Consumer Credit Risk Modeling

Bowen Baker
MIT Departments of Physics and EECS, 70 Amherst Street, Cambridge, MA 02142
(Dated: December 17, 2015)

We analyze and compare the performance of using Classification and Regression Trees (CARTs), Random Forests, and Logistic Regression to predict consumer credit delinquency. We also implement our own CART and Random forest algorithm and compare its results to the standard implementations.

I. Introduction

Consumer spending is one of the most important factors in the macroeconomic conditions and systemic risk of today's market. In 2013, there was approximately 3.09 trillion dollars of outstanding consumer credit in the United States, 856.8 billion of which was revolving consumer credit.[1] With so much revolving consumer credit and the increasing average charge-off rate, it is imperative that more sophisticated credit risk models be developed in order to attenuate the threat of further systemic dislocation.

We propose to use machine learning methods to analyze more subtle patterns in consumer expenditures, savings, and debt payments, than can the prominent models for consumer credit-default such as a logit, discriminant analysis, and credit scores. As the machine learning community is still unsure which models are best suited for credit card data, our study will aim to compare scalability, stability, and performance, of CARTs, Random Forests, and Logistic Regression.

Using proprietary data sets from a major commercial bank (from here on referred to as 'The Bank') we will train the model's forecasted scores with knowledge of whether or not customers did eventually default. Many lending entities stand to save significant amounts of money by employing these methods.

A. The Data

The data set we were given contained 50,000 unique customers with approximately 100,000 separate accounts and approximately 5,000,000 transactions over 13 months across these accounts. We have data given to The Bank by the Credit Bureau, along with general customer characteristics and account details. Each account has a marker indicating whether it had ever been delinquent and the date it first went delinquent. As is shown in Figure 1, most of the delinquencies are concentrated in 2013, which is convenient as we only have transactions data for 2013 and January of 2014.

In an attempt to conduct a causal experiment, our training set only has customers who went delinquent between May and August of 2013, and our validation set only has customers who went delinquent between September and December of 2013. Non-delinquent customers are then distributed randomly in a 70-30 split between training and testing sets. Once again, in an attempt to maintain causality, features for a customer were only created using data prior to that customer's window. For instance, a customer in the training set will only have features created from data that could be accessed prior to May, 2013.

We perform prediction on the customer level, which means we had to consolidate much of the data. A full feature list of the cleaned data can be found on the next page.

For all of the models run, our target variable was 0 if the customer was not delinquent for more than 31 days on a loan within the window. The target variable was set to 1 if the customer was delinquent within the window.

II. Success Metrics

Since the target variables are 0's or 1's, we can view the output of our predictors as a probability of delinquency. An intuitive metric to use, therefore, is the receiver operating characteristic (ROC) and the area under the ROC curve (AUC). The ROC is simply a plot of true positive rate (TPR) versus false positive rate (FPR) for a given probability cutoff. A completely random predictor will produce a straight line from $(0, 0)$ to $(1, 1)$ with an AUC of 0.5. A perfect predictor will produce a square ROC with an AUC of 1.

Feature List

Credit Bureau Data

Total inquiries
Total self-inquiries
Total co-borrower inquiries
Self-inquiries during last month
Self-inquireies during last 3 months
Credit Score
Event probability

Customer and Accounts Data

Number of accounts
Delinquent account flag
Average loan amount
Max loan amount
Most recent loan amount
Average loan term
Loan amount for each loan purpose
Loan amount for each product
Number of accounts for each loan purpose
Number of accounts for each product
Fraud flag
Suspicious flag
Revolving credit flag
Household number
Insured marker
Age group
Marital status
Occupation
Education

Loan Purposes

purchase
housing
facultative
instalment
commercial purpose
structured credit
reverse factoring
vehicle
employee credit card: 12 types

Transaction Data

Total cost
Max monthly cost
Min monthly cost
Total number of transactions
Max monthly number of transactions
Number of cities transactions occured
Max number of cities transactions occurred for 1 month
Number of countries transaction occured
Max number of countries transactions occurred for 1 month
Max spending, spending ratios, shock, moving average, and moving standard deviation for each purchase category

Purchase Categories

discount
restaurant
travel
clothing
entertainment
luxury
medical
food
utilities
vehicle
gas
online

