rate vs the False Positive Rate at various thresholds. The ROC AUC score by extension is expressed as the percentage of the total area under the ROC curve.

Benchmarks

Because only approximate 8% of borrowers have payment problems, the dataset is skewed in terms of the distribution of problematic and successful borrowers. It is possible to classify approximately 92% of our loans as not problematic, achieving a high (92%) accuracy rate, while in actuality, failing to detect any problematic loans. This naïve predictor was built as a benchmark for comparing the performance of various machine learning algorithms.

Because accuracy alone can be a misleading metric, we must also consider algorithmic performance in terms of precision and recall. These metrics are typically combined to create what is known as the F1 score, defined as the harmonic mean of precision and recall scores. More specifically, we use the F-beta score with a beta value of 0.5 to emphasize precision. The result ranges from 0 to 1, with one being the best possible score.

Our naïve predictor achieves an accuracy score of 0.9190 (~92%), and an F-Beta score of 0.9341. When looking at the ROC AUC score, the evaluation returns 0.5, which equates to the performance of random chance.

Our goal for success is to exceed the naïve predictor benchmark with an ROC AUC score > 0.5. Representing an improved ability to correctly identify borrowers that might default on their loans.

[2] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Data Exploration

The dataset provided for this contest is broken into multiple files. The primary source of information is application_train.csv, which contains both the bulk of the information about the applicant and metadata about the loan application itself.

This project begins with a significant amount feature exploration and data preprocessing, model comparison and tuning, and iteration over various hyperparameters and preprocessing steps. I ultimately augment the dataset with additional features from the bureau and bureau balance datasets, examining performance along the way to understand the impact of those new features. A detailed exploration can be found in the supplemental Jupyter notebooks.

Much of the data outside the application_train.csv file requires some feature engineering, as there are one-to-many relationships between the applicant and data points like individual credit card accounts, open balances, previous defaults, etc.

The application_train dataset itself is comprised of 122 individual features. Each application has a unique ID, which we can use as a key to join in data from the other data files. For basic analysis and sanity, I bucketed each feature into one of four categories and wrote a utility function to pull basic statistics about the data sufficient to validate basic sanity and identify any necessary preprocessing steps.

The initial dataset had 61 non-numeric (categorical) features, defined as features comprised of two or more possible strings, which reflect a particular categorization. As an example, the CODE_GENDER feature (Figure 1) was comprised of 'F', M', and XNA' values. These representations were analyzed for sanity, and then tracked for One-Hot Encoding during the preprocessing phase.

```
Value: Civil marriage Count: 29775 Percentage: 9.68258046053637
Value: Married Count: 196432 Percentage: 63.87804013514964
Value: Separated Count: 19770 Percentage: 6.429038310824653
Value: Single / not married Count: 45444 Percentage: 14.778007941179
Value: Unknown Count: 2 Percentage: 0.0006503832383231819
Value: Widow Count: 16088 Percentage: 5.231682769071676
```

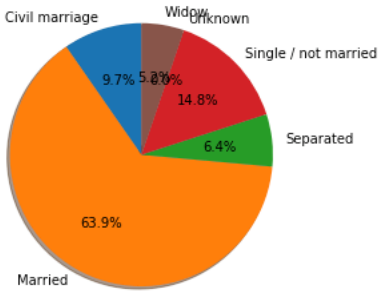


Figure 1 - NAME_FAMILY_STATUS Feature

```
Value: F Count: 202448 Percentage: 65.83439291602576
Value: M Count: 105059 Percentage: 34.164306317497584
Value: XNA Count: 4 Percentage: 0.0013007664766463637
```

