An Analysis of Loan Prepayment Using Competing Risks Random Forests

by

Joel Therrien

B.Sc., University of British Columbia (Okanagan), 2015

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Statistics and Actuarial Science
Faculty of Science

SIMON FRASER UNIVERSITY
Fall 2019

Approval

Name:	**Joel Therrien**
Degree:	**Master of Science (Statistics)**
Title:	**An Analysis of Loan Prepayment Using Competing Risks Random Forests**
Examining Committee:	**Chair:** Jinko Graham Professor
	Jiguo Cao Senior Supervisor Associate Professor
	Joan Hu Supervisor Professor
	Tom Loughin Internal Examiner Professor
Date Defended:	**November 27, 2019**

Abstract

Loan prepayment is a large cause of loss to financial institutions when they issue installment loans, and has not been well studied with respect to predicting it for individual borrowers. Using a dataset of competing risks times for loan termination, competing risks random forests were used as a non-parametric approach for identifying useful predictors, and for finding a tuned model that demonstrated that loan prepayment can be predicted on an individual borrower basis. In addition, a new software package we developed, largeRCRF, is introduced and evaluated for the purpose of training competing risks random forests on large scale datasets. This research is a firm first step for financial institutions to reduce their prepayment rates and increase their margins.

Keywords: installment loans; prepayment; competing risks; random forests; largeRCRF

Acknowledgements

Thank you to my supervisor, Dr. Jiguo Cao, for guiding me and supporting me throughout this research. Your help was critical and is much appreciated.

Thank to you Dr. Joan Hu for sitting on my committee, and for teaching me the survival analysis that set me on the path for this research topic.

Thank you to Dr. Tom Loughin for sitting on the committee and teaching me different machine learning algorithms and how to properly use them. Your teachings on random forests in particular were very helpful for this research topic.

Thank you to Dr. Jinko Graham for chairing my committee and teaching me statistical computing.

Special thanks to all the friends I've made throughout my time at SFU. You made this period of my life the best time of my life. I hope that the friendships we've grown will still thrive even as we move into the next period of our lives.

Thank you to Dr. Hemant Ishwaran for answering my many questions about his package, randomForestSRC.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Before a financial institution makes an installment loan[1] they first collect information about the potential borrower. This information is then analyzed using extremely accurate models (usually provided as a credit score) to predict the likelihood that the borrower will default on the loan. That likelihood is then used to determine whether to grant the loan, and if so, at what interest rate. Higher interest rates are used to offset the higher risk of expected default loss, so that on average the financial institution will still profit among every 'tranche' of default risk.

This loss due to default is the largest cause of loss, but it isn't the only one. Another type of loss occurs when a borrower prepays their loan by making larger than scheduled monthly payments (possibly paying the entire remaining balance at once), causing the financial institution to lose out on the interest that would have otherwise accrued; revenue that is used to offset the defaults of others and also to make up for the fixed overhead costs of making the loan. Most financial institutions are aware of and track overall prepayment rates, and adjust overall policies to account for it, but they apply these policies equally to all customers instead of predicting & targeting which customers are more likely to prepay. This means that potential borrowers that are unlikely to prepay their loans but want the option to may be discouraged from some financial institutions because that institution's policies on prepayment are too strict for them. If that institution could instead assign a measure of prepayment risk to each borrower, then they could only assign these discouraging

[1] An installment loan is a loan with a fixed monthly payment for a prespecified period of time; unlike a credit card or line of credit where you can draw and repay balance at will.

policies only to those likely to prepay, gaining new customers and reducing prepayment. Alternatively, they could offer discounted interest rates to someone likely to prepay if they'd agree to policies that restricted prepayment.

The goal of this analysis is to demonstrate that prepayment risk can be calculated, that it can be used to strongly differentiate borrowers that previously appeared to be identical, that the differences in expected loss to due prepayment are great enough to warrant further study into targeted prepayment policies, and to identify which currently measured credit measures are good indicators of prepayment risk. The analysis unfortunately cannot evaluate whether selective penalties or incentives to modify customer behaviour could save the financial institution money, as the data used for this analysis is strictly observational.

1.1 The Data

Before we can proceed, we must first talk about the dataset available and its limitations. The dataset covers $1,422,013$ three year loans made by Lending Club, an online peer-to-peer lender based in the United States. It contains 75 possible predictors and covers loans made from 2007 to 2018. Not all loans have had their full three years to mature, so there is censoring (about 39% of the loans are censored). For the response on every loan we know the times of loan termination that represent exactly one of:

- How long the loan survived until the borrower defaulted.
- How long the loan survived until the borrower paid it back.
- A lower bound on how long the loan survived.

The third option represents right-censoring, which only occurs due to when the data was collected. Since we know when the data was collected and the start time of each loan, this also gives us the censoring times for every loan, even if it terminated prior to this censor time. This allows us to trivially estimate the censoring distribution which some models may use (just use an empirical CDF).

What we *don't* know are the times or amounts of specific loan payments; only the time and cause of the final loan termination. This means we can't model something like the

average dollar amount of loan prepayment in a given month of the loan, and are restricted to only using methods that use the times described above. Given that we have two competing events (repayment and default) and only the loan survival times, we'll treat this as a competing risks problem.

1.1.1 Predictor Variables

This section will describe categories of predictors and point out some notable ones. For a complete listing, see Appendix B.

The predictors themselves can be categorized into four groups:

- Measurements from the credit report. Notable examples include:
  - `open_rv_12m` & `open_rv_24m`: Number of revolving trades[2] opened in the last 12 & 24 months.
  - `num_rev_accts`: The total number of revolving trades.
  - `revol_bal`: The balance of all revolving trades.
  - `total_bal_il`: The balance of all installment trades.
  - `delinq_amnt`: The total delinquent amount across all trades.
- Attributes of the loan. Notable examples include:
  - `loan_amnt`: The loan amount.
  - `int_rate`: The interest rate.
  - `installment`: The monthly payment.
- Information provided by the borrower. Notable examples include:
  - `annual_inc`: Borrower's self-reported income.
  - `home_ownership`: Whether they rent or own their home (or other).
  - `emp_length`: How long they've worked at their current employer.

[2]A revolving trade is a credit product where the borrower can draw and repay the balanc at will; such as a credit card or line of credit.

  - `purpose`: How the borrower plans on spending the loan (factor variable).

- Summary information derived from the above measures. Notable examples include:
  - `sub_grade_numeric`: Lending Club's internal overall default risk score.
  - `dti`: The borrower's debt to income ratio under the loan.

One issue among the predictor variables is that many of them were only introduced in the middle of the dataset (two groups, one midway through 2012, other at end of 2015). While the model we'll use can handle NA data, this will interfere with one of our goals of identifying which predictors are useful (as a financial institution starting research today would have access to all the predictors). We'll discuss this again in Chapter 4, but we will essentially repeat variable selection twice; once on the full dataset, and another only on loans issued after 2015.

1.2 Competing Risks Models

In competing risks models, we are concerned with the survival time of a subject that can terminate via one of several exclusive events (competing risks), or may be censored because they were not observed to terminate. We are often interested in estimating the cumulative incidence function (CIF) for one of the possible events for a subject; the probability that the survival time (T) has occurred by some time t and that the event that occurred (Δ) was the one that was specified (j).

$$\text{CIF}_j(t) = P(T \leqslant t \cap \Delta = j) \tag{1.1}$$

To do this we make the following assumptions and define the following notation for some subject i:

- Let T_i be a random variable to denote the survival time for subject i. T_i is not directly observed. For our dataset, we measure it in months from the loan issue date.
- Let C_i be a random variable to denote the censoring time for subject i. C_i is often not observed, but is fully available for this dataset. It is assumed that $\forall i$ T_i and C_i

are independent of each other. This independence assumption is reasonable for our dataset as the censoring times are caused purely by the date of data collection. We also measure C_i in terms of months from the loan issue date.

- Let J represent the number of possible competing events that may occur. For this dataset there are only two events - repayment and default.

- Let Δ_i be an integer ranging from 1 to J represent the cause for why subject i terminated at time T_i. Δ_i is not directly observed.

- Let $\tilde{T}_i = \min(T_i, C_i)$ and $\delta_i = \Delta_i \cdot I(T_i \leqslant C_i)$; these are the actual times and event indicators observed for subject i. We know that for a particular subject that if $\delta_i = 0$ then $\tilde{T}_i = C_i$, and that if $\delta_i \neq 0$ then $\delta_i = \Delta_i$ and $\tilde{T}_i = T_i$.

With these assumptions we can easily estimate the CIF for a group of data; the Aalen and Johansen estimator (Aalen and Johansen 1978) is commonly used. The challenge occurs when you want to estimate the CIF conditional on a group of predictors; in our case to estimate the CIF given a set of loan predictors.

Traditionally, survival analysis makes heavy use of parametric or semi-parametric models such as the Cox Proportional Hazards model, which have been applied for a particular event by treating other events as censoring. Other models have been designed particularly for competing risks data, such as the Proportional Subdistribution Hazards Model (Fine and Gray 1999).

These models unfortunately tend to have fairly strict assumptions, such as assuming both proportional hazards, and then usually a linear form for the predictors. These strict assumptions are necessary and useful in smaller datasets where reducing a lot of variance with a little bias is a good trade. However, our dataset has about 1.4 million rows - using assumptions that add even a small amount of bias won't reduce variability in a meaningful way, and we have no reason to believe that making one of these strict assumptions would only add a small amount of bias.

Instead, we'll look to use a flexible non-parametric model that is light on assumptions. Most machine learning algorithms have not been extended to competing risks data, but

random forests have. We'll therefore choose to use competing risks random forests to analyze this dataset.

