Predicting Cold Calling Success for Car Insurance Companies

By Austin Grosel

DePaul University
CSC 529 - Advanced Data Mining
Case Study 2

Introduction

Cold calling has been a function of American business for quite some time. Many companies hire entry level employees to use cold calling to sell or pitch their company's idea or product over the phone, hoping a few customers will listen and eventually buy their product. The task for cold calling is to call as many people as possible so an employee can increase their chances of a sale. If on average, 5% of customers buy a company's product, then the person who calls more people will most likely receive more sales. This is a very inefficient process, though. The 5% success rate in our hypothetical example doesn't factor in any discriminatory analysis that would filter out customers who may likely buy versus customers who most likely will not. A person who has a low amount of money in his or her bank account will probably not buy a product. So if we can cut everyone who has a low salary, or a low balance in their bank account, our success rate will most likely increase. This would be an example of creating a more efficient cold calling process.

Many companies like Mattersight (Mattersight) and ObservePoint (Lincoln) are using predictive analytics to increase their success rates. If we can create a machine learning algorithm to predict whether a customer will buy a product through cold calls, we can gain a leg on other competitors trying to do the same thing.

In this case study, I am given a dataset of cold calling logs from a car insurance company. My goal is to build a classification model to predict which types of customers are more likely to purchase car insurance from our cold calling system. To determine which model is the best, the most important metric I will test against is accuracy. I will be looking at a few different classification algorithms, but will mostly concentrate on support vector machines.

Data Description

The cold calling car insurance dataset was found on Kaggle.com and contains 4000 observations of call logs. There were 19 columns in the dataset, keeping track of different customer information, ranging from balance in their bank account to how long the call lasted. The table below shows the different column names, the value type, and the description.

Customer Call Log table

Variable name	Variable Type	Description
Id	Numeric	Call Log ID#
Age	Numeric	Age of customer
Job	Categorical	The customer's occupation, ranges from admin, blue-collar, entrepreneur, housemaid,

		management, N/A, retired, self-employed, services, student, technician, unemployed
Marital	Categorical	The marital status of the customer, levels are divorced, married, or single
Education	Categorical	Education level, ranges from none, primary, secondary, tertiary
Default	Categorical	If the customer has defaulted on a loan, 0 or 1
Balance	Numeric	Balance left in the customer’s bank account
HHInsurance	Categorical	Does the customer own household insurance, 0 or 1
CarLoan	Categorical	Does the customer have a car loan, 0 or 1
Communication	Categorical	What form of communication was used to contact the customer, levels are telephone, cellular, or null
LastContactDate	Numeric	The day of the month the customer was lasted contacted
LastContactMonth	Categorical	The month the customer was last contacted
NoOfContacts	Numeric	The number of contacts the customer has
DaysPassed	Numeric	The number of days passed since the customer was last contacted
PrevAttempts	Numeric	The amount of attempts the customer was contacted before the current call
PrevAttemptOutcome	Categorical	The outcome from the most previous call attempt, ranging from success, failure, other, or NA
CallStart	Date/Time	Time the phone call started
CallEnd	Date/Time	Time the phone call ended
CarInsurance	Categorical	Did the customer buy car insurance, 0 or 1 (response variable)

There is a lot to fix with this dataset, particularly because there are many outliers for the numerical variables and many of the categorical variables contain null values. The following sections of the data cleaning and exploratory analysis will be combined into one section, as I removed and created new features to play around with in the different classification algorithms.

Data Cleaning & Exploratory Data Analysis

Before I started my analysis, I wanted to check the distribution of the CarInsurance variable (which I renamed to Response). The dataset had a 40.1% success rate. Because there weren't a lot of successes or failures, accuracy seems like it will be a solid metric to evaluate going forward. The other task I wanted to do was remove the call start and end variables. While the data shows that longer calls lead to more sales, this is trivial, and doesn't help identify customers likely to buy insurance prior to attempting phone calls.

My first step of data cleaning came when I observed the numerical features of the dataset. I initially created a correlation plot to see if there was any strong numerical correlations with the response variable. Figure 1 below shows this in action.

