A Proposed Data Analytics Workflow and Example Using the R Caret Package

Simon Jones, Zhenghao Ye, Zhuoheng Xie, Chris Root, Theerakorn Prasutchai, Jeremy Anderson, Michael Roggenburg, Matthew A. Lanham
Purdue University, Department of Management, 403 W. State Street, West Lafayette, IN 47907
jone1107@purdue.edu; ye122@purdue.edu; xie176@purdue.edu; root2@purdue.edu; tprasutc@purdue.edu; ander650@purdue.edu; mroggenb@purdue.edu; lanhamm@purdue.edu

Abstract

This paper provides a comprehensive explanation of functions that are available in the R caret package, and proposed workflow in how one might use them to perform predictive modeling. The motivation for this paper is that the R statistical software is one of the most popular languages used by analytics professional for data analytics today, and the caret package has grown in popularity since its first release in 2007 to do predictive modeling. Unfortunately, the resources available that demonstrate the key functional components, expected runtime, and how an analyst might use those in a typical workflow for data mining and predictive analytics tasks are limited. We attempt to fill this void by providing the reader a valuable reference in how to get started using caret by motivating the functions and their use with a real-world dataset. The sample dataset we analyzed was acquired from the 2017 WSDM-KKBox's Churn Prediction Challenge provided on Kaggle.com, a platform for predictive analytics and data mining competitions. We demonstrate ways to clean, impute, pre-process, train, and assess popular machine learning models using caret. However, our contribution focuses on the actual process of getting results, as opposed to the results (e.g. model accuracy) themselves.

Keywords: R, caret, predictive analytics

Introduction

As the world enters a new age of information, it is becoming increasingly important to train people to become literate in analyzing and evaluating data for business and social use. In a popular study by McKinsey & Company, they estimated that by 2018 there would be a shortage of 140,000 to 190,000 people with deep analytical skills to take advantage of the data being generated (Manyika). In another survey conducted by IBM, the job market for Data Scientists will increase 28% by 2020, equating to nearly 364,000 new positions (Columbus, 2017). Clearly, industry is at a major growth stage in using data to support decision-making, and to do so effectively requires that their employees have analytics skills to get the job done.

Nowadays, as predictive analytics become a more important part of an organization's decision-making process, more tools are developed to facilitate the execution of predictive analytics. These tools can range from low in sophistication to those highly complicated tools that are designed for IT experts. Among these are commercial predictive analytic tools like MATLAB and SAS, as well as open-source software like R and Python, that have packages such as caret and scikit-learn. The Classification and Regression Training (caret) package from R for example is viewed as "*one of the best packages available in R to prototype various models* (Lanham, 2017).".

Caret provides one of the most comprehensive wrappers for any set of R packages and can be solely used to define an entire workflow starting from data cleaning and preprocessing, all the way through model training, prediction, and performance analysis. DataCamp.com describes it as a "*go-to package in the R community for predictive modeling and supervised learning... [and interfaces] with all of R's most powerful Machine Learning facilities* (DataRobot, 2016)." While some packages and libraries contain more detailed versions of some of the features in caret, the purpose of using it is to allow for a more streamlined process for modelers, as many packages have subtle differences that can lead to errors. The reason for this is there are many different packages that build different models (e.g. artificial neural network, support vector machine, random forest, etc.) and there is no one defined way in how a package developer must design their input and output arguments. For example, one package might require specifying the model argument as **modelA(Y ~ x1 + x1)**, while another package requires training the model by specifying the individual pieces as arguments like so; **modelB(y=Y, x=c(x1,x2))**. Thus, caret wraps both of these packages into one consistent function (i.e. train()) that allows the modeler to have a cleaner modeling workflow and allow for more efficient prototyping. For example, to use either of these generic functions might be **train(Y ~ x1 + x1, method="modelA")** or **train(Y ~ x1 + x1, method="modelB")**.

This data science community which consists of professionals ranging from Data Analyst to Data Scientist to Researcher, have embraced the power of open-source tools such as R as part of their workflow, even though they might be working at a company that has invested heavily in a commercial product (e.g. SAS). The reason is that as method methods are developed, they are often shared in near real-time. For example, the open-access *Journal of Statistical Software* (https://www.jstatsoft.org/index) is a popular avenue for researchers that develop new quantitative methods, visual tools, etc. to publish examples of R package as soon as their new methodology has been peer-reviewed and published elsewhere. While tutorial papers exist for caret, as we demonstrate in the literature review, we believe there is still a lack of information in how one can use these functions together to develop a predictive modeling workflow.

We structure this paper summarizing insights we found from several academic and professional papers on caret in the Literature Review. Next, we describe a data set we used in our study. In the Methodology section of the paper we develop a proposed workflow model that the authors have designed. This portion will contain a systematic process, as well as corresponding scripts in the R language, which will be used on the dataset example. In the Models section we describe ten separate machine learning algorithms used in our study and how we easily developed these models using caret. In the Results section we show how to compare the results of these models and discuss the runtime we observed to train them. Lastly, in the Conclusions we provide some key take-aways points for the reader to help them in their analytics journey using the R caret package.

Literature Review

While a lot has been said about the comprehensive nature of the caret package, for new data scientists, the process of learning and applying these data mining methods can be daunting. For this reason, we conducted a literature review to understand what is known that has been published, and thus frame our proposed methodological workflow in this space. Since our study is primarily aimed at creating a general framework for a workflow, we carried out our literature review analyzing papers that had been published regarding the basic functionalities of caret as well, as applied research that was performed using this package. **Table 1** provides a summary of these studies.

Most of these papers are demonstrative of the applications of caret and the data used in them are sample datasets, therefore we have chosen not to delve too deep into the motivations behind the studies. We were more interested in understanding common workflows used in the studies, than the results of the studies themselves, and we use these studies as support for our proposed recommendation.

Table 1: Summary of the literature

Year	Title	Author	Description
2008	The Caret Package	Max Kuhn	Provides descriptions of functions available when the package was first released. It explicitly lists out the constituent functions, function descriptions, usage, input values, arguments, and some sample code on how to apply it. Unlike the other literature it does not present a comprehensive application of the package using a sample data set.
2010	Variable Selection Using The Caret Package	Max Kuhn	Touches on the topic of dimension reduction and walks the reader through different ways by which they could go about it. The two primary sections in this paper cover feature selection using search algorithms, recursive feature elimination, and feature selection using univariate filters. In each section goes over functions in detail, the process by which these methods could be applied to most data sets. Unlike most of the papers published by Kuhn, this does not give any context as to how this fits into the larger data mining process.

