Santander Bank Customer Satisfaction Kaggle Competition

Ankit Sharma, as63437
ankit.sharma@utexas.edu
Sneha Shrotriya, sss2929
snehashrotriya@utexas.edu
Shruti Subramanian, ss53428
shruti.subramanian@utexas.edu

December 8, 2016

Abstract

Our team entered the Santander Bank Customer Satisfaction Kaggle Competition to learn more about classification data mining problems. Santander Bank is asking for help to identify dissatisfied customers early in their relationship with the company. This would allow Santander to take proactive steps to improve a customer's happiness before they decide to leave the bank. This problem proved to be a challenge because of the large number of customer features and the anonymized feature names. We evaluated our performance on this problem using the score on Kaggle's private leaderboard. In our attempts, we discovered the importance of data exploration, including understanding clearly what each feature represents, and the positive effect that data preprocessing has on improving models. We also used the models themselves, particularly XGBoost and Random Forests, to help with feature selection. Our overall approach consisted of some basic cleaning, followed by exploratory models, more in depth data preprocessing, model tuning, and some final ensembling. We spent a significant amount of time tuning our parameters for each of model: logistic regression, multilayer perceptron, random forests, XGBoost and neural networks with Keras, and used an AWS server to speed up our development and collaborate more effectively. We concluded our work with an ensembling of our successful models and a discussion on our key findings.

1.0 Introduction and Background

The problem discussed in this paper is the Santander Customer Satisfaction Kaggle Competition, which ran from 3/2/16 to 5/2/16. The goal of this competition is to design a model which will predict the probability that a customer is dissatisfied with their banking experience, using a training set with semi-anonymized features. The scoring metric used for this competition is area under the receiver operating characteristics (ROC) curve based on the probability we predict for each data point in the test set provided for the competition. Since this competition has concluded, we will discuss and compare our performance on both the public and private leaderboards, each of which contain about half of the test data randomly split.

From a business perspective, this problem is important because identifying unhappy customers before they leave can prevent the bank from incurring unnecessary extra costs and maintain the bank's reputation. According to the bank, this problem is important because "customer satisfaction is a key measure of success. Unhappy customers don't stick around. What's more, unhappy customers rarely voice their dissatisfaction before leaving" [1]. In addition, recognizing unhappy customers is important because poor customer experience costs firms more than $40 billion per year and customers are three times more likely to tell others about a negative experience than a positive experience [2]. Being able to identify an unhappy customer is important so that the bank can attempt to retain the customer, since it costs six to seven times more to acquire a new customer than retain an existing customer [3].

The data set for this competition is semi-anonymized, meaning the features are not clearly labeled; although, some of the feature have partial names and abbreviations in Spanish, allowing us to guess the true label. Not knowing for sure what all the features were was the most difficult part of this problem. Due to this lack of feature labels, we did not know which features were categorical and which were continuous and what amount of variance was significant for each feature, making preprocessing and cleaning the data difficult. To get around this issue, we examined the distribution of some of the important features, as determined by XGBoost and

random forests, and looked at some of the many posts on the forum dedicated to feature exploration.

Our approach for this problem included some initial attempts using neural networks, random forests, and logistic regression with very basic data cleaning. As a result of these attempts, we decided to focus more on data exploration and tuning our most successful models, XGBoost and random forests. Although our initial neural network using sklearn's multilayer perceptron (MLP) library was not very successfully, we continued iterating on neural networks since many of the top solutions used a neural network in their ensemble. This allowed our models to have more variety, since both random forests and XGBoost are tree-based model. Additionally, despite a relatively small data set (~60 MB for both training and test), for the later attempts, we decided to set up a Jupyter notebook server on Amazon Web Services (AWS) so that we could run large grid searches for our models in a time efficient manner. For the later attempts, we also relied more heavily on the forums and the code of some of the high scoring teams. We also examined the approaches in the posted code to tune our model parameters and get ideas for feature engineering and on how to ensemble models. The code for the model's link can be found in Appendix A and a summary of our approach is included Figure 1. We discuss this approach throughout sections three and four of the paper.

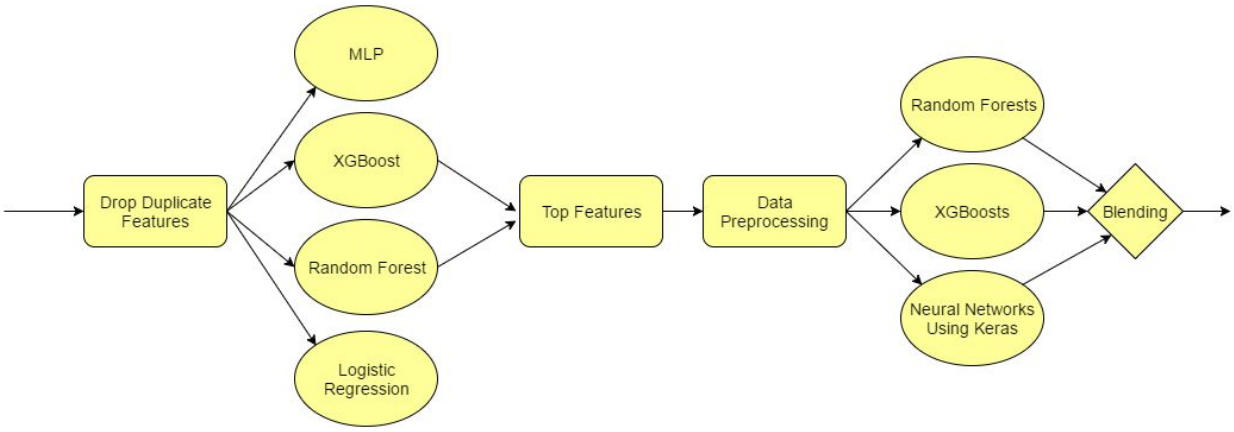

Figure 1: An overview of our approach

2.0 Data Description and Preprocessing

The data set consists of 76020 training samples with 368 anonymized features and two labeled features, id and target, per sample. The target feature takes on a value of one for unsatisfied customers and zero for satisfied customers. The test data consists of 48859 samples also with 370 features each. For the 368 features which are anonymized, we were initially unable to apply any intuition or external research to determine which features might be most important. In order to circumvent this issue, we initially trained an XGBoost and a random forest classifier on the unprocessed data. Since both of these models have built in feature selection, we used the output from these models to determine which features had a high predictive power and then examined the distribution and partial labels of these features in order to guess what they represented.