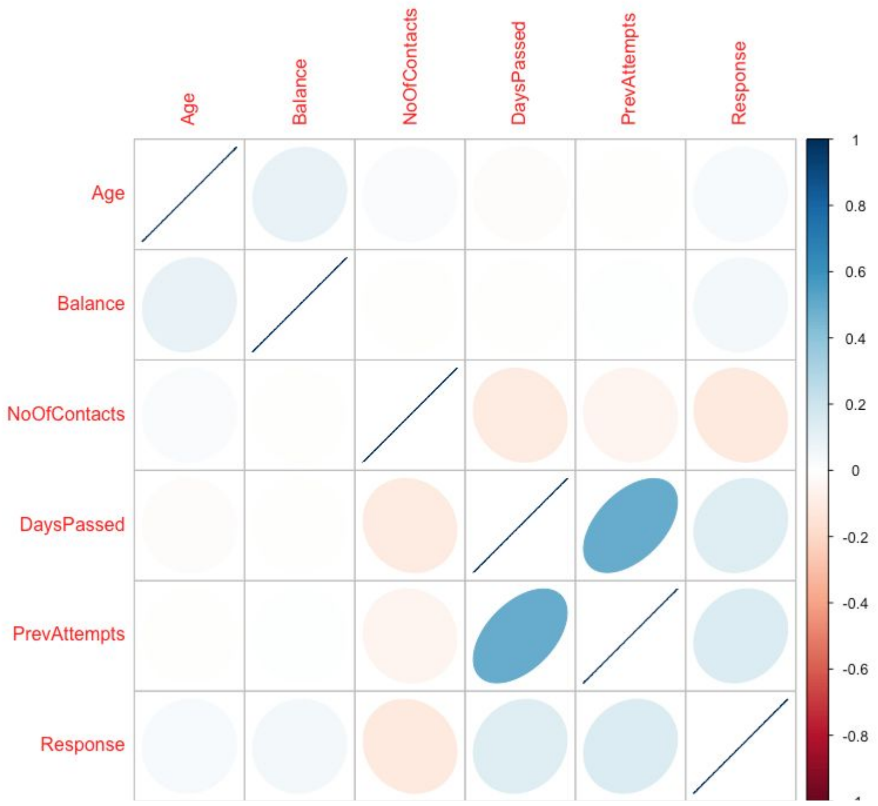


Figure 1. The correlation matrix comparing the numerical features and the response variable.

Unfortunately, there aren't a lot of strong correlations between Response and the numerical features. It looks like each of the numerical features have weak positive correlations, except for NoOfContacts, which has a weak negative correlation. PrevAttempts and DaysPassed seem to be highly correlated with each other, so that's something to keep an eye on going forward.

Ultimately, I'm pretty disappointed with these initial results. But after doing more data visualization, I realized there may be a reason for the poor correlation numbers. Below is the graph of the Balance distribution, regarding a customer's balance in their bank account.