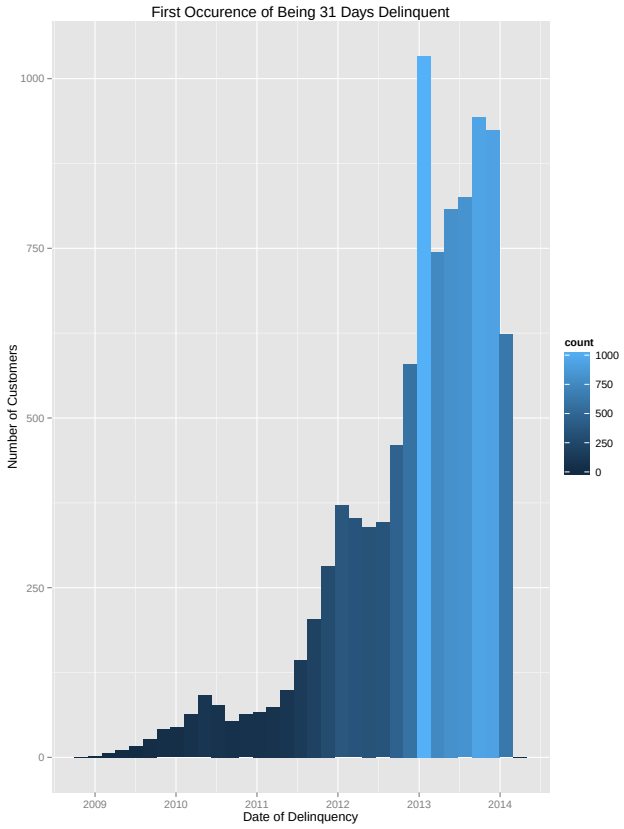

FIG. 1: Number of customers who went delinquent on a loan for 31 days or more for the first time.

More concretely, the AUC is probability that a randomly chosen positive sample (delinquent customer) will be ranked higher than a randomly chosen negative sample (non-delinquent customer).

Because models such as Credit Score and Logit produce probability like values over which one can compare customers, this is a natural metric to train on and compare with.

III. Classification and Regression Trees

Classification and regression trees recursively split the feature space into a binary tree structure. Because our outputs are real valued, we will just consider regression trees. Each node represents a rule consisting of the feature and value to split by. Thus if we split on feature j on value s at node N_m, we define the left and right children to be

$$\text{left}_{j,s}(N_m) = \{x \in N_m | x_j \geq s\} \tag{1}$$

$$\text{right}_{j,s}(N_m) = \{x \in N_m | x_j < s\} \tag{2}$$

Choosing the best order of features to split by for each node in the tree is NP-complete, so we will greedily make splits and minimize the error at each node. First we define the average output value at node m, to be

$$O_m = \text{mean}_{\{i|x^{(i)} \in N_m\}} y^{(i)} \tag{3}$$

and the error at node N_m to be

$$E(N_m) = \sum_{\{i|x^{(i)} \in N_m\}} (y^{(i)} - O_m)^2 \tag{4}$$

Thus, to find the greediest split at some node N_m we find the feature j and split value s that minimizes

$$E(\text{left}_{j,s}(N_m)) + E(\text{right}_{j,s}(N_m)) \tag{5}$$

Apart from this we can constrain the complexity of the tree by imposing limits on its maximum depth, minimum leaf size, and error reduction, i.e.

$$\Delta E = E(N_m) - \left(\frac{|\text{left}_{j,s}(N_m)|}{|N_m|} E(\text{left}_{j,s}(N_m)) + \frac{|\text{right}_{j,s}(N_m)|}{|N_m|} E(\text{right}_{j,s}(N_m)) \right) \tag{6}$$

The literature on CARTs posits that it is better to grow a relatively unconstrained tree and then prune it back after. The general procedure is to remove leaves from the tree such that each removal minimizes the error gain for the entire tree until the total error is within one standard error of the minimum.

The standard way to do a split on feature j when we have missing data is to introduce surrogate variables. Essentially, we try to find another feature k that is highly correlated to j so that we can split the remaining data points on k. By doing this we hope that splitting on k is similar to splitting on j. Usually one allows a few surrogate variables and then assigns points that have still not been split to the larger child.

We note here that from here on, we will define max node size to be k, max tree depth to be md, and minimum split error reduction to be mc.

A. Results

We used the **rpart** package in the statistics framework, R. The parameters that we chose to vary and do model selection on were minimum node size, maximum tree depth, and minimum split error reduction (given in Equation 6). We also compared this to a more simplistic custom CART we developed. We use the **optim** package to compute the optimal value for continuous variable splits based on the error function given in Equation 4. For categorical splits where the feature can be an element in the set S, we compute and compare the error for each split in the power set of S (excluding the empty and complete set). We, however, do not prune the tree nor use surrogate variables for missing features in our custom implementation. When there is a missing feature, we simply move the customer into the largest child (as is done when we run out of surrogate variables in the classic CART).

Figure 2 shows the ROC curves for both standard and custom implementations. Each plot shows the ROC for both the training and test data sets. The effect of not implementing surrogate variables and pruning is immediately obvious when comparing the standard and custom implementation curves. The custom ROC curve is much less steep in the low FPR region than the standard ROC curve. This can be more easily seen when comparing the area under each ROC curve as in Figure 3.

B. Discussion

One can see how pruning can reduce over fitting in the case of a low minimum node size. The standard implementation varies very little with this parameter, whereas the custom implementation varies wildly. However, the custom implementation does better with suboptimal values of minimum split error reduction and tree depth, which can be seen in Figure 3b. Overall, the optimal models from custom implementation was not overly worse than those of the standard implementation. The optimal models differed in AUC by only 0.04.