The plot in Figure 2 shows the distribution of F-scores for the different features as determined by our initial XGBoost and Figure 3 shows the same distribution determined by our initial random forests classifier. Both of these models were generated on data with no data preprocessing other than removing duplicate data. There is a significant drop off past saldo_var30 for the data generated by XGBoost and past ID for the data generated from random forests. In addition, both the models have the same top four features, var38, var15, ID, and saldo_var30, albeit in a slightly different ordering. This ranking could indicate that most of the variance comes from just the top four variables.

Using the feature importance reported by XGBoost and random forests, we developed a preprocessing strategy with consisted of three main stages 1) manipulating existing data, 2) dropping seemingly unnecessary features, and 3) engineering additional features. The code for the preprocessing can be found in the github link in Appendix A.

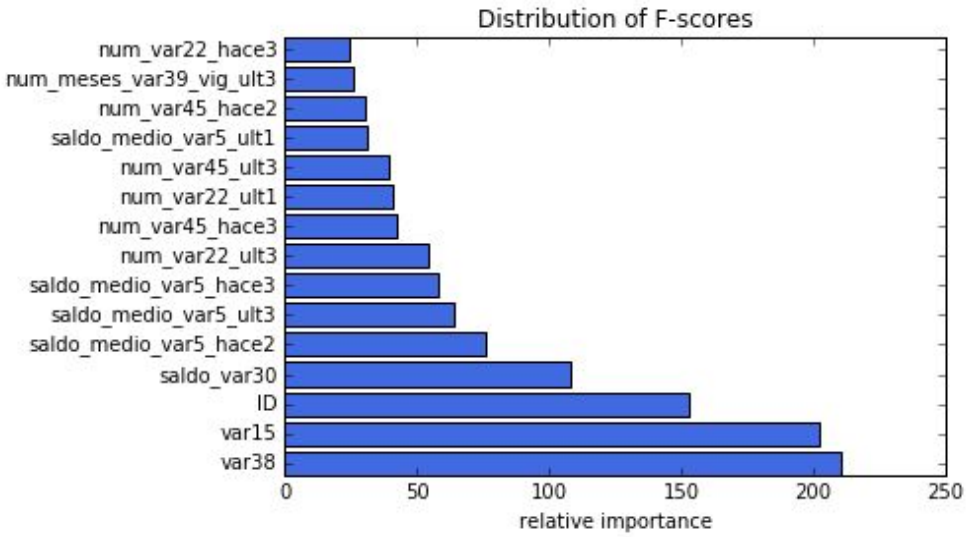

Figure 2: Top 15 features based on XGBoost on unprocessed data

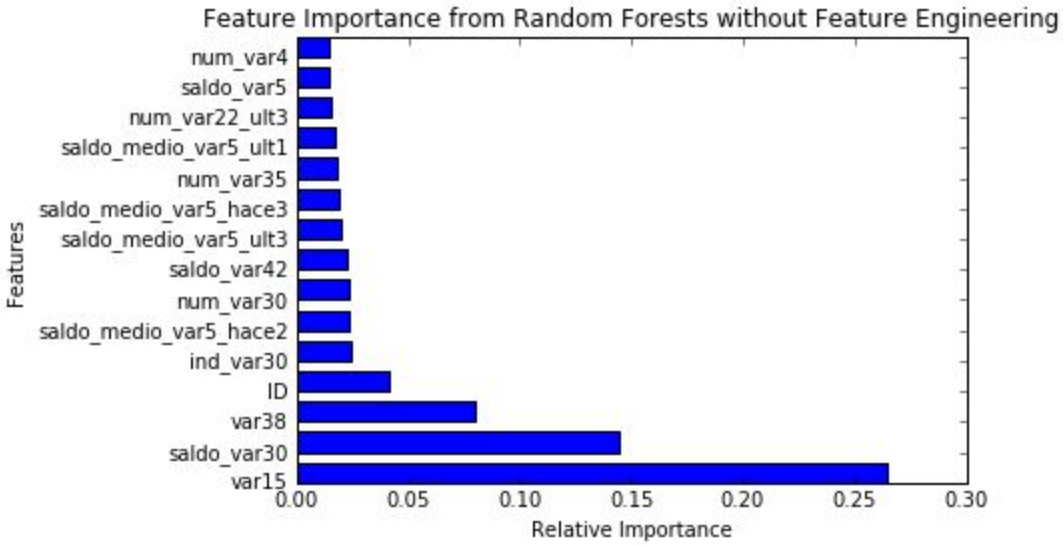

Figure 3: Top 15 features for Random Forests on unprocessed data.

2.1 Exploring Features and Manipulating Existing Data

Based on the relative feature importance, we started our exploration with looking at the top features, determining what the features were and the most effective way to handle the feature. We also consulted the forum to see which features some of the winning solutions had explored. In addition, in our preprocessing script, we created an option to allow the caller to standardize

the data, using sklearn's StandardScaler, and determined whether or not to use it based on the actual performance of the models. Finally, we also explored the partial labels of the features, such as saldo or medio, which could be abbreviations for Spanish words, which might give some insight into what the variable represents.

Exploration of var15

One of the most significant features shown in the figure 2 and 3 above is var15, which based on the range could be the age of the customer and the discussion on the forum seemly to largely support this conclusion. However, the distribution seems wrong for age. Firstly, the median is around 25 while the median age for Spain, where we are guessing most of the customer data is from, is closer to mid-thirties. Secondly, the population distribution in Spain is more bell shaped [4]. However, a possibility is that the data given is simply skewed towards younger customers. Another guess we had for this variable value was the amount of years the primary account linked to a customer has been active. Either way, treating var15 as a continuous variable seems reasonable. The plot in figure 4 shows the distribution of var15, with the happy and unhappy customers shown separately. The plot indicates that most of the unhappy customers have a var15 of between 23 and 50. Based on this examination of var15 values, we did not find a compelling reason to directly process var15 further in our preprocessing steps.