2013	Predictive Modelling with R and the Caret Package	Max Kuhn	Delves into the predictive modelling process as a whole then focuses on using the caret package to build predictive models. The paper is divided into 8 broad segments: Introduction to the predictive modelling process, data splitting, data preprocessing, overfitting and resampling, training and tuning model trees, training and tuning a SVM, comparing models, and parallel processing. The main difference with this piece of literature on caret is that not much emphasis is given to the workings behind the processes as opposed to how they all fit together to paint a larger picture workflow.
2013	Predicting Defects Using Change Genealogies	Kim Herzig, Sascha Just, Andreas Rau, Andreas Zeller	Analyzes and estimates software quality requiring the analysis of its version histories. The authors combine all changes into change genealogies and try to predict software quality based on these genealogies. They show that change genealogies provide sufficiently good classifiers and prove that the genealogy models outperform models based on code complexity which is the norm in this field. The authors specifically used caret package in R to generate 6 models: K-nearest neighbor (knn), Logistic regression (multinom), Recursive partition (rpart), Support Vector Machines (svmRadial), Tree Bagging (tree-bag), Random forest (raomForest).
2015	Building Predictive Models in R using the Caret Package	Max Kuhn	This paper runs the entire process of predictive modeling using data from computational chemistry. The author breaks the workflow down into its constituent steps i.e. data preparation, building the model, tuning the model, predictions, and evaluation. The following study is almost entirely carried out using functions available in the caret package therefore it would be the most comprehensive piece of published literature on the functionalities available within this package. To illustrate its breadth, the author builds and evaluates multiple models with data available emphasizing the theme of a unified interface developed to create different models.
2015	A Short introduction to the Caret Package	Max Kuhn	A shorter introduction to the caret package and is primarily an introduction to the functions available in it. Much like the other studies, it goes over a sample workflow from a relatively rudimentary dataset. Majority of the emphasis in this paper is given to the 'train' function and its applications while walking the reader through some sample code.
2015	A Comparison of Data Mining Techniques in Evaluating Retail Credit Scoring Using R Programming	Dilmurat Zakirov, Aleksey Bondarev, Nodar Momtselidze	The comparison conducted by the authors using retail credit scoring data for the most part encompasses most popular data mining techniques. The paper explores K-Nearest Neighbors(KNN), Support Vector Machines (SVM), Gradient Boosted Model (GBM), Naïve Bayes Classification, Classification and Regression Trees, and Random Forests as methods for predicting customer scores using an actual dataset. The paper concludes, that the Random Forest model with down-sampling had a higher accuracy when compared to the other models presented. The paper does a thorough job at summarizing

			the applications of each of these modeling methods but does not cover the data pre-processing stage. A lot is yet unknown as to the normalization methods used during the data preprocessing phase.
2016	Evaluation of four supervised learning methods for groundwater spring potential mapping in Khalkhal region(Iran) Using GIS-based features	Seyed Amir Naghibi, Mostafa Moradi Dashtpagerdi	A critical tool for water resource management in semi-arid and arid regions is the mapping of groundwater potential for planning and usage purposes. This paper evaluates four popular data mining models: K-Nearest Neighbors(KNN), Linear Discriminant Analysis (LDA), Multivariate Adaptive Regression Splines (MARS) and Quadratic Discriminant Analysis (QDA) to model groundwater potential maps (GPMS). The authors used caret for carrying out their Quantitative Discriminant Analysis and achieved an accuracy of roughly 61.2%. While this paper is not solely based on caret, it does represent an example of applied research carried out using it.

After a review of the literature, most articles come from the author of the package itself. We found that the, "*Building Predictive Models in R using the Caret Package* (Kuhn, 2015)" paper to be the closest to our study. Our study differs from this in that we demonstrate the functionality using a business example (e.g. churn), rather than a chemistry example. This is important because the target audience of this paper are those researchers or practitioners doing business analytics, and not physical science research.

Data

The business example we use in this study is from KKBOX, a Taiwanese music streaming service, and happens to be Asia's top provider founded in 2004. Its primary regions of operation are Taiwan, Hong Kong, Malaysia, and Singapore hosting the world's most extensive music library with over 30 million tracks. Their business model is funded through advertisements and paid subscriptions. KKBOX aims to accurately predict churn of their paid customers from who they receive most of their revenue.

KKBOX provided through Kaggle.com a large dataset of their customers listening and subscription behavior and allowed the analytics community to compete to build the best churn prediction problem. There are five sets of data: train, sample_submission_zero, transaction, user_logs, and members. Data descriptions and features are shown below.

Table 2 consists of customer user ids and whether those customers churned or not and have a subscription expiration in February 2017. These users were the main ones that competition participants use to build a predictive model.

Table 2: Train table

Variable	**Type**	**Description**
msno	Factor	The unique user identification
is_churn	Integer	Classify churner and non-churner

The scoring set of data consists of user ids and whether they have churned or not as shown in **Table 3**. These are users who have subscription expiration in March 2017. In this file, all is_churn values are zeros and the modeler is required to submit their probability prediction based on the modeling solution they generate. Only Kaggle knows the true is_churn flag, but when you submit your predictions to Kaggle you are scored compared to other participants.

Table 3: Sample_submission_zero table

Variable	**Type**	**Description**
msno	Factor	The unique user identification
is_churn	Integer	Predict churner and non-churner

Table 4 contains a description of the listeners transaction details of the users up until February 28, 2017. The data set consists of 9 features as shown below.

Table 4: Transaction table

Variable	**Type**	**Description**
msno	Factor	The unique user identification
payment_method_id	Integer	Classify payment method user used for subscription
payment_plan_days	Integer	The number of days of subscription
plan_list_price	Integer	The listed subscription price
actual_amount_paid	Integer	The actual paid price
is_auto_renew	Integer	Classify auto subscription renewal users
transaction_date	Integer	The date that transaction was made
membership_expire_date	Integer	The subscription expiration date
is_cancel	Integer	Classify churner

The user logs describe the users' behavior in using the KKBOX service and how they listened to music. For example, in **Table 5** we obtain measures about the quantity of songs played, and how much of the length of the songs were listened too.

Table 5: User_logs table

Variable	**Type**	**Description**
msno	Factor	The unique user identification
date	Integer	Date when the usage was recorded
num_25	Integer	Number of songs played less than 25% of the song length
num_50	Integer	Number of songs played between 25% and 50% of the song
num_75	Integer	Number of songs played between 50% and 75% of the song
num_985	Integer	Number of songs played between 75% and 98.5% of the song
num_100	Integer	Number of songs played over 98.5% of the song length
num_unq	Integer	Number of unique songs that user play
total_secs	Numeric	Total time songs are played in the unit of seconds

General information about a user such as age, gender, etc. as displayed in **Table 6**. We found that some of the observations in this table are incomplete or would be considered outliers.

Table 6: Members table

Variable	Type	Description
msno	Factor	The unique user identification
city	Integer	The city where user locates
bd	Integer	Age of the user
gender	Factor	Gender of the user
registered_via	Integer	The method user used for registration
registration_init_time	Integer	Initial registration date
expiration_date	Integer	Subscription expiration date

Table 7 contains generated features based on the attributes provide in the previous tables. We utilized some of the factors described above to create dummy variables which is indicative of the information. They were then used in the analysis.