Another interesting point is that the custom implementation actually performs *better* on the test set than the training set in many cases. This may seem odd; however, the overall accuracy is better for the training set, which is consistent with our implementation. Furthermore as another sanity check, we see that in cases where the model is allowed to overfit the training set, such as when $k = 1$, the training set AUC is indeed higher than that of the test set. In addition, the overall test AUC is still lower than that of the standard implementation, so the results are not malformed in terms of performance.

To reduce the model space over which we searched, we only varied one parameter at a time. We used a standard set of parameters to train with when they were not specifically being varied. These were: $k = 10$, $md = 20$, $mc = 0.001$.

Using these results, we find that the optimal parameters and CART model is the **rpart** implementation with $k = 10$, $mc = 0.001$, and $md = 20$ (labels defined in figure).

IV. Random Forests

Random forests are simply bagged CARTs. On each split we randomly choose a subset of fn features to consider splitting on. We then grow n trees and average predictors, i.e.

$$f_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{n} f_b(x) \tag{7}$$

where $f_b(x)$ is the predictor for the b^{th} tree. Random forests tend to greatly reduce the variance in the predictor, but are in general less interpretable than the CART algorithm.

From here on, we define nt to be the number of trees and fn to be the number of features randomly chosen to considered at each split.

A. Results

The standard implementation in the **randomForest** package in R does not use surrogate variables to characterize missing features. To deal with this it replaces missing values with the average value of that feature for continuous features and the mode for discrete features. The parameters that we chose to vary and do model selection on were minimum node size, maximum tree depth, minimum split error reduction (given in Equation 6), number of trees, and number of features chosen randomly to be considered at each node.

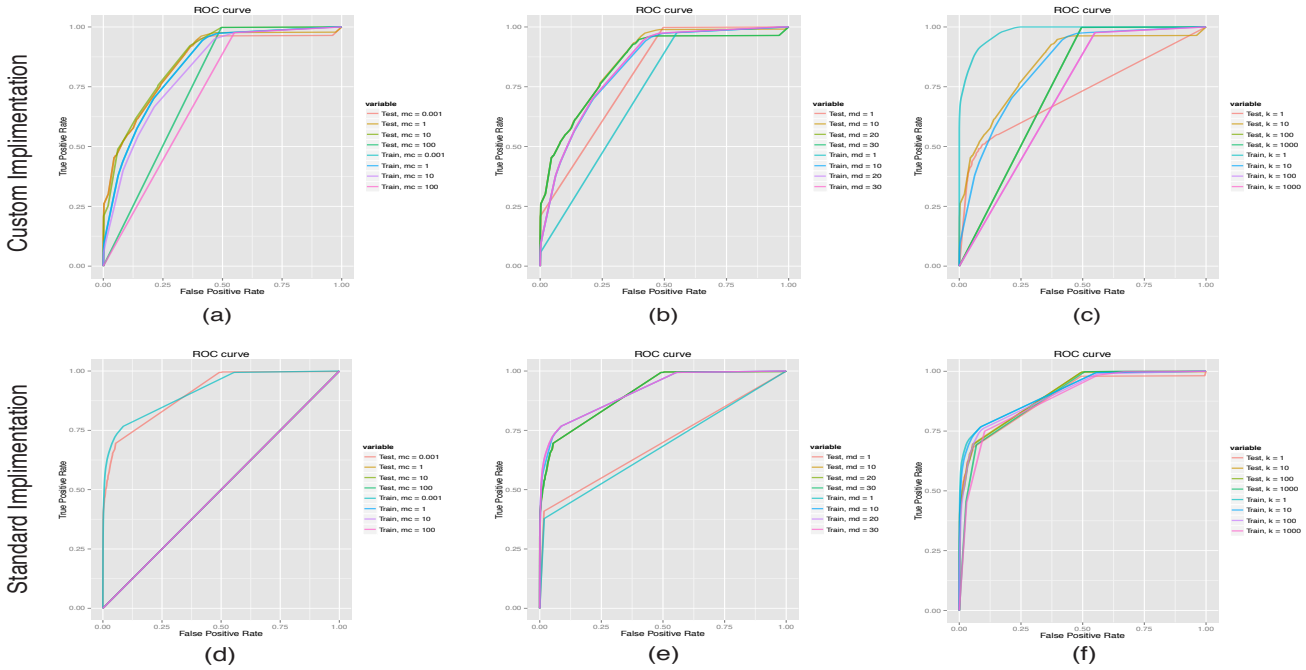

FIG. 2: Figures **a** and **d** show the ROC curve for varying values of minimum split error reduction (labeled mc). Figures **b** and **e** show the ROC curve for varying values of maximum tree depth (labeled md). Figures **b** and **e** show the ROC curve for varying values of maximum tree depth (labeled md). Figures **c** and **f** show the ROC curve for varying values of minimum node size (labeled k). The top row shows the performance of our custom implementation and the bottom row the standard implementation using **rpart**.