1.3 Outline

In Chapter 2 we'll discuss competing risks random forests. We'll start by going over the theory of random forests and how Ishwaran et al. 2014 adapted them to competing risks data. Afterward, we'll briefly discuss the one pre-existing software package able to train these forests, randomForestSRC, and how it isn't suitable for datasets of this size.

Chapter 3 will then introduce the software package we wrote for analyzing this particular dataset, largeRCRF, and evaluate its performance relative to randomForestSRC.

Chapter 4 will return to the application, where the data is described in greater detail, variable selection is performed, and forests are trained, and results are presented.

Finally, Chapter 5 will conclude this thesis, as well as go over areas of this research that we'd like to have improved on had we the time or resources, and could be looked at in the future.

Chapter 2

Competing Risks Random Forests

2.1 Theory

Random forests (Breiman 2001; Izenman 2008; Hastie, Tibshirani, and Friedman 2001) are a sequence of binary decision trees trained on bootstrap resamples of the original data. They were originally developed for classification and regression problems, but extensions have been developed for other responses, such as competing risks. For now, assume we have some generic response of unknown type and multiple predictor variables, and assume we have tuning parameters `ntree`, `nodeSize`, `maxNodeDepth`, `numberOfSplits`, and `mtry`.

To train the forest, we bootstrap the training data `ntree` times, and on each bootstrap re-sample we run `processNode(data, depth=0)`, which recursively grows the entire tree for that bootstrap re-sample. A simplified version of `processNode` without edge cases is described in Algorithm 1.

When we make predictions using the forest for a given row, we navigate the split nodes using the recorded split rule down to the appropriate terminal node on every tree. We then combine those terminal nodes together across the forest for our prediction.

These basic steps essentially define the random forest algorithm regardless of response type. There are just three details that then need to be specified for each type of random forest:

- How to calculate a score to find the best possible split.
- How to combine the responses to form a terminal node.
- How to combine the terminal nodes in the forest to make a prediction.

```
Function processNode(data, depth)
    if size of data ⩾ 2 × nodeSize or depth < maxNodeDepth then
        /* Split node                                                    */
        randomly select mtry predictor variables;
        bestSplit ← NULL;
        for each selected predictor do
            if numberOfSplits = 0 then
                Evaluate every possible split on predictor, replacing bestSplit with any
                  better splits;
            else
                Evaluate numberOfSplits random splits on predictor, replacing
                  bestSplit with any better splits;
            end
        end
        Using bestSplit, produce splitDataLeft and splitDataRight;
        childNodeLeft ← processNode(splitDataLeft, depth + 1);
        childNodeRight ← processNode(splitDataRight, depth + 1);
        return a split node with bestSplit, childNodeLeft, and childNodeRight;
    else
        /* Won't split; return a terminal node                           */
        return an average of data;
    end
```

Algorithm 1: How a node is created. A tree is created by running `processNode` on the root node's data, which recursively creates all of the child nodes under it.

In order to train a random competing risks forest, we need to specify these details. We cover the theory that Ishwaran et al. 2014 developed, with minor differences highlighted.

Let us restrict ourselves to a node in a decision tree that is currently being trained, and suppose that there are m observations in this node; the data we then work with is $\{(\tilde{T}_i, \delta_i, X_i) | i = 1, \ldots, m\}$.

Definitions:

- Let $Y(t) = \sum_{i=1}^{m} I(\tilde{T}_i \geqslant t)$, which gives the number of individuals at risk at time t.
- Define $d_j(t) = \sum_{i=1}^{m} I(\tilde{T}_i = t \cap \delta_i = j)$ which gives the number of events of type j that occur at time t.
- For discussing splitting rules, suppose that a potential split produces a 'left' and a 'right' group. Denote L and R to be the set of observation indices of the parent node that would get assigned to each group. Define $Y_L(t)$, $Y_R(t)$, $d_{Lj}(t)$, $d_{Rj}(t)$ for each side as above, i.e.,
  - $Y_L(t) = \sum_{i \in L} I(\tilde{T}_i \geqslant t)$; $Y_R(t) = \sum_{i \in R} I(\tilde{T}_i \geqslant t)$
  - $d_{Lj}(t) = \sum_{i \in L} I(\tilde{T}_i = t \cap \delta_i = j)$; $d_{Rj}(t) = \sum_{i \in R} I(\tilde{T}_i = t \cap \delta_i = j)$
- Index unique observed times $v_1, v_2, \ldots v_K \in \{\tilde{T}_i | \delta_i \neq 0\}$ as an increasing sequence.

2.1.1 Splitting Rules

There are two choices for splitting rules. The first is based on the generalized log-rank test statistic. For a fixed event type j, the log-rank test corresponds to a test of the null hypothesis $H_0 : \alpha_{Lj}(t) = \alpha_{Rj}(t) \quad \forall t \leqslant \max(\tilde{T}_i | \delta_i \neq 0)$, where $\alpha_{Lj}(t)$ and $\alpha_{Rj}(t)$ are the cause-specific hazard rates. The cause-specific log-rank test statistic is calculated as follows:

$$L_j^{\mathrm{LR}} = \frac{1}{\hat{\sigma}_j^{\mathrm{LR}}} \sum_{k=1}^{K} \left(d_{Lj}(v_k) - \frac{d_j(v_k) Y_L(v_k)}{Y(v_k)} \right) \tag{2.1}$$

where

$$(\hat{\sigma}_j^{\mathrm{LR}})^2 = \sum_{k=1}^{K} \frac{Y_L(v_k)}{Y(v_k)} \left(1 - \frac{Y_L(v_k)}{Y(v_k)} \right) \left(\frac{Y(v_k) - d_j(v_k)}{Y(v_k) - 1} \right) \tag{2.2}$$

When finding the best split, we try to find the split that maximizes $|L_j^{\mathrm{LR}}|$. This test is restricted to only event j, but it can be calculated and combined for multiple events, where we will try to maximize $|L^{\mathrm{LR}}|$ defined as

$$L^{\mathrm{LR}} = \frac{\sum_{j=1}^{\mathrm{J}} \hat{\sigma}_j^{\mathrm{LR}} L_j^{\mathrm{LR}}}{\sqrt{\sum_{j=1}^{\mathrm{J}} (\hat{\sigma}_j^{\mathrm{LR}})^2}}. \tag{2.3}$$

It should be noted that Ishwaran et al. 2014 contains a typo (confirmed in correspondence with Ishwaran), where they write Equation (2.3) with $(\hat{\sigma}_j^{\mathrm{LR}})^2$ in the numerator instead of $\hat{\sigma}_j^{\mathrm{LR}}$. This typo is only present in their papers; randomForestSRC correctly implements the splitting rule.

Ishwaran et al. 2014 also described a variant of this splitting rule that better handles competing risks data by instead calculating Gray's test statistic (Ishwaran et al. 2014, Section 3.3.2). This can be accomplished by reusing Equations (2.1), (2.2), (2.3), while replacing $Y(t)$ and $Y_L(t)$ with a cause-specific version when the censoring times are fully known.

$$Y_j^*(t) = \sum_{i=1}^{m} I(\tilde{T}_i \geqslant t \cup (\tilde{T}_i < t \cap \delta_i \neq j \cap C_i > t)) \tag{2.4}$$

Note that randomForestSRC approximates (2.4) by using the largest observed time instead of the actual censor times (even if the true censor times are available), while our package (to be introduced in Chapter 3) explicitly requires that all censor times be provided to use this test, and doesn't support any approximate version.

2.1.2 Creating Terminal Nodes

For generating a terminal node, we assume that the data has been split enough that it is approximately homogeneous enough to simply combine into estimates of the overall survival function, estimates of the cumulative incidence functions (CIFs), and estimates of the cumulative hazard functions using, respectively, the Kaplan-Meier estimator (Kaplan and Meier 1958), the Aalen and Johansen estimator (Aalen and Johansen 1978), and the Nelson-Aalen

estimator (Nelson 1972; Aalen 1978), which are expressed as

$$\begin{aligned}
\hat{S}(t) = \hat{P}(T \geqslant t) &= \prod_{i=1}^{m(t)} \left(1 - \frac{\sum_{j=1}^{J} d_j(v_k)}{Y(v_k)}\right) \\
\hat{F}_j(t) = \hat{P}(T \geqslant t \cap \Delta = j) &= \sum_{k=1}^{m(t)} \frac{\hat{S}(v_{k-1}) d_j(v_k)}{Y(v_k)} \\
\hat{H}_j(t) &= \sum_{k=1}^{m(t)} \frac{d_j(v_k)}{Y(v_k)}
\end{aligned}$$

where $m(t) = \max(k | v_k \leqslant t)$.

2.1.3 Averaging Terminal Nodes

To make a prediction, we simply average the above functions across terminal nodes at each time t. To be specific, assume we have M trees and are making a prediction for some predictors X. For each tree k we follow the split nodes according to X until we reach a terminal node, yielding functions $\hat{S}_k(t|X)$, $\hat{F}_{jk}(t|X)$, $\hat{H}_{jk}(t|X)$ for $\forall j = 1 \ldots J$. Then $\forall j = 1 \ldots J$ we define the overall functions that we return as:

$$\begin{aligned}
\hat{S}(t|X) &= \frac{1}{M} \sum_{k=1}^{M} \hat{S}_k(t|X) \\
\hat{F}_j(t|X) &= \frac{1}{M} \sum_{k=1}^{M} \hat{F}_{jk}(t|X) \\
\hat{H}_j(t|X) &= \frac{1}{M} \sum_{k=1}^{M} \hat{H}_{jk}(t|X)
\end{aligned}$$

2.2 randomForestSRC

Ishwaran, who extended random forests to competing risks, developed a software package called randomForestSRC (H. Ishwaran and Kogalur 2018) that can, among types of responses, train competing risks random forests. At the beginning of this analysis, this was the only software package available to fit this type of model.