Table 7: Generated features table

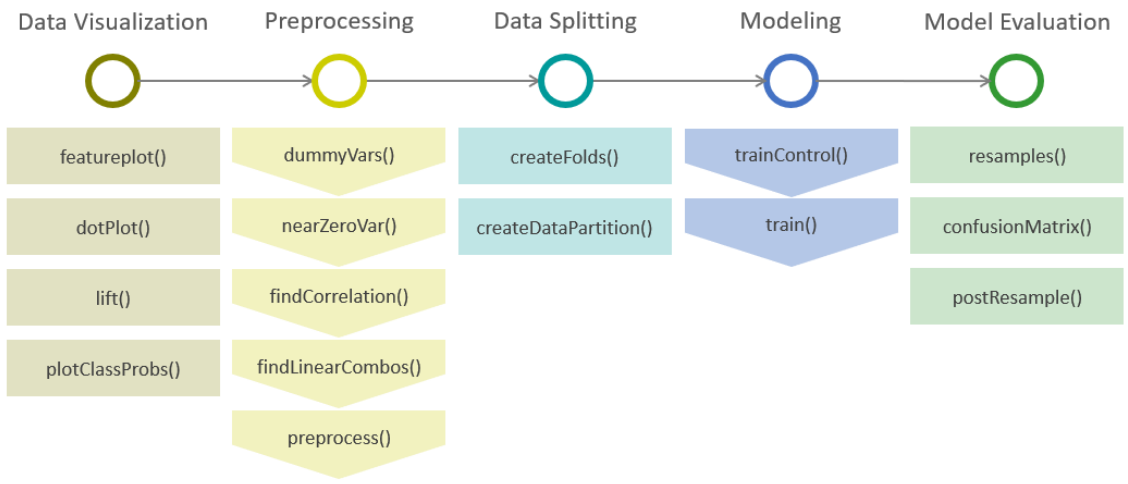
Variable	Type	Description
sub_Churn	Factor	Created based on the subscription model described by Kaggle's. According to Kaggle, some non-churners are indeed churners, and it's taken into consideration by this variable. This variable was not used in our modeling.
avgpayment	Numeric	The average payment to the service from the Transaction dataset
latest_renew	Date	What was the latest day to renewal from the Transaction dataset
latest_cancel	Date	What was the latest day of cancellation of the Transaction dataset
pct_cancel	Numeric	On average how many times the data had them listed as canceled on the Transaction dataset
Pricother	Factor	There were several different factor levels for the price. Most prices are put into their respective levels, and a small group that's left are grouped into "pricother"
Pric149	Factor	Dummy for price = 49
method	Factor	Payment method. Similar to price, most methods can be grouped to form their respective levels, and dummy variables are used.
meth36	Factor	Dummy for method = 36
methother	Factor	There were several different factor levels for payment method. The majority of methods came from like 6 or 7 different levels and grouped all the rest into "methother"
plan	Factor	Payment plan. The majority of methods came from 2 or 3 different levels. Thus, dummy variables are included.

planother	Factor	There were several different factor levels for the payment plan. The majority of payment plan came from like 2 or 3 different levels and grouped all the rest into " Planother".
plan7	Factor	Dummy for Plan=7
Daysrange	Numeric	How many days lasted for the service

Methodology

The caret package contains functions to perform the entire modeling process from pre-processing to modeling to feature selection to the final predictive output. **Figure 1** below details our proposed workflow. Here we focus on five different categories of functions: Data Visualization, Preprocessing, Data Splitting, Modeling, and Model Evaluation. Under each category are the caret functions that one will likely use when prototyping their predictive modeling solution.

Figure 1: Proposed caret workflow



Data Visualization

Data Visualization is the process of understanding the data used throughout the predictive workflow. It can be used at any step of the process to verify results or determine errors. Caret provides a variety of functions for gathering insights from analyzing the data visually. The various plot functions shown in **Table 8** provide a very simple way to create complex graphical displays for your data without the need for large amounts of code. Many analysts agree that visualizing your data before analyzing it can expedite finding non-linear relationships or high correlations. However, it is recommended that you utilize caret's visualization functionality at any step due to its ease of use.

Table 8: caret data visualization functions

Function	Description
featureplot	Outputs a range of visual plots i.e. scatterplot/boxplot
dotPlot	Create a dot plot of variable importance values
lift	Lift Plot
plotClassProbs	Plot Predicted Probabilities in Classification Models
plotObsVsPred	Plot Observed versus Predicted Results in Regression and Classification Models

Pre-processing

Pre-processing is the technique that prepares data for predictive applications. Normally, the raw data may be incomplete or noisy to use due to various problems such as obsolete fields, missing values, outliers, etc. Missing values should be dealt with first using a methodology consistent with the business problem. Also, R can improperly load in feature classes, therefore it is essential to evaluate every variable in the dataset and convert it to the intended class if necessary. Caret is extremely useful in pre-processing raw data beyond just removing missing information. To start with, if a dataset contains categorical features with many levels, then the **dummyVars** function can provide a very quick way to break apart the factor levels into separate columns and avoid certain instances of correlation. It is recommended to use a combination of **findLinearCombos** and **findCorrelation** functions to reduce dimensionality through linear combinations of features and remove correlated variables which harm the effectiveness of certain models. This step can confirm your findings in the visualization step if any exist as well. Finally, the **preProcess** function provides an extremely simple way to transform and re-scale your data. For example the z-score standardization can easily be done using **method=c("center","scale")** while min-max normalization can be done using **method=c("range")**. Sometimes, analysts also want to try and make their input features more Gaussian distributed, thus you could do this in isolation such as **method=c("Yeo-Johnson")**, or combine this transformation with scaling, such as **method=c("range","Yeo-Johnson")**. **Figure 2** shows an excerpt of the code using **dummyVars**, **nearZeroVar**, **findCorrelation**, and **preProcess** used in this analysis. **Table 9** provides a summary of the pre-processing functions.

Figure 2: Pre-processing code applications

```
#Dummy Variables
dummies <- dummyVars(is_churn ~ ., data = tr)     #Create a Dummy Variable object for the train data set
ex <- data.frame(predict(dummies, newdata = tr))  #Create a data table with all of the factors broken into dummy variables
names(ex) <- gsub("\\.", "", names(ex))           #Clean the names of the features to be visually appealing
tr <- cbind(tr$is_churn, ex)                      #Add the target variable column to the data table
names(tr)[1] <- "is_churn"                        #Make the tame of the target variable is_churn

#Zero Variance Detection
nzv <- nearZeroVar(tr[,2:61])          #Create a vector containing features with near 0 variance
tr_filtered <- tr[,2:61][,-nzv]        #Filter the train data to be only rebust features (those not in nzv)
tr <- cbind(tr$is_churn, tr_filtered) #Add the target variable column back to the data table
names(tr)[1] <- "is_churn"             #Make the name of the target variable is_churn

#High Correlation Analysis
descrCor <-  cor(tr[,2:ncol(tr)])                              #Get the correlation between all the features
highlyCorDescr <- findCorrelation(descrCor, cutoff = 0.90)  #Create a vector containing 90% correlated features
filteredDescr <- tr[,2:ncol(tr)][,-highlyCorDescr]          #Filter the data removing the previously identified features
descrCor2 <- cor(filteredDescr)                             #Get the correlation between remaining features
summary(descrCor2[upper.tri(descrCor2)])                    #Show the distribution of the new correlations
tr <- cbind(tr$is_churn, filteredDescr)                     #Add the target variable column back into the data table
names(tr)[1] <- "is_churn"                                  #Make the name of the target variable is_churn

#Pre-Processing
preProcValues <- preProcess(tr[,2:ncol(tr)], method = c("center", "scale"))  #Create preProcessing object
tr <- predict(preProcValues, tr)                                             #Apply preProcessing to data set
```

Table 9: caret pre-processing functions

Function	Description
dummyVars	Create A Full Set of Dummy Variables (turns an n-level factor into (n-1) separate boolean factors)
findCorrelation	Determine highly correlated variables
findLinearCombos	Determine linear combinations in a matrix
nearZeroVar	Identification of near-zero variance predictors
preProcess	Pre-processing of predictors
classDist	Class centroids and covariance matrix

Feature Selection

Feature Selection is the process of reducing the dimensionality of the data without significant loss in performance and predicting power with the effort to create a simpler model. In general, most approaches for feature selection (reducing the number of predictors) can be placed into two main methods: wrapper and filter. For wrapper method, caret has functions to perform recursive feature elimination, genetic algorithms, and simulated annealing as shown in **Table 10**. For filter method, caret has a general framework for using univariate filters. (Kuhn, 2017).