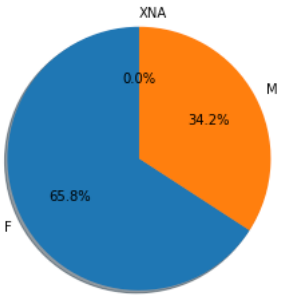


Figure 2 - COUNT_GENDER Feature

In addition, there was a subset of 6 categorical features which were actually binary in nature (Figure 3), typically characterized by "Yes/No" or "Y/N" fields. We classified those features as "String to Bool", and ultimately converted those fields to binary (1|0) representations. The meaning is identical, but the binary representation is more effectively processed by standard machine learning algorithms.

```
Value: N Count: 202924 Percentage: 65.98918412 Value: 0 Count: 202924 Percentage: 65.98918412
Value: Y Count: 104587 Percentage: 34.01081587 Value: 1 Count: 104587 Percentage: 34.01081587
```

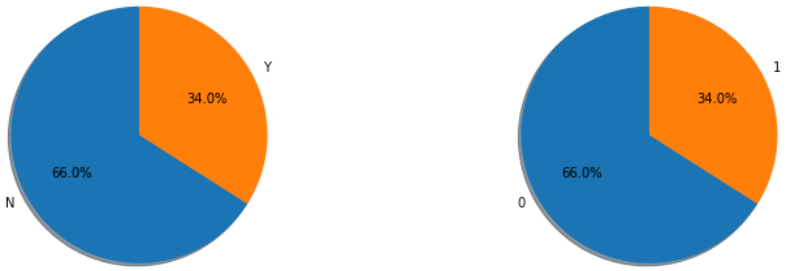


Figure 3 - Conversion of Y/N to Boolean Values

Machine learning algorithms tend to perform best under conditions when continuous fields represent a Gaussian distribution.

This graph shows the normal distribution of applicant's ages, represented as a negative integer, relative to the application date. The data is distributed along a reasonable curve and is well-centered.

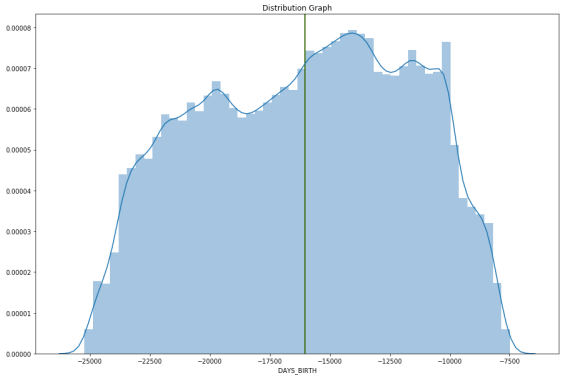


Figure 4 - Centered, reasonably normal distribution

In some features; however, we see heavily skewed datasets. This is relatively common, particularly when working with financial data, where you tend to see high counts near zero, and low counts of large values at the opposite end of the spectrum.

In the example below, we examine the applicant's total income. The range of applicant incomes is huge, with a minimum value of 25650, and a maximum value of 117,000,000. The average and mean values are 168797, with a standard deviation of 237122. As a result, you see that the graph is heavily skewed, with a small bump towards the left end of the range (the vertical bar is

the median), and a long tail of continuous values extending out to the right of the range (Figure 4).