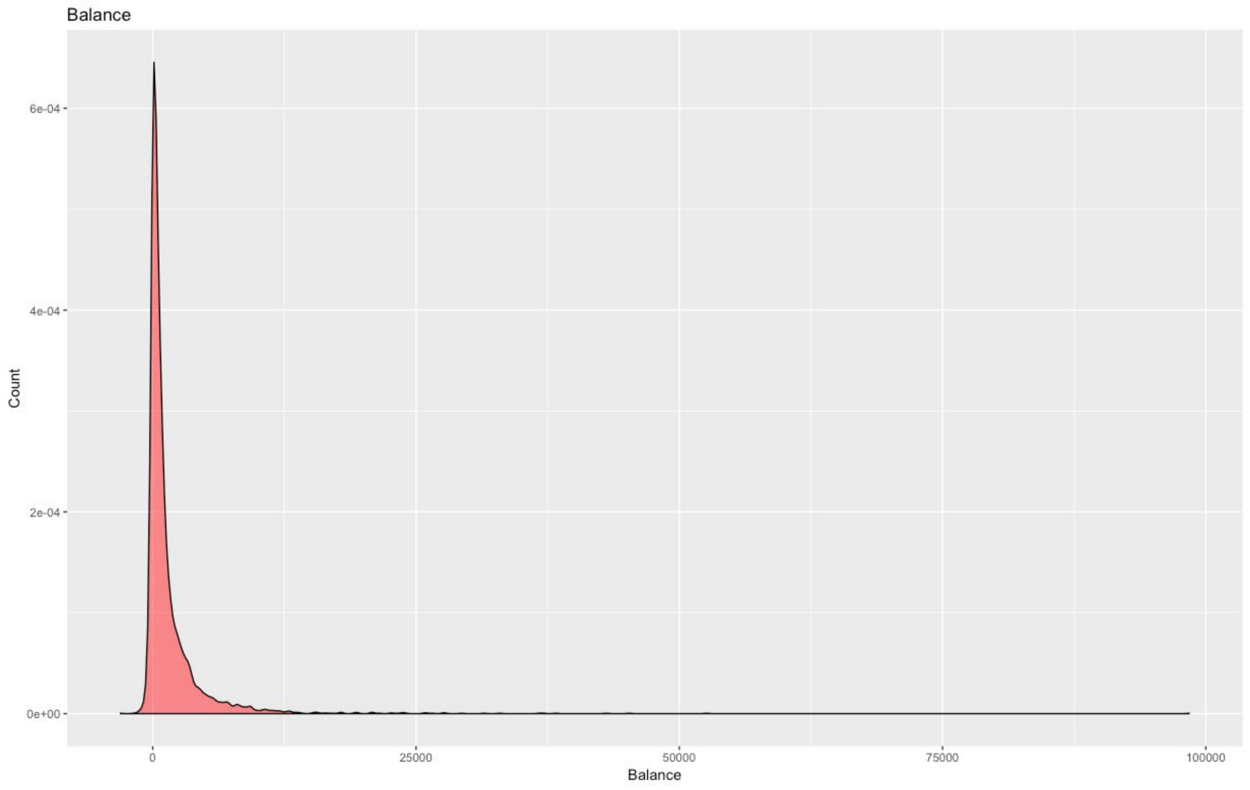


Figure 2. The histogram showing the distributions of the balance variable.

As we can see, this data is totally skewed right due to the presence of one giant outlier. This visualization also makes it seem like the minimum for Balance is zero, when in reality, about 300 customers have a negative bank account balance! The numerical value of Balance seems to be misleading, so I decided to bin the Balance feature into a categorical feature called BalanceBin. The bins of my new feature fell into "Negative", "Low" (below 250), "Normal" (below 2500), "High" (below 10000), and "Very High". Grouping this feature into bins shows that there's a clear discrepancy between the different categories:

Balance Bin	Observations	Success%
Very High	79	43.0%
High	765	48.1%
Normal	1772	42.8%
Low	1119	34.5%

Negative	265	21.9%

The table above shows that there is an increase of success rates when contacting people with sufficient money in their bank account. With such a big difference between the bins, I predict that at least one of these levels will be in our final model.

Next, I looked at the customer age variable. Like the balance feature, this variable also had a low linear correlation to the response variable. Let's graph some density plots to see if there are any age groups that tend to buy insurance.

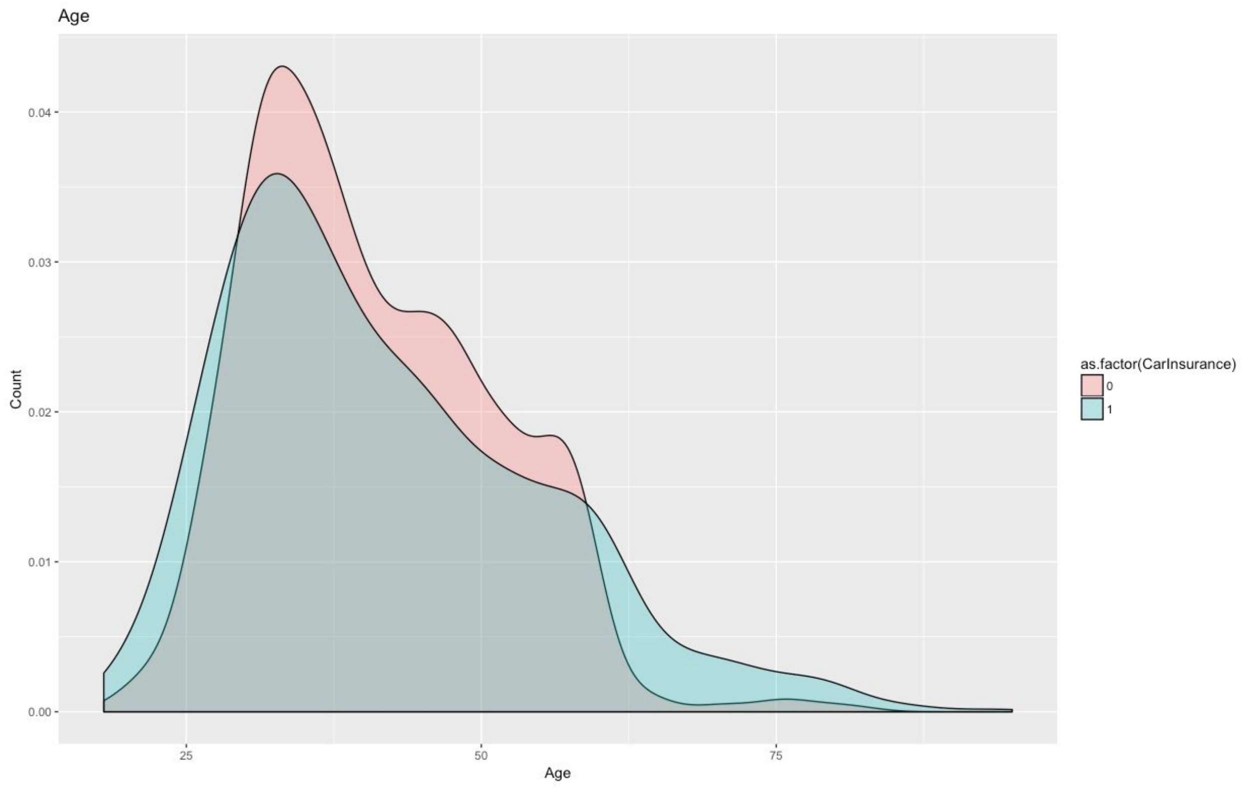


Figure 3. Density plots of showing the Age variable, with the response variable differing in color