If feature reduction was not completed in the pre-process step or this step, further reduction can be done in the modeling portion of the workflow. Certain models run through the **train** function have Principal Component Analysis added to them which can supplement the functions in the list and further reduce dimensionality.

Table 10: caret feature selection functions

Function	Description
filterVarImp	Calculation of filter-based variable importance
gafs	Genetic algorithm feature selection
gafsControl, safsControl	Control parameters for GA and SA
rfe, rfeIter	Backward Feature Selection
rfeControl	Controlling the Feature Selection Algorithms
safs	Simulated annealing feature selection
sbf	Selection By Filtering (SBF)
sbfControl	Control Object for Selection By Filtering (SBF)
varImp	Calculation of variable importance for regression and classification models

Data Splitting

Data splitting is the process of partitioning the data into two or more separate subsets with the intention of using one set to train the model, and the other set to test or validate it. Data can be split in several different ways depending on the problem at hand. The **createDataPartition** function will generate an index for splitting your dataset for any number of stratified random subsets of any size partition between train and test sets. **CreateFolds** is similar to this and will index your data into k-folds of randomly sampled subsets. Another useful splitting function is **createTimeSlices** which will index your data into training and test sets when it is in time series format and want a rolling train/test window. After splitting the data, it is common

to find, especially in classification problems, that a certain class is much larger. In such cases, using **upSample, downSample, SMOTE** or **ROSE** functions provide easy ways to resample your data to obtain more even distributions of each response and reduce bias when the classifier tries to learn. **Figure 3** shows an excerpt from the code used in this analysis showing the data partitioning step. **Table 11** summarizes these functions.

Figure 3: Data partitioning code

```
#Data Partitioning
trainIndex <- createDataPartition(tr$is_churn,  # target variable vector
                                  p = 0.80,     # % of data for training
                                  times = 1,    # Num of partitions to create
                                  list = F    # should result be a list (T/F)
)
train <- tr[ trainIndex,]
test  <- tr[-trainIndex,]


library(DMwR)
smotetrain<-SMOTE(is_churn~.,data = train,perc.over = 400,perc.under = 100,k=5)
```

Table 11: caret data splitting functions

Function	**Description**
createDataPartition	Creates a series of test/training partitions
createFolds	Splits the data into k groups
createTimeSlices	Creates cross-validation split for series data
downSample, upSample	Down- and Up-Sampling Imbalanced Data
groupKFold	Splits the data based on a grouping factor
maxDissim	Creates a sub-sample by maximizing the dissimilarity between new samples and the existing subset
SMOTE, ROSE	Mix of upsampling/downsampling
upSample	Samples with replacement until all class frequencies are equal
downSample	Randomly sample a dataset so that all class frequencies match the minority class

Modeling

Model training starts with specifying training parameters for a model that you may want to run. It feeds in the validation method, type of problem (i.e. regression/classification), and conditions for training. The second step specifies the model to train and the respective tuning parameters.

The two important functions are **trainControl**, which specifies the validation technique (e.g. k-fold, LOOCV, etc.) and type of modeling problem (e.g. regression, classification). The **train** function is one of the most extensive functions in R. It has integrated or wrapped 238 modeling packages as the time of writing this study, where each approach uses by default three different combinations of tuning parameters. A user can specify directly the tuning parameter combinations they would like to try or use a generic *tuneLength=* argument that will create a reasonable set of possibilities based on the number you specify. In addition, code-savvy users can create their own models and cleanly run them through the **train** function. The output for **train** identifies the "best" model obtained among the set of possible tuning parameters and uses that as the final model. Among other things, **Figure 4** shows the **trainControl** and **train** functions being applied for this paper's analysis. **Table 12** provides a summary of the modeling and tuning functions in caret.

Figure 4: Model Training and Statistic Recording

```
modeloutput=data.frame(matrix(nrow=0,ncol=10))
cname<-c("DeltaTime","Time To Train","Time to Predict",
         "Train Accuracy","Train Sensitivity","Train Specificity",
         "Test Accuracy","Test Sensitivity","Test Specificity"
         )


ctrl <- trainControl(method="cv",     # cross-validation set approach to use
                     number=5,       # k number of times to do k-fold
                     classProbs = T,
                     summaryFunction = twoClassSummary,
                     allowParallel = T
)

for (model in c("nb",'nnet','pcaNNet','ORFlog','treebag',"gbm","AdaBag","LogitBoost","C5.0",'svmRadialWeights')){
                                                                    #Loop over each model used
  cat(model)                                                        #Print off the model name
  start=Sys.time()                                                  #Take note of the time training starts
  modeled<-train(is_churn~.,data=smotetest,method=model,trControl=ctrl)    #Train the model
  cat(" Trained")                                                   #Print off to indicate progress
  DeltaTimeModel<-Sys.time()-start                                  #Take the difference of time to find how long training took
  TrainP <- predict(modeled, newdata=smotetest)                     #Get the model probabilities for the training data
  TestP <- predict(modeled, newdata=test)                           #Get the model probabilities for the testing data
  traincm<- confusionMatrix(data=TrainP,smotetrain$is_churn)        #Get the confusion matrix for the train data
  testcm<- confusionMatrix(data=TestP,test$is_churn)                #Get the confusion matrix for the test data

  DeltaTime<-Sys.time()-start                                       #Take the difference of time to find how long the whole process took
  DeltaTimeCM<-DeltaTime-DeltaTimeModel                             #Difference of the whole process and training gives prediction time

                                                                    #Put all of this information into a vector for later analysis
  x<-c(format(DeltaTime),format(DeltaTimeModel),format(DeltaTimeCM),     #Record the 3 times to process
       traincm$overall[1],traincm$byClass[1],traincm$byClass[2],     #Record Train Accuracy, Sensitivity, and Specificity
       testcm$overall[1],testcm$byClass[1],testcm$byClass[2]         #Record Test  Accuracy, Sensitivity, and Specificity
       )
  x<-t(data.frame(x))                                               #Transpose the data frame of the vector to form a row
  colnames(x)<-cname                                                #Name the analysis features
  row.names(x)<-model                                               #Name the row the model name
  modeloutput<-rbind(modeloutput,x)                                 #Add this new model to the bottom of the outputs
  rm(DeltaTime,DeltaTimeCM,DeltaTimeModel,traincm,testcm)           #Clean up variables
  cat('Data logged')                                                #Print to indicate progress
  write.csv(modeloutput,"Modelvalues.csv",sep="|")                  #Save the outputs for later analysis
  saveRDS(modeled,file=paste0(model,".rds"))                        #Save the full model for later loading
}
```

