Loan Default Prediction: Imperial College London Sponsored Kaggle Competition

Prepared by:

Shatabdi Choudhury
Rohan Mehta
Dan Rubach
Roza Salazar
Stephen D. Young
Justin Yun

Prepared for:

Dr. Jennifer Wightman
Northwestern University
Department of Predictive Analytics
Course: PREDICT 454 Advanced Modeling Methods

This document submitted 3/12/2017.

Introduction

The focus of this work is supervised learning as applied to loan default prediction and loss which may occur in the event of a default. The data used herein emanates from a Kaggle (www.kaggle.com) competition where participants were given training and testing dataset and asked to provide predictions of out-of-sample loss amounts for the testing dataset which excluded the loss value column entries. Competitors were to fit a model or ensemble of models on the training data and provide a submission file that included a record identification (i.e. id) and loss amount which would be non-zero only in the case of default. The competition was sponsored by researchers at Imperial College London.

This work falls into the machine learning category of supervised learning as our data set includes a large number of features from which we seek to classify unseen out-of-sample loans as those that will and will not default. Default is a binary classifier which while not included in the data set and is created based on loss values whereby we set default equal to one for those cases where a loss amount is greater than zero. In this work we follow the Cross-Industry Standard Process for Data Mining ("CRISP-DM") life-cycle which starts with an understanding of the business problem, check of data quality and exploratory analysis, data preparation which may include transformations, imputations, and other modifications, followed by modeling, evaluation, and deployment. While our section headings may differ, it is the CRISP-DM approach that guides our work. As one will read from our conclusions, we find models for each of the classification and regression parts of our work which, post fitting, serve to generalize well and predict out-of-sample defaults and loss amounts.

The remainder of this work is formatted as follows. In the next section we discuss the modeling problem. This is followed by a discussion of the data which then leads us into Exploratory Data Analysis ("EDA"). From the EDA come predictive modeling methods and results where we discuss the approaches used to predict default and estimate loss amounts. Post modeling methods we provide for results and compare and contrast. Finally, we provide conclusions followed by a bibliography and an appendix with our R code. R is the statistical programming language that we use throughout this work.

The Modeling Problem

The modeling problem which we are faced with is to predict defaults and loss amounts on future unseen data records. The main focus is on loss but losses generally only occur as a result of default and, as such, our modeling focuses on out-of-sample prediction of both default and loss as these may be combined to estimate expected losses where one could take the probability from default and loss amount from the latter to calculate an expected loss. Modeling defaults and losses is of paramount importance for banks, finance companies, and those that may purchase loan securitizations made up of a large number of individual loans. Loan default modeling is also of concern to regulators that seek to understand the quality of a bank's loan portfolio as these financial institutions are overseen by various agencies whose goal is to ensure the safety and soundness of individual entities and the financial system as a whole.

In contrast to unsupervised learning which may be focused on broad categorizations and will often have no desired particular output, supervised learning is used to learn-by-example with the goal of generalization to future cases. For example, supervised learning may be focused on classifying an email as "spam" or "not spam", a financial transaction as "fraudulent" or "not fraudulent", and a loan as likely to "default" or "not default." Supervised learners focused on classification include such modeling methods as logistic regression, linear discriminant analysis,

decision trees, ensembles of decision trees which come together to form so-called random forests, and other approaches. Whichever approach, the overarching goal is the generalization of the model from in-sample fitting to out-of-sample test records and "real world" future use upon deployment. What generally varies from model-to-model is the modeling approaches ability to determine a decision boundary, or boundaries for classification problems which go beyond binary, that serve to separate the data into a respective category where in our case it is default or no default.

Thus, our modeling problem is well defined and is fully focused on the classification of loans to a binary outcome which includes default and no default. Further, beyond default we are concerned with the loss or severity of the outcome which is typically a fraction of the overall value of any loan as there is typically collateral (e.g. property) that may be liquidated and therefore result in recovery. When one combines the classification and subsequent loss the modeling effort is one of frequency and severity which are characteristic of default modeling. It is important to note that while frequency and even the outcome from select models (e.g. logistic regression which models log odds which are translated into a probability) may be continuous, in our modeling a loan will either be classified as one that will or will not default. Thus we have a classification problem associated with default and a regression problem in modeling loss.

Given our description of the modeling problem, our work starts by modeling default which is a binary classification and defined as follows:

$$default = \begin{cases} 0, & loss = 0 \\ 1, & loss > 0 \end{cases}$$

For the above, we define default based on the loss feature in the training data where loss is zero for no default and positive in the event of default. That is, a default flag is not included in the training data but is derived and given by a binary predictor.

While we start by modeling default which is a binary classifier, our work also includes a prediction for the loss amount in the event of default. Loss is not binary but rather a continuous value and bounded by zero and one as per the training data (i.e. $loss\ [0,1]$). Given the range of the loss value we model it using a regression based approach where in the machine learning literature this simply refers to the use of methods applicable to those response variables that are continuous as opposed to categorical or ordinal. Finally, if one assumes that default and loss amounts are independent than one may predict the former and model the latter conditional upon the occurrence of the former. That is, one may use a binary classification model to determine when a default occurs and then calculate the loss amount using a model which is not statistically coupled where we use one model for default prediction and another for the loss given default ("LGD"). Also, one may use calculate a probability from the classification models (i.e. in lieu of $loss\ [0,1]$) and the product of the probability and loss amount would represent an expected loss which exists for all loans at inception and post-inception.

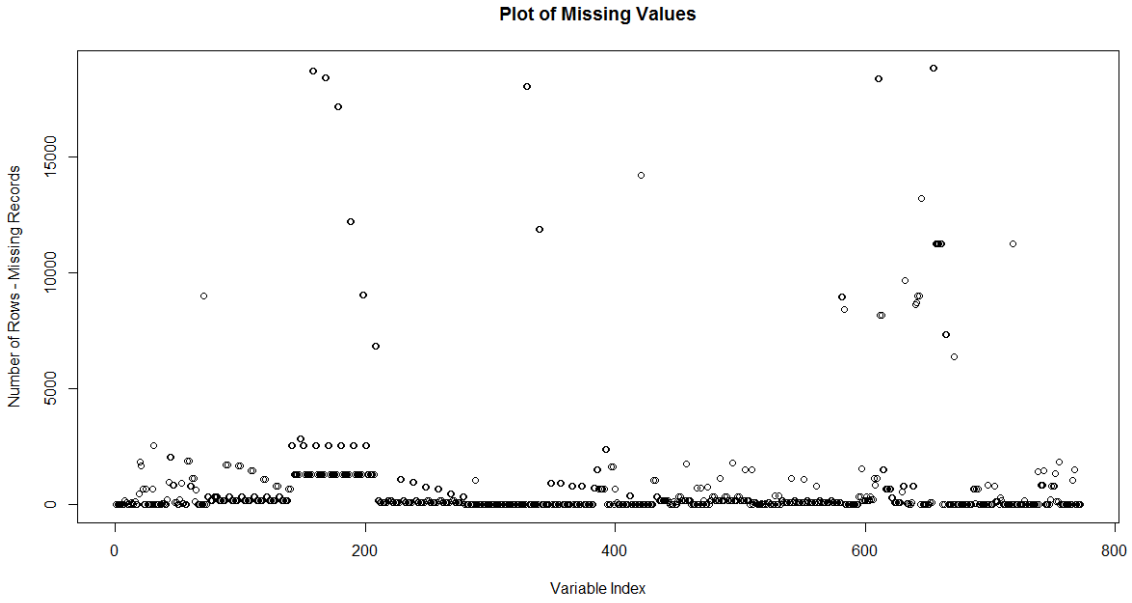
The Data

The data includes 129,494 total records of which there are 105,471 included in a file deemed training and 24,023 entries in a file labeled test for which the latter does not include a loss amount. There are 771 columns included in the training data and, as previously highlighted, we use the loss amount to specify a binary classifier for default which takes the value of zero or

one. Finally, given that there is a column labeled "id" there are 769 features which we may use to fit our models in the training set.

As we treat the test data as future unseen records, we focus herein on the training data. Out of the 105,471 records in the training data, there are 9,783 records for which there are positive loss amounts which constitutes 9.28% of the overall records. Similar to most large data files, there are missing values for several predictors for which we must ultimately decide to impute or discard the feature. In Figure 1 we provide for a plot that highlights those predictors which include missing values. There are two features (i.e. f662 and f663 which may be redundant as the summary statistics for each are extremely similar) for which there are 17.86% of observations missing, 25 predictors which greater than 10% of their observations missing, and 43 where the missing values exceed 5%. Thus far we have decided to impute using the median for each predictor that has missing values. The data is anonymized such that we do not know the exact nature of the predictors.

Figure 1. Plot of missing values for the training data set.



There are 9 features (i.e. f33, f34, f35, f37, f38, f678, f700, f701, and f702) for which the entire columnar values are zeros and two features whose values are constants (i.e. f736, and f764). As such, these 11 features are removed from the dataset.

We look for those predictors whose observations may include so-called outliers. We calculate the distance of the minimum and maximum values respectively, from the mean and normalize by dividing by the predictor's standard deviation. These values are analogous to z-scores and point us towards those features whose minimum or maximum values may appear to be outliers. There are 53, 97, and 119 predictors respectively that have minimum values which are 10, 5, and 4 standard deviations from their means for which the largest deviation is approximately 143 standard deviations in distance from its minimum to its mean value. There are 223, 465, and 533 predictors that have maximum values which are 10, 5, and 4 standard deviations from their means for which the largest is 233 standard deviations in distance from the maximum to the mean when normalized by the standard deviation. We can be highly confident that there are a large number of predictors which have outliers or the series is highly non-normal (i.e. non-zero skewness).

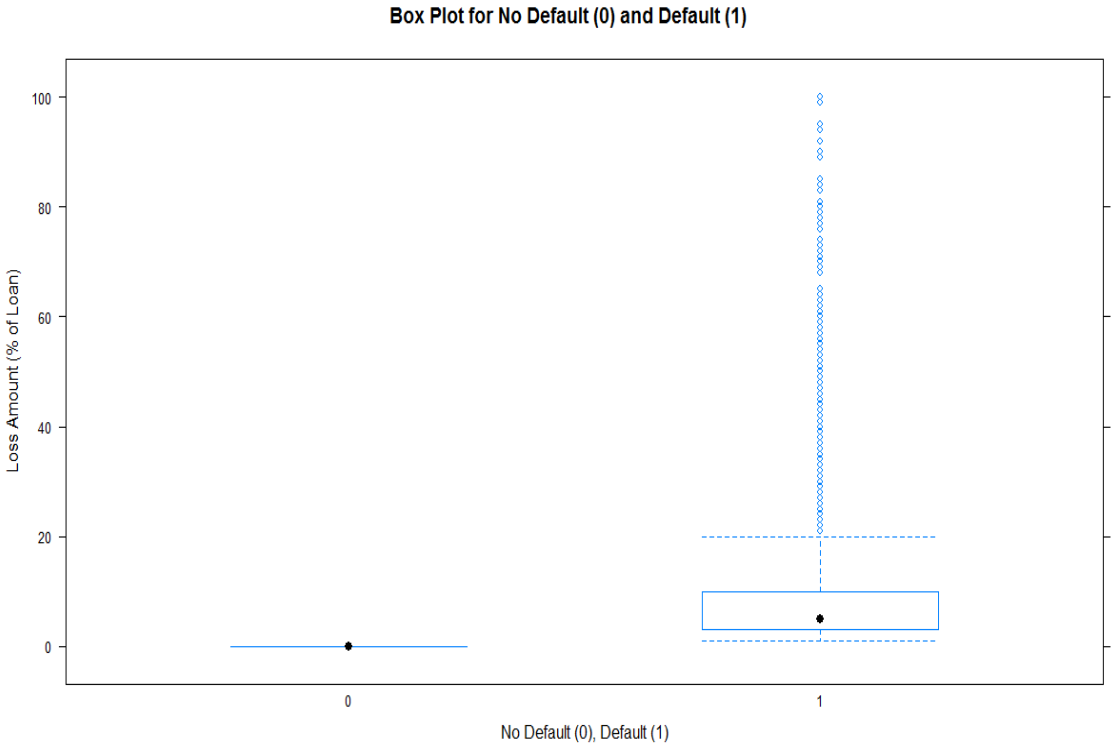
Finally, the data also includes duplicate columns and variables which are highly correlated. While one may get rid of one of each of the pairwise highly correlated variables, we first take a linear combination of these variables to derive so-called "user defined" variables from which we can explore importance. We highlight that these "user defined" variables were discussed on the Kaggle competition discussion board during the time of the competition thereby making them known to all participants. The discussion board is located at https://www.kaggle.com/c/loan-default-prediction/discussion. Logically, one may ask about the meaning of the differences of highly correlated variables and seek to determine meaning. First,

we note that without variable identifiers it is impossible to know the exact nature of the meaning of these differences. Second, it is extremely possible that these differences are quite logical where an example could be that the data includes a variable called income, a variable called expenses, and one's income and expenses may be strongly linearly related where the difference represents a net income which would be important for modeling default and loss. One can easily fathom similar relationships among a dataset that includes such a large number of features.