Unfortunately, this package has serious performance issues with large datasets. When we later compare it against our package, we'll see that it takes almost an hour to train and predict for a relatively small forest of 100 trees on a dataset of $n = 10,000$ and $p = 3$, and

about 4.5 days when n is increased to 100,000. Even if the package somehow took only 4.5 days for our dataset ($n = 1,422,013$), it would still be completely impractical to use given that we'd still need to train many different forests for both tuning and variable selection. This meant that in order to proceed with this analysis, we needed to write our own package.

Chapter 3

largeRCRF

largeRCRF is an R (R Core Team 2018) package we developed for training competing risks random forests. It's composed of two components - the logic for training and using forests is contained in a Java code base, while the user interface is written in R. This gives a performance benefit, as Java is a much faster language than pure R code.

The package takes advantage of object-oriented design to be extensible; the three details in Section 2.1 that a random forest implementation needs to specify (how to score a split, how to create a terminal node, and how to combine terminal nodes) is represented using interchangeable 'objects'. The rest of the algorithm is kept generic. Thus, to extend support for a different type of response only a small bit of code for those three tasks needs to be specified to 'fill in the blanks' of the generic algorithm. While for this project it was only necessary to add support for competing risks data, we kept largeRCRF extensible so that other researchers can easily extend random forests for a new type of data. Internally in the code, this extensibility is demonstrated with support for regression.

Before we can use it for this project, we first have to evaluate its accuracy and performance for competing risks data. We'll do so by comparing its accuracy against randomForestSRC on smaller datasets that both packages can handle.

3.1 Simulation Studies

We run two simulation studies. The first simulation is used to verify the accuracy of largeRCRF by comparing it with randomForestSRC at a small sized dataset ($n = 1000$). The

second simulation is used to measure the time performance of both packages at varying simulation sizes.

For the first simulation, we tune models for both packages on the data using the naive concordance error as described in Wolbers et al. 2014, Section 3.2 on a validation dataset, and then calculate an estimate of the integrated squared error on the CIFs using a final test dataset for the tuned models. We repeat this procedure 10 times for a sample size of $n = 1000$. We do not tune the number of trees and fix it at 100.

For the second simulation, we fix the tuning parameters and train both packages on simulated datasets of varying sizes, recording for each package the sum of the time used for training and the time used for making predictions on the validation dataset.

One important note; as of version 2.9.0 randomForestSRC adjusted their default algorithm to sample without replacement 63.2% of the data for each tree, instead of using bootstrapping resampling normally associated with random forests. For these simulations we manually keep randomForestSRC at its previous default of bootstrap resampling.

3.1.1 Generating Data

Every training, validation, and test dataset in both simulations are generated in the same way. We first generate covariate vectors $X_1, X_2, X_3 \overset{iid}{\sim} N(0, 1)$. We subset the space created by $X_1, X_2,$ and X_3 into 5 regions (see Table 3.1). In each region, for each response to generate, we randomly select which competing risks event should occur based on prespecified probabilities, and then depending on the event, generate the competing risks time according to a prespecified distribution. By specifying the probabilities of each event and the distribution used to generate each event's time, we then know the true population cumulative incidence function (CIF) for any event j (see (3.1)), which we can compare against the estimates for our error measure.

$$\text{CIF}_j(t|X) = P(T \leqslant t \cap \Delta = j|X) = P(T \leqslant t|\Delta = j, X)P(\Delta = j|X) \tag{3.1}$$

Set	Conditions	$P(\Delta = 1\|X)$	$P(\Delta = 2\|X)$	Dist. of $T\|\Delta = 1, X$	Dist. of $T\|\Delta = 2, X$
1	$X_1 < 0$ & $X_2 < 0$ & $X_3 < 1$	0.4	0.6	Weibull($k = 5, \lambda = 6$)	Exp(1)
2	$X_1 < 0$ & $X_2 \geqslant 0$ & $X_3 < 1$	0.1	0.9	Lognormal$(0, 1)$	Truncated positive $N(0, 1)$
3	$X_1 \geqslant 0$ & $X_2 < 0$ & $X_3 < 1$	0.7	0.3	Exp(1)	Exp(1) offset by $+1$
4	$X_1 \geqslant 0$ & $X_2 \geqslant 0$ & $X_3 < 1$	0.6	0.4	Weibull($k = 1, \lambda = 2$)	Lognormal$(0, 1)$
5	$X_3 \geqslant 1$	0.5	0.5	Exp($\lambda = 10$) offset by $+2$	Exp($\lambda = \frac{1}{4}$)

Table 3.1: Overview of the distributions used to generate the simulated datasets.

Table 3.1 contains details on these weights and distributions. We also let censor times $C \sim \text{Exp}(\lambda = 1/15)$, regardless of the covariates. We only allow largeRCRF and randomForestSRC access to $((\tilde{T}, \delta), X_1, X_2, X_3)$ where $\tilde{T}$ and δ are defined as in Section 1.2.

3.1.2 The First Simulation Study - Assessing Accuracy

Tuning

When we tune both packages for each of the 10 times, we want to maximize the concordance index error calculated on the validation dataset, except that we have quantities for each event to consider that aren't necessarily on the same scale. Let J be the number of events to consider (J=2). Let $\hat{C}_{ijk}$ be the concordance index as defined in Wolbers et al. 2014, Section 3.2 for tuning parameter combination i, event j, package k; evaluated with the predicted mortality of each validation observation being the integral of the estimated CIF for event j for each of the observations from time 0 to the largest non-censored event time in the training dataset. $\hat{C}_{ijk}$ can be thought of as an estimate of the probability that the forest associated with i and k correctly predicts the ordering of two random event times for event j. More specifically, it looks at the proportion of pairs where the shorter event time had a correctly predicted larger integrated CIF (a larger integrated CIF suggests that the event time occured earlier), weighted and adjusted to handle the competing risks and censoring.

We then calculate $\hat{C}^*_{ijk}$, the centered and scaled $\hat{C}_{ijk}$ according to $\underset{i}{\text{mean}}(\hat{C}_{ijk})$ and $\underset{i}{\text{sd}}(\hat{C}_{ijk})$, respectively, since we'd like to tune according to both events equally.

$$\hat{C}^*_{ijk} = \frac{\hat{C}_{ijk} - \underset{i}{\text{mean}}(\hat{C}_{ijk})}{\underset{i}{\text{sd}}(\hat{C}_{ijk})}$$

Then let the concordance error that we finally use to tune be

$$\epsilon_{ik}^{\text{Concordance}} = -\frac{1}{\text{J}} \sum_{j=1}^{\text{J}} \hat{C}^*_{ijk} \tag{3.2}$$

We take the negative so that our intuition of minimizing error remains. For each package k, we then select the tuning parameters associated with the i that minimized $\epsilon_{ik}^{\text{Concordance}}$, which we store for later use to calculate the CIF errors.

The tuning parameters we consider are a grid formed by:

- Number of splits tried (`nsplit`): [1, 50, 100, 250, 1000]
- Node size (`nodeSize`): [1, 10, 50, 100, 250, 500]
- Number of covariates tried at each split (`mtry`): [1, 3]

The splitting rule used for both packages is the composite log-rank rule defined in Equation (2.3) as it would not be equivalent to compare the two different Gray test implementations (since randomForestSRC uses an approximation and largeRCRF requires that censor times be available).

CIF Error

For the 10 selected models for each package we then calculate the error on the estimates of the cumulative incidence functions. For every observation i and event j in the test dataset we determine the true CIF according to Equation (3.1) and Table 3.1, which we denote as CIF_{ij}. Using the selected model trained on the training set we then predict the corresponding estimate of the CIF for observation i and event j which we denote as $\widehat{\text{CIF}}_{ij}$.

Let $\tau = 20$. We let τ be a constant number so that errors between training sets are comparable; 20 is otherwise arbitrary except that it encompasses most of the response

times. We then calculate an error for each observation i as follows:

$$\epsilon_{ij}^{\text{CIF}} = \sqrt{\int_0^\tau \left(\text{CIF}_{ij}(t) - \widehat{\text{CIF}}_{ij}(t)\right)^2 \text{d}t}$$

We then average over the test dataset to calculate the mean error for event j.

$$\epsilon_{\cdot j}^{\text{CIF}} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \epsilon_{ij}^{\text{CIF}}$$

Finally we combine the CIF errors for our events together by averaging them.

$$\epsilon^{\text{CIF}} = \frac{1}{J} \sum_{j=1}^{J} \epsilon_{\cdot j}^{\text{CIF}}$$

Putting it All Together

We generate 10 training, validation, and test datasets with size 1000 each. Both largeRCRF and randomForestSRC are trained on each training dataset with the different parameter combinations described earlier. $\epsilon_{ik}^{\text{Concordance}}$ is calculated on the validation dataset using each package's code for naive concordance (as shown in Equation (3.2)). In addition, we also calculate $\epsilon_{ik}^{\text{Concordance}}$ for randomForestSRC using largeRCRF's implementation of naive concordance (referred to as randomForestSRC/Alt), as there appears to be significant disagreement in the concordance error returned between the two packages.

For these three combinations (largeRCRF, randomForestSRC, and randomForestSRC/Alt), the sets of tuning parameters that minimized $\epsilon_{ik}^{\text{Concordance}}$ is calculated. Using the generated test sets, estimates of the error on the cumulative incidence functions are then calculated.