This is a significant observation. One of the reasons why age had such a weak correlation was that the tails of the variable are more likely to buy insurance than the middle of the group. Customers younger than 25 years old or older than 60 tend to buy insurance more than other age groups. Therefore, if we divide the age variable into bins, we can get a better of which groups are more likely to buy than others. I created a new variable called AgeBin that had 4 groups: "College-Aged" (younger than 25), "Post-College" (younger than 40), "Middle-aged" (younger than 60), and "Senior".

Next, I wanted to address the different categorical variables in the dataset. I first addressed the date variables. While I was unable to find any correlations of calling at the end of months, I was able to find some interesting observations looking strictly at months. The bar graph below shows the index of each month and the distribution of the response variable.

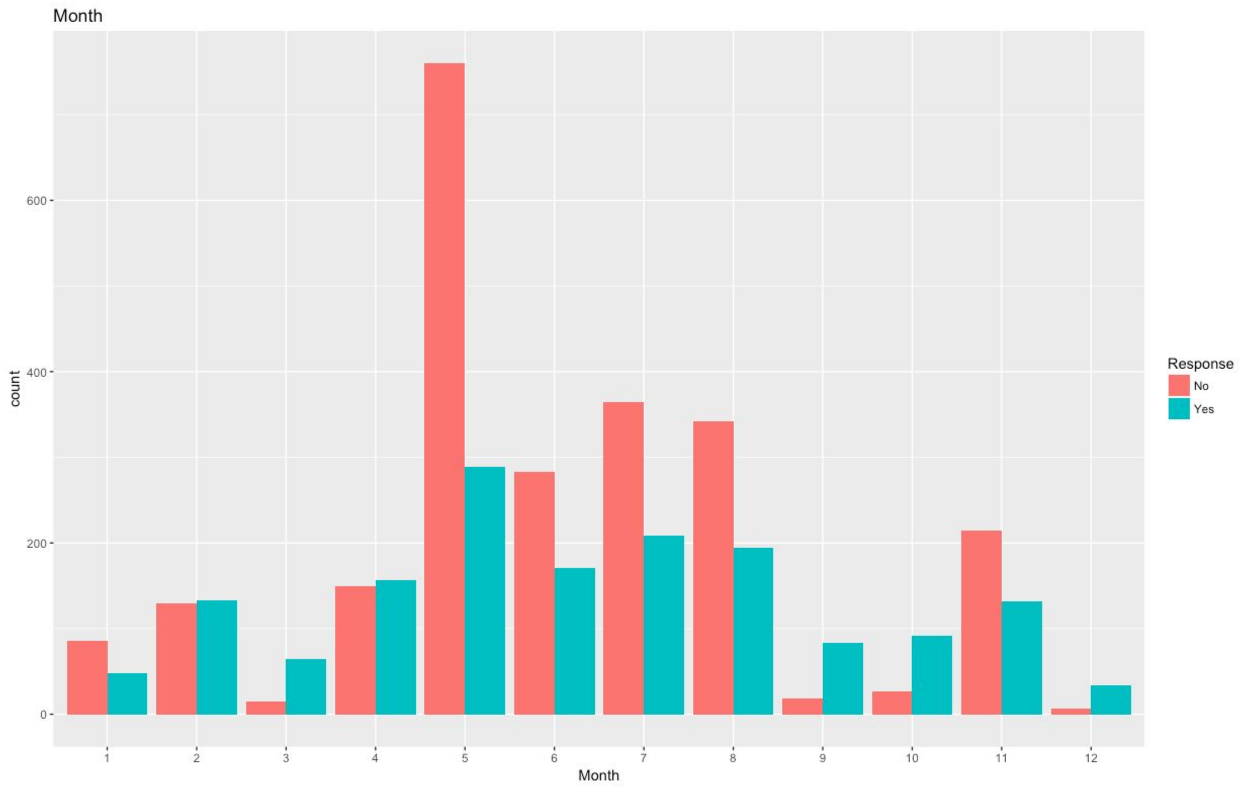
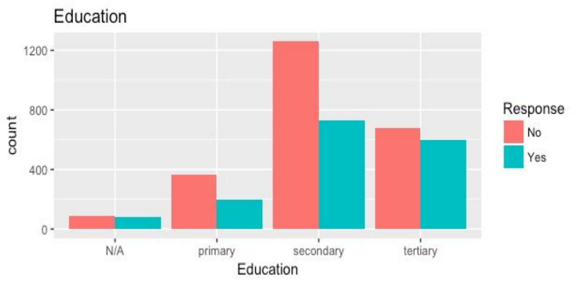
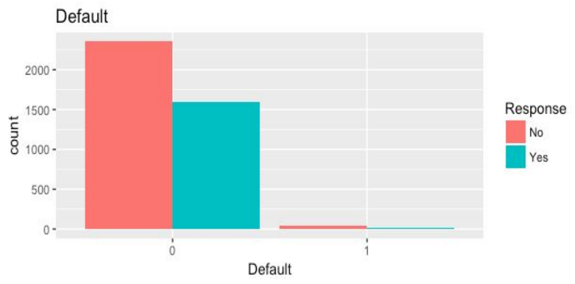