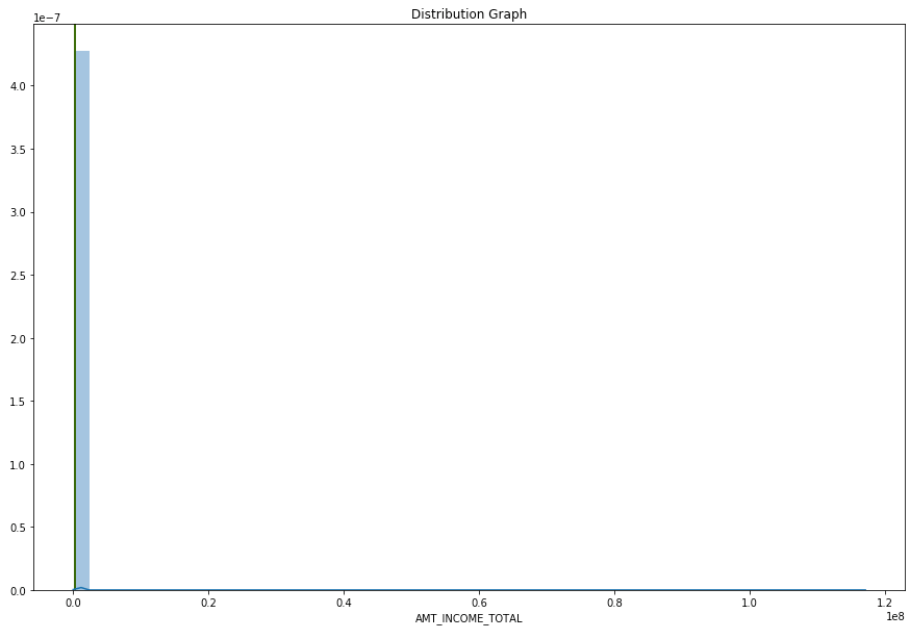


Figure 5 - AMT_INCOME_TOTAL Unprocessed

Financial data like this tends to be lognormally distributed, and by taking the log of the values, we can typically restore symmetry to our data. After processing, we now see a symmetrical curve, nicely centered around the median. In addition to the 61 continuous numeric fields, we identify another 11 fields for log transformation during the pre-processing phase.

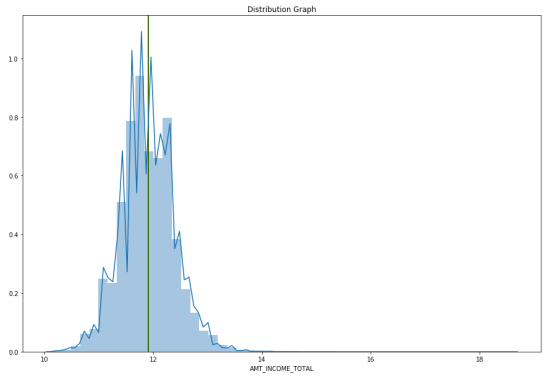


Figure 6 - AMT_INCOME_TOTAL - Log Transformation

Anomalous Data

This data exploration exercise revealed a handful of features that contained datapoints that were problematic.

DAYS_EMPLOYED is represented as a negative integer, relative to the date the local application was submitted. The field included values that indicated that some people had worked for 365243 days, or approximately 1000 years.
This seemed like they were probably a result of a bug, and in order to avoid penalizing the applicants for a software issue, the erroneous value was replaced with the Inter-Quartile mean, in an effort to avoid shifting the center of the values in the reasonable range.

A number of normalized housing data fields like “HOUSETYPE_MODE” contained categorical String values, with the exception of NaN when a value was not specified. This doesn’t work when one-hot encoding categorical values, so the string value “not specified” was substituted.

Preprocessing

Converted Date Values to Positive Integers

For some reason, the Home Credit team represents dates as negative integers, relative to the date that a loan application was submitted. A 20 year-old applicant would be representated as -7300 in the DAYS_BIRTH column. While we know that older applicants tend to be better about repaying loans than younger applicants, the negative values presented DAYS_BIRTH as a strong indicator of risk for default, instead of considering increasing age as a strong negative indicator. When comparing feature correlations, having the borrower’s income as a contraindicator for default and age as an indicator seemed counter-intuitive, and I applied an absolute value transformation to the date features to deal with this.

Converted Two-Category String Features to Boolean

A number of logically Boolean features (e.g. “Does the borrower own a car?”, “Does the borrower own a house?”) were represented as “Y” or “N” instead of binary values. To aid in efficient processing, I employed the sklearn LabelBinarizer feature to convert these to integer 1 and 0 values, respectively.

One-Hot Encoding for Categorical Features

In instances where a feature has more than two categories (i.e. Gender: M (Male), F (Female), XNA (Decline to State / Not Applicable), you can choose to represent those values through label encoding (M=1, F=2, XNA=3); however, lacking context, the machine learning algorithm may interpret the integer value of 3 as being more important than an integer value of 1.