Exploratory Data Analysis

We begin our EDA by examining the distribution of loss amounts and through its relationship to defaults the prevalence of defaults in the training data. As previously mentioned, out of the 105,471 records in the training data, there are 9,783 records for which there are positive loss amounts which constitutes 9.28% of the overall records. In Figure 2 we present box plots of the loss amounts conditioned on no default (i.e. 0) and default (i.e. 1). As can be seen in Figure 2, loss amounts as a percentage of a loan amount and so bounded by zero and one are generally below 20% with a mean of 8.62%. This suggests that there is collateral such that upon a loss the recovery represents a sizeable amount of the overall loan. It is important to note the loss amount is right-skewed with a large number of values in the 0 – 20% range but the maximum loss is 100%.

Figure 2. Box plots of loss amounts conditioned on no default and default.



In Figure 3 we present a kernel density plot of the loss amount conditioned on the loss value exceeding zero. Figure 3 shows the pronounced right-skew associated with the loss value. To further explore the loss amount in Figure 4 we provide a kernel density of the loss amount conditioned on the amount being greater than zero but less than 40%. As with Figure 3, Figure 4 shows the pronounced right-skew and further highlights that loss amounts are generally less than 20% which amounts to significant recovery rates.

Figure 3. Kernel density of loss amount conditioned on the value being greater than zero.

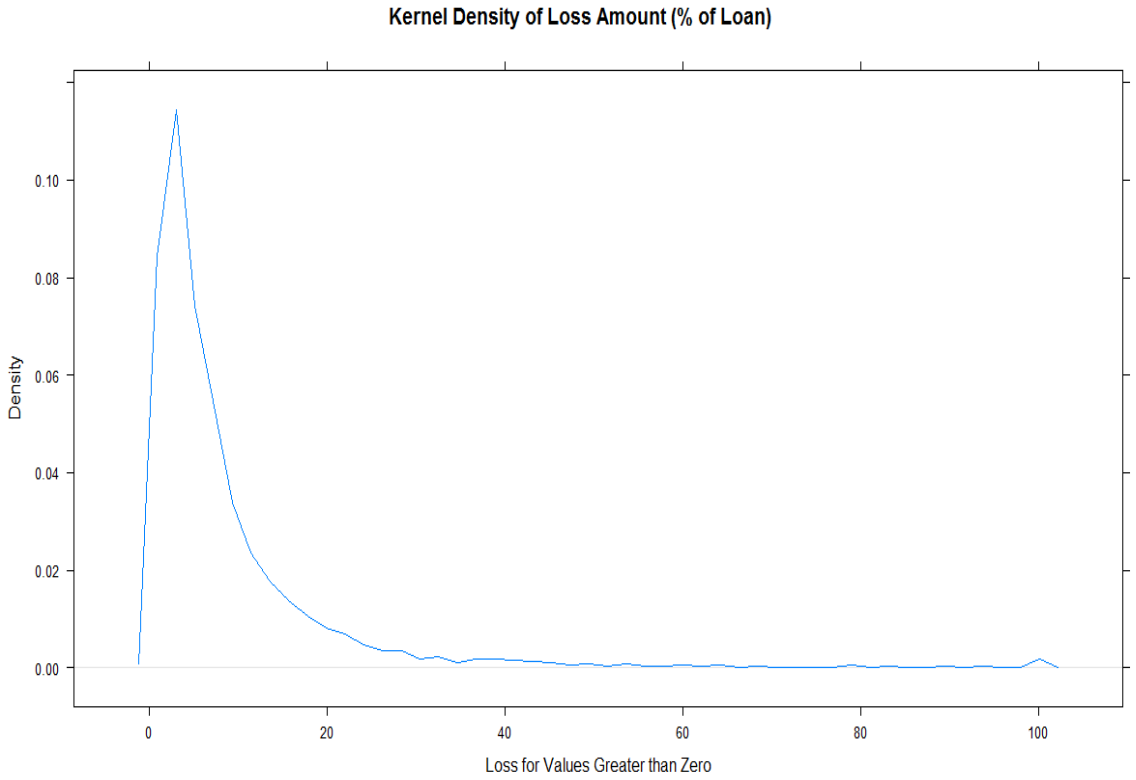
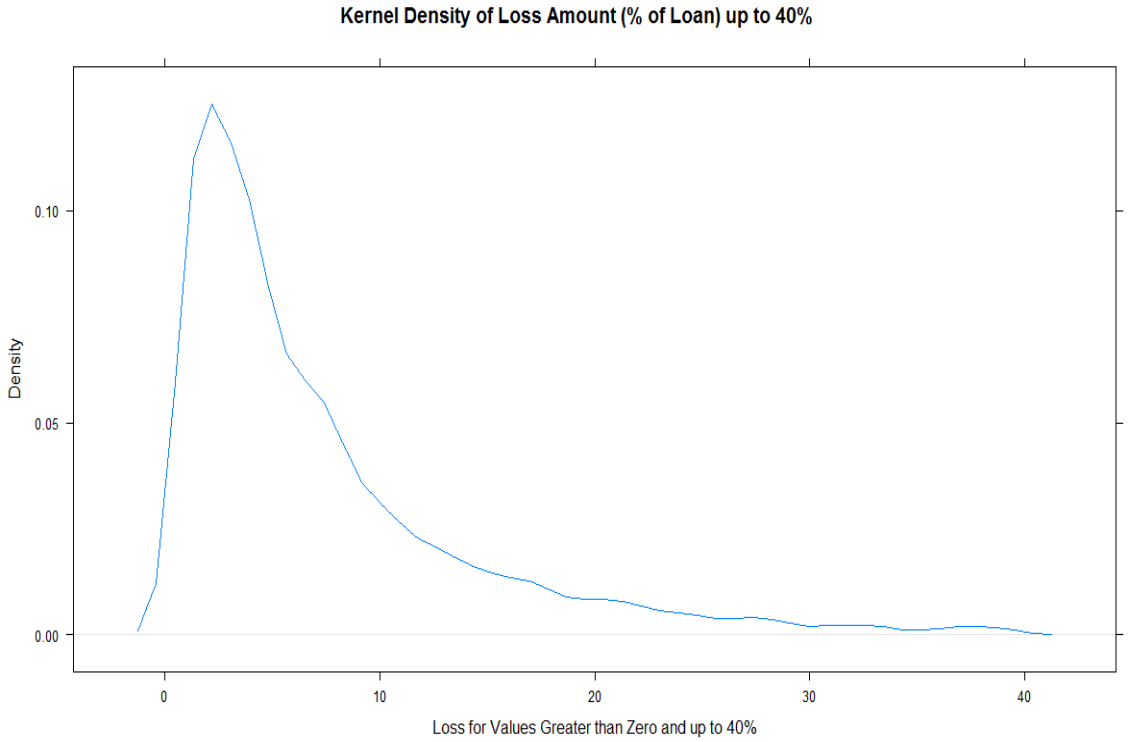


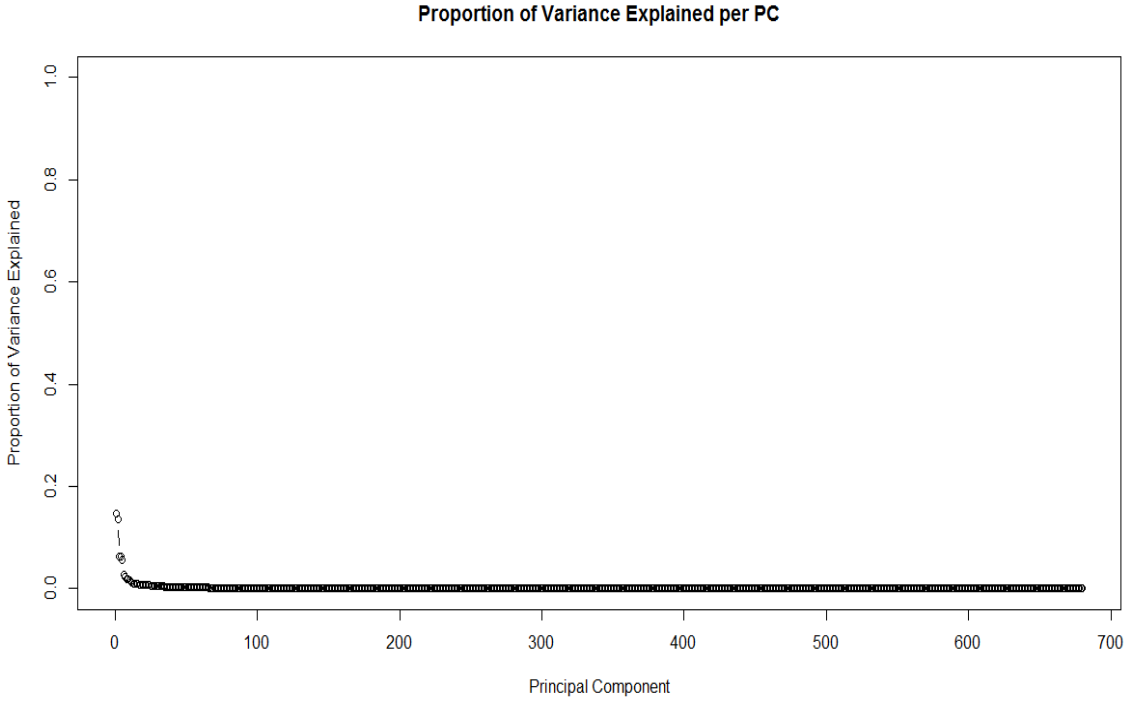
Figure 4. Kernel density of loss amount conditioned on the value being greater than zero and less than 40%.



Given the large dimensionality of our feature space (i.e. 769 predictors), it is natural to explore whether this space may be reduced through Principal Components Analysis ("PCA"). PCA seeks to find the best low-dimensional representation of the variation of the data in a multivariate setting. In short, it is often possible to find a small number of principal components that may then be used to explain the variability of our response variable (i.e. in our case the loan status which we seek to classify) where by construction these components are orthogonal. We use PCA here to see if our predictor space can be reduced significantly to a few principal components that account for a sufficient amount of variability in the predictors. At times the components have an intuitive explanation but one drawback of PCA is that there may be no natural explanation. As is typical for PCA, we first standardize our predictors. In Figure 5 we

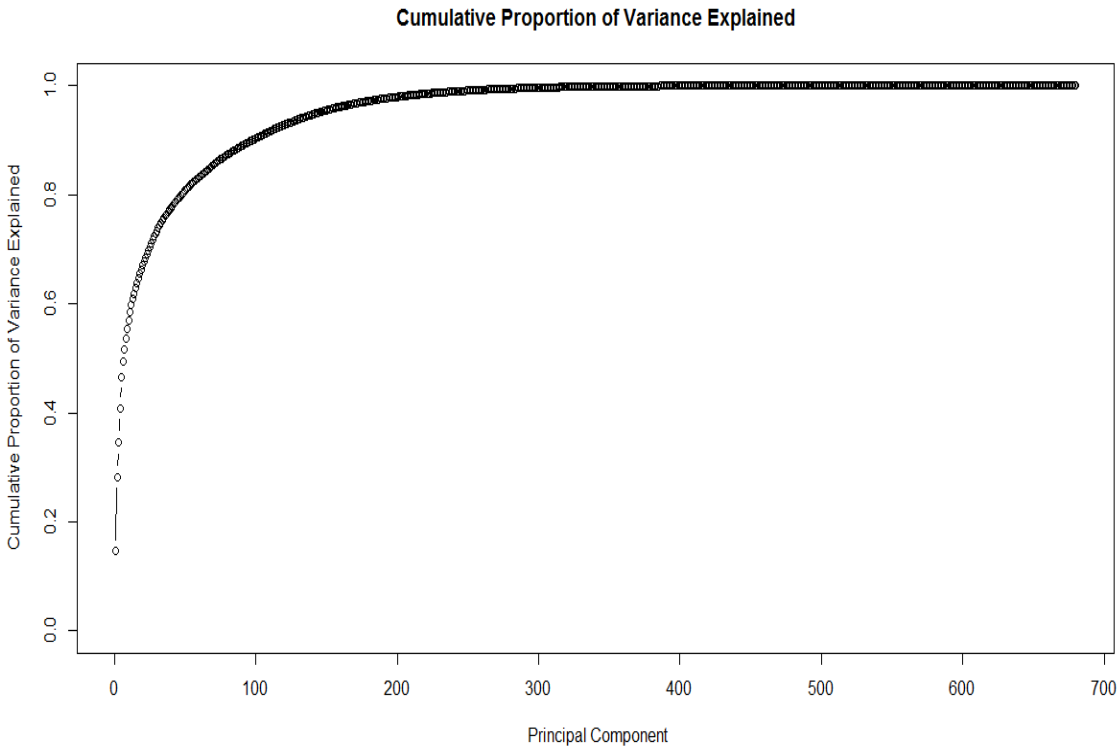
provide for a plot of the proportion of the variance explained by each Principal Component ("PC").

Figure 5. Proportion of variance explained by Principal Components ("PCs").