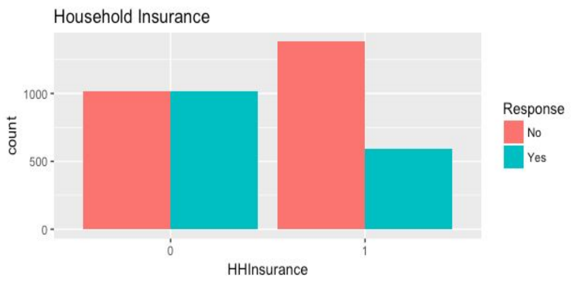
Figure 4. The distribution of the Response variable versus the month indices.

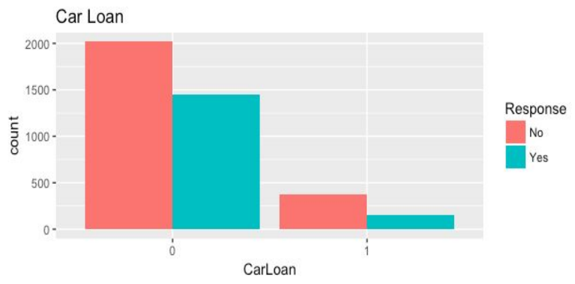
The first observation I noticed is that apart from November, the company seemed to have more success towards the end of the year. This makes sense if the company is located in a colder climate and the weather conditions can make driving less safe. Diving a little deeper, we can see that there seems to be some significance to the different seasons. There's more success in the winter than there is in the spring and summer. I added two more categorical variables to the dataset. The first was called MonthBin, which took had two categories: "Beginning of the year" and "End of the year". The second variable was Season, which had the normal four season categories: "Winter", "Spring", "Summer", and "Fall".

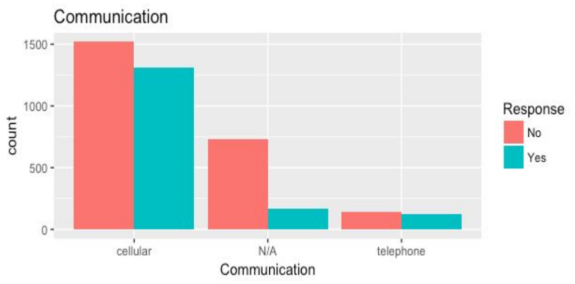
A few of the other categorical variables needed to be fixed before being analyzed. There were empty values in the Education, Communication, and Job features. Because there was no way to determine if a null feature actually meant none, like trying to figure out if a customer had no education background or decided to leave it blank, I decided to insert the category variable "N/A" in each of these features. If an "N/A" value proves to look like a significant value in these features, I will hypothesize why this value may have been left blank. The graph below shows the different categorical variables and their distributions of the response variable.