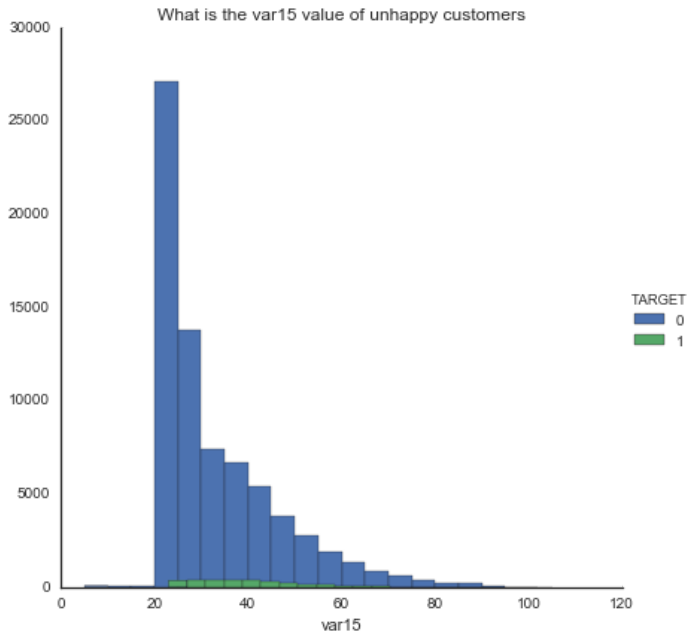

Figure 4: Distribution of var15. The green indicates unsatisfied customers and the blue indicates satisfied customers.

Log Transforming var38 and Splitting var38 into Two Features

The XGBoost on the unprocessed data indicates that var38 is the most important feature and the random forest indicates var38 is the third most important feature. The plot in figure 5 shows the distribution of the log of var38. Based on the range and distribution of var38, we guessed that var38 is the money in a customer's account. However, on the forum, several people with domain knowledge suggested that this was the mortgage value of the account held by the customer [5]. In the plot in figure 5, there is a suspiciously large mode, so we plotted the data removing the mode value and found the data as log normally distributed. As a result, we decided to split var38 into two features, one for the log of var 38, with zero in place of the mode, and one categorical feature that indicated whether the customer has the mode at the var38 value or not.

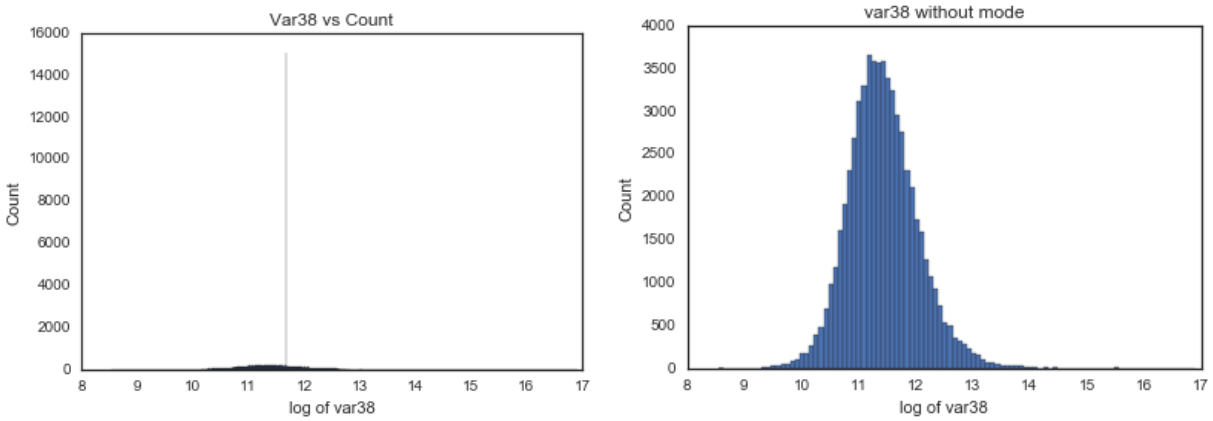

Figure 5: The plot on the left shows the log distribution of var38 and the plot on the right shows the log distribution of var38 except for values in mode bin.

Replace Missing var3 Values With Mode and One Hot Encode

Finally, although var3 was not considered one of the top 15 features for XGBoost or random forests, we also did some processing on var3 based on suggestions we found in the competition forum. Based on the distribution of var3, shown in figure 6, the posts on the forum guess that var3 is the country of origin for the bank and the spike seen in the plot for var3 at a value of two is probably for accounts based in Spain and the other values around two may represent other Spanish speaking countries [6]. This finding suggests why var3 maybe did not show up as an important variable in the initial xgboost: we were treating var3 as a continuous variable, when the variable was actually categorical. As a result we used one-hot-encoding for representing the feature. Additionally, some entries for var3 had the value as -99999, most likely indicating an omitted value. We chose to replace this value with the mode, although making -99999 its own category would have been a reasonable alternative since the value may have been left out on purpose. The plot on the right hand side of figure 6 suggests that there may be some relationship between var3 and target since there seem to be three distinct clusters of unsatisfied customers. However, we also noticed the test set included 32 var3 values that were not in the training set, for which we had no information. Hence, var3 might not be as useful for predicting the labels in the test set.