Table 12: caret modeling and tuning functions

Function	Description
trainControl	Specifies the type of problem, conditions for training a model and methods for validation
train	Specifies the desired model and corresponding tuning parameters

Model Performance & Evaluation

Model evaluation directly follows the model training step. In this section, the user benchmarks the model(s) against some predefined metric for the problem or each other. The **confusionMatrix** function is a powerful tool for classification problems and produces the distribution of type-I, type-II errors, and correct/incorrect classifications. The regression counterpart to this is the **postResample** which outputs the MSE and RS for the model. Using the **predict** function will return a multitude of probability predictions based on the model and can be applied to both classification and regression models. There are also a variety of further statistics that can be reported using the rest of the functions from the list. After evaluating the model, if it does not achieve the required performance, iteration of the modeling stage is recommended using different tuning parameters. **Figure 4** above showed the performance evaluation **confusionMatrix** on the 9th and 10th lines of the loop. **Table 13** summarizes the functions.

Table 13: caret performance and evaluation functions

Function	Description
calibration	Probability Calibration Plot
confusionMatrix	Create a confusion matrix

defaultSummary	Default function to compute performance metrics in train
postResample	Calculate the MSE and R2 given two numeric vectors of data
twoClassSummary	Computes sensitivity, specificity, and AUC
multiClassSummary	Computes some overall measures of performance (accuracy and Kappa)
extractPrediction	Extract predictions and class probabilities from train objects
extractProb	Extract class probabilities from train objects
negPredValue, posPredValue, sensitivity	Calculate sensitivity, specificity and predictive values
recall	Calculate recall
precision	Calculate precision
F_meas	Calculate F values
resamples	Collation and Visualization of Resampling Results
resampleSummary	Summary of resampled performance estimates
thresholder	Generate Data to Choose a Probability Threshold

Other

In **Table 14** there are other functions that do not necessarily fall into any of the other categories that are useful in the data analysis workflow.

Table 14: caret other functions

Function	**Description**
getSamplingInfo	Get sampling info from a train model
learning_curve_dat	Create Data to Plot a Learning Curve
modelLookup, checkInstall, getModelInfo	Tools for Models Available in train
SLC14_1, SLC14_2, LPH07_1, LPH07_2, twoClassSim	Simulation Functions

Models

In our business example, the response variable (is_churn) is a binary variable; therefore, making this problem binary classification problem. In this study, we used ten different models among the many available to compare their performance including accuracy and runtime. **Table 15** provides a summary of these methods. Some are common models such as neural network and naïve bayes, some are models suggested in section 7.0.50 (Two Class Only) of *The Caret Package* (Kuhn, 2017) such as AdaBoost Classification Tree, and Support Vector Machines with class weights.

Table 15: caret models used in this study

Model	**Function**	**Type**	**Tuning Parameters**
Naïve Bayes	nb	Classification	·Laplace Correction (fL, numeric) ·Distribution Type (usekernel, logical) ·Bandwidth Adjustment (adjust, numeric)
Neural Network	nnet	Classification Regression	·Size (#Hidden Units) ·Decay (Weight Decay)

Neural Network with Feature Extraction	pcaNNet	Classification Regression	·Size (#Hidden Units) ·Decay (Weight Decay)
Oblique Random Forest	ORFlog	Classification	·Number of Randomly Selected Predictors (myry, numeric)
Bagged CART	treebag	Classification Regression	No tuning parameters for this model.
Stochastic Gradient Boosting	gbm	Classification Regression	·Number of Boosting Iterations (n.trees, numeric) ·Max Tree Depth (interaction.depth, numeric) ·Shrinkage (shrinkage, numeric) ·Min. Terminal Node Size (n.minobsinnode, numeric)
Bagged AdaBoost	AdaBag	Classification	·Number of Trees (mfinal, numeric) ·Max Tree Depth (maxdepth, numeric)
Boosted Logistic Regression	LogitBoost	Classification	·Number of Boosting Iterations (nIter, numeric)
C5.0 Tree	C5.0	Classification	·Number of Boosting Iterations (trials, numeric) ·Model Type (model, character) ·Winnow (winnow, logical)
Support Vector Machines with Class Weights	svmRadial Weights	Classification	·Sigma (sigma, numeric) ·Cost (C, numeric) ·Weight (Weight, numeric)

1. Naïve Bayes

Naïve Bayes classifiers are a family of simple probabilistic classifiers that are based on Bayes Theorem. These classifiers are particularly well suited to datasets where the dimensionality of the inputs is high. Naïve Bayes is also fast in terms of computation speed and simple in terms of implementation. On the other hand, Naïve Bayes relies on the assumption that all features are independent, which is where the "naïve" in its name comes from. If the naïve assumption is violated, the classifier might perform rather badly.

2. Neural Network

The concept of neural network is inspired by how animal brains work: with a vast network of interconnected neurons. It contains an input layer, an output layer, and defined number of hidden layers and neurons. It is widely applied in artificial intelligence such as speech and image recognition. A Neural Network is a powerful algorithm to model non-linear data, especially with a large number of input features such as sounds and pictures. Meanwhile, it is nearly impossible to understand the classification boundaries intuitively. For large data sets, a Neural Network is also computationally expensive.

3. Neural Network with Feature Extraction

A Neural Network works well with feature extraction because of its powerful parallel computation. The model can lead an exploratory network which can perform dimension reduction or feature extraction on the training dataset in both linear and nonlinear neurons. "*First, the formulation of a single and a multiple feature extraction are presented. Then a new projection index that favors directions possessing multimodality is presenter* (Intrator)."

4. Oblique Random Forest

The Oblique Random Forest shares basic ideas with the Random Forest algorithm such as bagged trees. The main difference is the procedure of optimal splits direction are sought at each node, for the Oblique Random Forest, it focuses on the multivariate models for binary splits in each node. There are two types of Oblique Random Forests: oRF-lda, which performs an unregularized split and oRF-ridge that optimizes regularization parameter at every split (Menze).

5. Bagged CART

Bagging is also referred to bootstrapped aggregation, which is an ensemble method from the "ipred" package which creates different models of the same type using different sub-samples of the same dataset. The bagged CART is very effective when the methods have high variance because all models will be combined to provide a final result (Browniee, 2014).

6. Stochastic Gradient Boosting

Stochastic Gradient Boosting is basically a gradient boosting model with a minor modification. It combines the randomness as an integral part of the procedure. Stochastic Gradient Boosting makes "*the value of f smaller [reducing] the amount of data available to train the base learner at each iteration. This will cause the variance associated with the individual base learner estimates to increase* (Friedman, 1999)."

7. Bagged AdaBoost

AdaBoost is best used to boost the performance of decision trees on binary classification problems. AdaBoost is a very good approach as it corrects upon its mistakes. The boosting for this model is achieved through ensembling methods. However, the biggest problem with this algorithm is that it is sensitive to outliers.