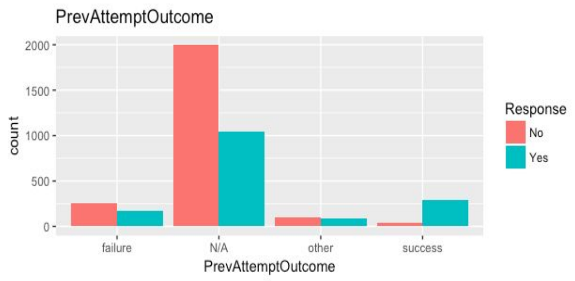












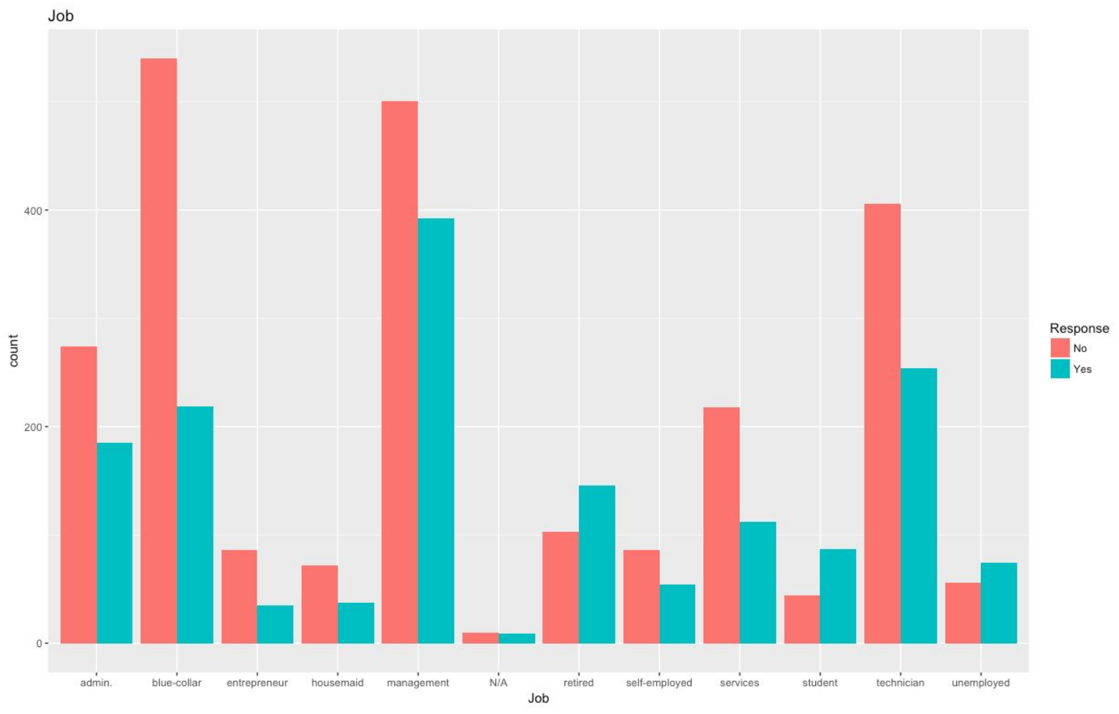
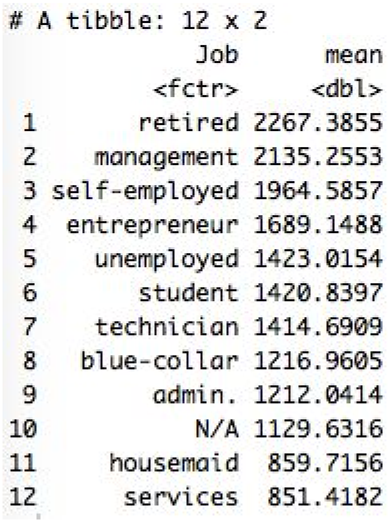


Figure 5. The distribution of the Response variable versus different categorical variables.

There's a lot more to take away from these original categorical variables, starting with HHInsurance. If a customer has household insurance, he or she tends to stay away from buying car insurance. That seems like it will be a significant variable in our models going forward. Another interesting observation is the "N/A" variable in communication. There's very little success using this form of communication. My assumption would be that this "N/A" category is from door-to-door sales, since this method tends to have less success than cold calling. Ultimately, it seems like communication over the phone is much more successful. In PrevAttemptOutcome, "N/A" also shows up as having very little success. However, we see that if we had success with a customer previously, they are much more likely to buy insurance again.

The last categorical variable that has some insights is Job. There are only three jobs that have a greater than 50% chance at selling insurance: "Student", "Retired", and "Unemployed". While the "Student" and "Retired" categories make sense because of what we showed earlier in the age groups, the "Unemployed" category is slightly confusing. We tend to think unemployed customers have lower balances in the bank account, which we showed seems to have a significant effect on the response variable. However, when grouping these customers by their jobs and observing their average balance, we can see that unemployed customers rank 5th out of 12th. This evidence makes their success rate more understandable.



```
# A tibble: 12 x 2
             Job      mean
          <fctr>     <dbl>
 1       retired 2267.3855
 2    management 2135.2553
 3 self-employed 1964.5857
 4  entrepreneur 1689.1488
 5    unemployed 1423.0154
 6       student 1420.8397
 7    technician 1414.6909
 8   blue-collar 1216.9605
 9        admin. 1212.0414
10           N/A 1129.6316
11     housemaid  859.7156
12      services  851.4182
```

Figure 6. The average Balance value for each of the different Job categories.

There were a lot of modifications made to the dataset, but my hope is that some of these categorical variables will appear as significant variables in our models.