In Figure 6 we provide for a plot of the cumulative proportion of the variance which may be explained through successive accumulation of the PCs. As one can see, it may be feasible to reduce the dimensionality of the predictor space as the cumulative plot reaches 80% for a small number of components especially when compared to the 769 predictors that characterize the entire feature space. However, as is typically the case with PCA, the resulting PCs do not point towards specific variables or characteristics of the data that are easily interpretable.

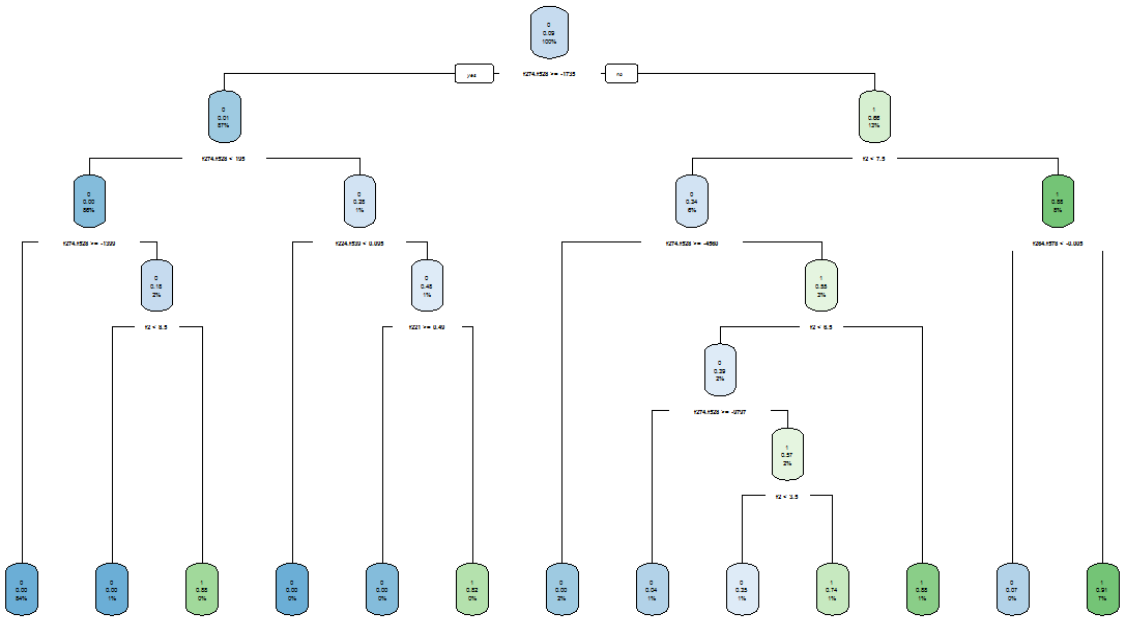
Figure 6. Cumulative proportion of the variance explained through successive Principal Components ("PCs").



Post eliminating variables, imputing, and making transformations we have a total of 2,267 values including loss and default. We first focus on default and using a decision tree and the entire 105,471 records we examine the splits and variable importance from a tree. A decision tree is a multistage classification approach made up of nodes where the terminal nodes representing a class label. The splitting that occurs at a node is driven by a measure such as Entropy, Gini, or Classification error where each are values of impurity where one seeks to minimize this upon a split. The induction process proceeds based on measuring the information gained by making successive splits until all attributes have been taken account of or the increase in information is below some threshold. There are various algorithmic approaches to constructing a decision tree and we leverage those available in the rpart R package. In Figure 7 we provide a plot of a decision tree with default the response variable. As one can see from the

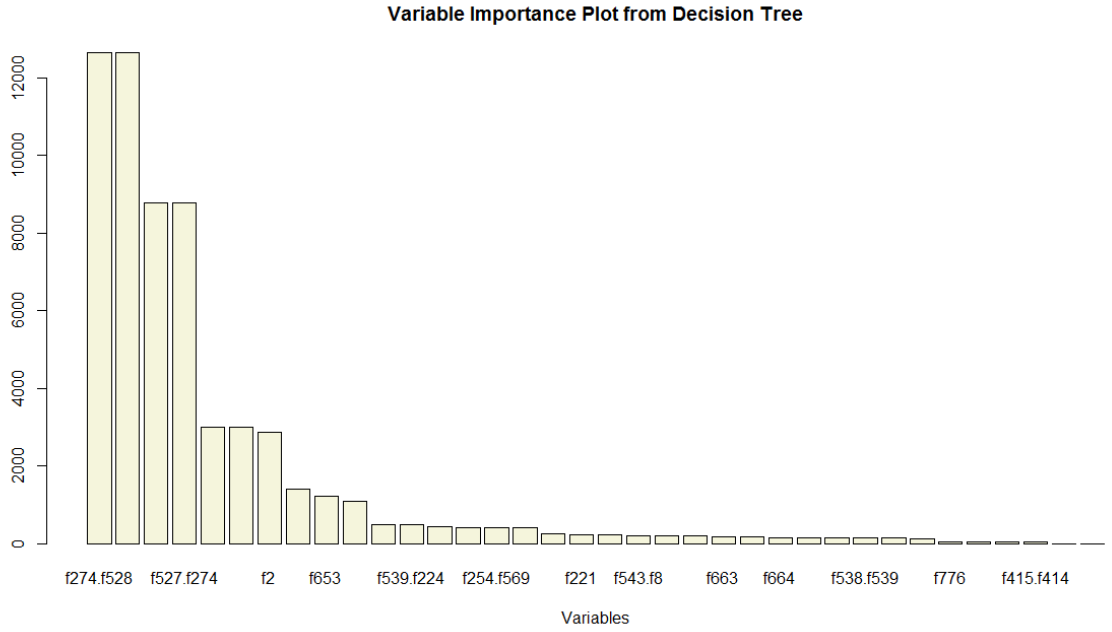
below, are data is first split on predictor f274.f528 and followed by splits on f527.f274 and f527.f528 which are all differences between originally provided variables. As such, we can be reasonably confident that these predictors will prove fruitful in modeling default which is our binary target.

Figure 7. Plot of a decision tree fit on the entire dataset for EDA with default the response variable.



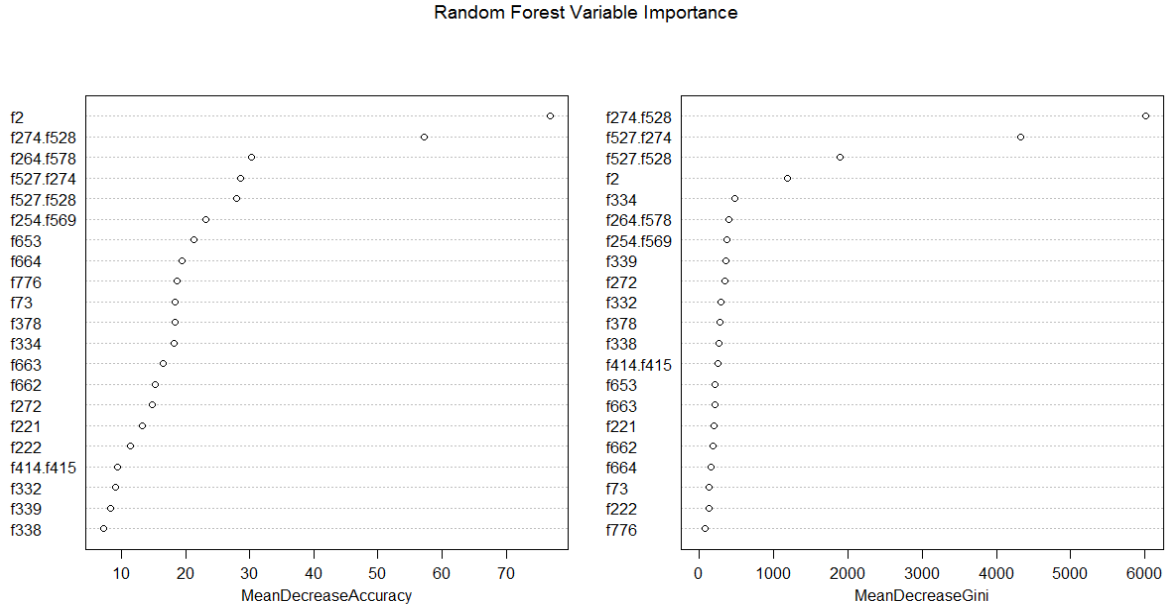
In Figure 8 we provide for a variable importance plot from the decision tree with default the response variable. As one will see, the plot highlights additional variables that may prove useful in predicting default such as f2, f653, f221, f663, f664, and f776. Again, we note that without a data dictionary we do not know the meaning of these variables but rather that they are simply important as per our exploratory analysis.

Figure 8. Variable importance plots for EDA from a decision tree and the entire dataset with default as the response variable.



In Figure 9 we provide a variable importance plot from a random forest model based on 200 trees and with default the response variable. The random forest further supports the importance of select variables.

Figure 9. Variable importance plot for EDA from Random Forest and the entire dataset with default as the response variable.



Finally, as it pertains to default, we use variable selection to fit a logistic regression to determine which variables, from those identified via the random forest, would prove to be statistically significant. In Table 1 we provide the results of the results. The variable names V1, …,V21 are a subset of those from the variables identified as important via the random forest where these are stored in a new data frame and comma separated value file to be loaded and used to fit and evaluate models. From the 21 variables identified as important from random forest, logistic regression shows that 18 of those variables are statistically significant and selected through backwards variable selection.

Table 1. Coefficients from logistic regression using backwards selection with default the response variable.

Coefficients:

	Estimate	z	Significance
(Intercept)	-53.46	2E-16	***
V1	-83.55	2E-16	***
V2	-60.09	2E-16	***
V4	7.38	1.5E-13	***
V5	-6.14	8.51E-10	***
V6	6.14	8.48E-10	***
V7	50.08	2E-16	***
V8	-21.26	2E-16	***
V9	14.87	2E-16	***
V10	28.24	2E-16	***
V11	12.56	2E-16	***
V13	10.24	2E-16	***
V14	-51.77	2E-16	***
V15	4.45	8.73E-06	***
V16	14.84	2E-16	***
V17	-14.26	2E-16	***
V19	3.42	0.00062	***
V20	-10.56	2E-16	***
V21	4.96	6.9E-07	***

Model equation:

as.factor(V22) ~ V1 + V2 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V13 + V14 + V15 + V16 + V17 + V19 + V20+ V21

--

Signif. codes:

0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Similar to default, we examine loss via EDA. Default and loss are inextricably linked and so we expect that variables which may be used to predict the former are possibly related to modeling the latter. In Figure 10 we provide a variable importance plot from a random forest model based on 200 trees and with loss the response variable. As we can see from Figure 10, there are important variables that appear in both the modeling of default and loss where one uses random forest to determine variable importance.

Finally, in Figure 11 we plot the RMSE from random forest as a function of the number of trees with loss as our response variable. As one can see, the RMSE tapers off in the 200 – 300 range for the number of trees used to grow the forest.

Figure 10. Variable importance plot for EDA from Random Forest and the entire dataset with loss as the response variable.