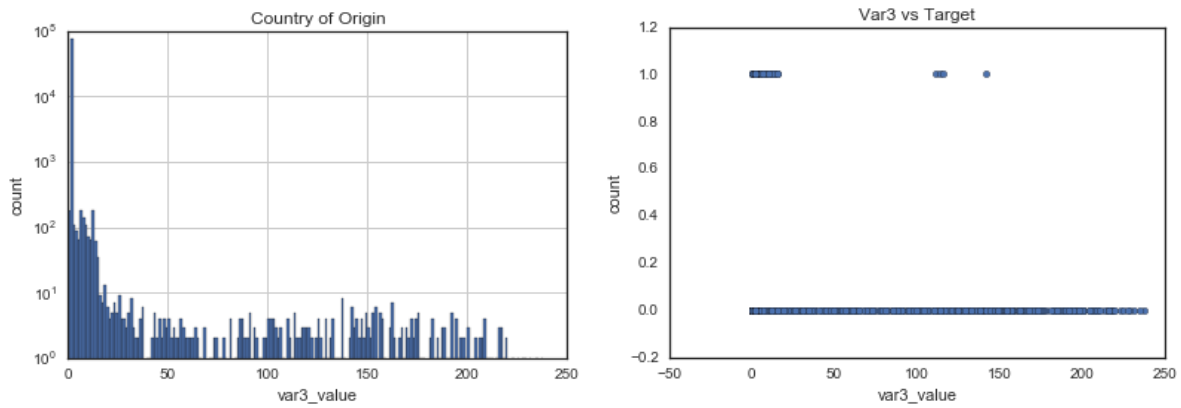

Figure 6: The left figure shows a histogram of var3, which is probably the country of origin for the customer. We believe the spike at the value of 2 represents Spain. The right figure shows var3 as a function of the target, which visually indicates that there is a relationship between var3 values and customers more like to have a target value of 1 (i.e. be unsatisfied).

Exploring the Partial Labels

While we were not provided with clear labels to the data, we were given some partial labels, which we used to determine what some of the other features might represent. We guessed the labels were in Spanish since Santander is a popular bank in Spanish-speaking areas. In addition, the partial labels kind of make sense if they are interpreted in Spanish. For example, saldo is balance and medio is average in Spanish. We decided that saldo_var30, is probably the balance of some account, so we should treat the value as a continuous variable. In another example, hace means ago and many of the variables containing hace end with a number, so the features containing hace could indicate the balance of some account some time units ago. Using our limited knowledge of Spanish, Google translate, and the forum posts, we gathered the translations to perform a similar analysis on the other features.

2.2 Removing Features

Since 370 features is a lot to work with and we suspected that some of the features were not actually related to the customer's satisfaction, we worked on dropping features that might be unimportant and those features with low variance. We discuss the ID feature here since although

we ultimately elected the leave the feature in for our final models, we experimented with dropping the feature.

Drop the ID...Maybe

The most surprising find from the feature ranking by XGBoost and random forests was that the ID was a highly important feature for both models. Although the data set is semi-anonymized, the ID is one of the two features clearly labeled. The ID feature is a unique identifier number that corresponds to a single customer. The ID is supposed to have no utility other than uniquely identifying the data for submissions. In submitting the test data to Kaggle, we have to submit our predictions as key value pairs of ids and probabilities. Had we realized the ID feature was in the training set when performing the initial models on the unprocessed data, we would have dropped the column. However, we accidentally left the feature in when training the model. Surprisingly, the id feature is the third most important feature for XGBoost and fourth most for random forests. This finding suggests that there might have been some sort of data leakage in the test set, where the way the data was entered or sorted and organized was related to the customer's satisfaction level. However, despite searching in the forum, we did not find any discussion on this finding as most people had simply dropped the id column before doing any preprocessing or training any models. To further investigate, in the preprocessing we developed, we had an option to drop the ID or not, and experimented both with and without the id. We saw the best results when leaving the ID for our final model.

Removing features with Low Variance

Although the data set had a lot of features and we were not able to examine each one individually, the plot of the features important for XGBoost and random forests suggests that most of the variance was coming from a few of the features, so dropping some of the less important features would be a good idea. We calculated the variance of each of the features, which we thought were continuous features; however, since we did not know most of the feature labels, we could not determine what variance was significant for each value. We noticed that about half of the features had a variance of less than one, while the remaining features had

variances orders of magnitude higher. These features also did not show up in the top 15 features for XGBoost or random forests. Using this finding, in our preprocessing, we dropped all of the continuous features with variance less than one.

2.3 Feature Engineering

As the last step of our preprocessing of the data, we did some feature engineering based on our observation of the data and used principal component analysis (PCA) to generate high variance features.

Adding a feature for the sum of zeros

Through a quick glance of the data set we noticed that a large number of values for features were zeros and wanted to determine whether there was some relationship between how sparse the data was and a customer's dissatisfaction. Perhaps more zeros indicate that a customer does not interact with the bank much or does not use as many bank products, which intuitively seems like a factor that may indicate unhappiness. The plot in figure 7 shows a histogram of the number of zeros in a row, where the blue lines indicate the satisfied customers and the green lines the unsatisfied customers. The plot suggests that customers with more zeros for feature entries are more likely to be unsatisfied since the taller green bars occur at larger values of the number for zeros. This finding seemed promising so we added the count of the number of zeros as an additional engineered feature to the training set.

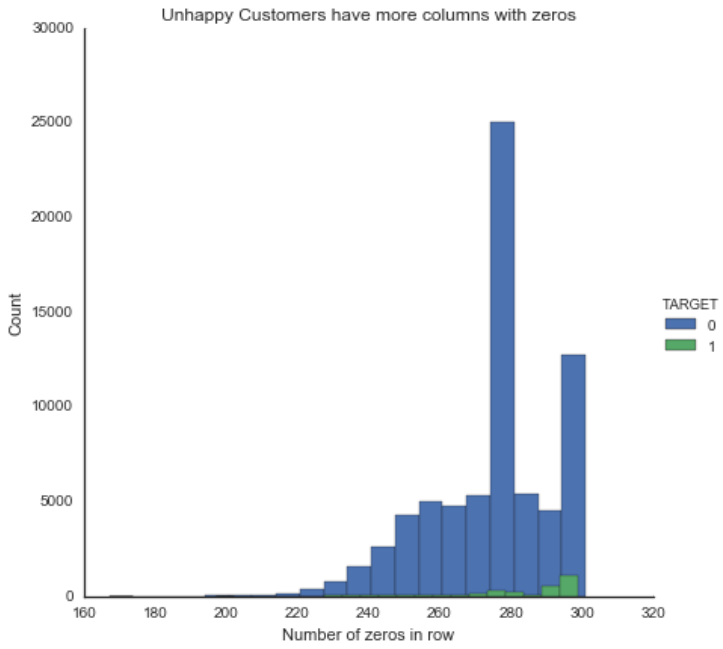

Figure 7: Distribution of number of zeros in a row for satisfied and unsatisfied customer

Adding Principal Components

Since we planned to drop many features but still wanted to capture the maximum variance for the model, we performed PCA on the data and used the top five principal components as features. Recognizing what each component means is usually difficult; however, since in this case we did not know the data anyways, adding the components did not really affect our interpretation of the data and allowed us to more confidentially drop features with low variance. We chose the top five components by trying a range from zero to seven components and having the most success with five.