Experimental Results and Analysis

To reiterate my goal of this case study, I want to build and select a model that will accurately predict if a customer will purchase insurance or not, with an emphasis on support vector machines. SVM's tend to do well with accuracy, and they perform well on smaller datasets over larger ones because the time to build an SVM can be very long (Bambrick) One of the cons of SVM's is scrutability. Often, SVM's get into high dimensionality and it's tough to find and interpret variable importance. I decided to implement a random forest on the dataset to get an idea of which features were important when building this model. The random forest results are shown below.

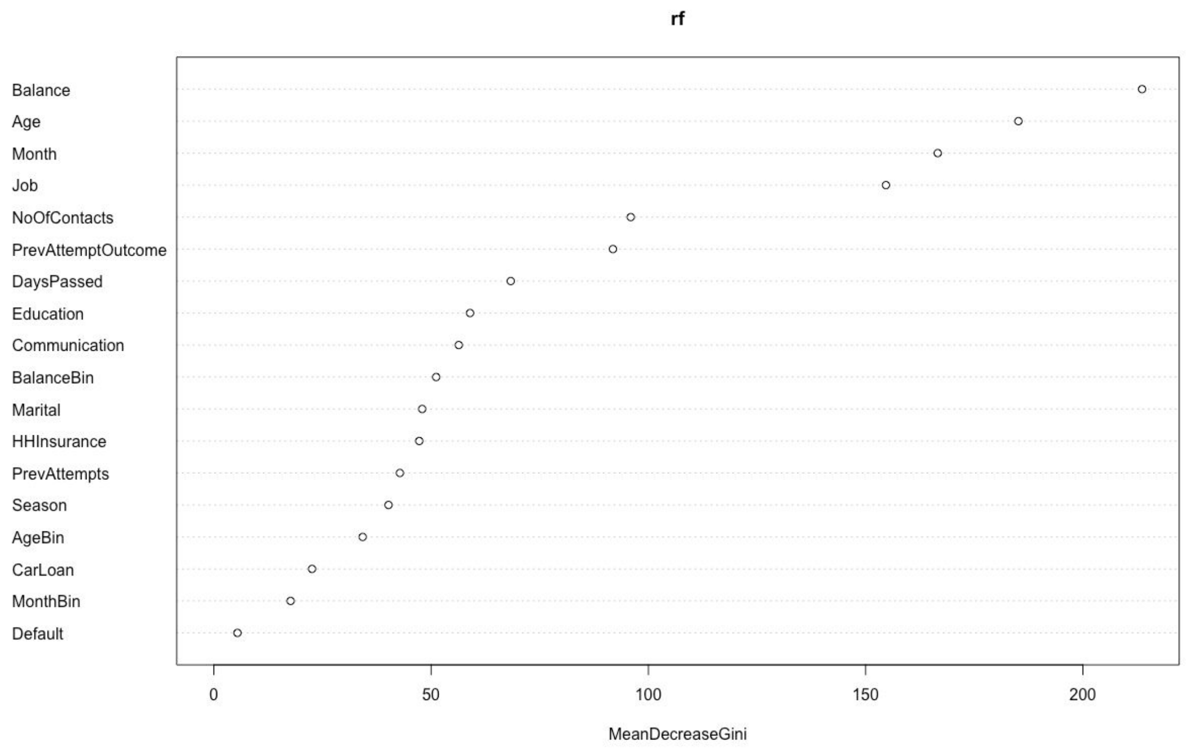


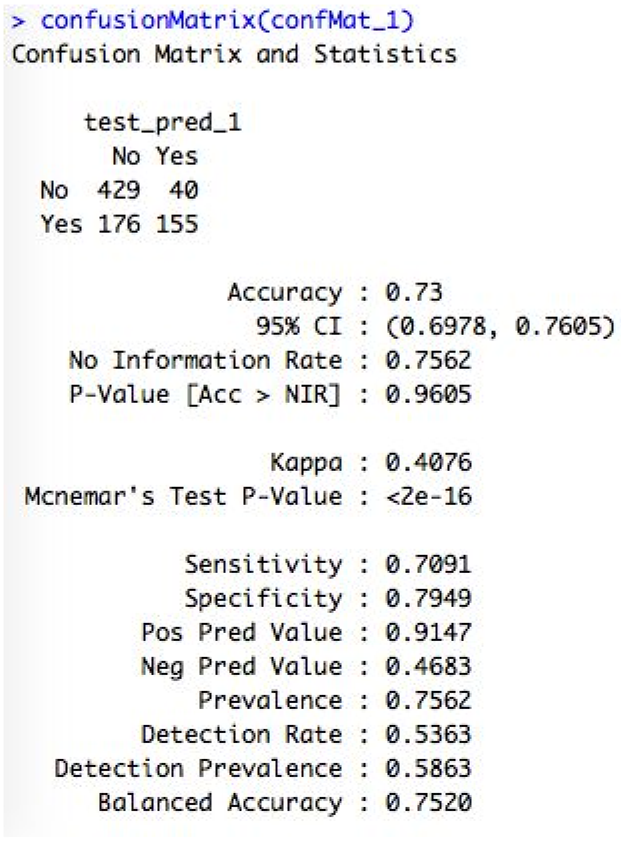
Figure 7. The variable importance plot from the random forest model.