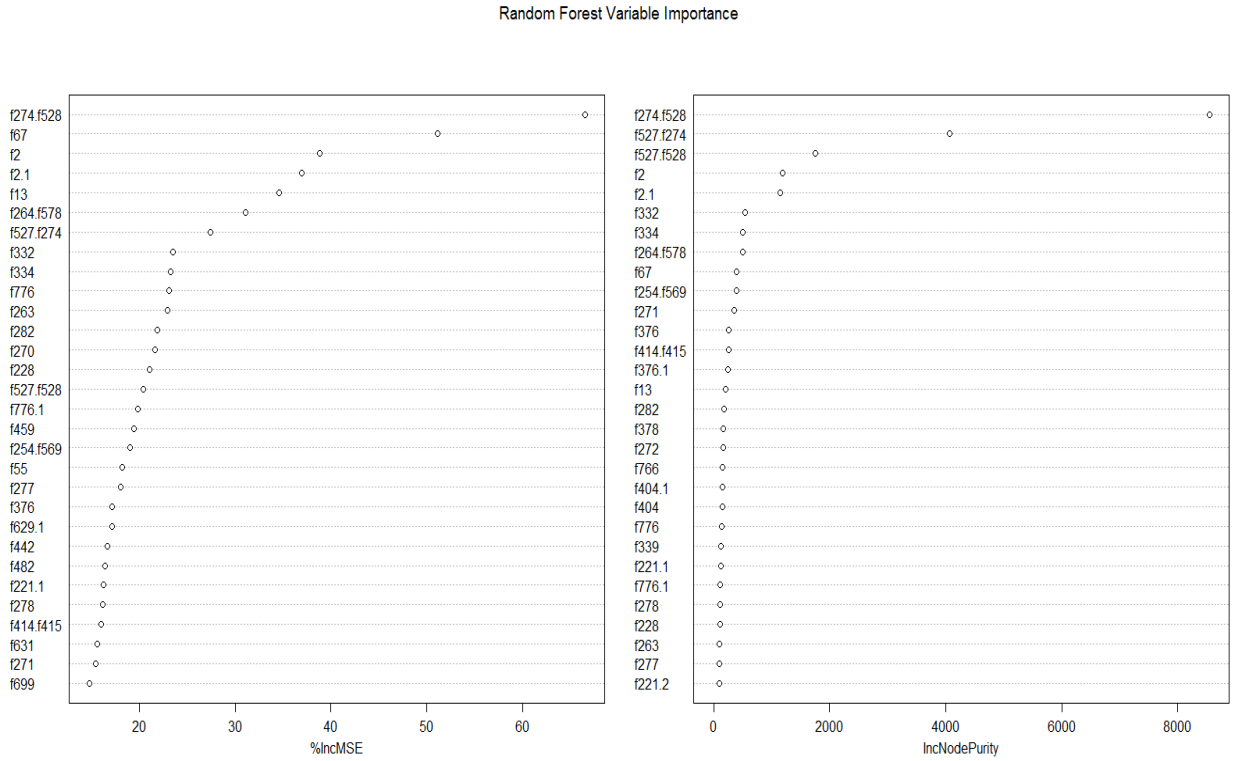
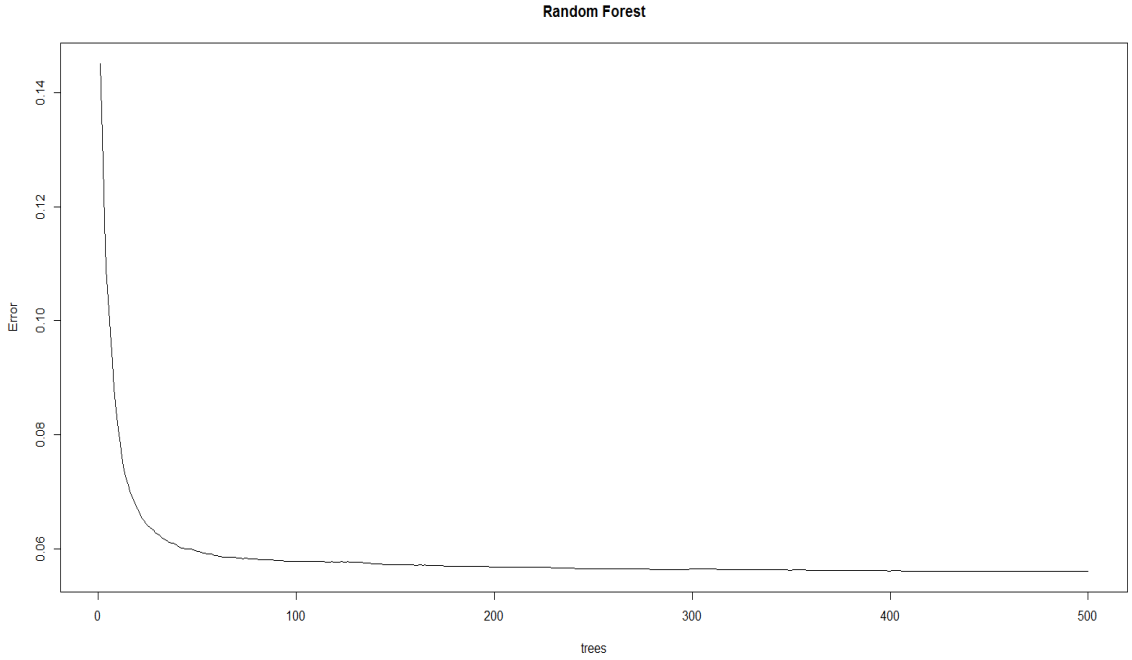


Figure 11. RMSE from Random Forest used for EDA to determine error as a function of number of trees and variable importance with loss as the response variable.



Predictive Modeling: Methods

Default

In what follows we first describe the three main modeling methods which were used thus far in the effort to predict loan default. The description of methods is a synthesis emanating from several resources as included as references. With our training data which has 105,471 records, we create a smaller training dataset and a validation dataset. These datasets have 73,829 and 31,641 records respectively representing a 70/30 split of the larger training dataset. While we have 2,267 possible predictors to fit our models, our EDA has reduced this space considerably as we have identified 18 important predictors to consider for logistic regression, decision tree, and random forests on the training dataset with 73,829 records.

In Table 2 we provide for information around the 70/30 split of the full training dataset into smaller training and validation datasets with our focus on those 18 important predictors from our EDA related to default.

Table 2. 70/30 split of full training data into smaller training and validation datasets and used to model default.

Summary of Data Sets		
Full Training Data Se	[1:105471, 1:2267]	
Records	105,471	
# features	18	
loss > 0	95,688	90.72%
loss = 0	9,783	9.28%
Training Data	Random Split	
Records	73,830	
% of Data	70.00%	
y = yes	66,982	90.72%
y = no	6,848	9.28%
Validation Data	Random Split	
Records	31,641	
% of Data	30.00%	
y = yes	28,706	90.72%
y = no	2,935	9.28%

Logistic Regression

Linear regression is widely used and can be applied in the case of a dichotomous or binary response independent variable. However, if one wishes to apply regression methods whereby the independent variable is of a "yes" or "no" type such that the output would be a probability of "belonging" or "not belonging" to a particular group or "having" or "not having" a particular attribute then a linear regression will produce results but they will suffer from certain shortfalls. Two such shortfalls are particularly salient. First a Bernoulli type outcome of either "yes" or "no" does not satisfy the property of normality. Second, given a simple linear model there is nothing that constrains the resulting output to be bounded by $[0,1]$ and therefore be a valid probability measure. As such, if one wishes to model a dichotomous type independent

variable with the resulting output being a probability one would typically use Logit or Probit type models where here our focus is on the former.

Logistic regression posits a linear relationship between the independent and dependent variables but where the latter is the "log odds ratio." We define the log odds as the natural logarithm of the probability P_i divided by one minus P_i and call this the logit which represents the logistic or logit transformation.

$$L_i = \left[\frac{P_i}{(1 - P_i)}\right]$$

We relate the natural logarithm of the odds ratio to our right-hand-side (RHS) linear model and upon simplification we have a simple linear relationship between the logit L_i and the RHS intercept (β_0) and product of β_1 and X_1. An example of simple univariate model is:

$$L_i = \left[\frac{P_i}{(1 - P_i)}\right] = \beta_0 + \beta_1 \cdot X_1$$

In our example, what we have done is specified a model that is linear in logits but non-linear in terms of probability which may be recovered based on the relationship of odds to probability and taking relationship between the exponential and natural logarithm where the probability P_i is related to L_i as:

$$P_i = \frac{e^{L_i}}{(1 + e^{L_i})} = \frac{1}{(1 + e^{-L_i})}$$

Where for our univariate example we have that $L_i = \beta_0 + \beta_1 \cdot X_1$. Thus the logistic model is linear in log odds but non-linear in probability but allows us to do regression or logistic regression on binary response variables and express our results in terms of proper probabilities

through the transformation from logits which are unbounded $(-\infty, \infty)$ to probabilities $[0,1]$ which are bounded. This transformation allows us to capture the non-linearity between the RHS regression equation and the probabilities. It is easy to see that P_i is bounded by $[0,1]$ as:

$$\lim_{L_i \to -\infty} \left[\frac{1}{(1 + e^{-L_i})}\right] = 0$$

$$\lim_{L_i \to \infty} \left[\frac{1}{(1 + e^{-L_i})}\right] = 1$$

And, given the nature of the function we can also see that changes in probability are larger in the middle of the range and smaller at the extremes. There are two important items worth further discussion. First, when we solve for parameters via simple linear regression we minimize the sum of the squared errors which can be done using basic calculus and first and second order conditions (i.e. first derivative set to zero and sign of function at second derivative should be positive for a minimum). In contrast, when we solve for the parameters for logistic regression we use Maximum Likelihood Estimation ("MLE") and maximize the log of the likelihood function. Put succinctly, when we use MLE we solve for those model parameters that maximize the likelihood of having produced the data which we observe. We work in log space as by the rules of logs we turn products into summations and the natural log is monotonic making the maximum invariant to working in the original or log space. Second, with ordinary regression we have a wealth of statistics that help us with interpretation of the results. With logistic regression we get coefficients where for the Beta (β_1) tells us the change in the log odds for a one unit change in the independent variable. To go to probabilities we need to make the appropriate transformations and, as mentioned earlier, while the model is linear in log odds it is non-linear in probabilities. So, we must be careful in our interpretation of the output and define

a threshold in probability space that defined the boundary between no default and default for our loan modeling problem.

Decision Trees and Random Forests

A decision tree is a multistage classification approach made up of nodes where the terminal nodes representing a class label. Decision trees are a form of supervised learning which is part of the larger set of data mining techniques. In their simplest form a decision tree is an induction method whereby a lattice is constructed based on a series of binary or multi-way splits. The goal of the decision tree is to classify data items. Decision trees have found wide use in areas such as biology, ecology, medicine, psychology, finance (credit scoring), and other areas of applied sciences.

Decision trees are composed of a root node, internal nodes, and leaf nodes where the size of the tree is based on algorithmic considerations and the splitting associated with node formation generally a function of minimizing an impurity measure such as the entropy or Gini values. Given their formulaic expressions, these impurity measures are greatest for equal probability type outcomes (e.g. 50/50 outcome such as toss of fair coin) which imply a split that does not serve to enhance classification. As such, the goal of measuring impurity is to provide for appropriate value-added splits where there is an information gain as a result of the split and growing the tree forward.

In general, one may stop growing a decision tree based on stopping criteria where any further growth results in an information gain as measured by the change in impurity from a parent to child node that is set to a constant. Alternatively, one may grow a decision tree "completely" and subsequently go back and prune the tree by examining smaller variations such

that there is minimal loss in explanatory power with respect to a "hold-out or out-of-sample" set of data.

On the entropy and Gini measures, they are both formulaic expressions of impurity and are similar in that their first and second derivatives are the same sign over the $[0,1]$ probability domain (e.g. the both reach maximums with zero first derivative at a value of .5 where to the left of the maximum they both have positive first derivatives and negative second derivatives and to the right of.5 they both have negative first derivatives and negative second derivatives). They are both concave functions with one difference their rate of change where entropy increases and decreases at a faster rate. By examining the derivatives of the respective functions, one can see that over much of the $[0,1]$ domain tests based on these measures would yield similar conclusions. It is at the extremes that they differ.

Random forests are at their core ensembles of decision trees. Random forests are based on the concept of bagging whereby successive trees do not depend on earlier trees as they are constructed using a bootstrapped sample of the dataset. Based on the bootstrapping the trees tend to be uncorrelated or, at the least, not very highly correlated which results in a reduction in variance. In addition to bootstrapping, random forests include an additional layer of randomness where at each node there is a split using the best among a subset of predictors that are randomly chosen at the node. Thus, random forests are an extension of a decision tree where there are two key additional parameters that include the number of variables in the random subset for splitting and the number of trees in the forest. Finally, with random forests for each bootstrap iteration there is data that is not included in the sample and deemed "out of bag" or OOB. The OOB data is used for purposes of calculating error rates and to estimate the importance of a variable by

looking at how much the prediction error increases when OOB data for the variable is permuted while all others remain unchanged.

Loss

For loss modeling, in addition to using random forests on the full predictor space and plotting the variable importance, we also use lasso, and gradient boosting with 20,000 records to assist in reducing the number of possible predictors. Ultimately, we use 89 variables to model loss where we transform loss to log-space (i.e. we take natural log of one plus the loss amount and define our predictor) for our regression based methods fit for prediction. The methods that we use include linear regression with step-wise selection, ridge regression and Lasso, gradient boosting, and random forest. In what follows we first describe the regression and gradient boosting methods which are used to model our log-transformed loss. Given our previous description of random forest we refer the reader to that section for that approach. As with our default modeling, the description of methods is a synthesis emanating from several resources that are included as references.

In Table 3 we provide for information around the 70/30 split of the full training dataset into smaller training and validation datasets with our focus on those 89 important predictors from our EDA related to loss.

Table 3. 70/30 split of full training data into smaller training and validation datasets and used to model loss.

Summary of Data Sets		
Full Training Data Se	[1:105471, 1:2267]	
Records	105,471	
# features	89	
loss > 0	95,688	90.72%
loss = 0	9,783	9.28%
Training Data	Random Split	
Records	73,830	
% of Data	70.00%	
y = yes	66,982	90.72%
y = no	6,848	9.28%
Validation Data	Random Split	
Records	31,641	
% of Data	30.00%	
y = yes	28,706	90.72%
y = no	2,935	9.28%

Linear Regression with Subset Selection

Linear regression is a workhorse in modern statistical modeling, is well known, and we therefore do not provide exhaustive details here. Linear regression is often quite competitive in performance in comparison to other methods of prediction and the method does provide advantages in terms of interpretability and ease of use. A main aspect of the model that can affect both of these is the number of observations and predictors used in the linear regression model. Particularly, if the number of observations is not significantly larger than the number of predictors when using least squares there can be a lot of variability in the fit. Also, interpretability is reduced as complexity is added through the use of unnecessary variables. Hence techniques such as subset selection and shrinkage can be used to yield more interpretable and better performing models.