8. Boosted Logistic Regression

Logistic regression is a special case of a Generalized Linear Model (GLM) where the response variable is binary. It uses a logistic function to measure the relationship between the responsible variables and predictors. Unlike other models in GLM family, the outputs of a logistic regression are probabilities, which could be converted to binary results (0 or 1) based on a specified cutoff threshold. One of the advantages of using logistic regression is that it is simple and efficient. It does not use large memory space to run. In addition, the output probability scores are easy to interpret, and it is possible to manually change the cut-off to obtain the best prediction results. However, logistic regression does not perform well for small data sets as it will tend to overfit to the training data.

9. C5.0 Tree

The C5.0 tree model extends the C4.5 classification algorithms developed by Ross Quinlan (1992). It is a decision tree model with some improvement from its previous C4.5 tree. C5.0 is a sophisticated data mining tool for discovering patterns that delineate categories, assembling them into classifiers, and using them to make predictions (Information on See5/C5.0, 2017). Like other decision trees, C5.0 is able to handle non-linear features and make intuitive decision rules. However, it does not provide ranking scores and sometimes it is extremely vulnerable to overfitting.

10. Support Vector Machines with Class Weights

Like Neural Networks, Support Vector Machines (SVMs) are a very popular machine learning technique. SVMs were developed by Cortes & Vapnik in 1995 for binary classification and can handle non-linear decision boundaries. SVMs can handle a large feature space, but are not very efficient (Sachan, 2015).

Results

Runtime

Runtime is one of the key factors to consider when choosing among a possible set of predictive solutions to support the business problem. It is also important when deciding on the technology or architecture to use. Some detractors of R claim that it does not perform as well as other languages such as Python or SAS. Thus, we provide an idea of runtime for a large dataset such as the one this large company uses to understand if their customers will churn or not.

There is no concrete relation between runtime and accuracy of the predictive model. However, more complex models will typically require more time to perform the prediction. Essentially, an accurate model with short runtime is ideal, however this is unlikely. Depending on the business problem at hand, a quicker, less accurate model or a slower, more accurate model may be preferred. **Table 16** shows the total run for these ten models ranges from one minute to 11 hours.

Table 16: Run time by algorithm

Algorithm	Total Run Time	Training Time	Predict/Score Time
nb	18.94176 mins	9.188566 mins	9.753199 mins
nnet	10.46171 mins	10.44262 mins	0.01909322 mins
pcaNNet	7.621718 mins	7.584865 mins	0.03685278 mins
ORFlog	10.00892 hours	9.759153 hours	0.2497653 hours
treebag	6.574278 mins	4.305433 mins	2.268845 mins
gbm	2.763423 mins	2.741724 mins	0.02169898 mins
AdaBag	34.03858 mins	33.18765 mins	0.850931 mins
LogitBoost	1.094711 mins	49.7573 secs	15.92534 secs
C5.0	26.34021 mins	25.73059 mins	0.6096223 mins
svmRadialWeights	11.82071 hours	11.73735 hours	0.08335575 hours

According to runtime recorded in the chart above, the Oblique Random Forest and Support Vector Machines with class weights are two most time-consuming models among ten predictive models we chose for the study. They both took more than ten hours to run.

Accuracy, Sensitivity, and Specificity Performance

When we get the results, we must first understand the difference between the training set and the testing set and be sure that we did not overfit to the training set. During the "Data Splitting" of the workflow, we partitioned the data into two sets: train and test sets, where the training set contains the majority of the data. This training set will be used to train the model. Once the model is trained, the test set data will be fed into the trained model. If the model is well-trained we will expect to see the accuracy of the test set to be similar to the accuracy of the training set. What we do not want is to overfit the data to the training set, which

means that the model we trained only gives us good results for the training set. If we put in the testing data into the overfitted model, we see that the results we get for the test set are not similar to the training set results. **Figure 5** shows the accuracy of each model on the train and test sets. Accuracy gives a percentage of the amount of overall correctly identified targeted variable.

Figure 5: Accuracy by model

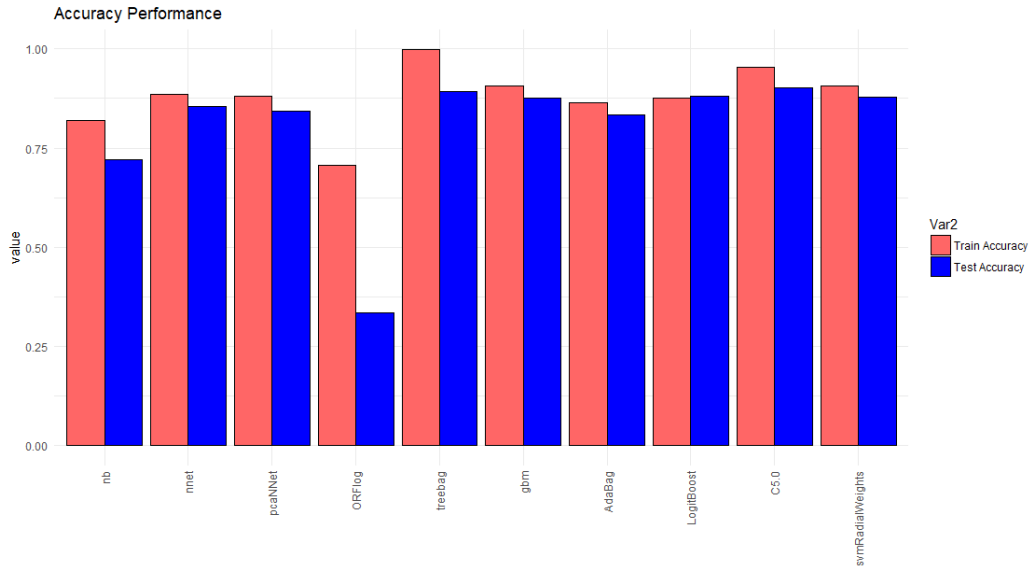


Figure 6 provides the sensitivity (or true positive rate). Sensitivity is the percentage of users that were correctly predicted as churners whom were actually churners. We see in this business case, most of the models tend to perform well at predicting churners on the train set, but do not generalize well as the test set performance is much lower. We believe this is due to the imbalance in the data set, as most customers are non-churners.