2.4 Results of Preprocessing

After developing the preprocessing produce discussed above, we ran a random forest and a XGBoost classifier on the processed data and extracted the top fifteen features for each model. Figure 8 shows the feature importance for random forests and Figure 9 shows the features for xgboost. A number of the additional features we added showed up in the model including the PCA components, log of var38, and the number of zeros in a row (n0) for XGBoost and var38ismode for random forests. In the next section, we discuss our XGBoost and random forests

model in greater depth as well as the additional models we used, and examine how the preprocessing improved our score.

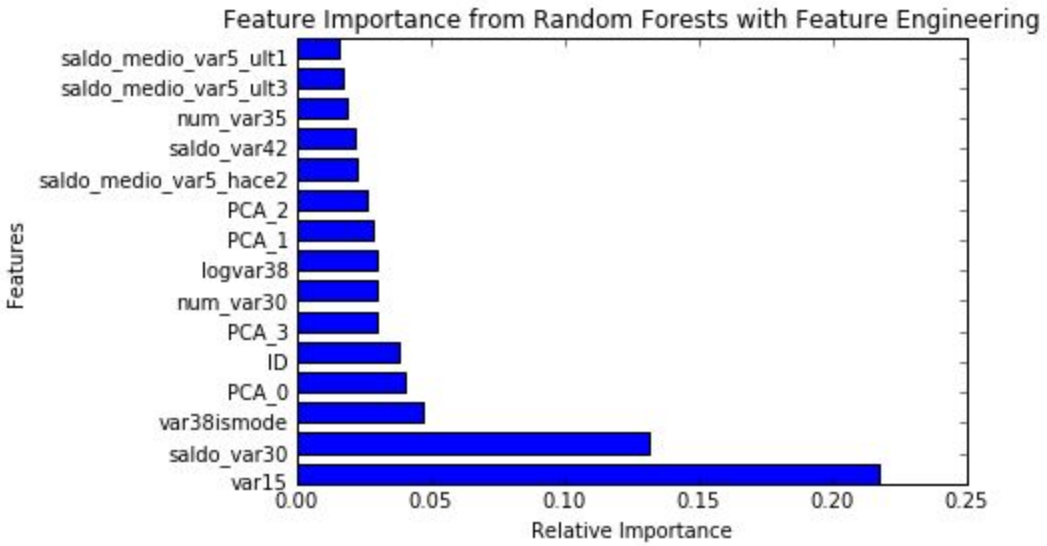

Figure 8: Top 15 features based on random forests on processed data

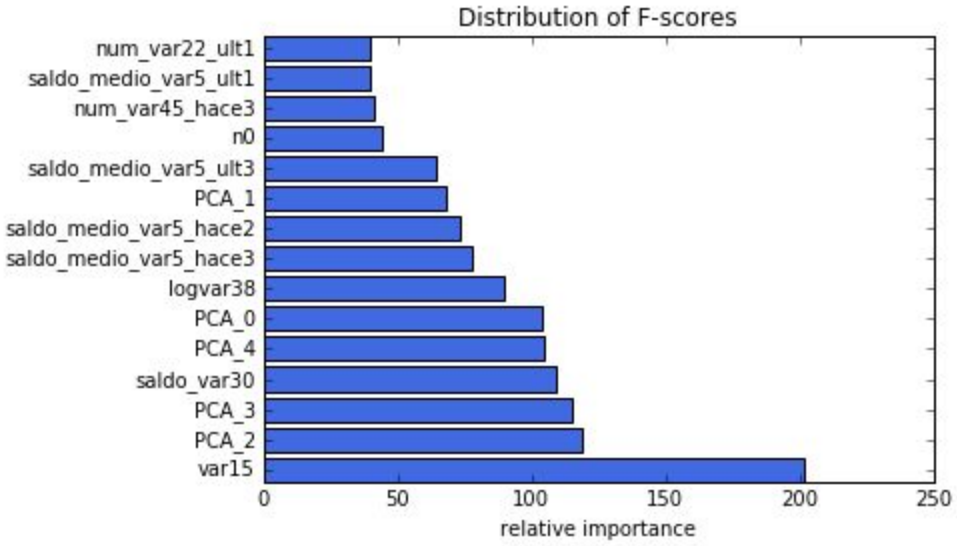

Figure 9: Top 15 features for XGBoost after preprocessing

3.0 Computing

Exploring the data set led us to understanding the scope of this classification problem. Although the data set itself is small, in order to try multiple parameters for preprocessing and perform grid

searches to tune our model efficiently, we needed additional computing power. For our initial attempts, we were able use laptop computers to carry out modeling without too much latency. Later in the design process, we had to wait for many hours for grid searches to finish running so we looked into cloud servers to handle our computing tasks. We learned about AWS and their EC2 product while working on this project.

Certain EC2 products proved to be more suitable for this problem than others. We initially thought that we should select a compute optimized instance as these machines would be able to carry out computations faster. However, we saw no noticeable improvement in the speed of running our code with this selection. We discovered that these compute optimized machines did not have enough RAM for the model fitting process. Instead, we switched to using memory optimized instances. We eventually settled on an instance with 64 GiB of random access memory and 8 virtual processors. With this modification, we saw a considerable improvement in how fast we could train models.