With variable subset selection, one uses some measure of performance to compare model performance both before or after a variable is included in a model to determine if there is enough of an increase, as determined by some threshold parameter setting, to warrant including this variable in the final model. It is easy to say that one should just try every possible combination of variables until able to find the best subset and arrive at an ideal model. Usually this is handled by a computer and performed automatically with appropriate settings specified for the algorithm to be run. However, it can be easily be seen that a "best-subsets" approach can become computationally expensive in the case of a large number of predictors as there are 2^p subsets to choose from to find the best possible subset. Further, with an increasingly large search space, there is also an increasing probability of finding models that 'look good on the training data, but have no predictive power on future data' (i.e. they are over-fit and do not generalize well). Various methods have been designed to more efficiently search through a more restricted set of models.

For our loss modeling, we employ a type of automatic variable subset selection called stepwise selection. This method allows us to add or delete variables at each step. Backwards stepwise selection starts with all predictors and then deletes one of these. It then tests the performance of the model. It will then delete another and test the performance again. Before deleting the third, it will check to see if adding the first one back in improves the performance criterion. It may then either leave it in or delete it again before moving on to the next variable. The algorithm proceeds in this bidirectional manner, each time choosing the least significant variable and checking to see if all but the most recently removed variable may affect the judgement criterion in a statistically significant way. We must set two significance parameters to

use this algorithm. One for deletion from the model and one more stringent setting for adding a parameter back into the model.

Lasso and Ridge Regression

Both Lasso and ridge regression are deemed shrinkage methods in the literature. Shrinkage involves using all the predictors and a technique that constrains the coefficients of the predictors, shrinking them towards zero and giving us another method of reducing or even removing (i.e. by actually making the coefficient zero) the effects of the least significant variables from the model. This also has the effect of improving the variance of the model coefficients. We used both of the two best known shrinkage methods which include the Lasso and ridge regression in our tests of potential loss modeling methods.

Lasso regression helps to overcome certain weaknesses associated ridge regression. In contrast to ridge regression the Lasso uses a slightly different shrinkage penalty that, when large enough, forces the coefficient to zero. The Lasso specification is given by the below where the penalty function includes a scalar multiplier and serves to constrain the sum of the absolute value of model coefficients.

$$\min\left(\sum_{i=1}^{n}\left(y_i + \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right)$$

Given the form of the model, it is arguably an algorithm that performs variable subset selection. Further, the Lasso yields sparse and more easily interpretable models than basic linear regression and ridge regression.

Ridge regression is similar to least squares regression except that instead of just looking to minimize the residual sum of squares, it seeks to minimize the residual sum of squares plus a shrinkage penalty.

$$\min\left(\sum_{i=1}^{n}\left(y_i + \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right)$$

The first portion can be recognized as the residual sum of squares and the latter is the shrinkage penalty with tuning parameter λ applied only to the coefficients for the predictors. As $\lambda \to 0$ the penalty is reduced and the coefficients are allowed to grow, where $\lambda = 0$, this will produce the least squares estimates for the coefficients. As $\lambda \to \infty$, the penalty is increased and the coefficients are reduced to towards zero and leaving a null model (i.e. intercept only). The idea is to test various values of the tuning parameter using cross validation to arrive at a best model. However, none of the coefficients can ever actually be set to zero by this algorithm unless $\lambda = \infty$, which can be a problem for interpretability with a large number of predictors.

Gradient Boosting

This is similar to random forests in that it is a machine learning technique that uses an ensemble of other models. It is a means primarily of improving the prediction ability of a relatively weak learner. But instead of building each model of the ensemble on the training data, only the first model is built directly on the data. The next model is built on loss function values of each prediction from the initialization model and the actual training data. In practice the first "model" can be a simple median or average of the response under consideration. It is just a starting point from which we can determine our first set of loss calculations. The prediction result for each observation is compared with the initialization model prediction and then another

loss value is produced. This process is then iterated until the improvement in loss value on the next iteration is sufficiently small. As with random forest, boosting (i.e. a common resampling method) and cross validation are also normally implemented with this algorithm.

The gradient portion of this algorithm is implemented by the computation of a minimum step magnitude multiplier that is used to help moderate the change in the next model. In decision trees, this multiplier can be different for each node in the tree. Once the gradient γ is arrived at, it is used as to update the subsequent prediction.

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{Jm} \gamma_{jm} I\left(x \in R_{jm}\right)$$

The output of the model is simply: $\hat{f}(x) = f_M(x)$. The two basic tuning parameters are the number of iterations *M* and the sizes of the constituent trees J_m , $m = 1,2, \ldots, M$.

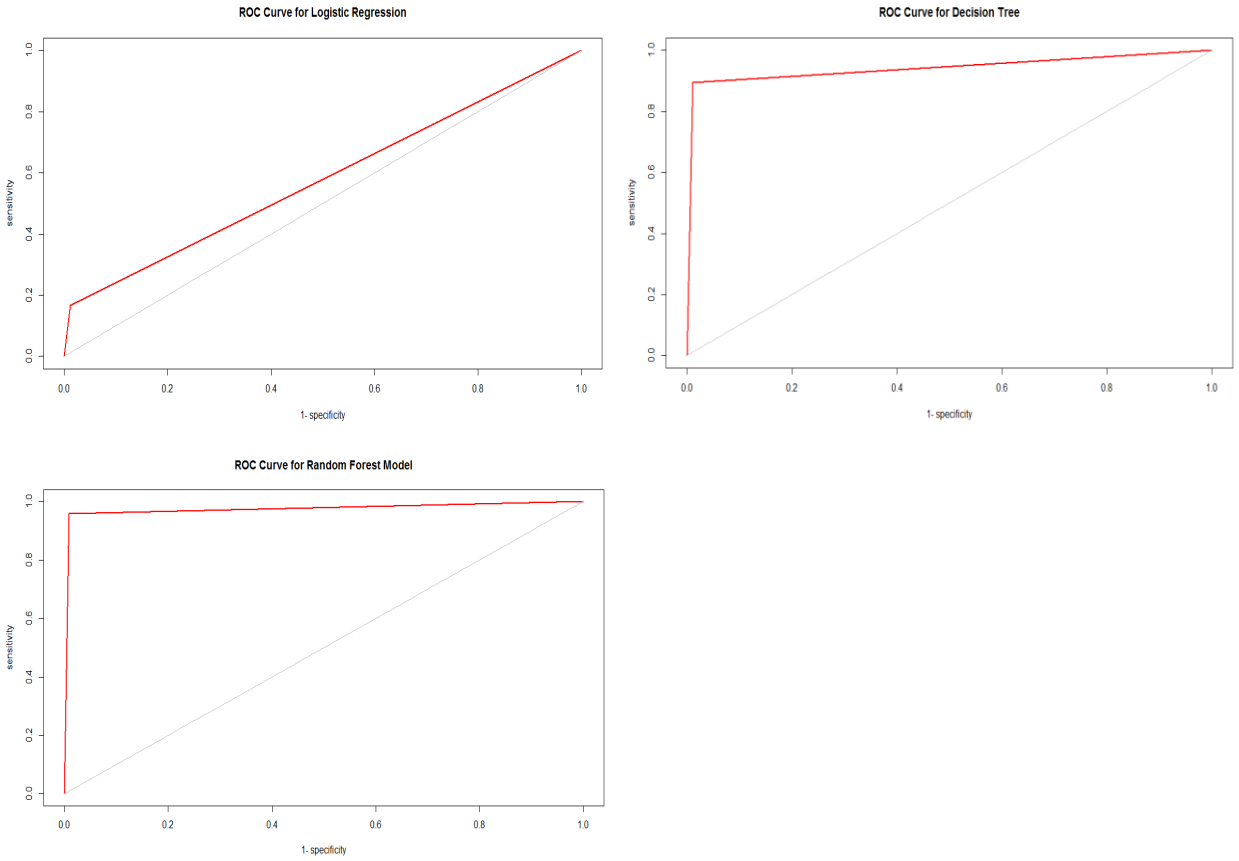
Comparison of Results

Default

Receiver Operating Characteristic ("ROC') curves are a useful tool to select classification models based on their performance with respect to sensitivity (or true positive rate) and one minus specificity (or true negative rate) where the 45 degree line represents random guessing. In short, a ROC curve is a visualization of the tradeoff between true positives and negatives where increasing the former will generally come at the expense of increasing the latter and decreasing the latter would be based on decreasing the former. Modelers face tradeoffs and in the absence of a perfect classifier may be able to enhance a particular evaluation measure at the expense of

another measure. In Figure 11 we provide ROC curves for each of our classifiers which include logistic regression, decision tree, and random forest (i.e. moving clockwise from the upper left hand corner). As one can see, the random forest is extremely effective and, as such, results in a classifier that does not require modifying to increase the true positives or true negatives. We note that the Area Under the Curve ("AUC") is 97.69% for the random forest fit on training data and used to predict on our validation data. The in-sample AUC for random forest is 100%.

Figure 12. ROC curves for each of Logistic Regression, Decision Tree, and Random Forest models fit on training data and used to predict out-of-sample on the validation data.



In Table 4 we provide confusion matrices for each of our classification methods with model evaluation measures from each of the training and validation datasets so that one can see the in-sample and out-of-sample values. As one can see from Table 3, the logistic, decision tree, and random forest methods perform similarly on the training (LHS) and validation (RHS)

datasets and do not point towards overfitting on the training data. The random forest does not predict default perfectly out-of-sample, but increasing the sensitivity would only come at the expense of decreasing the specificity where both are at more than acceptable levels for our problem of predicting default.

Table 4. Confusion Matrix for training and validation datasets with the latter the out-of-sample data for each of Logistic Regression, Decision Tree, and Random Forest.

Logistic Regression - Fit Train, Results Train

		Observed	
	Train	No Default	Default
Predicted	No Default	66229	5667
	Default	753	1181
TP Rate		98.88%	
FP Rate		82.75%	
Precision (Sensitivity)		92.12%	
Specificity		17.25%	

Logistic Regression - Fit Train, Results Validation

		Observed	
	Train	No Default	Default
Predicted	No Default	28382	2442
	Default	324	493
TP Rate		98.87%	
FP Rate		83.20%	
Precision (Sensitivity)		92.08%	
Specificity		16.80%	
AUC		57.83%	

Decision Tree - Fit Train, Results Train

		Observed	
	Train	No Default	Default
Predicted	No Default	66257	702
	Default	725	6146
TP Rate		98.92%	
FP Rate		10.25%	
Precision (Sensitivity)		98.95%	
Specificity		89.75%	

Decision Tree - Fit Train, Results Validation

		Observed	
	Train	No Default	Default
Predicted	No Default	28386	308
	Default	320	2627
TP Rate		98.89%	
FP Rate		10.49%	
Precision (Sensitivity)		98.93%	
Specificity		89.51%	
AUC		94.20%	

Random Forest - Fit Train, Results Train

		Observed	
	Train	No Default	Default
Predicted	No Default	66982	0
	Default	0	6848
TP Rate		100.00%	
FP Rate		0.00%	
Precision (Sensitivity)		100.00%	
Specificity		100.00%	

Random Forest - Fit Train, Results Validation

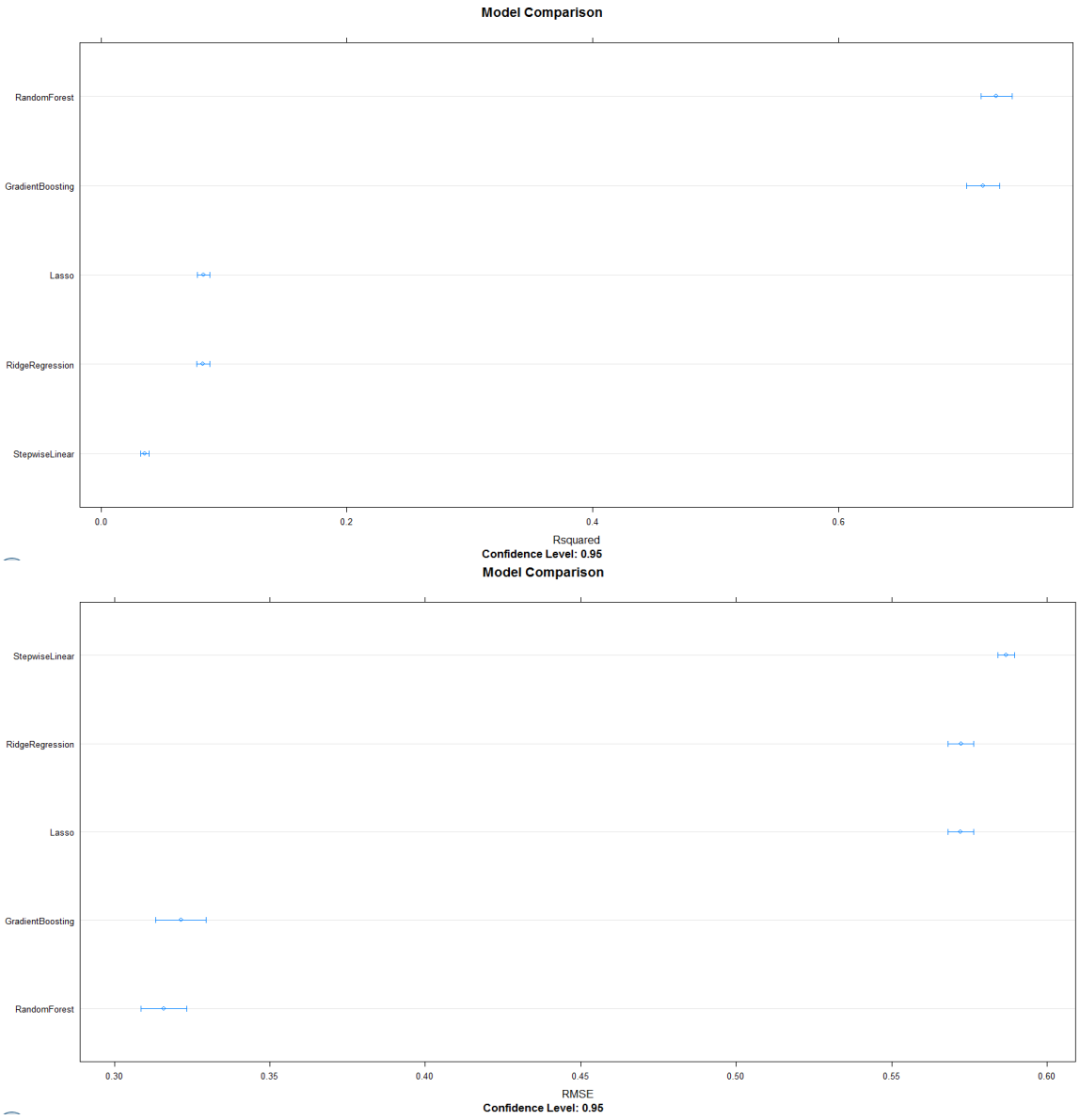
		Observed	
	Train	No Default	Default
Predicted	No Default	28453	114
	Default	253	2821
TP Rate		99.12%	
FP Rate		3.88%	
Precision (Sensitivity)		99.60%	
Specificity		96.12%	
AUC		97.69%	