Simulation results

Figure 3.1 shows the errors on the CIFs for the different models. Since all of the models consider the same dataset in each simulation, we can make the plot more informative by dividing each CIF error by the smallest error produced by the 3 models on that dataset. Figure 3.2 shows these results.

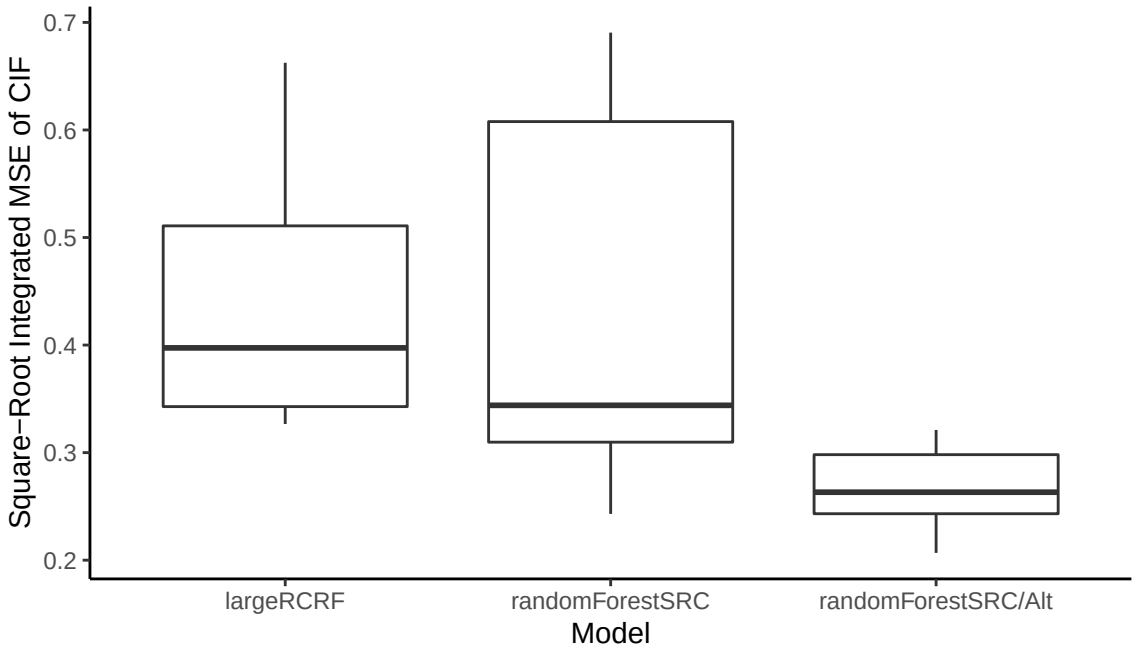

Figure 3.1: Boxplot of cumulative incidence function (CIF) errors by model. randomForestSRC/Alt is the model produced by randomForestSRC, but tuned using the concordance error as calculated by largeRCRF.

It's clear from both plots that largeRCRF results are only slightly worse than randomForestSRC's, demonstrating that largeRCRF is a viable alternative for researchers to use. Interestingly, randomForestSRC using largeRCRF's implementation of naive concordance for tuning greatly outperforms both packages, suggesting that future versions of randomForestSRC may easily further improve their accuracy by modifying their implementation of naive concordance.

That said, the next simulation will show that largeRCRF runs significantly faster which could result in better performance if largeRCRF were tuned on a denser grid.

3.1.3 The Second Simulation - Assessing Speed

For the timing simulation, we generate 6 datasets; two of size 1000, two of size 10,000, and two of size 100,000. For each pair of datasets we'll train forests on one and then use that forest to predict on the other, timing how long it takes to train and make predictions for each package, for each size dataset. For both packages we'll repeat the procedure 10 times for $n = 1000$ and 5 times for $n = 10,000$. For $n = 100,000$ we train largeRCRF 5 times

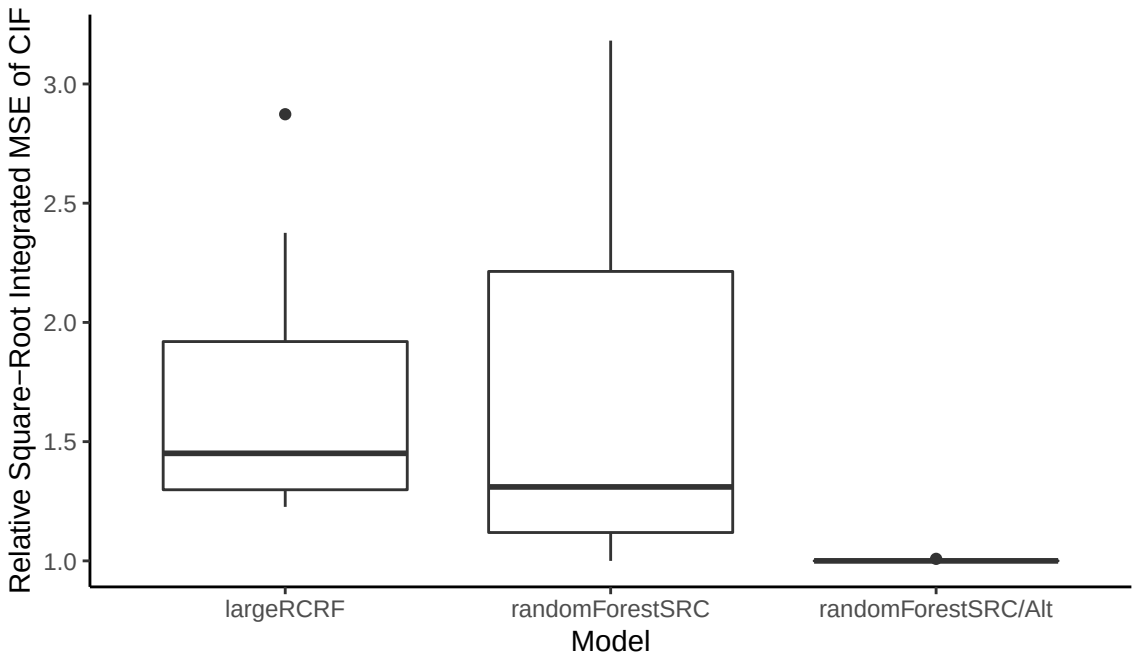

Figure 3.2: Boxplot of relative cumulative incidence function (CIF) errors by model. For each dataset, all errors are divided by the smallest error produced; values of 1 means that that model produced the lowest error. randomForestSRC/Alt is the model produced by randomForestSRC, but tuned using the concordance error as calculated by largeRCRF.

	largeRCRF			randomForestSRC		
	Min.	Median	Max.	Min.	Median	Max.
$n = 1000$	2.10	2.17	2.32	7.04	7.14	7.21
$n = 10,000$	35.7	36.2	37.0	3274	3350	3439
$n = 100,000$	736	742	748	-	384061	-

Table 3.2: The recorded minimum, median, and maximum times (in seconds) recorded for training and predicting for each package for the different sizes of data.

but randomForestSRC only once (due to computational constraints). The forest parameters used are constant and are:

- Number of splits tried (`nsplit`): 1000
- Node size (`nodeSize`): 500
- Number of covariates tried at each split (`mtry`): 1
- Number of trees (`ntree`): 100

The simulations were run on a desktop computer running Linux with 16GB of RAM and a 4 core (8 thread) 3.5 GHz CPU.

The results, summarized in Table 3.2, show that largeRCRF is faster than randomForestSRC at all three sizes. At $n = 1000$ largeRCRF is only about 3x times faster, but that factor quickly grows as the sample sizes increases. At $n = 10,000$ largeRCRF is about 90x faster, and at $n = 100,000$ largeRCRF is about 500x faster. This speed increase allows researchers to perform more accurate and denser tuning, especially at larger sample sizes.

3.1.4 Implementation Differences

A few times now there's been a note made of a difference in largeRCRF and randomForestSRC's implementation. This section will formally note them.

First, randomForestSRC approximates Gray's test (Equation 2.4) by using the largest observed censor time with no way to use the true censor times if available; largeRCRF explicitly requires the true censor times with no approximate version available.

Second, as of version 2.9.0, randomForestSRC by default trains trees on 63.2% of the data without replacement instead of bootstrapping the full dataset. largeRCRF sticks with the original random forest algorithm and uses bootstrap samples of size n. When running the simulations we kept randomForestSRC at the old default of performing bootstrap samples of size n.

Third, another difference is how NAs are handled. This wasn't relevant for the simulations where there were no NAs but will be important for the loan dataset. randomForestSRC tries to impute NAs through training unsupervised random forests, such that every predictor in a row could be predicted based off of the other values. Training an additional forest is computationally expensive and not suitable for our large data.

Instead, we use an NA strategy similar to the one described in Tang and Hemant Ishwaran 2017, Section 2.3 (steps 1 - 3 only). This strategy essentially says that when finding a best split for a predictor, missing data is excluded. After the best split (and its corresponding score) is found, the rows with missing data are assigned to the left and right nodes randomly based on the proportions of how the non-missing data was split.

Unfortunately this strategy of random assignment *after* the best split has been found means that we might split on a predictor that provides good splits but is mostly missing over a similar predictor that is slightly noisier with less missingness. Ideally we'd like to

penalize a predictor with greater missingness. We modify the strategy slightly - for each predictor we'll split on we find the best split using only non-missing data. We then assign the missing rows randomly as before and recalculate the split score for each predictor's previously identified best split. We then take the worst of the two scores, and then choose the predictor that still has remaining best score. This penalizes predictors with missingness - they'll only be split on if even with random assignments they are better than the other available predictors.