Additionally, we set up a Jupyter notebook server running in the background on our EC2 instance that could be accessed by group members from anywhere by just sharing a link without having an active ssh session into our EC2 instance. The Jupyter notebook was helpful for collaboration because executed code was visible to all members of the group. This was particularly handy when one group member had inaccurately tuned or made an error in programming that another could easily spot. This setup was the primary environment we used for developing and tuning our models.

4.0 Predictive Modeling and Model Comparisons

Our modeling efforts were comprised of two stages, exploration and production. During the exploratory stage of predictive modeling, we tried to apply several models taught in class to understand which types work most effectively. These included a logistic regression, sklearn's MLP neural network, random forest, and gradient boosting using xgboost. We chose models in this phase based on ease of tuning, our familiarity with the model, and speed of results. The

conclusions drawn from our exploratory attempts and guidance from the Kaggle forums suggested that we should work on further tuning our random forest and XGBoost models. In addition, although we had little success with MLP, in our final model, we still used a neural network implemented using Keras since a number of people on the forum had success with neural networks. We later created an ensemble of these models to give us our final predictions. To improve the performance of each model we alternated between adding iterations of data preprocessing and tuning the model on more parameters. The winning score on Kaggle's private leaderboard was 0.829072 and we were able to achieve a score slightly below this but above 0.82000 using our models. The code for the model's link can be found in Appendix A

4.1 Logistic Regression

Since this problem is a classification problem and logistic regression is easy to tune, the first model we used for this problem was a logistic regression. Initially, we used the non-preprocessed data and default parameters. In the second iteration, we performed a grid search over both the L1 and L2 penalties and varying the C value (inverse of regularization strength) from [0.001, 0.01, 0.1, 1, 10, 100, 1000]. For the third iteration we added in data preprocessing and in our fourth iteration we tuned the model over the same grid search with the preprocessed data. Figure 10 presents a summary of our private and public results on kaggle with logistic regression over each iteration.

Overall, the logistic regression performed poorly compared to our other models and significantly lower than the top scores on the leaderboard. In addition, the logistic regression actually got slightly worse with more tuning and preprocessing. Possibly the extra tuning overfit the model causing poor performance on the test set. However, the preprocessing improved all the other models, so we were surprised that the logistic regression performed worse. As a result of these findings, we determined that logistic regression was not well suited for this problem and did not include the model in our ensemble.

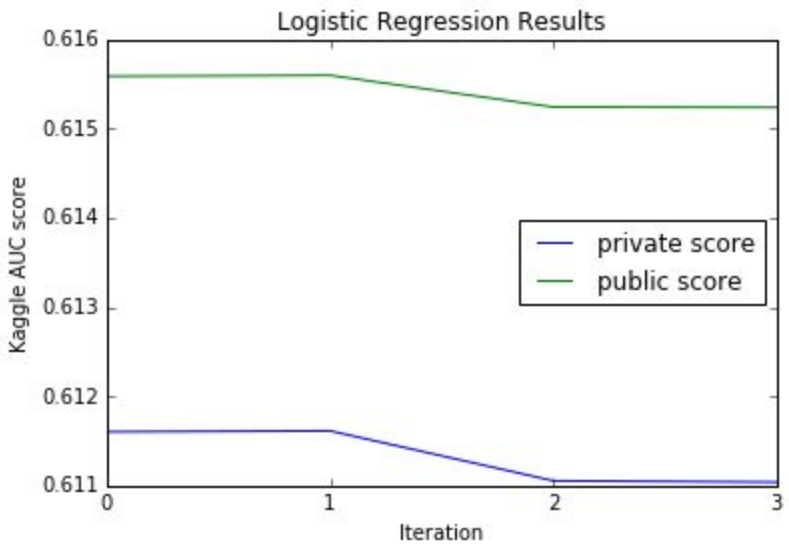

Figure 10: Logistic regression performance over various iterations, which are described in section above.

4.2 MLP

Another model we attempted was MLP because some of the top scoring models used neural networks in their final ensemble. As with the logistic regression, we planned to fit MLP first on the non-preprocessed data to get a benchmark and later tune the parameters on processed data. We did a small grid search over different parameters for the number of hidden layers, the maximum number of iterations, and the activation function and scored 0.66265 on the training set. However, the MLP model we trained predicted all 0s for the test set, which is interpreted as all customers are satisfied with their banking experience. This model scored 0.50 on the test set, which is no better than a model which randomly guesses. This model was significantly worse than our logistic regression discussed above; however, we suspect that the reason for the failure of the MLP model was not because neural networks are not suited for this problem, but because we did not tune the model well or possibly because the implementation of MLP is sklearn is not as effective as other neural network implementations in dedicated libraries. The evidence for this guess is that later we tuned a neural network using Keras which performed much better on the test set and other successfully applied neural networks to this problem.

4.3 Random Forests

Our initial attempt at using random forests showed some promising results which prompted us to further refine this model. In our first iteration, as seen in figure 11, we began with some basic tuning of the model by including a small grid search over the n_estimators (the number of trees in the forest) and the max_features (the number of features considered at each split) and we saw that the output became more accurate according to our AUC scoring metric [7]. Next, we decided to preprocess our data rather than trying to fit all features with our model, so in our second through fifth iterations we incrementally added in our data cleaning and preprocessing discussed above. The trend we noticed was that as our preprocessing script became more advanced, our model was able to fit the data more accurately. In our sixth iteration we began to tune our parameters for the random forest classifier with a larger grid search over n_estimators, max_features and criterion (a function that measure the quality of a split) [7]. In our seventh iteration, we decided to try to drop the columns that had low variance which also improved our score.

Since random forests have so many parameters, we tried to run a larger search over more of the parameters. This expanded grid search still included n_estimators, criterion, and max_features, but it also included max_depth, min_samples_leaf, and random_state. We initially tried to run this locally, but after 12 hours there seemed to be no end in sight to find the tuned parameters. Once we were able to set up our AWS server, we were able to relatively quickly find the fine tuned parameters for our model. In our last iteration we added the ID back into the features since we saw that ID seemed important as seen in figure 8. With the ID added back in as a feature we saw that our model performed better. The improvement in performance supports our hypothesis of data leakage that we previously discussed.