An alternative approach would be to create new features for each category, and provide a binary indication of which category is chosen as a 1 in the corresponding category.
This technique is known as one-hot encoding. While this technique increases the number of features in the dataset, it does not arbitrarily signal information that might potentially be interpreted as weighting to the machine learning algorithm. This is the approach that used for features with more than one category.

Categorical Encoding Examples

Original Encoding

ID	Gender
1	M
2	F
3	XNA

Label Encoding

ID	Gender
1	1
2	2
3	3

One-Hot Encoding

ID	Gender_M	Gender_F	Gender_XNA
1	1	0	0
2	0	1	0
3	0	0	1

Removed non-numeric values from numeric features

In order to scale a group of continuous numeric fields to a comparable range, those fields must exclusively contain numerical data.

In a number of instances, fields where numeric data was not supplied were populated with NaN (Not A Number) values. To facilitate further processing, those values were converted to 0 using the NumPy library's nan_to_num() function, which converts NaN to 0, -inf to Python's MININT, and inf to Python's MAXINT values.

Outlier Removal

In looking at our dataset, it was clear that some numeric fields included exceptional values that skewed the mean for the majority of records. This is very apparent in AMT_INCOME_TOTAL and AMT_CREDIT, where some individuals earn and borrow sums that are orders of magnitude larger than the average borrower. It's unclear if these are data entry records or legitimate transactions performed by very wealthy individuals, but they're not useful for making predictions about the typical borrower.

Taking a cue from earlier projects, we perform outlier removal by taking the Inter-Quartile mean of our data (i.e. the mean of data between the 25th and 75th percentiles of the range), so as to remove any skew from values at the extreme upper or lower bounds of the range. We then substitute those mean values for values that exceed values above +/- 1.5 * the inter-quartile mean.

The result is that we remove the outliers without moving the mean of the data for the majority of records.

Feature Scaling

When machine learning algorithms evaluate numeric features, they may apply weight to larger numerical values. In order to ensure that the algorithm used applies equal weight to values of independent features, we put them all in a relative scale from 0 to 1. This scaling approach maintains the shape of the value distributions in each feature, while solving the problem of artificially weighting various features.

Because some fields may have contained outliers, the scikit learn RobustScaler class was initially considered to mitigate the effect of any outliers in the set; however, when evaluating the performance of our candidate algorithms between data preprocessed with RobustScaler and StandardScaler, and MinMaxScaler, StandardScaler actually performed slightly better in practice.

As a further sanity check, we also compared performance using data with and without log transformations. As expected, we found that the log-transformed data performed marginally better.

Modeling

Using the scikit-learn flowchart[3] for choosing an algorithm as a guide, I ultimately selected three algorithms to compare based on the characteristics of our data.

The dataset is based on historical loan data, and is already labeled by whether or not the application payed on time. Because we're working with a labeled dataset, we're simply looking for a Classification algorithm that can predict the binary state of whether or not a loan will be paid on time.

Our dataset is large and feature-rich, with hundreds of features and hundreds of thousands of records. This rules out Support Vector Curves and Naïve Bayes algorithms, both of which are too computationally intensive to be practical with datasets of this size.

The candidates that we selected for evaluation were LinearRegression (which is actually a classification algorithm), as well as the RandomForest and AdaBoost ensemble methods.

Linear Regression models the response between a dependent variable and one or more explanatory variables by attempting to derive a line that best separates the two classes of data (loans with problems and loans without problems). The algorithm does this by choosing a random line, measuring the distance between points in the dataset to that line as "error", then moving the line to minimize the total error against all points. Error is generally calculated as Ordinary Least Squares, or the sum of the squared error value.

[3] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Random Forests and AdaBoost are an ensemble methods, which use multiple instances of estimators to produce meta-results that are more accurate than individual instances of the underlying estimators themselves. Random Forests uses a number of Decision Tree classifiers on subsets of the training data to improve the predictive accuracy of the algorithm and control the tendency of a single Decision Tree to overfit the training data[4].

AdaBoost takes a slightly different approach. It still uses an ensemble of Decision Tree Classifiers, but it fits to the original dataset, then increases the weights on incorrect answers in order to focus on misclassified data points with each subsequent iteration, in an effort to reach cases that are more difficult to classify.