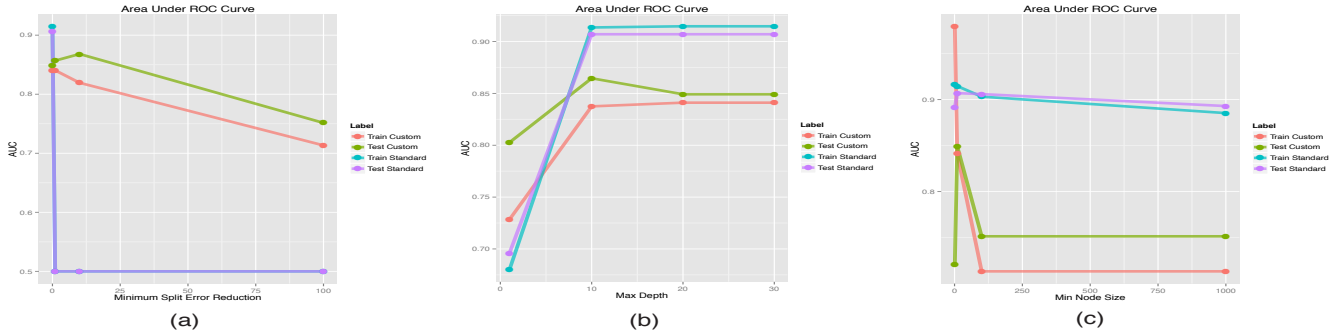

FIG. 3: Each figure shows the area under the ROC curve for both test and training sets and both standard and custom implementations. Figure **a** shows AUC versus minimum split error reduction, figure **b** shows AUC versus maximum tree depth, and figure **c** shows AUC versus minimum node size. The maximum AUC for the test occurred at $mc = 0.001$ with AUC=0.907, $md = 20$ with AUC=0.907, and $k = 10$ with AUC=0.907 (all with the standard implementation).

We also compared this to a more simplistic custom random forest we developed. We made a slight modification to the custom CART implementation presented in Section 3 to choose a random subset of features to consider at each split. We grew each tree on a random sample 80% the size of the original data set evenly distributed based on delinquency (we used the same percentage in the standard implementation). Figure 4 shows the ROC curves for the custom RF implementation. We see similar trends as with the custom CART model, though with increased AUC values on average. The optimal custom RF model had 0.025 greater AUC than the optimal custom CART model.

Figure 5 shows the ROC curves for the standard RF implementation.

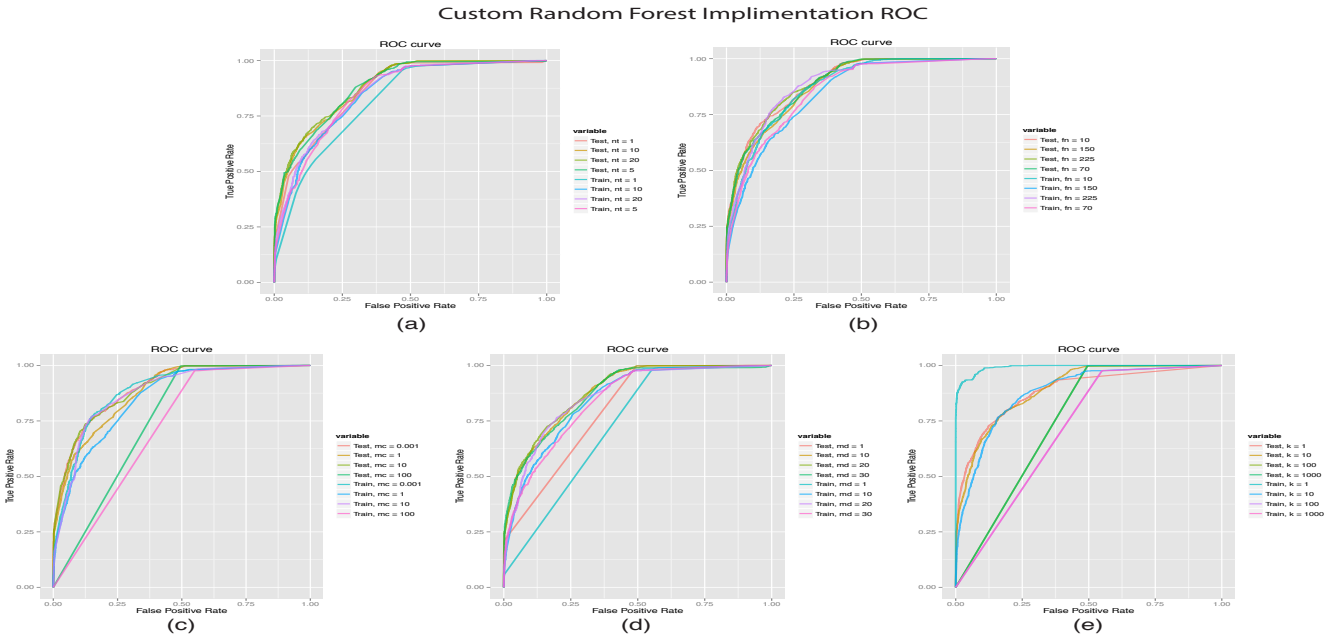

FIG. 4: Each figure the ROC curve for both test and training sets for the custom RF implimentation. Figure **a** shows the ROC for varying number of trees, figure **b** shows the ROC for varying number of features considered at each node, figure **c** shows the ROC for varying minimum split error reduction, figure **d** shows the ROC for varying maximum tree depth, and figure **e** shows the ROC for varying minimum node size. The maximum AUC for the test data occurred at $nt = 20$ with AUC=0.888, $fn = 225$ with AUC $= 0.895$, $mc = 0.001$ with AUC=0.897, $md = 20$ with AUC=0.886, and $k = 10$ with AUC=0.895.