Loss

In contrast to default which is a classification problem, loss is modeled via regression based methods and therefore model evaluation is based on measures such as R-squared and

RMSE. In Figure 13 we provide for the R-squared and RMSE for each of the loss modeling methods employed.

Figure 13. R-squared and RMSE for various modeling methods with loss as the response variable.



As one can see from Figure 13, random forest has the highest R-squared and lowest RMSE. The model ordering remains the same when one considers R-squared and RMSE with random forest

the best followed by gradient boosting, Lasso, ridge regression, and linear regression with stepwise variable selection.

Finally, in Table 5 we provide for R-squared and RMSE value for all methods fit on training data with evaluation measures from training and out-of-sample from our validation dataset. As evident from Table 5, random forest and gradient boosting significantly outperform the more standard statistical regression based approaches even with the penalty functions to constrain the model coefficients via Lasso and ridge regression. The performance of random forest is likely driven by the aggregation of trees, resampling, and bagging where the underlying decision trees can easily accommodate data that is correlated. The gradient boosting augments model building by adjusting 'residuals' to continuously improve prediction. Both of the random forest and gradient boosting methods are far superior to the alternatives with random forest the model of choice.

Table 5. Table of RMSE and R-squared values for loss modeling methods fit on training data.

	Training Sample		Validation Sample	
Loss Modeling Method	**RMSE**	**R-squared**	**RMSE**	**R-squared**
Random Forest	0.2374	0.8411	0.2397	0.8461
Gradient Boosting	0.2798	0.7818	0.2860	0.7806
Lasso	0.5722	0.0833	0.5814	0.0805
Ridge Regression	0.5722	0.0831	0.5815	0.0801
Stepwise Linear Regression	0.5869	0.0384	0.5954	0.0362

Conclusions

From a modeling perspective, the random forest proves superior to all other approaches considered in modeling default and loss. Random forest generalizes well in both cases including the classification and regression associated with predicting default and loss in our out-of-sample validation dataset. This work has demonstrated the importance of machine learning techniques that go beyond the more standard statistical based approaches. While machine learning algorithms may, at times, be more opaque, if the goal is prediction as opposed to inference, these methods are extremely useful. As such, in modeling default or loss or even in extracting probabilities to be combined with loss amounts in modeling expected loss, we would recommend the use of random forest for the data under consideration in this work.

The data used herein is from Imperial College London and was used in a Kaggle competition. Similar to others that have worked with this data (see Kaggle discussion boards) we found the lack of a data dictionary frustrating as eliminates any use of domain knowledge even by novices. The exercise amounts to a 'blind' machine learning competition which is generally not the way that data scientists operate. While data scientists and predictive analytics professionals are often unconcerned with inference and possibly fully focused on out-of-sample prediction, they are typically reasonably informed around the underlying data which may be messy and necessitate cleaning and transformations to obtain satisfactory results. But, it is rare that analysts 'fly totally blind' in using fully anonymized data in an effort to fit and generalize a model for prediction.

However, even under such harsh conditions random forests proved extremely useful and served to result in models that used in isolation to predict default or loss result in good out-of-sample performance and used in conjunction could be used to estimate expected loss.

Bibliography

1. Flach. Machine Learning: *The Art and Science of Algorithms that Make Sense of Data.* First edition, 2012, Cambridge University Press.

2. Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second edition, 2009, Springer.

3. James, Witten, Hastie, and Tibshirani. *An Introduction to Statistical Learning with Applications in R.* First edition, 2014, Springer.