Figure 6: Sensitivity by model

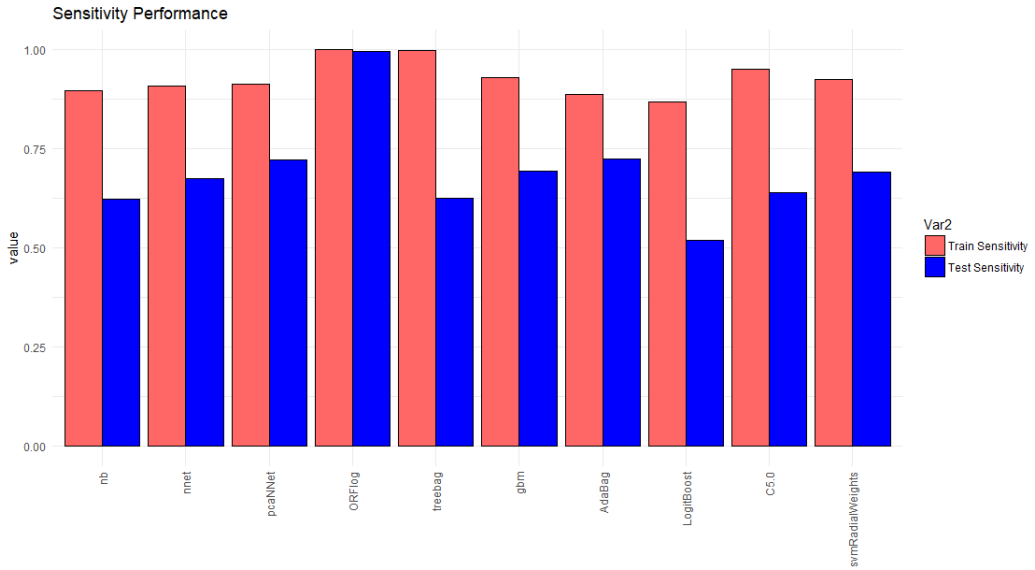
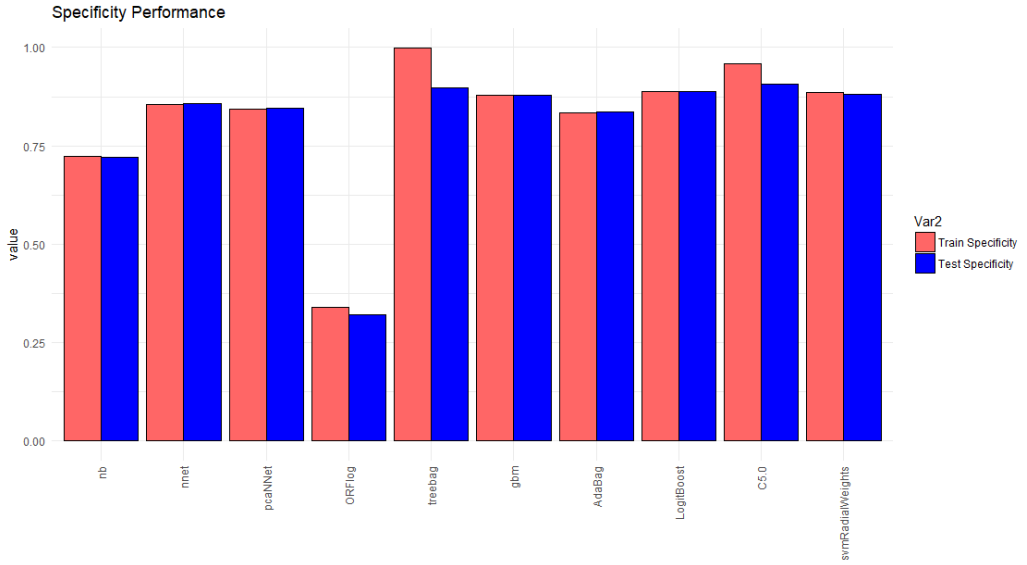


Figure 7 provides specificity (or true negative rate). Specificity measures the proportion of predicted non-churners whom were actually non-churners. We see similar performance on the train and test sets for all algorithms on the specificity performance measure. The logic is the same as that for sensitivity in that since most records are non-churners, the algorithms will tend to predict those better than the churners.

Figure 7: Specificity by model



The Oblique Random Forest and Bagged CART are overfitting models whose accuracy of train dataset is much higher than that of test dataset, especially Oblique Random Forest with test accuracy less than 35%, not mentioning its 10-hour runtime. The best model based on accuracy is C5.0 Tree since its accuracy of test dataset is a bit higher than train dataset and results of two datasets are close. With respect to the sensitivity for our Oblique Random Forest we get 100% which means that we have correctly identified all of the people who churn in our data. As for the specificity for the Oblique Random Forest, we see that it is low which means that we tended to assume that people were going to churn more than they actually were. We get these results because the Oblique Random Forest tended to guess that a customer was going to churn more often than not churn which leads to a high sensitivity but to a low specificity.

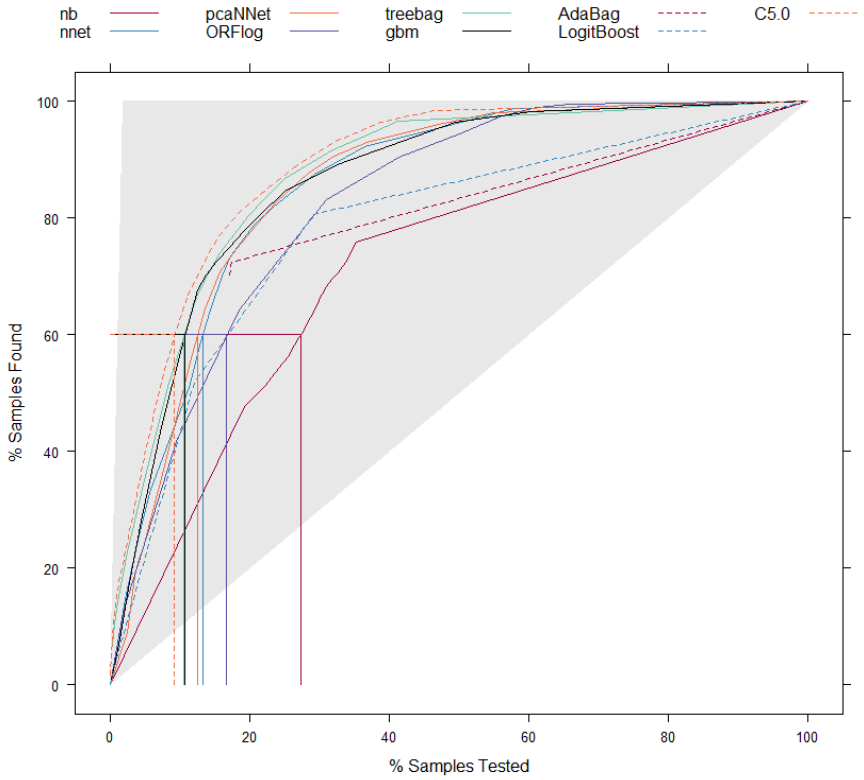
Table 17: Train and test statistics by algorithm.

Model	Train Accuracy	Train Sensitivity	Train Specificity	Test Accuracy	Test Sensitivity	Test Specificity
nb	0.819057	0.895409	0.723617	0.719743	0.621522	0.72156
nnet	0.884458	0.907025	0.856248	0.853869	0.673895	0.857197
pcaNNet	0.880958	0.911831	0.842366	0.843237	0.722586	0.845468
ORFlog	0.70673	1	0.340142	0.333839	0.995908	0.321595
treebag	0.998637	0.99818	0.999207	0.893044	0.625205	0.897998
gbm	0.906069	0.928848	0.877595	0.875332	0.693535	0.878695
AdaBag	0.863256	0.886962	0.833623	0.833735	0.723813	0.835767
LogitBoost	0.876175	0.86698	0.887667	0.880318	0.51964	0.886988
C5.0	0.954131	0.951099	0.95792	0.902837	0.638298	0.907729
svmRadialWeights	0.906989	0.924594	0.884983	0.878074	0.690671	0.88154

Lift Curve

Lift is the concept of how much of an identified population is captured at a given percentile, that is to say, for a given percentile cutoff what is the proportion of positives were identified over how many positives were identified by the model.