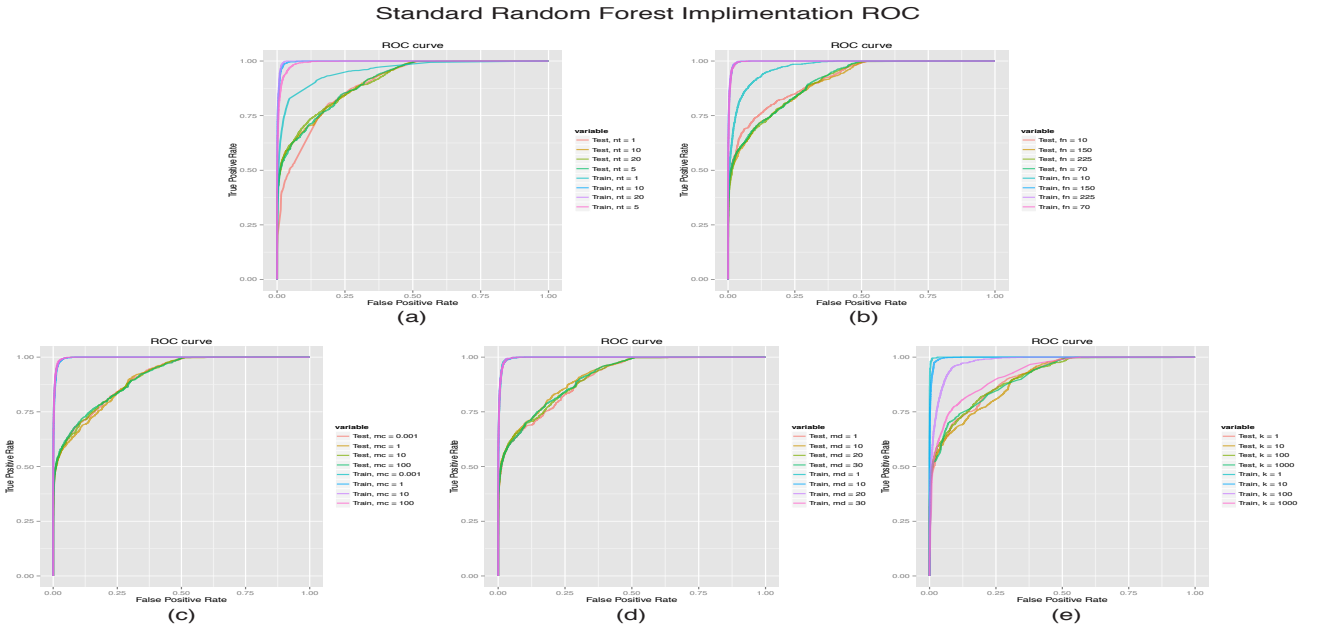

FIG. 5: Each figure the ROC curve for both test and training sets for the standard RF implimentation. Figure **a** shows the ROC for varying number of trees, figure **b** shows the ROC for varying number of features considered at each node, figure **c** shows the ROC for varying minimum split error reduction, figure **d** shows the ROC for varying maximum tree depth, and figure **e** shows the ROC for varying minimum node size. The maximum AUC for the test data occurred at $nt = 20$ with AUC=0.908, $fn = 70$ with AUC $= 0.907$, $mc = 100$ with AUC=0.908, $md = 30$ with AUC=0.908, and $k = 100$ with AUC=0.91.

B. Discussion

An interesting point to note are that many times the standard RF overfits the training data. This can be seen by both the very high AUC training values shown in Figure 6 and also in the almost completely square ROC training curves. This overtraining could be due to improper pruning or not utilizing surrogate variables during the splitting process. It also seems as though the complexity control parameter, mc, and the max tree depth, md, have very little affect on the standard RF implementation performance.

To reduce the model space of which we searched we only varied one parameter at a time. Our standard set of parameters were $k = 10$, $md = 20$, $mc = 0.001$, $nt = 10$, and $fn = 70$. After doing model selection we found that the optimal model was the standard RF with 20 trees, 225 features at each node, minimum split error reduction of 100, maximum depth of 30, and minimum node size of 100.