4. Kuhn, and Johnson. *Applied Predictive Modeling.* First edition, 2013, Springer.

5. Larose, and Larose, C. *Data Mining and Predictive Analytics.* Second edition, 2015, John Wiley & Sons.

6. Ledolter. *Data Mining and Business Analytics with R.* First edition, 2013, John Wiley & Sons.

Appendix – R Code

```
# Northwestern University
# PREDICT 454
# Loan Default Prediction

# Install packages
# Can comment out (#) if installed already
install.packages("stats", dependencies = TRUE)
install.packages("pastecs", dependencies = TRUE)
install.packages("psych", dependencies = TRUE)
install.packages("lattice", dependencies = TRUE)
install.packages("plyr", dependencies = TRUE)
install.packages("corrplot", dependencies = TRUE)
install.packages("RColorBrewer", dependencies = TRUE)
install.packages("caTools", dependencies = TRUE)
install.packages("class", dependencies = TRUE)
install.packages("gmodels", dependencies = TRUE)
install.packages("C50", dependencies = TRUE)
install.packages("rpart", dependencies = TRUE)
install.packages("rpart.plot", dependencies = TRUE)
install.packages("modeest", dependencies = TRUE)
install.packages("randomForest", dependencies = TRUE)
install.packages("caret", dependencies = TRUE)
install.packages("MASS", dependencies = TRUE)
install.packages("e1071", dependencies = TRUE)
install.packages("ggplo2", dependencies = TRUE)
install.packages("readr", dependencies = TRUE)
install.packages("AUC", dependencies = TRUE)

# Include the following libraries
library(stats)
library(pastecs)
library(psych)
library(lattice)
library(plyr)
library(corrplot)
library(RColorBrewer)
library(caTools)
library(class)
library(gmodels)
library(C50)
library(rpart)
library(rpart.plot)
library(modeest)
library(randomForest)
library(caret)
library(MASS)
library(e1071)
library(ggplot2)
library(readr)
library(AUC)

# Get list of installed packages as check
search()

# Variable names start with Capital letters and for two words X_Y
# Examples: Path, Train_Data, Train_Data_Split1, etc.

# Set working directory for your computer
```

```
# Set path to file location for your computer
# Read in data to data file
setwd("C:/Users/Stephen D Young/Documents")
Path <- "C:/Users/Stephen D Young/Documents/Stephen D. Young/Northwestern/Predict 454/Project/Train Data/train_v2.csv"
# setwd("C:/Users/IBM_ADMIN/Northwestern/454/Project/train_v2.csv/")
# Path = ("C:/Users/IBM_ADMIN/Northwestern/454/Project/train_v2.csv/")

Train_Data <- read.csv(file.path(Path,"train_v2.csv"), stringsAsFactors=FALSE)

# Get structure and dimensions of data file
str(Train_Data)
dim(Train_Data)

# Characteristics of data frame from above
# 105471 rows(records), 771 columns, 770 not including loss which is last column
# there is no indicator variable for default but can be 0 or 1 for when there is
# an observed loss.  So we can create binary default flag.
# Summary statistics for loss variable
# options used to define number and decimal places
options(scipen = 100)
options(digits = 4)
summary(Train_Data$loss)
# Get mode using mlv from modeest package
mlv(Train_Data$loss, method = "mfv")
# Results of loss are min = 0, max = 100, mean = .8, mode = 0
# Loss is not in dollars but as percent of loan amount (e.g. 50 is loss of
# 50 out of 100 loan amount)

# Check for missing values in loss column
sum(is.na(Train_Data$loss))
# There are no missing values in loss column

# Add in default indicator and determine number and proportion of defaults
# Use ifelse to set default to 1 if loss > 0 otherwise default is 0
# Get table of 0 and 1 values (i.e. no default, default)
Train_Data$default <- ifelse(Train_Data$loss>0,1,0)
table(Train_Data$default)
dim(Train_Data)

# Box plot and density plot of loss amount
# Lattice package for bwplot and densityplot
# Boxplot is for no default (0) and default (1)
# Boxplot should have zero loss for no default and range of loss upon default
bwplot(loss~factor(default), data=Train_Data,
      main = "Box Plot for No Default (0) and Default (1)",
      xlab = "No Default (0), Default (1)",
      ylab = "Loss Amount (% of Loan)")

# Density plot of loss should skew right as 100% loss is max but uncommon
densityplot(~(loss[loss>0]), data = Train_Data, plot.points=FALSE, ref=TRUE,
          main = "Kernel Density of Loss Amount (% of Loan)",
          xlab = "Loss for Values Greater than Zero",
          ylab = "Density")

# Density plot of loss should skew right even with 40% as max value for plot
densityplot(~loss[loss>0 & loss<40], data = Train_Data, plot.points=FALSE, ref=TRUE,
          main = "Kernel Density of Loss Amount (% of Loan) up to 40%",
          xlab = "Loss for Values Greater than Zero and up to 40%",
          ylab = "Density")

# Basic summary statistics for loss values when greater than zero
# Get mode using mlv from modeest package
```

```
summary(Train_Data$loss[Train_Data$loss>0])
mlv(Train_Data$loss[Train_Data$loss>0], method = "mfv")
# Results of loss are min = 1, max = 100, mean = 8.62, mode = 2

# Function to compute summary statistics
myStatsCol <- function(x,i){

  # Nine statistics from min to na
  mi <- round(min(x[,i], na.rm = TRUE),4)
  q25 <- round(quantile(x[,i],probs = 0.25,  na.rm=TRUE),4)
  md <- round(median(x[,i], na.rm = TRUE),4)
  mn <- round(mean(x[,i], na.rm = TRUE),4)
  st <- round(sd(x[,i], na.rm = TRUE),4)
  q75 <- round(quantile(x[,i], probs = 0.75, na.rm = TRUE),4)
  mx <- round(max(x[,i], na.rm = TRUE))
  ul <- length(unique(x[complete.cases(x[,i]),i]))
  na <- sum(is.na(x[,i]))

  # Get results and name columns
  results <- c(mi, q25, md, mn, st, q75, mx, ul, na)
  names(results) <- c("Min.", "Q.25", "Median", "Mean",
                      "Std.Dev.", "Q.75", "Max.",
                      "Unique", "NA's")
  results
}

# Check dimensions and names for calculation of summary statistics
dim(Train_Data)
names(Train_Data)

# Call to summary statistics function
# There are 9 summary statistics and 769 predictors excluding id, loss,
# and default which are columns 1, 771, and 772
Summary_Statistics <- matrix(ncol = 9, nrow = 770)
colnames(Summary_Statistics) <- c("Min.", "Q.25", "Median", "Mean",
                                  "Std.Dev.", "Q.75", "Max.",
                                  "Unique", "NA's")
row.names(Summary_Statistics) <- names(Train_Data)[2:770]

# Loop for each variable included in summary statistics calculation
for(i in 1:770){

  Summary_Statistics[i,] <- myStatsCol(Train_Data,i+1)

}

# Create data frame of results
Summary_Statistics <- data.frame(Summary_Statistics)
# View select records for reasonableness
head(Summary_Statistics,20)

# Output file of predictor variable summaries
write.csv(Summary_Statistics, file = file.path(Path,"myStats_Data1.csv"))

# Creates summary table from which we get n which is number of missing
# records (i.e. Summary$n)
Summary <- describe(Train_Data, IQR = TRUE, quant=TRUE)

# Calculate number of missing records for each variable
Missing_Var <- nrow(Train_Data) - Summary$n
plot(Missing_Var, main = "Plot of Missing Values", xlab = "Variable Index",
     ylab = "Number of Rows - Missing Records")
```

```
# Get variable names missing more than 5000 records as we may want to remove
# those with n missing > 5000 as imputation could be problematic
Lot_Missing <- colnames(Train_Data)[Missing_Var >= 5000]
print(Lot_Missing)
# Variables for which n missing > 5,000
# [1] "f72"  "f159" "f160" "f169" "f170" "f179" "f180" "f189" "f190" "f199"
# [11] "f200" "f209" "f210" "f330" "f331" "f340" "f341" "f422" "f586" "f587"
# [21] "f588" "f618" "f619" "f620" "f621" "f640" "f648" "f649" "f650" "f651"
# [31] "f653" "f662" "f663" "f664" "f665" "f666" "f667" "f668" "f669" "f672"
# [41] "f673" "f679" "f726"

# Impute missing values
# Need to check size of data here to get for loop correct
dim(Train_Data)
# Loss and default have no missing values and will not be imputed
Imputed_Data <- Train_Data
for(i in 1:772){

  Imputed_Data[is.na(Train_Data[,i]), i] = median(Train_Data[,i], na.rm = TRUE)

}

Train_Data <- Imputed_Data
dim(Train_Data)

# Remove duplicate columns which we know exist from summary statistics
# and inspection
Train_Data <- Train_Data[!duplicated(as.list(Train_Data))]
dim(Train_Data)

# Remove constant columns which we know exist from summary statistics
# and inspection
Train_Data <- Train_Data[,apply(Train_Data, 2, var, na.rm=TRUE) != 0]
dim(Train_Data)

# Principal Components Analysis (PCA) for Dimensionality Reduction
# Run PCA removing observations with missing data as first pass and scaling
# which is to unit variance
PCs <- prcomp(Train_Data[complete.cases(Train_Data),], scale=TRUE)

#Variance explained by each principal component
PCs_Var <- PCs$sdev^2

#Proportion of variance explained
Prop_Var_Expl <- PCs_Var / sum(PCs_Var)

# Plot of proportion of variance explained and cumulative proportion of variance explained
plot(Prop_Var_Expl, main = "Proportion of Variance Explained per PC", xlab="Principal Component", ylab="Proportion of
Variance Explained", ylim=c(0,1),type='b')
plot(cumsum(Prop_Var_Expl), main = "Cumulative Proportion of Variance Explained",xlab=" Principal Component", ylab
="Cumulative Proportion of Variance Explained", ylim=c(0,1),type='b')
# Based on PCA may be able to reduce features to ~ 200

# Find Correlations
High_Correlation <- cor(Train_Data)
High_Correlation1 <- findCorrelation(High_Correlation,cutoff=0.99)
High_Correlation1 <- sort(High_Correlation1)
High_Correlation <- High_Correlation[-1,-1]
Corr_Vars <- colnames(High_Correlation)[c(High_Correlation1)]
dim(Train_Data)
names(Train_Data)
```

```
Imputed_Predictors <- Train_Data[,-1] # id
dim(Imputed_Predictors)
names(Imputed_Predictors)
# 679 values at this point

# Calculate differences in all correlated variables
Imputed_Data <- Imputed_Predictors
diff <- NA
diff_mat <- rep(NA,nrow(Imputed_Predictors))
for (i in 1:ncol(High_Correlation)) {
  for (j in 1:nrow(High_Correlation)) {
    diff <- NA
    if (High_Correlation[j,i] >= 0.99 & High_Correlation[j,i] < 1.00){
      diff <- Imputed_Data[,i] - Imputed_Data[,j]
    } #ends if
    if (sum(as.numeric(diff),na.rm = TRUE)==0){
      temp_delete <- rep(NA,nrow(Imputed_Data))
    } else {
      diff_mat <- data.frame(cbind(diff_mat,diff))
      colnames(diff_mat)[ncol(diff_mat)] <- paste(colnames(High_Correlation)[i],"-",colnames(High_Correlation)[j],sep="")
    }
  } #ends j
} #ends i
head(diff_mat)
names(diff_mat)
dim(diff_mat)
dim(Imputed_Data)
dim(Train_Data)
names(Imputed_Data)
names(Train_Data)

# Bring data back together to run analysis and determine important variables
# 1587 differences from diff_mat
# 679 predictors from Imputed_Data
# default
# New_Train_Data should be 105471 x 2266
New_Train_Data <- cbind(Train_Data,diff_mat)
dim(New_Train_Data)
#names(New_Train_Data)

#-------------------------------------------------------------------------------------
#  Rerun all previous lines and jump to 'loss' model after this
#-------------------------------------------------------------------------------------

# Decision trees using rpart, rpart.plot and variable importance
Decision_Tree_Train <- rpart(as.factor(default)~. -loss, data=New_Train_Data)
rpart.plot(Decision_Tree_Train)
summary(Decision_Tree_Train)

# Splits and Variable importance
Decision_Tree_Train$splits
Decision_Tree_Train$variable.importance

barplot(Decision_Tree_Train$variable.importance, main="Variable Importance Plot from Decision Tree",
        xlab="Variables", col= "beige")

# Random Forest Model for Variable Importance EDA
set.seed(1030)
Random_Forest_Var_Imp <- randomForest(as.factor(default) ~ f274.f528 + f527.f274 + f527.f528 + f414.f415 + f264.f578 +
f254.f569 +
                                      f2 + f334 + f339 + f378 + f338 + f221 + f222 + f272 + f653 + f663 + f662 +
```

```
                        f664 + f73 + f776 + f332, data=New_Train_Data, importance=TRUE, ntree=200)
VI_Fit1 <- importance(Random_Forest_Var_Imp)
varImpPlot(Random_Forest_Var_Imp, main = "Random Forest Variable Importance")
VI_Fit1

# Using 21 variables from decision tree importance and random forests importance
# we create a new data frame and output to csv so it can easily be read in at
# this point and used for model fitting.  Reading in entire dataset is too cumbersome
# and not necessary
New_Datafile <- as.data.frame(cbind(New_Train_Data$f274.f528, New_Train_Data$f527.f274,
  New_Train_Data$f527.f528, New_Train_Data$f414.f415, New_Train_Data$f264.f578,
  New_Train_Data$f254.f569,New_Train_Data$f2, New_Train_Data$f334, New_Train_Data$f339,
  New_Train_Data$f378, New_Train_Data$f338, New_Train_Data$f221, New_Train_Data$f222,
  New_Train_Data$f272, New_Train_Data$f653, New_Train_Data$f663, New_Train_Data$f662,
  New_Train_Data$f664, New_Train_Data$f73, New_Train_Data$f776, New_Train_Data$f332,
  New_Train_Data$default, New_Train_Data$loss))
dim(New_Datafile)
names(New_Datafile)

write.csv(New_Datafile, file = file.path(Path,"New_Datafile.csv"))
Reduced_Dataset <- read.csv(file.path(Path,"New_Datafile.csv"), stringsAsFactors=FALSE)
dim(Reduced_Dataset)
names(Reduced_Dataset)

New_Dataset <- Reduced_Dataset[,-1]
dim(New_Dataset)
names(New_Dataset)
# We now have V1, V2,..., V22, V23 where V22 is default and V23 loss

# Stepwise logistic regression
Logistic_Regression_EDA <- glm(as.factor(V22)~. -V23, data = New_Dataset, family=binomial)
summary(Logistic_Regression_EDA)
backwards <- step(Logistic_Regression_EDA, trace = 0)
formula(backwards)
summary(backwards)

# Stepwise logistic results in following formula:
#  as.factor(V22) ~ V1 + V2 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
#  V11 + V13 + V14 + V15 + V16 + V17 + V19 + V20 + V21

# Partitioning training data for training and validation
# Set seed for reproducibility
set.seed(999)
sample <- sample.split(New_Dataset$V22, SplitRatio = .70)
Training_Default <- subset(New_Dataset, sample == TRUE)
Validation_Default  <- subset(New_Dataset, sample == FALSE)
dim(Training_Default)
dim(Validation_Default)
table(Training_Default$V22)
table(Validation_Default$V22)

# Logistic Regression fitting in and out of sample results
Logistic_Regression <- glm(V22 ~ V1 + V2 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
                        V11 + V13 + V14 + V15 + V16 + V17 + V19 + V20 + V21, data=Training_Default, family = binomial)
# In sample
Logistic_Regression_Pred <- predict(Logistic_Regression,Training_Default, type="response")
Logistic_Default_Pred <- ifelse(Logistic_Regression_Pred >= 0.5,1,0)
table(Logistic_Default_Pred,Training_Default$V22)
# Out of sample
Logistic_Regression_Pred <- predict(Logistic_Regression,Validation_Default, type="response")
Logistic_Default_Pred <- ifelse(Logistic_Regression_Pred >= 0.5,1,0)
table(Logistic_Default_Pred,Validation_Default$V22)
```

```
# Plot ROC curve and calculate Area under curve for Logistic Regression
# Out of sample
Log_ROC_Data <- cbind(Logistic_Default_Pred,Validation_Default$V22)
Log_ROC <- roc(predictions = Log_ROC_Data[,1] , labels = factor(Validation_Default$V22))
Log_AUC <- auc(Log_ROC)
plot(Log_ROC, col = "red", lty = 1, lwd = 2, main = "ROC Curve for Logistic Regression")
Log_AUC

# Decision Tree for Prediction on Validation
Decision_Tree <- rpart(as.factor(V22) ~ V1 + V2 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
                    V11 + V13 + V14 + V15 + V16 + V17 + V19 + V20 + V21, data=Training_Default, method = "class")

# In sample
Decision_Tree_Pred <- predict(Decision_Tree,Training_Default, type="class")
table(Decision_Tree_Pred,Training_Default$V22)
# Out of sample
Decision_Tree_Pred <- predict(Decision_Tree,Validation_Default, type="class")
table(Decision_Tree_Pred,Validation_Default$V22)

# Plot ROC curve and calculate Area under curve for Decision Tree
Tree_ROC_Data <- cbind(Decision_Tree_Pred,Validation_Default$V22)
Tree_ROC <- roc(predictions = Tree_ROC_Data[,1] , labels = factor(Validation_Default$V22))
Tree_AUC <- auc(Tree_ROC)
plot(Tree_ROC, col = "red", lty = 1, lwd = 2, main = "ROC Curve for Decision Tree")
Tree_AUC

# Random Forest Model for Prediction on Validation
set.seed(1010)
Random_Forest <- randomForest(as.factor(V22) ~ V1 + V2 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