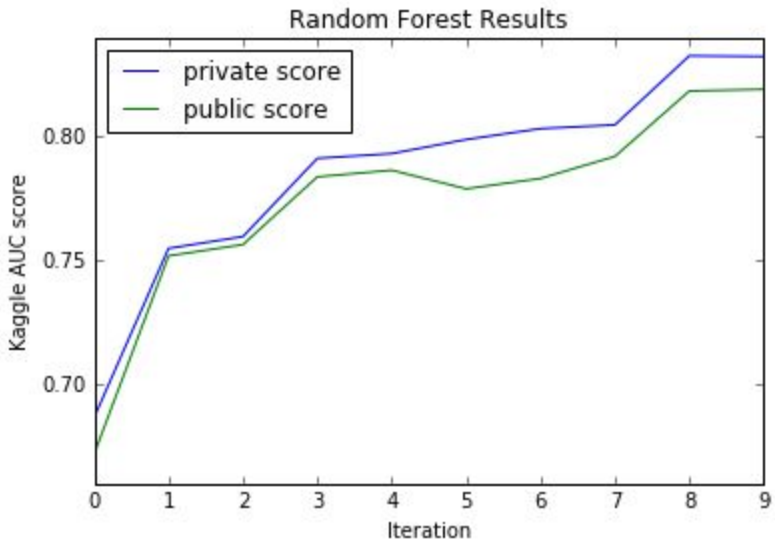

Figure 11: The effect of each iteration on the kaggle AUC score. Each iteration is discussed in depth in the above paragraph.

4.4 XGBoost

XGBoost has emerged as a popular model for classification and regression problems in Kaggle. XGBoost, or eXtreme Gradient Boosting, is a tree-based classifier that uses an ensemble of short decision trees and decision stumps to make a prediction. Predictions using gradient boosting models includes many hyperparameters to tune these weak predictors and the ensemble itself, so our group had to make extensive use of sklearn's cross-validated parameter searching tools (GridSearchCV) to find the suitable parameters for a successful prediction. We realized that the computing power and memory of our AWS instance could afford a miniscule learning rate and use a large subsample of a training set to fit a large number of trees.

Score optimization efforts led to many interesting conclusions. For example, we discovered that that XGBoost performs much better when the features are scaled between zero and one, and we saw improvements to our private scores when this was added to our preprocessing steps. In addition, several of our engineered features had high importance scores in the predictions. However, this problem only required a small subset of the initial feature set to be included.

We also noticed that XGBoost may have some problems with overfitting after observing that predictions had very high AUC scores on the training data but much lower scores on Kaggle. To combat this, we also made use of the XGBoost early stopping feature. This feature works by recording the point during tuning where the prediction accuracy on the validation set begins to decrease as the accuracy on the training set continues to improve. The hyperparameters at this "epoch" are used for fitting and prediction. We were able to achieve private scores greater than 0.82 using XGBoost.

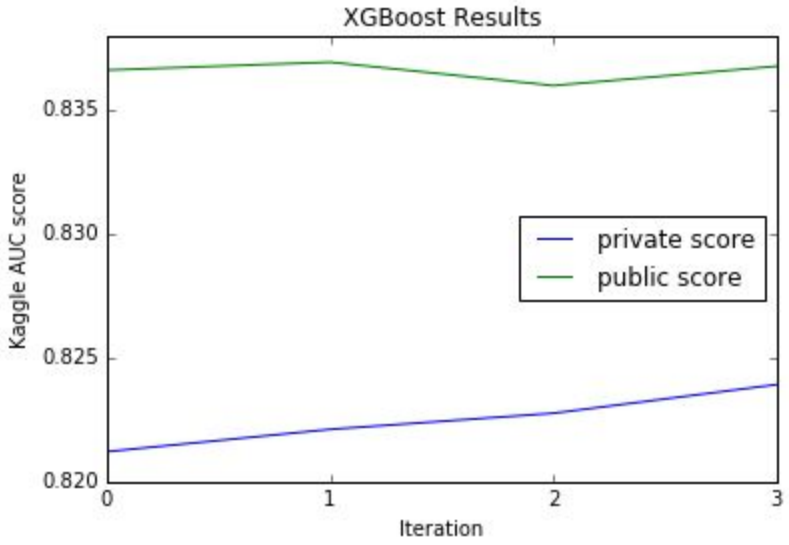

Figure 12: The effect of each iteration with XGBoost on the kaggle AUC score. At each iteration we tuned applied more preprocessing and tuned our model slightly.

4.5 Neural Network Using Keras

While the initial failure of sklearn's multilayer perceptron neural network model deterred our group from applying neural networks to this problem, consulting the forum helped us give neural networks a second look. For this project, we had the most success with Keras, a neural networks library in python. Keras executes neural network code by forwarding compute jobs to parallel processors using Theano or TensorFlow. Group members were relatively unfamiliar with feedforward neural networks prior to working on this project, and we learned a lot about working with these models in Keras.

One of the biggest challenges associated with using neural networks was designing the layers. Our group was not too familiar with the analysis that goes into this, but we made use of the Kaggle forums to bridge this gap in knowledge. We found that the third place team made use of Keras in their neural network. We borrowed ideas from this group's solution, which was a sequential model with two layers [8]. The third place team made use of the parametric rectified linear unit (leaky ReLU) and softmax activation function between layers. They also included normalization steps and used an adaptive learning-rate gradient descent (Adagrad) as an optimizer. We were excited to try out these new tools on the Santander competition, but unfortunately our implementation of this neural network performed poorly.

We decided to create a more simple neural network. Our next neural network was comprised of two layers, and first reduced the dimension of the input data using a hyperbolic tangent activation function. The next layer does a linear transformation to one dimension, and we use a sigmoid function here. Though our score is far from the top of the leaderboard, we did manage to achieve a private score of about 0.78 after scaling the features and applying our preprocessing ideas.