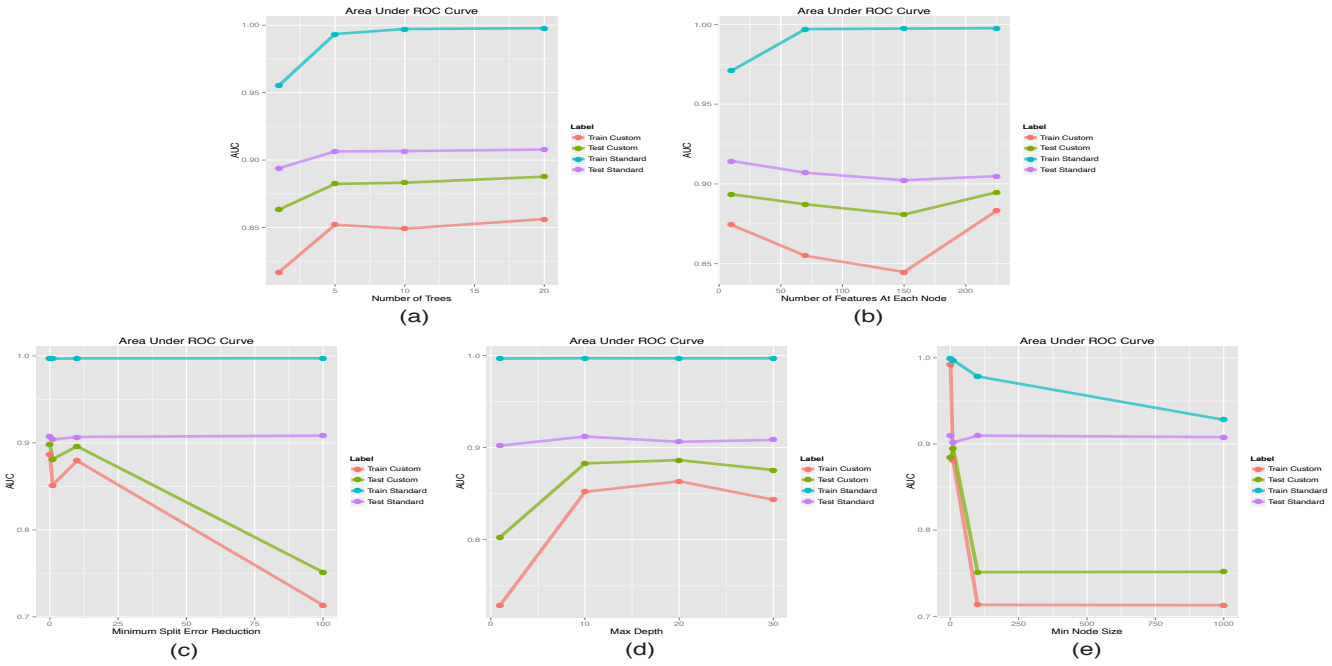

FIG. 6: Each figure shows the area under the ROC curve for both test and training sets and both standard and custom RF implementations. Figure **a** shows AUC versus number of trees, figure **b** shows AUC versus number of features considered at each node, figure **c** shows AUC versus minimum split error reduction, figure **d** shows AUC versus maximum tree depth, and figure **e** shows AUC versus minimum node size. The maximum AUC for the test occurred at $nt = 20$ with AUC=0.908, $fn = 225$ with AUC = 0.905, $mc = 100$ with AUC=0.908, $md = 30$ with AUC=0.908, and $k = 100$ with AUC=0.91.

In both the RFs and CARTs we found that the most significant features in predicting delinquency were housing loan amount, facultative loan amount, loan amounts for 3 of 26 distinct loan products (anonymized by The Bank), max monthly online purchase amount, household number, and revolving credit marker.

V. Logistic Regression

Logistic regression (LR) is a classic model used in this field, and as such provides a good baseline with which to compare our decision trees and random forests. LR uses the maximum likelihood of the sigmoid probability distribution to classify feature vectors. In the two class case, where the target variable $y \in \{0, 1\}$, the probability that $y_i = 1$ takes the form

$$\begin{aligned} p(y_i = 1|\mathbf{x_i}, \mathbf{w}) &= sigmoid(\mathbf{x_i} \cdot \mathbf{w} + w_0) \\ &= \frac{1}{1 + \exp\left(-(\mathbf{x_i} \cdot \mathbf{w} + w_0)\right)} \end{aligned}$$

To find the optimal weights, we want to minimize the negative log likelihood of the data given the weights, i.e.

$$\text{NLL}(W) = -\sum_{i=1}^{n}\Big(sigmoid\Big(W^T x^{(i)}\Big) - y^{(i)}\Big)x^{(i)} \tag{8}$$

To reduce the magnitude of the weights, we can introduce a penalty on the L2 norm of the weights into NLL(W). One can also introduce a penalty on the L1 norm of the weights, which is a "pointier" penalty that theoretically will set many weights to zero. However, it is much harder to converge on a solution with an L1 penalty, so we restricted ourselves to just doing model selection with penalties to the L2 norm.

A. Results

We used the R library **penalized** to perform LR. The results of varying the L2 norm can be seen in Figures 7. The best performance on the test set occurs when L2 penalty constant is 10 resulting in an AUC of 0.888.