Figure 8: Lift curves by algorithm



For our problem, the lift curves for Naïve Bayes, Bagged ADABoost, and LogicBoost did not perform well since they need a higher percentile of the population to find the same amount of sample results as other models. The C5.0 classification tree was the best model based on lift.

Area Under Curve (AUC)

The Area Under Curve is obtained from the area under the receiver operating characteristic (ROC) curve, which plots sensitivity versus 1- specificity. The plot is also known as ROC plot. The area under the curve increases when the model identifies true positives accurately regardless of the false positive performance.

Table 18: AUC by algorithm

Model	AUC
nb	0.772650517
nnet	0.866525556
pcaNNet	0.86538195
ORFlog	0.836493277
treebag	0.887684069
gbm	0.873647309
AdaBag	0.779440607

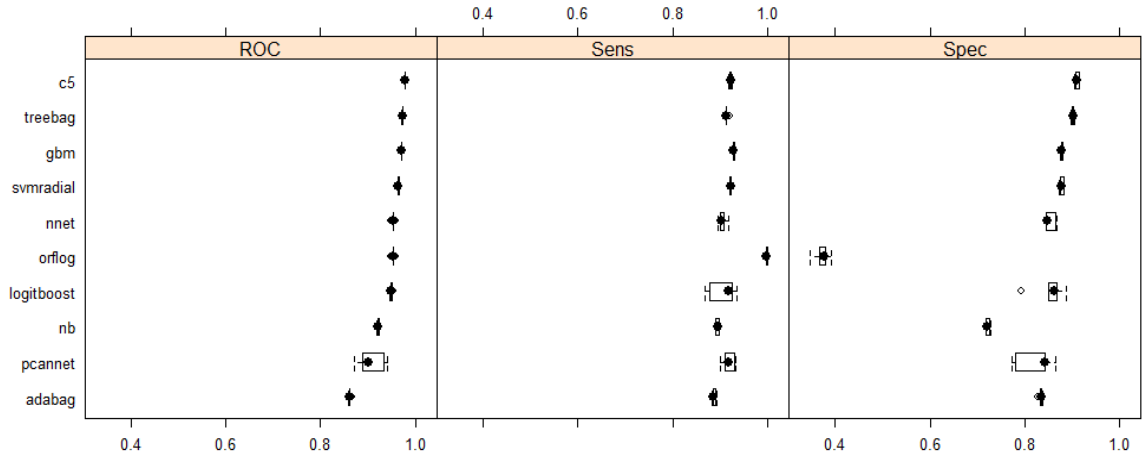
LogitBoost	0.820858647
C5.0	0.899662705
svmRadialWeights	0.775597068

In general, all ten models performed well based on AUC as all values are greater than 0.77. We observed that the C5.0 tree and Bagged CART models were the two best models for this problem among the ten models.

Observational Resampling of Statistics

From a statistical point of view, as the ROC, sensitivity, and specificity are calculated by observational data, the statistical true value of these statistics should be best summarized as a confidence interval. In order to generate the graphs below, the created models on subsets of the test data set and calculated the AUC, sensitivity, and specificity 1,000 times. **Figure 9** shows the robustness of these models.

Figure 9: Boxplot of results



For our data, as there were significantly many observations in the training set, the values of each of the statistics are relatively robust, indicating the model is well trained and that the above conclusions made previously in this paper are not due to incidental values, namely the Oblique Random Forest's markedly low specificity.

Conclusion

With the development of various kinds of predictive analytic tools and models, the selection of predictive models has become more important for companies in the business world. This study demonstrates how the caret package in R provides many sophisticated functions to generate a complete predictive analytics workflow. Caret provides one of the most comprehensive wrappers for any set of R packages and can be solely used to define an entire workflow starting from data cleaning and preprocessing, all the way through model training, prediction, and performance analysis.

In our study, we summarized the available function, show how to use them in order, and show results from a real business problem, using different kinds of algorithms. We believe those new to R or caret will find

this paper useful and a go-to reference to get up to speed more quickly and use some of these functions for their predictive modeling task.

We are currently working to extend this study on the same dataset using the popular scikit-learn package. Since R and Python are most popular analytics languages used by professionals today, we hope to identify functionality that does and does not exist in each, as well as compare both model performance and runtimes. As stated in our paper, some detractors of R believe the performance is not well-equipped enough to support developing and predicting on large datasets and languages such as Python are.

References

Batuwita, R., & Palade, V. (2012). *CLASS IMBALANCE LEARNING METHODS FOR SUPPORT VECTOR MACHINES* (pp. 1-2). University of Oxford Retrieved 12 December 2017, from http://www.cs.ox.ac.uk/people/vasile.palade/papers/Class-Imbalance-SVM.pdf

Brownlee, J. (2017). K-Nearest Neighbors for Machine Learning - Machine Learning Mastery. Machine Learning Mastery. Retrieved 11 December 2017, from https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/

Cortes, C. & Vapnik, V. (1995). Support-vector network. Machine Learning, 20, 1–25.

ctufts/Cheat_Sheets. (2017). GitHub. Retrieved 12 December 2017, from https://github.com/ctufts/Cheat_Sheets/wiki/Classification-Model-Pros-and-Cons

Information on See5/C5.0. (2017). Rulequest.com. Retrieved 12 December 2017, from http://www.rulequest.com/see5-info.html

Lanham, M. (2017) Project Problem: A Comparison of R Caret and Python scikit-learn for Predictive Analytics.

Kuhn, M. (Oct 4, 2007) Caret Package v2.27. Retrieved from https://www.rdocumentation.org/packages/Caret/versions/2.27

Kuhn, M. (2017). The Caret Package. GitHub. Retrieved 14 December 2017, from https://topepo.github.io/Caret/

Sachan, L. (2015). Logistic Regression vs Decision Trees vs SVM: Part II - Edvancer Eduventures. Edvancer.in. Retrieved 11 December 2017, from https://www.edvancer.in/logistic-regression-vs-decision-trees-vs-svm-part2/

Scikit-learn Developers (Nov 21, 2017). Scikit-learn User Guide Release 0.19.1. Retrieved from http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf

Friedman, J. H. (1999, March 26). Stochastic Gradient Boosting. Retrieved February 22, 2018, from https://statweb.stanford.edu/~jhf/ftp/stobst.pdf

Intrator, N. (n.d.). A Neural Network for Feature Extraction. Retrieved February 23, 2018, from https://pdfs.semanticscholar.org/970a/2fa8e2a8a3139a87fa9379dbda0536654a77.pdf

Menze, Bjoern H., et al. On Oblique random Forest. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.207.7485&rep=rep1&type=pdf

Non-Linear Classification in R with Decision Trees. (2016, September 21). Retrieved February 23, 2018, from https://machinelearningmastery.com/non-linear-classification-in-r-with-decision-trees/