4.6 Ensembling

Once we designed and tuned several different models, we began to explore the potential benefits of model ensembling. The models we used in our blending ensemble included XGBoosts, random forests, and neural networks as they were the better suited models for this problem as was discussed previously. We fit every model in our blending ensemble solution using cross validation.

For the blending approach, we divided the training set with a 10/90 split [9]. We split the training set into two so that we could train our models with 90% of the data and use the other 10% to generate predictions using each of our previously discussed models. We appended to a Pandas dataframe blended_ensemble_train to appended all the predicted probabilities from the 90% portion of our split. On the blended_ensemble_test dataframe, we used each of our models to

predict the target and appended to this in a similar fashion. We then tuned another logistic regression to fit blended_ensemble_train to the corresponding given target values for the samples. We used the meta-model from this blending ensembling method fit the predictions of blended_ensemble_test [9].

By using this blending technique, we aimed to reduce the inherent biases of each model such that we could see an improvement on our private kaggle score, which improved up to 0.824485. Blending did better than our basic predictive models on the Kaggle leaderboard, and we found that there was a smaller spread in scores with the use of blending in comparison to each standalone model. Due to the decrease in spread, we believe that with more training this blending method could be even more successful.

5.0 Key Findings and Lessons Learned

As we worked on this classification problem, we learned about the importance of knowing the feature labels for a data mining problems, how to use AWS to increase our computing power, and discovered that tree-based models tended to work well for this problem.

Working on a classification problem without knowing what the features are is incredibly difficult. We could not apply any domain knowledge to the problem and had a lot of difficulty in figuring out which features were important. In addition, we spent a lot of time working on preprocessing or simply exploring the features, which we could have spent on tuning the models and working on more effective ensembling methods.

We also discovered the power of cloud-based computing resources in using AWS and learned the importance of having enough RAM in the machine to perform the modeling. Additionally, we found the AWS set up to be somewhat expensive. Although we did not actually pay for our usage, since Amazon gives students some credits to use, because we often left our server running even though no one was using it and initially set up the wrong type of instance, we would have spent what amounts to ~$40. In the future, if we were to use AWS again, we would make sure to

only set up the server when actually necessary and making sure the select the correct instance type for our problem.

Finally, our modeling approaches revealed that tree-based models perform much better than logistic regressions or neural networks. While possible that our neural network did not perform well due to our lack of experience working with them, we hypothesize that this finding reveals the nature of client relationships with Santander Bank. Many decisions in corporate finance are made using decision trees [10], and customer service particulars may be made this way too. Client opinions on the bank may heavily depend on their dealings with customer service, which could be less favorable if a particular client is delinquent. Nevertheless, we also belatedly found that XGBoost is heavily favored by "Kagglers," so another possibility is that because people spend so much time tuning XGBoost models, they perform so well in competitions. When undertaking future machine learning problems, we would like to spend more time exploring other methods.

6.0 Conclusion

The team was able to design a data mining model that scored highly on the private data set in the Santander Kaggle competition. This problem was important to the bank because knowing which customers were unsatisfied allows them to target these customers and improve their banking experience before the customer leaves. We initially had considerable difficulty getting started on the problem, since there were 370 features with unknown labels, but we used XGBoost and random forests to determine which features had a high predictive power and then focused on exploring those features. We ultimately settled on a four part approach that included data preprocessing, feature engineering, modeling, and ensembling. While creating our models, we decided to focus on tuning our random forests, XGBoosts, and keras neural networks. Tuning our parameters for these models was incredibly slow using our local machines, so we set up a Jupyter notebook server on AWS to increase the computation power. We used a blending technique with these models with ensembling to achieve our highest score. Our final score was 0.824485, which was less than 0.005 from the winning score. Through this strategy, we were

able to improve our private score to within five percent of the top score. This effort allowed us to learn more about classification problems and Kaggle competitions in general. In the future, we hope to make use of key preprocessing findings, such as scaling and removal of unimportant features, and focus on more interesting aspects of a machine learning problem such as engineered features and designing neural networks.

References

[1] "Santander customer satisfaction," in Kaggle Inc, 2016. [Online]. Available: https://www.kaggle.com/c/santander-customer-satisfaction.

[2] "The EY Customer Experience Series ™ — Cost of Complaining," Ernst & Young Global Limited, Australia, 2014. [Online]. Available: http://www.ey.com/Publication/vwLUAssets/EY_-_Customer_Experience_Series_-_Cost_of_complaining/$FILE/EY---Cost-of-Complaining-report.pdf.

[3] G. Winfrey, "The Cost of Unhappy Customers (Infographic)," Inc.com, Jul. 2014. [Online]. Available: http://www.inc.com/graham-winfrey/the-cost-of-unhappy-customers.html.

[4] "Median age / countries of the world,". [Online]. Available: http://world.bymap.org/MedianAge.html.

[5] "Var38 is mortgage value? - Santander Customer Satisfaction," in Kaggle Inc, 2016. [Online].Available:www.kaggle.com/c/santander-customer-satisfaction/forums/t/19895/var38-is-mortgage-value.

[6] "Var3 - Santander Customer Satisfaction," in Kaggle Inc, 2016. [Online]. Available: https://www.kaggle.com/c/santander-customer-satisfaction/forums/t/19367/var3?forumMessageId=111169.

[7] "3.2.4.3.1. Sklearn.ensemble.RandomForestClassifier — scikit-learn 0.18.1 documentation," in scikit-learn, 2010. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[8] diefimov, "The 3rd place solution for Santander Customer Satisfaction competition," GitHub, 2016. [Online]. Available: https://github.com/diefimov/santander_2016.

[9] "Kaggle Ensembling Guide," in MLwave, 2015. [Online]. Available: http://mlwave.com/kaggle-ensembling-guide/.

[10] A. Pinkasovitch, “Using Decision Trees In Finance,” *Investopedia*, 23-Aug-2015. [Online]. Available: http://www.investopedia.com/articles/financial-theory/11/decisions-trees-finance.asp.

Appendix A: Pointer to Code

https://github.com/asharma94/SantanderKaggle