Since recalculating the split score for every predictor with missing data can be expensive, and matters little with variables with very few NAs, this randomized penalty is only applied to predictors that have at least 5% of its values missing (the 5% threshold is default but configurable).

3.1.5 largeRCRF Evaluation

The simulations showed that largeRCRF's training speed is vastly superior to randomForestSRC. It also showed, however, that there is an ongoing issue of accuracy as largeRCRF does not perform as well as it could relative to a correctly tuned randomForestSRC (randomForestSRC/Alt).

Unfortunately, despite a lot of effort the reason for the poor accuracy has not been found. We have some idea of where in the code there isn't an issue by comparing specially crafted deterministic forests between largeRCRF and randomForestSRC, but the location and cause of the bug is still unknown. Based on past discrepancies found between randomForestSRC's implementation and its published papers, there is a chance that its authors found a beneficial tweak to the random forest algorithm that remains unpublished as of yet. This lines up with our observation that randomForestSRC's accuracy drastically improved with version 2.9.0 (largeRCRF beat randomForestSRC on an earlier version). That said, while largeRCRF has room for improvement, we feel that it is good enough (and our only option) for use on our dataset.

Chapter 4

Application on Loan Prepayment

As mentioned in Section 1.1.1, some of the variables were only introduced part way through the dataset. Since our random forest implementation penalizes splitting on variables with NAs, this will cause variable importance metrics to underestimate the importance of those variables. Therefore we'll work with two datasets -

- The full dataset - we'll apply variable selection to help reduce the number of variables for faster tuning, and then we'll tune the random forest hyper-parameters to find the best forest. We'll use the best forest from this dataset for our analysis. The predictors included part ways through the dataset will be NA for the loans where they're unknown.

- A restricted dataset of loans that were issued starting in January 2016 so that it contains all of the predictors that were previously NA. We'll apply variable selection to this dataset to identify important variables that might inform financial institutions who wish to perform their own analysis. We won't, however, use this dataset for our full analysis as the oldest of the 36-month loans in this dataset is only 33 months, meaning that any estimates of the loan prepayment cumulative incidence functions will not be available for the last 3 months of the loans.

Since censor times are available, we'll use the Gray test splitting rule described in Section 2.1.1 focused entirely on the repayment event only.

4.1 Variable Selection

To determine which variables are actually important, we'll apply the random forest variable selection algorithm described in Genuer, Poggi, and Tuleau-Malot 2010. Because computational resources are limited and this variable selection algorithm trains upwards of hundreds of forests per run, we restrict it to training on random samples of 5000 loans. To get an idea of how variable the selections are, we will repeat this 10 times (no overlap of random samples).

An brief overview of the algorithm is presented below. Steps:

1. A pre-screening step is done to eliminate variables that are certainly unimportant, to lower later computation costs. For this step 50 random forests are trained using all the variables, with variable importance metrics calculated for each variable for each forest. The mean and standard deviation of each variable's variable importance is calculated from the 50 runs, and the variables are sorted in descending importance according to the mean. Genuer et al. claim that the standard deviations of importance for important variables are larger than the standard deviations of unimportant variables, so we can then expect that the standard deviations should decrease in roughly the same order as the rankings by the mean. A regression tree, where the response are the standard deviations and the one predictor is the ranks from sorting, is then fit. Using that tree, the smallest predicted standard deviation is retrieved. Variables whose mean importance is less than or equal to this smallest standard deviation are removed. Assume we have m variables remaining.

2. Relying on the previous rankings of the remaining variables, this step computes the average OOB errors (over 25 runs) for when the k most important variables are included for all $k = 1 \dots m$. We then select the smallest model whose average OOB error is less than the minimum average OOB error plus its standard deviation. I.e. let m' be the smallest k that satisfies this constraint:

$$\overline{\text{OOB}}_k^{\text{Err}} < \overline{\text{OOB}}_{k_{\min}}^{\text{Err}} + \underset{i=1\dots25}{\text{sd}}(\text{OOB}_{i,k_{\min}}^{\text{Err}})$$

If our goal is only to identify important variables for the purpose of interpretation, we could stop here. However, we can continue onto the next step to further reduce it to try to get the best possible predictions.

3. For this step, we follow a stepwise addition procedure to add variables based on minimizing OOB error, using only the variables we identified in Step 2. However, we only add variables if the decrease in error exceeds the threshold given in the following equation, based on the OOB errors from Step 2.

$$\frac{1}{m-m'}\sum_{k=m'}^{m-1}|\overline{\text{OOB}}_{k+1}^{\text{Err}}-\overline{\text{OOB}}_{k}^{\text{Err}}|$$

In other words, we estimate what the mean decrease in OOB error was among the variables we eliminated in Step 2 to get an estimate of the decrease in OOB error that adding a noise variable would cause, so that we can avoid adding too many variables.

For this algorithm all forests will be trained with hyper-parameters of `nsplit` of 100, `nodeSize` of 250, and `mtry` of one third of the available predictors, rounded up. `nsplit` is 100 for performance reasons, but one of the 100 splits should be close to the true optimal split. `nodeSize` is rather large at 250, but it was a necessary compromise for performance reasons. The algorithm was originally designed for regression or classification problems, so to extend it we'll need to specify how we calculate error and variable importance.

For error, we use the mean weighted integrated Brier score as described in Ishwaran et al. 2014, Section 4.2 for the repayment CIF only; integrating from 0 to 36 months for the full dataset, and 0 to 33 months for the restricted dataset. We don't care about the accuracy of the repayment CIF past 36 months as that would represent someone who was late repaying their loan, and so isn't prepayment. The weights $\hat{w}_i(t)$ are derived from an estimate of the censor distribution $\hat{G}(t)$, which we can easily obtain using an empirical CDF.

$$\hat{w}_i(t)=\frac{I(\tilde{T}_i\leqslant t\cap\delta_i\neq 0)}{\hat{G}(\tilde{T}_i)}+\frac{I(\tilde{T}_i>t)}{\hat{G}(t)}$$

$$\text{IBS} = \frac{1}{n}\sum_{i=1}^{n}\int_0^{36} \hat{w}_i(t)[\widehat{CIF}_{i,1}(t) - I(\tilde{T}_i \geqslant t \cap \delta_i = 1)]^2 dt \tag{4.1}$$

For variable importance, we use permutation variable importance as described in Izenman 2008, Section 14.4.5 with this above IBS error, in which we record the difference in error before and after scrambling the values of a particular predictor.

For the number of trees, we start at 100, measure the OOB IBS error, and increment the number of trees by another 100, recording the error again. We track the minimum error, and keep incrementing the number of trees until the error fails to decrease for two 100 tree increments in a row. Note that adding more trees in random forests generally should never hurt predictive performance, so we use the latest error with all the trees despite the fact that the sampled minimum OOB error was always two increments ago.

4.1.1 Full Dataset

Table 4.1 is the result of applying the variable selection procedure described in Section 4.1; only variables that were selected at least once are displayed.

These results show that most of the original 75 predictors appear to have no importance and may be dropped from the analysis. Most of the variables listed appear to measure overall debt utilization, and how many credit products a borrower has. This might match our intuition; someone who has many credit options available to them will find it easier to consolidate their debt with Lending Club at a much lower interest rate, showing up to us as prepayment.

Some of these variables are close cousins to others; for instance `all_util_custom` and `all_util_custom_no_mortgage`; both of which measure the borrower's credit utilization; except that `all_util_custom_no_mortgage` excludes mortgage payments (it's not necessarily fair to count mortgage payments as debt payments since the borrower is saving on not paying rent and is gaining an asset; a very different situation from paying down a credit card). Including both of these predictors would add unnecessary noise, especially because the algorithm never selects both variables in the same trial. Instead, the variables we'll use

	Trial										
Variable	1	2	3	4	5	6	7	8	9	10	Count
num_rev_tl_bal_gt_0	✓	✓	✓	✓	✓	✓	✓		✓	✓	9
acc_open_past_24mths	✓	✓	✓	✓	✓			✓		✓	7
annual_inc	✓			✓	✓	✓		✓	✓		6
earliest_cr_line	✓			✓		✓	✓		✓		5
all_util_custom_no_mortgage	✓	✓		✓	✓				✓		5
num_tl_op_past_12m	✓					✓	✓		✓		4
total_rev_hi_lim					✓	✓		✓			3
num_rev_accts	✓					✓	✓				3
num_il_tl	✓			✓	✓						3
num_actv_rev_tl		✓			✓			✓			3
tot_hi_cred_lim			✓	✓							2
revol_bal						✓		✓			2
pct_tl_nvr_dlq		✓	✓								2
mort_acc				✓			✓				2
loan_amnt	✓				✓						2
dti			✓							✓	2
total_acc		✓									1
tot_cur_bal										✓	1
purpose										✓	1
num_op_rev_tl			✓								1
num_actv_bc_tl					✓						1
mths_since_recent_inq										✓	1
mths_since_recent_bc_dlq	✓										1
mths_since_last_delinq	✓										1
installment						✓					1
all_util_custom			✓								1

Table 4.1: The predictors that were selected at least once for the full dataset, along with which trials they were selected in. The predictors are sorted according to how often they were included in each trial.

when training on the full dataset will be those variables that were selected at least 3 times; meaning we have 10 predictors:

- acc_open_past_24mths
- all_util_custom_no_mortgage
- annual_inc
- earliest_cr_line
- num_actv_rev_tl

- num_il_tl
- num_rev_accts
- num_rev_tl_bal_gt_0
- num_tl_op_past_12m
- total_rev_hi_lim

Interestingly, this list includes num_rev_accts, which tracks the number of revolving trades, and num_rev_tl_bal_gt_0, which tracks the number of revolving trades that carry a positive balance. Our intuition may suspect that the model is really looking for "proportion of revolving trades that carry a positive balance", so we create a new variable called proportion_rev_bal_gt_0[1] that we'll also include, giving 11 predictors in total.