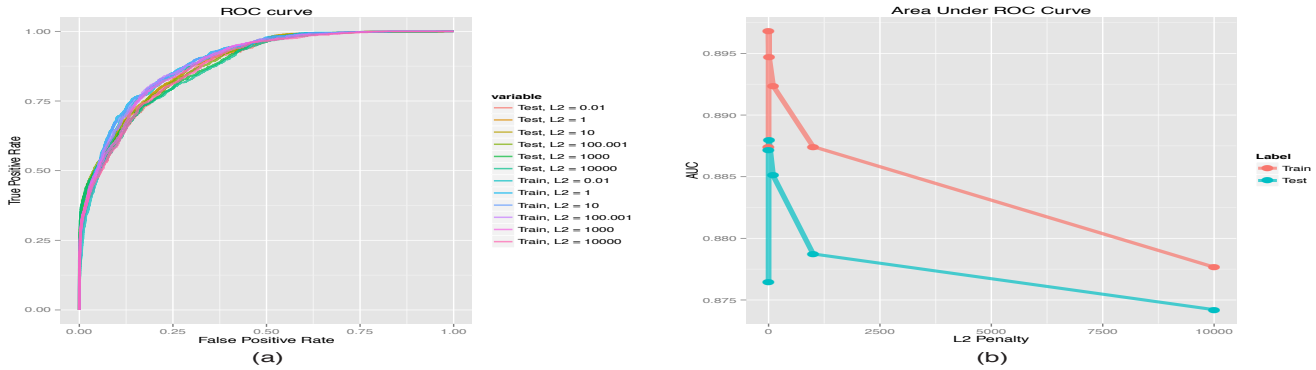

FIG. 7: Logit ROC, figure **a**, and AUC, figure **b**, curves for both training and testing sets. L2 refers to the scaling factor on the weight penalty. The best performance on the test set occurs when L2 penalty constant is 10 with resulting AUC of 0.888.

Interestingly, the highest weighted features from LR, i.e. the most significant predictors, were not exactly the same as those chosen by RF and CART. The most significant features for LR were total transaction cost, maximum monthly transaction cost, facultative loan amount, max loan amount, average loan amount, and loan amounts for other various products and purposes.

VI. Model Comparison

Now that we have found a set of relatively optimal models for CART, RF, and LR, we would like to compare their performance against each other and against another industry standard, the Credit Score. We received a set of credit scores from the The Bank, with no information on how they were actually calculated. The credit score ranges from 321 to 613 for both the training and test data sets, where a higher credit score presumably means a customer is less likely to default. To compare this score to CARTs, RFs, and LR, we associated a probability with the customer to be 1 minus the normalized credit score.

Figure 8 shows the performance of each model discussed compared with Credit Score. We were able to create an ROC curve for CScore using the probabilities described above. Each model did significantly better than CScore. The standard RF model had an AUC 0.269 greater than that of CScore. The standard RF model also did definitively better than the Logit model by an AUC of 0.039. These results are very compelling arguments for RF models as a good substitute for that current industry standards.

Overall, the custom implementations did not do better than the standard libraries, though this is to be expected as they were simpler models. They also took much longer to run, which we attribute to some inefficient loops in the code.

Another useful way to visualize these results is to actually compare the probability assigned by each models to the set of test customers. This can be see in Figure 9, which compares **randomForest**'s RF, **penalized**'s LR, and CScore.

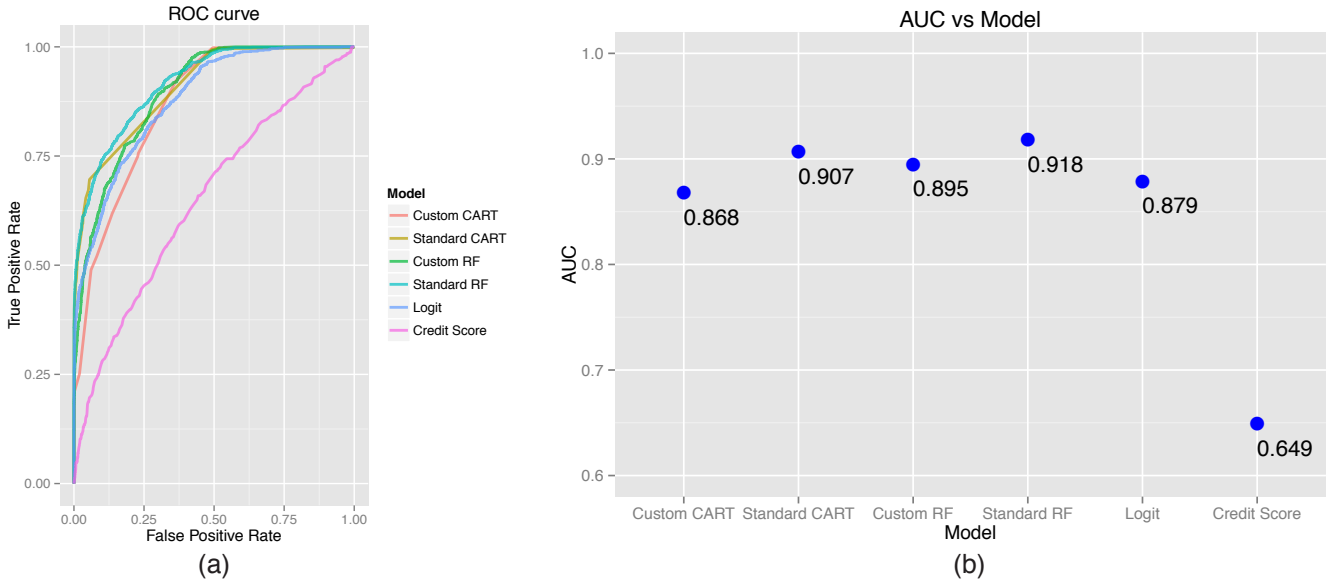

FIG. 8: CART (Custom and Standard), RF (Custom and Standard), LR, and Credit Score ROC curves in figure **a** and AUC in figure **b** for the test set.