In practice, our evaluation showed that LinearRegression fails miserably in this particular application failing to identify any records as potentially problematic. Random Forests and AdaBoosts performed comparably. While Random Forests is about twice as fast as AdaBoost in training and prediction tasks, AdaBoost performs slightly better in terms of F1 score.

Model Tuning

AdaBoost was selected as the classification algorithm for this project for superior prediction scores and still-reasonable execution times. The values for AdaBoost's n_estimators and learning_rate parameters were tuned using an automated Grid Search, comparing ROC AUC scores for predictions generated by various combinations of those parameters.

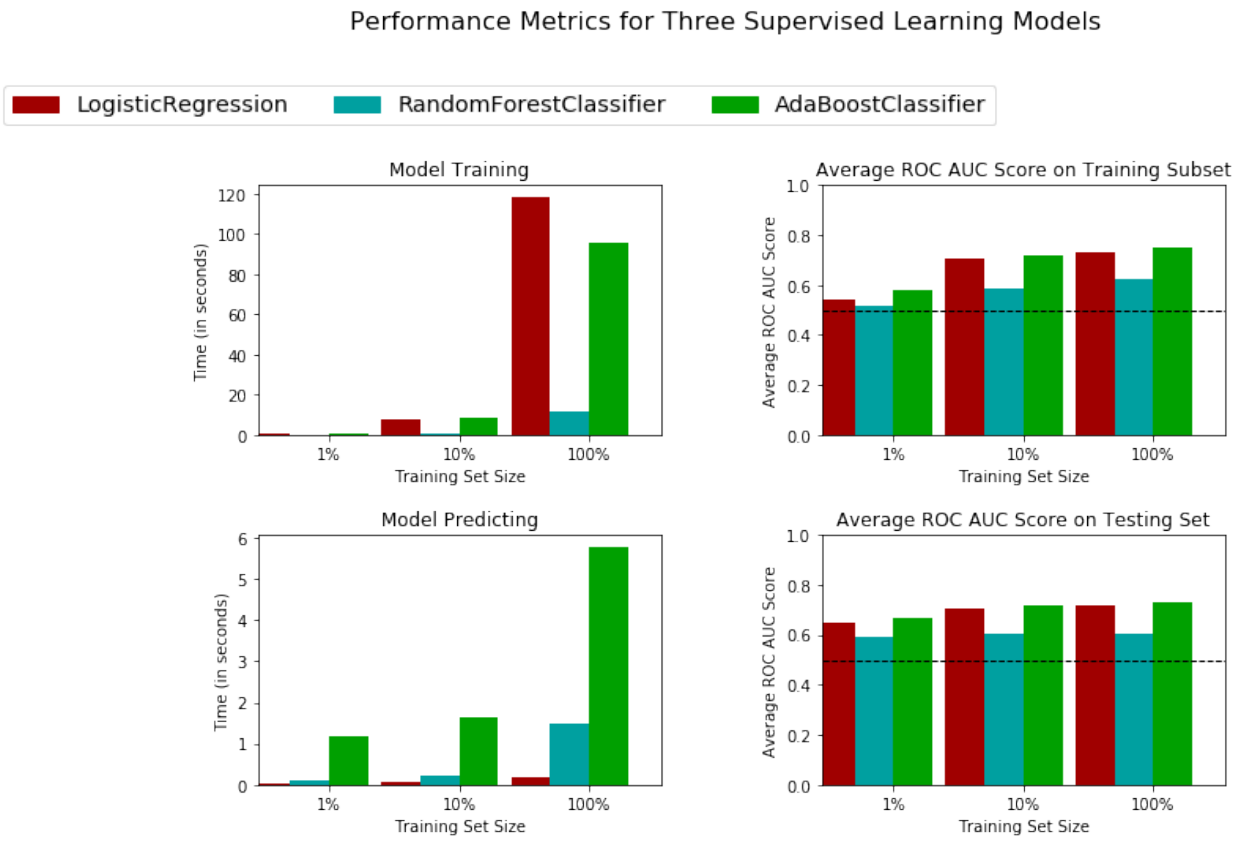
We ultimately arrived at values of 1 for the learning rate and 1000 for the n_estimators parameter. A value of 100 estimators produced reasonable results with much faster execution, so we use those throughout the project for intermediate sanity checks, with the full complement of estimators reserved for final evaluation passes.