4.1.2 Restricted Dataset

When applied to data where many more predictors are not NA (data from after end-of-2015), the variable selection procedure described in Section 4.1 produced Table 4.2. Again, only variables that were selected at least once are displayed.

We again see that only a small number of the original 75 predictors appear important. However, contrasted with the full dataset, there are differences in the variables that are selected. For instance, open_il_24m, which under the full dataset is about 40% NA but virtually never NA under the restricted dataset, was selected five times for the restricted dataset and never for the full dataset.

This suggests that some of the predictors that were only recently introduced are in fact useful for estimating prepayment, and that the error produced by the full dataset is even more likely to be an overestimate of the error that a financial institution would observe should they analyze loan prepayment using their full datasets.

If we use the same rule of requiring that a predictor appear at least 3 times, we have the following predictors available to use:

[1]There was a division by 0 in exactly one case; this was solved by just dropping that row.

	Trial										
Variable	1	2	3	4	5	6	7	8	9	10	Count
dti	✓		✓	✓	✓	✓	✓	✓	✓	✓	9
acc_open_past_24mths		✓	✓	✓	✓		✓	✓	✓	✓	8
all_util_custom_no_mortgage	✓	✓	✓		✓	✓		✓		✓	7
num_actv_rev_tl			✓	✓	✓		✓	✓	✓		6
open_il_24m	✓		✓	✓	✓			✓			5
num_rev_tl_bal_gt_0		✓			✓		✓			✓	4
earliest_cr_line			✓	✓			✓			✓	4
total_cu_tl	✓				✓					✓	3
tot_hi_cred_lim				✓			✓			✓	3
revol_bal	✓	✓									2
num_rev_accts							✓		✓		2
num_il_tl					✓	✓					2
mths_since_rcnt_il						✓				✓	2
installment						✓	✓				2
total_rev_hi_lim		✓									1
total_il_high_credit_limit							✓				1
total_bc_limit	✓										1
total_bal_ex_mort			✓								1
total_acc								✓			1
num_bc_tl								✓			1
num_actv_bc_tl	✓										1
mort_acc						✓					1
mo_sin_old_rev_tl_op			✓								1
loan_amnt				✓							1
inq_fi	✓										1
annual_inc							✓				1

Table 4.2: The predictors that were selected at least once for the restricted 2016+ dataset, along with which trials they were selected in. The predictors are sorted according to how often they were included in each trial.

- dti
- acc_open_past_24mths
- all_util_custom_no_mortgage
- num_actv_rev_tl
- open_il_24m
- num_rev_tl_bal_gt_0
- earliest_cr_line

- `total_cu_tl`
- `tot_hi_cred_lim`

4.2 Bayesian Optimization

For the full dataset, now that we have the subset of variables we want to include, we use Bayesian optimization (Snoek, Larochelle, and Adams 2012) through the rBayesianOptimization package (Yan 2016) to tune the following random forest hyper-parameters:

- `mtry`: The number of predictors to attempt to split on. We let it range from 1 to the number of available predictors.
- `nodeSize`: Forces any sibling terminal nodes to have a minimum pairwise average size of at least this size; indirectly controls the depth of each tree. We let it range from 50 to $75,000$.
- `nsplit`: The number of random possible splits attempted for each predictor. We let it range from 1 to 100.

Bayesian optimization tries to estimate the error function using a Gaussian process, and based off of all previous trained models it will try and guess the hyper-parameters for the most optimal model, train it, and incorporate the evaluated error into the posterior to optimize a new best guess. There are different variants of Bayesian optimization; here we choose to evaluate the model that minimizes the current 95% lower confidence bound of the error function.

For our full dataset, we'll first split the full dataset into training, validation, and test components with sizes (1,322,012; 50000; 50000). We'll again use the IBS error as our error function, but this time with the validation dataset instead of out-of-bag error. We'll also apply the same procedure for determining how many trees to grow as we did for variable selection, instead of tuning that with Bayesian optimization (which would be wasteful, as more trees are generally always better).

4.3 Results

After applying the Bayesian optimization method to the full dataset for 70 trained forests, the tuning parameters selected were as follows: `mtry` was 6, `nodeSize` was 354, and the selected `nsplit` was 71. This produced a mean validation error of 5.842 and a final test error of 5.875.

To demonstrate that our model is useful for estimating loan prepayments, we're going to look for two loans that have a strong difference in their predicted loan repayment CIF with the constraint that the two loans should otherwise appear equivalent from a financial institution's perspective. This means we're going to require that the loans have identical loan amounts, similar interest rates, identical annual incomes, and identical default risk scores. Obviously some of the credit metrics will vary (they have to if there's to be different predictions), but financial institutions generally only use these metrics to determine the interest rate and default risk and are then ignored. Table 4.3 contains the values for these loans' predictors. Their predicted repayment cumulative incidence functions (CIFs) are plotted in Figure 4.1.

Predictor	**Low Prepayment**	**High Prepayment**
`loan_amnt`	\$11000	\$11000
`int_rate`	6.83%	6.97%
`installment`	\$338.80	\$339.50
`annual_inc`	\$105000	\$105000
`sub_grade`	3	3
`earliest_cr_line`	August 2004	February 1987
`total_rev_hi_lim`	\$67400	\$20850
`acc_open_past_24mths`	18	5
`num_actv_rev_tl`	5	5
`num_il_tl`	29	4
`num_rev_accts`	32	11
`num_rev_tl_bal_gt_0`	5	5
`num_tl_op_past_12m`	8	4
`all_util_custom_no_mortgage`	0.32	0.90
`proportion_rev_bal_gt_0`	0.16	0.45

Table 4.3: The values of the relevant metrics for the two selected loans.

Seeing in Figure 4.1 the estimated repayment CIFs is useful, but we need to put everything back into the units that the financial institution sees - money. We can use our

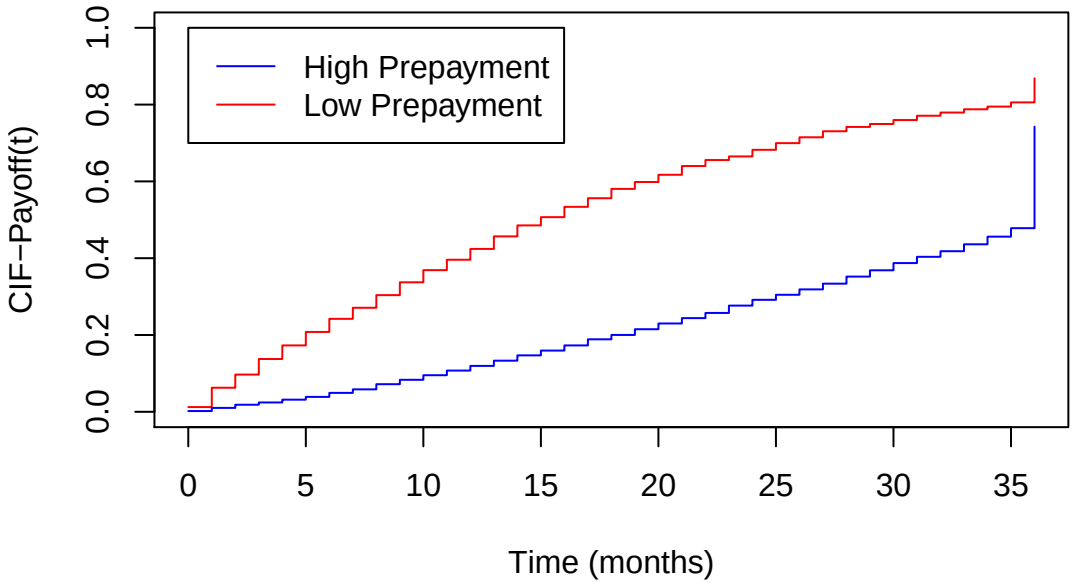

Figure 4.1: Plot of the predicted repayment cumulative incidence functions (CIFs) for two loans with identical provided default-risk levels. The high jump at 36 months is associated with all of the loans that finished repaying right on time.

estimates of the repayment CIF (and trivially our estimate of the default CIF) to calculate an 'expected loss due to loan prepayment' by using the CIFs to get the joint distribution of event time and event type. There's one caveat though - *when* a loan prepays doesn't perfectly map to the loss due to interest. A given loan can be made to have prepaid by t months by the borrower making all prepayment at the beginning of the loan, by paying off the remaining balance at t months, or something in between. Those first two situations represent the largest and smallest, respectively, loss to the financial institution. We'll therefore use both as assumptions in order to map time of loan prepayment to loss due to prepayment.

- **Prepayment Early**: The borrower immediately repaid a balance once they got the loan and then made regularly scheduled payments afterward, causing them to repay at the specified time. In this situation the financial institution loses the most because interest was not able to accrue on the immediately repaid balance.

- **Prepayment Late**: The borrower made regularly scheduled payments, and then on the day they repaid their loan they prepaid the entire remaining balance back. In this situation the financial institution loses the least because interest was able to accrue on the balance for the longest possible period.

To formalize our 'loss due to prepayment' functions for a given prepayment date:

Let $f(t, \delta)$ and $g(t, \delta)$ denote the loss functions for a given time and event under the two assumptions, prepayments early and late, respectively. Because we are calculating expected loss *due to prepayment*, let $f(t, \delta = 2) = g(t, \delta = 2) = 0$ and $\forall t \geqslant 36$ $f(t, \delta) = g(t, \delta) = 0$, since a default or late repayment is not a prepayment.