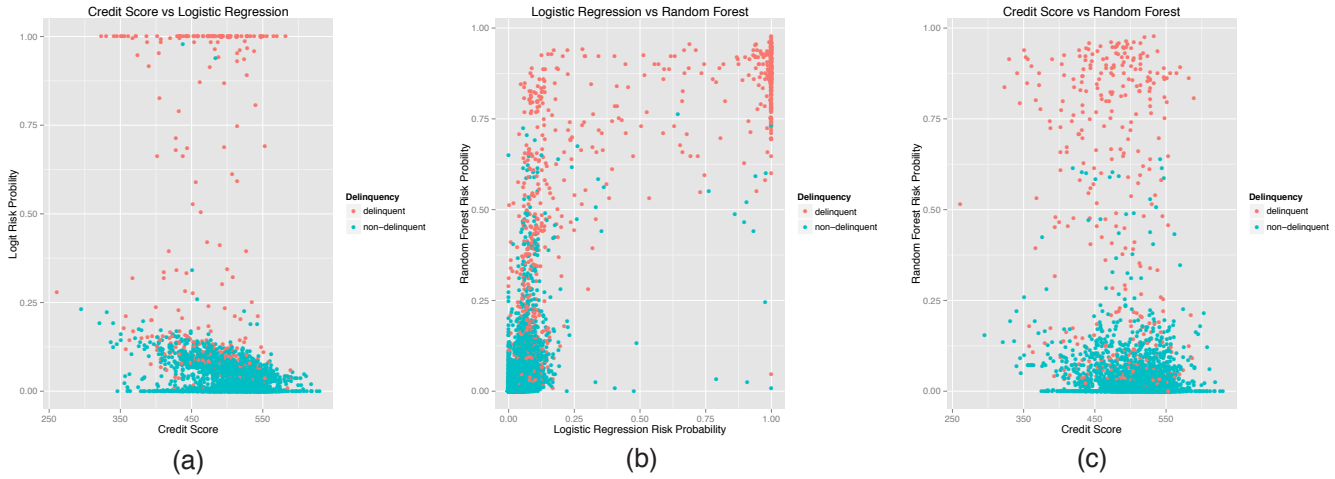

FIG. 9: Model probability comparison with delinquency labeled. Figure **a** shows CScore versus LR; figure **b** shows LR versus RF; figure **c** shows CScore versus RF.

VII. Further Work

There are many ways in which we could have improved this study which leaves much room to conduct future studies. The **randomForest** library does not introduce surrogate variables when the feature that is being split is missing. Many features in the data set had a significant amount of missing values, so we think we could stand to gain much by introducing this feature. Furthermore, we did a very simplistic model selection, only varying one parameter at a time. In the future we could consider searching a more complete model space with a decreasing granularity. We could also consider doing model selection on a different metric. For instance, one could do model selection on the aggregate expected income from loans..

Characterizing how much money could be saved by switching predictive models would further strengthen our claim. We would also like to broaden our study with data sets from other loaning entities, so that we may identify the most significant features in the space as a whole and not for just a single region or bank.

There are also other models we would like to test the performance of on consumer credit data, including Support

Vector Machines (even though they wouldn't produce probabilities), Artificial Neural Networks, and Deep Learning. However, even were these models to perform on par with or better than RFs, they are significantly harder to interpret.

VIII. Acknowledgements

I would like to acknowledge my collaborators, Andrew Lo, David Fagnan, Yuwei Zhang, and Danny Yuan. Everything presented in this paper was original work done by myself except the original data cleaning for which Yuwei, Danny, and David collaborated.

---

[1] Troyer, Matthias. Lecture. "Introduction to many-body quantum mechanics." Computational Quantum Mechanics. Institute For Theoretical Physics at Swiss Federal Institute of Technology Zurich. 2012. www.itp.phys.ethz.ch/education/fs12/cqp/chapter04.pdf

[2] Khandani, Amir E., Adlar J. Kim, and Andrew W. Lo. "Consumer credit-risk models via machine-learning algorithms." Journal of Banking & Finance 34 (2010): 2767-2787.

[3] Bellotti,Tony,andJonathanCrook."Support vector machines for credit scoring and discovery of significant features." Expert Systems with Applications 36.2 (2009): 3302-3308.

[4] Atiya, Amir F."Bankruptcy prediction for credit risk using neural networks: A survey and new results." Neural Networks, IEEE Transactions on 12.4 (2001): 929-935.

[5] Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.