                        V11 + V13 + V14 + V15 + V16 + V17 + V19 + V20 + V21, data=Training_Default, importance = FALSE,
ntree=1000)

# In sample
Random_Forest_Pred <- predict(Random_Forest, Training_Default, type = "class")
table(Random_Forest_Pred,Training_Default$V22)

# Out of sample
Random_Forest_Pred <- predict(Random_Forest, Validation_Default, type = "class")
table(Random_Forest_Pred,Validation_Default$V22)

# Plot ROC curve and calculate Area under curve for Random Forest
RF_Roc_Data <- cbind(Random_Forest_Pred,Validation_Default$V22)
RF_ROC <- roc(predictions = RF_Roc_Data[,1] , labels = factor(Validation_Default$V22))
RF_AUC <- auc(RF_ROC)
plot(RF_ROC, col = "red", lty = 1, lwd = 2, main = "ROC Curve for Random Forest Model")
RF_AUC

#-------------------------------------------------------------------------------
#                          Analysis of 'Loss'
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#                          Transformation
#-------------------------------------------------------------------------------
# boxcox power transformation using forecast library
library(forecast)
#  find optimal lambda
lambda = BoxCox.lambda( Train_Data$loss )
lambda
#0.0005656
# Lambda = 0.00: natural log transformation
```

```
# natural log transformation is used for 'loss'

# new dataframe to analyse 'loss'
Train_Data_loss = Train_Data

# Transformed target = log of (loss + 1). 1 is added because loss has data points with '0' values.
Train_Data_loss$logloss = log(Train_Data_loss$loss + 1)

# Get rid of loss
Train_Data_loss$loss = NULL

# Get rid of ID
Train_Data_loss$ID = NULL

# Get rid of default
Train_Data_loss$default = NULL

dim(Train_Data_loss)
#105471    679
#-------------------------------------------------------------------------------------------
#                              Variable Selection Process
#-------------------------------------------------------------------------------------------
#sample before modeling
set.seed(1234)
smpl1 <- Train_Data_loss[sample(nrow(Train_Data_loss), 20000),]

t.test(Train_Data_loss$logloss,smpl1$logloss)
# p-value = 0.3
# fail to reject the null hypothesis, which means there is no difference in the mean.
#-------------------------------------------------------------------------------------------
#                            Random Forest as variable selection
#-------------------------------------------------------------------------------------------
set.seed(100)
Random_Forest <- randomForest(logloss ~ ., data=smpl1, importance=TRUE, ntree=50)
Random_Forest

varImpPlot(Random_Forest)
VF = varImp(Random_Forest)

# top 20 importance variables
importanceOrder=order(VF, decreasing = TRUE)
names=rownames(VF)[importanceOrder][1:20]
names

#[1] "f674" "f289" "f514" "f640" "f71"  "f442" "f624" "f384" "f20"  "f340" "f436" "f23"  "f433"
# "f631" "f278" "f424" "f468" "f663" "f368" "f282"

VF = varImp(Random_Forest)
# Output file of predictor variable summaries
write.csv(VF, file = file.path(Path,"vfrf.csv"))
#-------------------------------------------------------------------------------------------
#                Lasso as varriable selection
#-------------------------------------------------------------------------------------------
set.seed(100)
library(glmnet)
grid=10^seq(10,-2,length=100)
x=model.matrix(logloss~.,smpl1)[,-1]
y=smpl1$logloss

lasso.mod=glmnet(x,y,alpha=1,lambda=grid)
lasso.mod
```

```
set.seed(1)
cv.out=cv.glmnet(x,y,alpha=1)
bestlam=cv.out$lambda.min

lasso.pred=predict(lasso.mod,s=bestlam,newx=x[])
mean((lasso.pred-y)^2)

out=glmnet(x,y,alpha=1,lambda=grid)

#lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:679,]
lasso.coef = lasso.coef[lasso.coef!=0]
lasso.coef
# Output file of predictor variable summaries
write.csv(lasso.coef, file = file.path(Path,"vflass.csv"))

# following variables have non-zero coefficients in Lasso model.

#  f2         f13         f23         f55         f211         f221
#f228        f251        f261        f263        f270        f277        f308
#f343        f360        f376        f404        f406        f411        f416
#f459        f471        f482        f514        f536        f621        f629
#f648        f657        f674        f699        f768        f776
#-------------------------------------------------------------------------------
#                        Gradient Boosting as variable selection
#-------------------------------------------------------------------------------
set.seed(123)
gbmVarFit <- train(logloss ~ ., data = smpl1, method = "gbm",metric='RMSE')

gbmVarFit
# View variable importance
VFGBM = varImp(gbmVarFit)
VFGBM
#20 most important variables
# f536 f468  f471  f674  f211  f221  f766  f363  f376  f67 f404  f271 f629  f368  f28
# f250 f213 f568 f273 f387

# Output to a file
sink(file.path(Path,"vfgbm.txt"), append=FALSE, split=FALSE)
VFGBM
sink()
#-------------------------------------------------------------------------------
#                    Varriable selection
#-------------------------------------------------------------------------------
# Following variables are selected for modeling:
# (1) All variables from Lasso +
# (2) Top 20 variables from Random forest +
# (3) Top 20 variables from GBM +
# (4) Variables from the classification model above.
# Some variables overlap.

# Transformed target = log of (loss + 1). 1 is added because loss has data points with '0' values.
New_Train_Data$logloss = log(New_Train_Data$loss + 1)

subset_data = subset(New_Train_Data, select=c(logloss,f2,f221,f222,f254.f569,f264.f578,f272,
                                    f332,f334,f338,f339,f378,f414.f415,f527.f274,
                                    f527.f528,f653,f662,f663,f664,f73,f776,f13,f2,
                                    f20,f211,f211,f221,f221,f228,f23,f23,f251,f261,
                                    f263,f270,f271,f274.f528,f277,f278,f28,f282,
                                    f289,f308,f340,f360,f363,f368,f368,f376,f376,
                                    f384,f404,f404,f406,f411,f416,f424,f43,f433,
```

```
                              f436,f442,f459,f468,f468,f471,f471,f482,f514,
                              f514,f536,f536,f55,f621,f624,f629,f629,f631,
                              f640,f648,f657,f663,f67,f674,f674,f674,f699,
                              f71,f766,f768,f776))
dim(subset_data)
#105471     90
# Save the subset
write.csv(subset_data, file = file.path(Path,"subset_train_v2.csv"))

#subset_data <- read.csv(file.path(Path,"subset_train_v2.csv"), stringsAsFactors=FALSE)
#subset_data$X = NULL

#-------------------------------------------------------------------------------------
#                           Modeling Process
#-------------------------------------------------------------------------------------
#sink(file='console.txt')
library(caret)
#-------------------------------------------------------------------------------------
#                           Preparation for Modeling
#-------------------------------------------------------------------------------------
# Partition the data into training and test sets
set.seed(998)
inTraining = createDataPartition(subset_data$logloss, p = 0.70, list = F)
training <- subset_data[ inTraining,]
testing  <- subset_data[-inTraining,]

#-------------------------------------------------------------------------------------
#                       Linear Regression with stepwise Selection
#-------------------------------------------------------------------------------------
set.seed(1234)
lm.stepwise = train(logloss~., data = training, method = "leapSeq")
# Model Summary
lm.stepwise
# In-Sample
#nvmax  RMSE    Rsquared
#4      0.5869  0.03843
#4      0.5868  0.03573

# predict using test data
lm.stepwise.pred = predict(lm.stepwise, testing)

#Out of sample
lm.stepwise.results = postResample(pred = lm.stepwise.pred, obs = testing$logloss)
lm.stepwise.results
# Run 1
#RMSE   Rsquared
#0.59544  0.0361
#0.59585  0.03483
#-------------------------------------------------------------------------------------
#                           Random Forest
#-------------------------------------------------------------------------------------
set.seed(1234)
rffit <- randomForest(logloss ~ ., data=training, importance=TRUE, metric='RMSE')
#rffit = train(logloss~ .,data = training, method = "rf", metric='RMSE')
# View summary information
rffit
plot(rffit, main="Random Forest")
# In-Sample
#Type of random forest: regression
#Number of trees: 500
#No. of variables tried at each split: 29
#Mean of squared residuals: 0.05637945
```

```
#% Var explained: 84.11

#Mean of squared residuals: 0.05623
#% Var explained: 84.15
sqrt(0.05637945)
plot(rffit)

#For Regression, %IncMSE is the mean decrease in accuracy,
#and IncNodePurity is the mean decrease in MSE.
#IncNodePurity (increase node impurity)
varImpPlot(rffit, main="Random Forest Variable Importance")
plot(rffit, log="y")
VF = varImp(rffit)

# top 20 importance variables
importanceOrder=order(VF, decreasing = TRUE)
names=rownames(VF)[importanceOrder][1:20]
names
#[1] "f274.f528" "f67"       "f2"        "f2.1"      "f13"       "f264.f578" "f527.f274" "f332"
#[9] "f254.f569" "f228"      "f263"      "f527.f528" "f776.1"    "f282"      "f776"      "f270"
#[17] "f334"      "f424"      "f277"      "f55"

# predict using test data
rffit.pred <- predict(rffit, testing)

#Out of sample
rffit.results = postResample(pred = rffit.pred, obs = testing$logloss)
rffit.results
#     RMSE  Rsquared
#0.2400085 0.8456089
#0.2397   0.8460

#RMSE
RMSE.rffit <- sqrt(mean((rffit.pred-testing$logloss)^2))
RMSE.rffit
#0.2397

#MAE
MAE.rffit <- mean(abs(rffit.pred-testing$logloss))
MAE.rffit
#0.06621

# add the predicted value in testing data
testing[ ,(ncol(testing)+1)] <- rffit.pred

dim(testing)
write.csv(testing, file = file.path(Path,"testing_pred.csv"))
#------------------------------------------------------------------------------------
#                           Random Forest with ntree = 1500
#------------------------------------------------------------------------------------
set.seed(1234)
#rffit = train(logloss~., data = training, method = "rf",metric='RMSE')

#set.seed(100)
rffit2 <- randomForest(logloss ~ ., data=training, importance=TRUE, metric='RMSE', ntree=1500)
# View summary information
rffit2
# In-Sample
#Type of random forest: regression
#Number of trees: 500
#No. of variables tried at each split: 29
#Mean of squared residuals: 0.05637945
```

```
#% Var explained: 84.11
sqrt(0.05637945)
#0.2374
plot(rffit2)

VF = varImp(rffit2)

# top 20 importance variables
importanceOrder=order(VF, decreasing = TRUE)
names=rownames(VF)[importanceOrder][1:20]
names

# predict using test data
rffit2.pred <- predict(rffit2, testing)

#Out of sample
rffit2.results = postResample(pred = rffit2.pred, obs = testing$logloss)
rffit2.results
#     RMSE  Rsquared
#0.2397   0.8461
# Conclusion: ntree=1500 did not improve the RMSE much.
#-------------------------------------------------------------------------------------
#                              Gradient Boosting
#-------------------------------------------------------------------------------------
# GBM
set.seed(825)
gbmFit <- train(logloss ~ ., data = training, method = "gbm",metric='RMSE',verbose = TRUE)
# View summary information
gbmFit
# In-Sample
#interaction.depth  n.trees  RMSE    Rsquared
#3                  150      0.2798  0.7818
gbmFit$finalModel

# View variable importance
varImp(gbmFit)
plot(varImp(gbmFit))

# predict using test data
gbmFit.pred <- predict(gbmFit, testing)

#Out of sample
gbmFit.results = postResample(pred = gbmFit.pred, obs = testing$logloss)
gbmFit.results
#RMSE  Rsquared
#0.2860   0.7806

#RMSE
gbmFit.results[1]
#
#Rsquared
gbmFit.results[2]
#
#-------------------------------------------------------------------------------------
#                              Ridge Regression
#-------------------------------------------------------------------------------------
set.seed(825)
ridge <- train(logloss ~., data = training, method='ridge',lambda = 4,
               preProcess=c('scale', 'center'))
ridge
# In-Sample
#lambda  RMSE    Rsquared
```

```
#0.0001  0.5722  0.08314

ridge$finalModel

# View variable importance
#varImp(ridge)
#plot(varImp(ridge))

# predict using test data
ridge.pred <- predict(ridge, testing)

#Out of sample
ridge.results = postResample(pred = ridge.pred, obs = testing$logloss)
ridge.results
#RMSE   Rsquared
#   RMSE Rsquared
#0.58149  0.08011
#-------------------------------------------------------------------------------------
#                                Lasso
#-------------------------------------------------------------------------------------
set.seed(825)
lasso <- train(logloss ~., training, method='lasso',
               preProc=c('scale','center'), metric='RMSE')
lasso
# In-Sample
#fraction  RMSE    Rsquared
#0.1       0.5722  0.08327
lasso$finalModel

# View variable importance
varImp(lasso)
plot(varImp(lasso))

# predict using test data
lasso.pred <- predict(lasso, testing)

#Out of sample
lasso.results = postResample(pred = lasso.pred, obs = testing$logloss)
lasso.results
#RMSE   Rsquared
# RMSE Rsquared
#0.58136  0.08051
sink()

#-------------------------------------------------------------------------------------
#                    Model comparison: Training sample
#-------------------------------------------------------------------------------------

resamps1 = resamples(list(RandomForest = rffit,
                          RidgeRegression = ridge,
                          Lasso = lasso,
                          StepwiseLinear  = lm.stepwise,
                          GradientBoosting = gbmfit ))

resamps1

resamps1$values

# summary of the models
summary(resamps1)

#RMSE plots
```

```
bwplot(resamps1,metric="RMSE",main="Model Comparison")        # boxplot
dotplot(resamps1,metric="RMSE",main="Model Comparison")       # dotplot


#Rsquared plots
bwplot(resamps1,metric="Rsquared",main="Model Comparison")    # boxplot
dotplot(resamps1,metric="Rsquared",main="Model Comparison")   # dotplot

#-------------------------------------------------------------------------------------
#                  Model comparison table: Training and Test samples
#-------------------------------------------------------------------------------------
# rffit2 values are used.
library(formattable)
DF <- data.frame(Model=c("Random Forest", "Gradient Boosting", "Lasso",
                         "Ridge Regression", "Linear Regression Stepwise"),
                 Training.RSquared=(c(0.8411, 0.7818, 0.08327, 0.08314, 0.03843)),
                 Training.RMSE=(c(0.2374, 0.2798, 0.5722, 0.5722, 0.5869)),
                 Test.RSquared=(c(0.8461, 0.7806, 0.08051,0.08011,0.0361  )),
                 Test.RMSE=(c(0.2397,0.286, 0.58136, 0.58149, 0.59544)))


#Sort based on AUC
DF <- DF[order(DF$Test.RMSE, decreasing=FALSE),]
formattable(DF)

formattable(DF, list(
  Test.RMSE = color_tile("lightgreen", "lightpink"),
  Test.RSquared = color_tile("lightpink", "lightgreen")))

DF
#-------------------------------------------------------------------------------------
# Final Model: Random Forest. Target: logloss, Test.RMSE = 0.23970, TEST.MAE = 0.06621
#-------------------------------------------------------------------------------------
#            Back transform the MAE to compare with Kaggle contestants.
#          https://www.kaggle.com/c/loan-default-prediction/leaderboard
#-------------------------------------------------------------------------------------
testing_pred <- read.csv(file.path(Path,"testing_pred.csv"), stringsAsFactors=FALSE)

dim(testing_pred)

# (1) back transform loss
testing_pred$loss_back_tr = exp(testing_pred$logloss) - 1

# (2) back transform predicted loss
testing_pred$loss_back_tr_pred = exp(testing_pred$V91) - 1

# abs(1 - 2)
testing_pred$abs = abs( testing_pred$loss_back_tr - testing_pred$loss_back_tr_pred)

# sum of (1 - 2)^2
sum = sum(testing_pred$abs)
sum

n = nrow(testing_pred)
n
#MAE
sqrt(sum/n)
#0.7149

#-------------------------------------------------------------------------------------
#          After transforming back logloss to loss, the Test.MAE = 0.7149
#             Our rank would be 193 out of 675 in the leaderboard.
```