For the remaining values of t and δ (i.e. where prepayment actually occurs) the two functions are calculated using compound interest to determine what the loan balance would have been at time t had no prepayment occurred. For the 'prepayment early' case we then need to undo the compound interest to get the prepayment amount at the beginning of the loan. For the other case, 'prepayment late', the prepayment is just the remaining loan balance. We then sum up the amount of payments that were made in total, and subtract it from the amount of payments that should have been made over the course of the entire loan (36 payments of `installment`). Appendix C contains the R code that describe these loss functions.

At this point we can now calculate the expected loss due to prepayment for the two scenarios.

$$\begin{aligned}
\hat{E}_{T,\Delta}[f(T,\Delta)|X] &= \sum_{\delta \in \{1,2\}} \sum_{t=0}^{\infty} f(t,\delta) \cdot \hat{P}(T = t \cap \Delta = \delta | X) \\
&= \sum_{t=0}^{35} f(t, \delta = 1) \cdot \hat{P}(T = t \cap \Delta = 1 | X) \\
\hat{E}_{T,\Delta}[g(T,\Delta)|X] &= \sum_{\delta \in \{1,2\}} \sum_{t=0}^{\infty} g(t,\delta) \cdot \hat{P}(T = t \cap \Delta = \delta | X) \\
&= \sum_{t=0}^{35} g(t, \delta = 1) \cdot \hat{P}(T = t \cap \Delta = 1 | X)
\end{aligned}$$

where $\hat{P}(T = t \cap \Delta = 1|X)$ is easily derived from the estimated repayment CIF.

Loan	**Prepayments Early**	**Prepayments Late**
Predicted High Prepayment	$760	$459
Predicted Low Prepayment	$347	$158
Difference	$413	$301

Table 4.4: Expected loss due to prepayment for two loans under the two assumptions of prepayment.

For the two loans, their expected losses due to prepayment are calculated and shown in Table 4.4. We see from the table that the difference in prepayment risk between the two loans under both assumptions are high. If a financial institution was able to somehow influence the behaviour of a borrower who is likely to prepay to behave like the other customer, then they could save somewhere between $300 to $400; which would substantially increase their margins. This suggests that research into predicting loan prepayment has value, and that financial institutions should take a closer look using their complete datasets where loan prepayment can be directly tracked.

Chapter 5

Conclusion and Discussion

In Chapter 1 we introduced the problem of loan prepayment. We were interested in seeing if we could identify which predictors were useful, and whether we could use those predictors to distinguish between loans that were more or less likely to prepay. Because the dataset only contained times of loan termination instead of a dollar amount of prepayment, we were restricted to using competing risks models.

The size of the dataset and the unknown relationship between the response and the predictors made a non-parametric model the only real option. In Chapter 2 we looked into competing risks random forests, which was implemented in an existing package, randomForestSRC, but found that it was not suitable for a dataset at our scale.

In order to analyze the dataset we decided that we need to write our own package. We wrote largeRCRF which we introduced and evaluated in Chapter 3 by comparing its accuracy and speed with randomForestSRC. While its accuracy was not as good, its speed was stellar, finally providing us a tool to analyze this dataset.

In Chapter 4 we went about determining which predictors were important. Since many of the predictors were missing in the earlier part of the dataset, we ran variable selection on both the full dataset and a subset where these predictors were rarely missing. In both cases the variable selection algorithm greatly reduced the number of available predictors to a more reasonable level.

Afterward, we went about using the chosen predictors for the full dataset to train and tune a model that could be used to successfully predict loan prepayment. We demonstrated this model on a pair of loans that most financial institutions would see as equivalent,

predicting vastly different prepayment behaviour. We also mapped the predicted cumulative incidence functions for repayment into a range of dollar loss by the financial institution, putting the prepayment behaviour into terms a financial institution can understand, and showed that for this pair the difference was a few hundred dollars.

This research is preliminary. With only time data available, we can only indirectly predict loan prepayment. With only observational data available, we cannot predict how successful targeted methods to change customer behaviour would be. What we have been able to do with this research though is demonstrate that loan prepayment can be predicted to a useful degree, and we've identified a set of variables that should be useful in a more thorough study. We believe that this research provides a good justification for further research by a financial institution, and that it provides a good first step towards that research.

5.1 Areas for Future Work

It's worth pointing out that the author recognizes a few areas of improvement, that if not for computational and time constraints would be dealt with. First, largeRCRF's accuracy as compared to randomForestSRC (when randomForestSRC is properly tuned) is poor. Ideally we'd like to solve this issue, and great quantities of time was put into it with little success. Eventually we had to move on because of time constraints.

Second, the variable selection algorithm used was applied to small datasets of only size 5000, with a `nodeSize` tuning parameter of size 250. This meant that the trained trees were much more shallow than if a larger subset had been used. Predictors with a small (but present) predictive ability might not have been picked up by variable selection algorithm, giving us a smaller set of variables than might have been optimal on the full dataset.

Third, the Bayesian optimization tuning was stopped too soon; in fact the last model it trained was the one selected. With more time and computational resources the model error could be further reduced.

Fourth, one of the selected variables, `earliest_cr_line`, which is the encoded date of the earliest credit line, might leak information about *when* the loan was issued. That might unrealistically improve model performance, because it's not unreasonable to assume that

all loans issued in one year might have different performance than those issued in another year. In essence, the model could memorize past performance from previous years to get better results; but that's not useful because the model needs to predict *future* performance. Instead, `earliest_cr_line` should have been encoded to instead measure how old the earliest credit line was at the time of loan application.

Unfortunately, this was discovered after both variable selection and model tuning had been performed. However, we do not think this leakage affects results too badly. We tested this by repeating the training on the same chosen model with an adjusted `earliest_cr_line` and got a test error of 5.897 (versus the previous 5.875). Using the same chosen loans, the plot of the CIFs is similar, and the expected loss due to prepayment estimates are all within \$15 of their previous values, with the differences within \$5. Thus, while we'd ideally like to rerun the variable selection and tuning with the corrected predictor, we don't believe this mistake detracts from our conclusion that loan prepayment can be predicted to a useful degree.

Bibliography

Aalen, Odd O. (1978). "Nonparametric Inference for a Family of Counting Processes". In: *The Annals of Statistics* 6.4, pp. 701–726. URL: http://www.jstor.org/stable/2958850.

Aalen, Odd O. and Søren Johansen (1978). "An Empirical Transition Matrix for Non-Homogeneous Markov Chains Based on Censored Observations". In: *Scandinavian Journal of Statistics* 5.3, pp. 141–150. URL: http://www.jstor.org/stable/4615704.

Breiman, Leo (Oct. 2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32. DOI: 10.1023/A:1010933404324.

Fine, Jason P. and Robert J. Gray (1999). "A Proportional Hazards Model for the Subdistribution of a Competing Risk". In: *Journal of the American Statistical Association* 94.446, pp. 496–509.

Genuer, Robin, Jean-Michel Poggi, and Christine Tuleau-Malot (2010). "Variable selection using random forests". In: *Pattern Recognition Letters* 31.14, pp. 2225–2236. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2010.03.014. URL: http://www.sciencedirect.com/science/article/pii/S0167865510000954.

Grolemund, Garrett and Hadley Wickham (2011). "Dates and Times Made Easy with lubridate". In: *Journal of Statistical Software* 40.3, pp. 1–25. URL: http://www.jstatsoft.org/v40/i03/.

Grothendieck, G. (2017). *sqldf: Manipulate R Data Frames Using SQL*. R package version 0.4-11. URL: https://CRAN.R-project.org/package=sqldf.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7.

Ishwaran, H. and Udaya B. Kogalur (2018). *Random Forests for Survival, Regression, and Classification (RF-SRC)*. R package version 2.9.0. manual. URL: https://cran.r-project.org/package=randomForestSRC.

Ishwaran et al. (2014). "Random Survival Forests for Competing Risks". In: *Biostatistics* 15.4, pp. 757–773. DOI: 10.1093/biostatistics/kxu010.

Izenman, Alan J. (2008). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer-Verlag New York. ISBN: 978-1-4939-3832-2. DOI: 10.1007/978-0-387-78189-1.

Kaplan, E. L. and Paul Meier (1958). "Nonparametric Estimation from Incomplete Observations". In: *Journal of the American Statistical Association* 53.282, pp. 457–481. DOI: 10.1080/01621459.1958.10501452.

Nelson, Wayne (1972). "Theory and Applications of Hazard Plotting for Censored Failure Data". In: *Technometrics* 14.4, pp. 945–966. DOI: 10.1080/00401706.1972.10488991.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: arXiv: 1206.2944 [stat.ML].

Tang, Fei and Hemant Ishwaran (2017). "Random Forest Missing Data Algorithms". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 10.6, pp. 363–377. DOI: 10.1002/sam.11348.

Urbanek, Simon (2018). *rJava: Low-Level R to Java Interface*. R package version 0.9-10. URL: https://CRAN.R-project.org/package=rJava.

Wickham, Hadley (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag. ISBN: 978-3-319-24277-4. URL: http://ggplot2.org.

Wickham, Hadley, Jim Hester, and Winston Chang (2019). *devtools: Tools to Make Developing R Packages Easier*. R package version 2.2.1. URL: https://CRAN.R-project.org/package=devtools.

Wickham, Hadley et al. (2019). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.3. URL: https://CRAN.R-project.org/package=dplyr.