Goodness of Fit

In order to ensure that our algorithm is not either overfitting or underfitting the data, the ROC AUC score was compared against both the Training and Testing subsets (in this case, we used sklearn's test_train_split function to generate testing and training subsets of our labeled application_train.csv dataset), to rule out the possibility that we are memorizing the training data and applying it poorly to test predictions.

As our visualization below indicates, each algorithm performs better than our naïve predictor (indicated by the dotted horizontal line), and we see reasonable performance against both the training and testing sets. We're not over-learning the training set, and our predictions exceed the performance of our naïve predictor.

[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



Results

An exhaustive Grid Search was conducted over the AdaBoost hyperparameters for algorithm, n_estimators and learning_rate. The search took about a day to run, and ultimately found that the SAMME.R algorithm with a learning rate of 1 and 1000 estimators produced the best results.

The SAMME (Stagewise Additive Modeling using a Multi-class Experimental loss function[5]) algorithm is an ensemble learning method designed for multi-class boosting. The SAMME.R algorithm is similar, but typically converges faster than SAMME, with lower test error and fewer boosting iterations[6]. The n_estimators parameter defines the threshold at which boosting is terminated, and learning_rate is a value [0 .. 1], which shrinks the contribution of each individual classifer. Since those parameters represent a direct trade-off, I conducted a pretty thorough examination of possibilities in the grid search.

Scores are ROC AUC with 3-fold cross-validation

AdaBoost	Score
Unoptimized	0.7294682667290626
Optimized	0.7506304243694171

[5] https://web.stanford.edu/~hastie/Papers/samme.pdf
[6] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Our ROC AUC score of 0.7506304243694171exceeds the naïve predictor score of 0.5.

Additional Exploration

After selecting the initial model, I explored further opportunities for improving prediction accuracy. While better than nothing, the predictive power of my model isn't a massive improvement over the naïve predictor.

To validate my preprocessing approach, I iterated over each preprocessing step, passing the results back through the model to ensure that each step had a netural or positive outcome on my prediction scores. This included experimentation with various scaling approaches, outlier removal and what data fields log transformations were applied to. Surprisingly, I ultimately came to the conclusion that no log transformations produced superior scores, at least at this point in the exercise.

Feature Importance and Dimensionality Reduction

Adding additional features to our dataset increases the dimensionality of our data, which in turn, increase the amount of computation time required. Given that our existing dataset isn't producing great results at this stage of the project, it makes sense to prune our dataset of features that don't emit useful signals about which applications may be problematic.

The AdaBoostClassifier provides a feature_importances_ property, which enumerates the features and their relative importance to determining a successful result. The importance of a feature is also known as the Gini Importance, described in the original Random Forests paper as the sum of the decreases of the gini impurity criterion, used as a fast, consistent approximation of the permutation importance measure[7].