Wolbers, Marcel et al. (Feb. 2014). "Concordance for Prognostic Models with Competing Risks". In: *Biostatistics* 15.3, pp. 526–539. DOI: 10.1093/biostatistics/kxt059.

Yan, Yachen (2016). *rBayesianOptimization: Bayesian Optimization of Hyperparameters*. R package version 1.1.0. URL: https://CRAN.R-project.org/package=rBayesianOptimization.

Appendix A

Computational Details

All training of forests can be run on computers with a 4 core / 8 thread CPU and 32 Gb of memory, running Ubuntu Linux.

Source code for largeRCRF can be retrieved at `https://github.com/jatherrien/`, and it can be easily installed in R using the devtools (Wickham, Hester, and Chang 2019) package with `devtools::install_github("jatherrien/largeRCRF")`.

Software Used

The following software was extremely useful throughout this research and its authors deserve to be credited.

- The open-source software project R (R Core Team 2018) was used throughout the entire process and provided the user interface that largeRCRF was able to build on.
- R package ggplot2 (Wickham 2016) was used for several of the plots.
- R packages sqldf (Grothendieck 2017), dplyr (Wickham et al. 2019), and lubridate (Grolemund and Wickham 2011) were used for cleaning and processing data.
- R package rJava (Urbanek 2018) is a critical package that largeRCRF uses for connecting the R interface with the backend Java code that most of largeRCRF is written in.
- R package devtools (Wickham, Hester, and Chang 2019) is a package that helped in the development for largeRCRF.
- R package rBayesianOptimization (Yan 2016) is the package used for tuning in the analysis.

Appendix B

Predictor Variables

All predictors are collected at the time of loan application.

- `acc_now_delinq`: The number of items that the borrower is currently delinquent on.
- `acc_open_past_24mths`: The total number of debt items opened in the past 24 months.
- `addr_state`: The address state the borrower lives in (factor variable).
- `all_util_custom`: The credit utilization (current balance / credit limits) across all trades.
- `all_util_custom_no_mortgage`: The credit utilization (current balance / credit limits) across all trades, excluding mortgages.
- `annual_inc`: Borrower's self-reported income.
- `avg_cur_bal`: The average balance of all accounts.
- `chargeoff_within_12_mths`: The number of debts that defaulted within the past 12 months.
- `collections_12_mths_ex_med`: The number of credit items that have gone to collections in the past 12 months, excluding medical debt.
- `delinq_2yrs`: The number of times in the past 2 years the borrower was 30+ days late on a payment.
- `delinq_amnt`: The total delinquent amount across all trades.
- `dti`: The borrower's debt service ratio; borrower's current monthly non-mortgage debt payments divided by their self-reported monthly income.
- `earliest_cr_line`: The month the borrower's earliest reported credit line was opened.
- `emp_length`: How long they've worked at their current employer; with values between 0 and 10 (truncated).

- `home_ownership`: The borrower's home ownership status; factor variable with levels RENT, OWN, MORTGAGE, or OTHER.
- `inq_fi`: The number of credit report inquiries.
- `inq_last_12m`: The number of credit inquiries in the past 12 months.
- `inq_last_6mths`: The number of debt inquiries on the credit report in past 6 months (excluding auto and mortgage inquiries).
- `installment`: The monthly payment.
- `int_rate`: The interest rate.
- `loan_amnt`: The loan amount.
- `max_bal_bc`: The maximum balance ever seen on every revolving trade, totaled.
- `mo_sin_old_il_acct`: The number of months since the oldest installment debt item was opened. If one was never opened, a very high value is given.
- `mo_sin_old_rev_tl_op`: The number of months since the oldest revolving debt item was opened. If one was never opened, a very high value is given.
- `mo_sin_rcnt_rev_tl_op`: The number of months since the youngest revolving debt item was opened. If one was never opened, a very high value is given.
- `mo_sin_rcnt_tl`: The number of months since the youngest debt item was opened. If one was never opened, a very high value is given.
- `mort_acc`: The number of mortgage debt items.
- `mths_since_last_delinq`: The number of months since the borrower was last 30+ days late on a payment; if this never happened a very high value is given.
- `mths_since_last_major_derog`: The number of months since the borrower has been 90+ days late on a payment; if this never happened a very high value is given.
- `mths_since_last_record`: The number of months since the last public record; if there are none then a very high value is given.
- `mths_since_rcnt_il`: The number of months since the youngest installment account was opened. If one was never opened, a very high value is given.
- `mths_since_recent_bc`: The number of months since the most recent bankcard item was opened. If one was never opened, a very high value is given.
- `mths_since_recent_bc_dlq`: The number of months since the most recent bankcard item was delinquent. If this never happened, a very high value is given.
- `mths_since_recent_inq`: The number of months since the most recent credit inquiry. If this never happened, a very high value is given.

- `mths_since_recent_revol_delinq`: The number of months since the borrower has been over 30 days late on a payment for a revolving debt item. If this never happened then a very high value is given.
- `num_accts_ever_120_pd`: The number of debt items where the borrower was ever more than 120 days late on their payment.
- `num_actv_bc_tl`: The number of currently active bankcard accounts.
- `num_actv_rev_tl`: The number of currently active revolving accounts.
- `num_bc_sats`: The number of bankcard accounts in good standing.
- `num_bc_tl`: The number of bankcard accounts.
- `num_il_tl`: The number of installment accounts.
- `num_op_rev_tl`: The number of open revolving accounts.
- `num_rev_accts`: The total number of revolving trades.
- `num_rev_tl_bal_gt_0`: The number of revolving accounts that carry a non-zero balance.
- `num_sats`: The number of accounts in good standing.
- `num_tl_90g_dpd_24m`: The number of times in the past 2 years the borrower was 90+ days late on a payment.
- `num_tl_op_past_12m`: The number of accounts opened in the past 12 months.
- `open_acc`: The number of open credit lines in the borrower's credit report.
- `open_acc_6m`: The number of 'open' debt items opened in the last 6 months.
- `open_act_il`: The number of currently active installment (fixed monthly non-mortgage payments) debt items.
- `open_il_12m`: The number of installment debt items opened in the past 12 months.
- `open_il_24m`: The number of installment debt items opened in the past 24 months.
- `open_rv_12m`: The number of revolving debt items opened in the past 12 months.
- `open_rv_24m`: The number of revolving debt items opened in the past 24 months.
- `pct_tl_nvr_dlq`: The percent of trades that have never had payments more than 30+ days past due.
- `percent_satisfactory_accounts`: The percent of all accounts in good standing.
- `percent_satisfactory_bank_cards`: The percent of bankcard accounts in good standing.
- `provided_employer`: A true / false indicating if the borrower named their employer.

- `pub_rec`: Number of derogatory public records on the borrower's credit report.
- `pub_rec_bankruptcies`: The number of bankruptcies on file.
- `purpose`: How the borrower plans on spending the loan (factor variable with 14 levels)
- `revol_bal`: The balance of all revolving trades.
- `sub_grade_numeric`: Lending Club's internal overall default risk score, represented as an ordinal variable.
- `tax_liens`: The number of tax liens on file.
- `tot_coll_amt`: The total amount that has ever gone to collections for a borrower.
- `tot_cur_bal`: The total current debt of a borrower.
- `tot_hi_cred_lim`: The total credit limit on all trades (for installment trades, this would be the original loan amount).
- `total_acc`: The total number of credit items in the borrower's credit report.
- `total_bal_ex_mort`: The total credit balance of all trades, excluding mortgages.
- `total_bal_il`: The balance of all installment trades.
- `total_bc_limit`: The credit limit on bankcards.
- `total_cu_tl`: The number of finance items on the credit report.
- `total_il_high_credit_limit`: The total of all installment loan amounts at their highest balances.
- `total_rev_hi_lim`: The total credit limit of revolving debt items.
- `verification_status`: A true/false indicating if Lending Club verified the borrower income.

Appendix C

Code for Loss Functions

```
# Function returns the current balance of a loan, assuming that the
# borrower is making only their scheduled payments on time.
balance = function(time_month, monthlyPayment, loanAmount,
             yearlyInterestRate){
  day = round(time_month*30)
  bal = loanAmount
  dailyInterestRate = yearlyInterestRate/360

  if(day == 0){
    return(bal)
  }

  for(d in 1:day){
    bal = bal + bal*dailyInterestRate
    if(d %% 30 == 0){
      bal = bal - monthlyPayment
    }
  }

  return(bal)

}

# Calculates the amount of interest the financial institution would
# have lost had the loan been paid off by time_month, given that
# all prepayments were made on day one of the loan
# (and normal payments afterward)
prepayments_early_loss =
 function(time_month, monthlyPayment, loanAmount,
    yearlyInterestRate){
  time = floor(time_month)
```

```
  dailyInterestRate = yearlyInterestRate / 360

  remainingBalance = balance(time_month, monthlyPayment, loanAmount,
                       yearlyInterestRate)
  days = round(time_month*30)

  prepayment = remainingBalance/((1+dailyInterestRate)^days)

  #Loss: [amount that should be paid] - [amount actually paid]
  return(monthlyPayment*36 - (monthlyPayment*time_month
           + prepayment))

}

# Calculates the amount of interest the financial instituion would
# have lost had the loan been paid off by time_month, given that
# all prepayments were made on the last day and previous payments
# had been normal.
prepayments_late_loss =
 function(time_month, monthlyPayment, loanAmount,
      yearlyInterestRate){
  numPayments = floor(time_month)

  prepayment = balance(time_month, monthlyPayment, loanAmount,
                 yearlyInterestRate)

  #Loss: [amount that should be paid] - [amount actually paid]
  return(monthlyPayment*36 - (monthlyPayment*numPayments
          + prepayment))
}
```