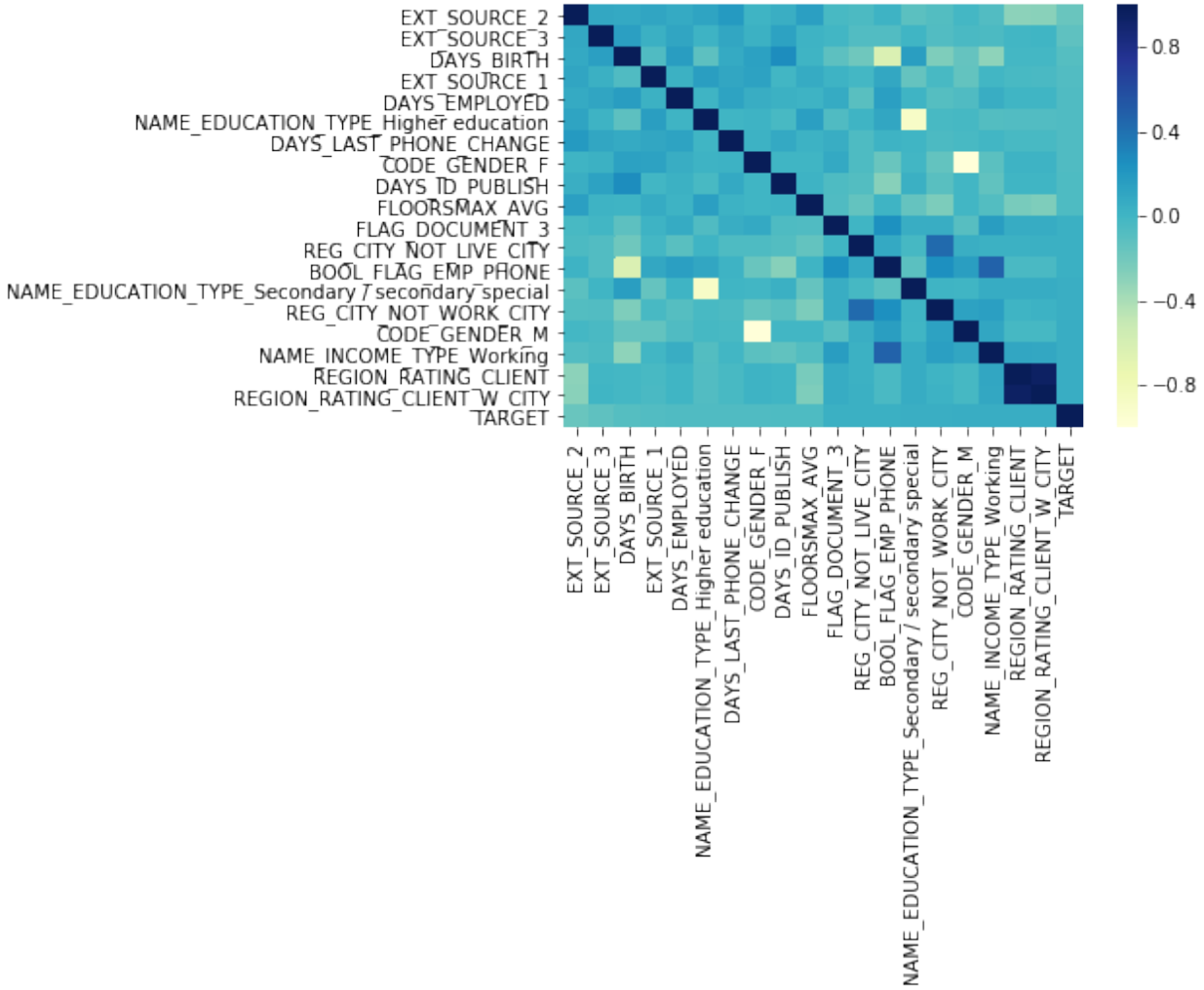
Through experimentation, it is clear that a subset of the most significant 40 features produces almost identical prediction performance results our optimal AdaBoost configuration on the original dataset, using a fraction of the original features; however, I was concerned about blindly applying this kind of reduction until all datapoints in the set were fully considered. To that end, I regularly compared results with both the full and reduced datasets, and recalculated the list of important features as new datapoints were added.

Proprietary predictive scores, labeled EXT_SOURCE_* actually prove to be the strongest predictors of whether or not an applicant will pay their loan, in addition to the applicant's age, length of employment and education (college vs. secondary). The applicant's hometown and region are actually the strongest predictors of default. Men are more likely to default, as are applicants that only graduated high school, and those that commute to a different city for work.

[7] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Feature Engineering and Additional Datasets

In the application data, I looked at features with strong correlations for interesting data points.



Some interesting highlights include strong negative correlations (the white squares) between applicants who graduated high school and those that graduated college, men vs women, and the applicant's age, and whether or not they supplied a phone number for their employer (I presume that in practice, the employer's phone is an indicator of verifiable employment).

Strong positive correlations (dark squares) are shown between people "registered" in the city but don't work there, and people "registered" in the city that don't live there. In both instances, they have a commute to the city in common. There's a correlation between people that are employed as "working" and those that have an employer phone number, which intuitively makes sense. The age of a person's identification documents and their actual age is strongly correlated. My guess is that false identities probably have newer documents, and older applicants are more likely to pay loans in general, and would typically have well-established documentation.

These aspects are well-represented in our feature importance list already, but understanding the trends lends some confidence that I'm at least on the right track.

I initially experimented with creating polynomial features out of the list of existing features to see if interactions between features might produce stronger signals, but I ultimately abandoned this approach, as adding the most important polynomial features to the dataset actually reduced prediction scores significantly.

The next attempt was to conduct a Primary Component Analysis of the 10 most important features, examining the composition of each dimension for further inspiration. (see jupyter notebook: Step 4.1 – Feature Engineering – Application Data)

The first grouping of features explains 28.27% of the variance, and includes the amount of the loan, the amount of the purchase (for commercial loans), and the amount of the annual payment.

The majority of PCA dimensions were dominated by the proprietary EXT_SOURCE_* features, and by various combinations of AMT_CREDIT, AMT_GOODS_PRICE and AMT_ANNUITY, appearing both in concert and opposition.

In thinking about what these signals might mean, it seemed like understanding the loan and loan payment as a percentage of a person's income might offer additional insight, as well as how much the person's down payment was on that commercial purchase (represented as the percentage of the purchase price borrowed).

When adding these features to the reduced dataset, PER_GOODS_CREDIT, PER_ANNUITY_INCOME and PER_CREDIT_INCOME all appear in the top 40 features.

Scores are generated using 3-fold cross-validation with the roc_auc scorer:

Dataset	ROC AUC Score	Number of Predicted Targets
Application – Full Feature Set	0.7472294334657973	746
Application – Reduced Feature Set	0.7390721573995197	496
Application – Full + New Features	0.7483329192450814	888
Application – Reduced + New Features	0.746894528394559	840

Conclusion

In summary, this project attempts to determine whether a particular loan repaid, by applying supervised learning to a historical set of loan data. Supervised Learning techniques are employed to classify loans as problematic by building an accurate predictive model based on the historical dataset.

The original dataset was examined, problematic records and outliers were normalized, and data was processed and structured in a manner suitable for machine learning. We then explored a

number of potential learning algorithms, selecting the best one based on predictive performance and feasible execution time.

We ultimately employ the ensemble method AdaBoost to aggregate and weigh the results of multiple instances of simple learning algorithms to better tune and focus the model on difficult edge cases. Hyperparameters for the algorithm were tuned automatically, using Grid Search with cross-validated scoring to determine the best performance.

Once a model was selected and tuned, the individual features in the dataset were weighted for importance, and important features were examined through PCA analysis to see how features worked together to explain classifications. Correlations between datapoints were also examined to understand relationships between dependent features. Insights into those relationships led to the composition of new features that were relevant to the problem domain, which improved predictive accuracy of our model.

Finally, additional datasets with many-to-one relationships to the original application data were aggregated and added to our dataset to improve the predictive accuracy of our model.

While my model is not competitive in real-world scenarios at this point, the exercise was educational. I was able to explore new scoring methods (ROC AUC) and model selection techniques (cross-validated grid search). The project also required new skills for joining and merging multiple dataframes, as well as engineering and validating new features. A significant amount of troubleshooting and double-checking results led to a much deeper understanding of the Pandas, Numpy and Sci-Kit Learn libraries.

I intend to continue to refine this project in the future, with the goal of getting a competitive score in the Kaggle competition. The inclusion of additional features, deeper exploration of feature engineering (particularly around domain-specific insights), and the employment of automated feature selection techniques all seem like promising next steps.