The random forest resulted some pretty surprising observations. First, even though Age and Balance had low correlations with the response variable, they were much more important in this random forest. However, the explanation of this is that decision trees bin numerical features already, so the binned variables are redundant. If I was using a logistic regression, it'd make more sense to use AgeBin and BalanceBin. Another observation is something we pointed out earlier: the month and job variables are very significant. Calling students and retirees will increase success along with identifying which months have been successful in the past. The random forest was able to correctly classify 73.6% the test set, with a 95% confidence interval of (70.4%, 76.7%). I will use that as a benchmark going forward.

The next part of my experiment will be comparing the accuracies of the different kernel support vector machines. For each of these SVM's, I will use be using 10-fold cross validation with three repeats to build the model and compare against the different model parameters. Whichever parameter set gives the highest mean accuracy, I will use that set along with the method on the test set for prediction.

The first support vector machine I tried was using the Gaussian kernel. This is the default kernel used in programming languages like R and Python. In R, there are two parameters used for inputs: sigma and C. Sigma is generally defined as the Gaussian standard deviation. The higher the sigma value, the more generalized and flexible the classifier boundary will be. C is somewhat the opposite of sigma. The more flexible boundary comes when C is lower. This can cause the training accuracy to decrease, but it generalizes the model better for new data.

For tuning purposes, I fluctuated the value of C between 1 and 3, and sigma between 0.1, 0.15, and 0.2. After using the 9 parameters, the most accurate results were when C = 2 and sigma = 0.1. The build time for each of these models took about 36.7 seconds. The accuracy of the Gaussian SVM was 73.0% with a 95% confidence interval of (69.8%, 76.1%).

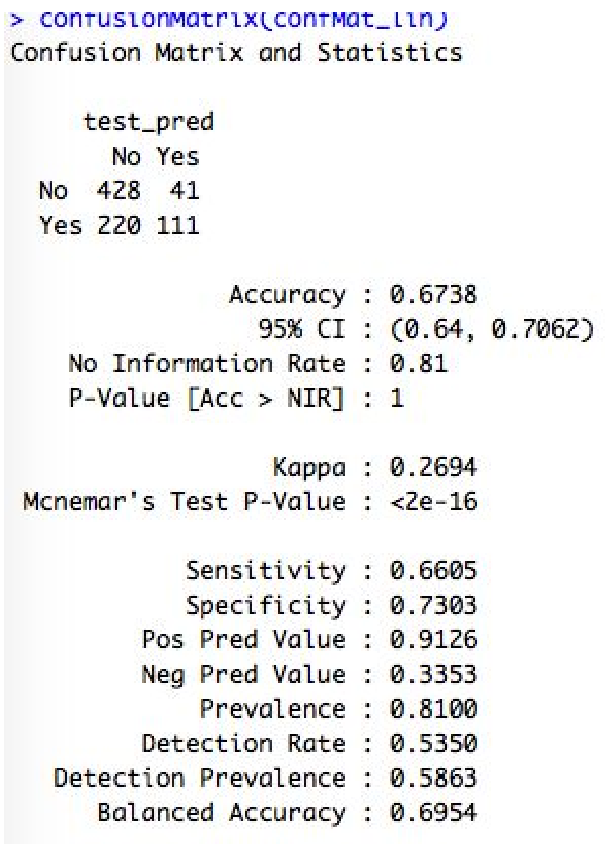


```
> confusionMatrix(confMat_1)
Confusion Matrix and Statistics

     test_pred_1
       No Yes
  No  429  40
  Yes 176 155

               Accuracy : 0.73
                 95% CI : (0.6978, 0.7605)
    No Information Rate : 0.7562
    P-Value [Acc > NIR] : 0.9605

                  Kappa : 0.4076
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.7091
            Specificity : 0.7949
         Pos Pred Value : 0.9147
         Neg Pred Value : 0.4683
             Prevalence : 0.7562
         Detection Rate : 0.5363
   Detection Prevalence : 0.5863
      Balanced Accuracy : 0.7520
```

Figure 8. The confusion matrix from the Gaussian kernel SVM

The second support vector machine kernel attempted was the linear kernel. The linear kernel is generally recognized as a better alternative to the Gaussian kernel when there are more features because there is not much of a need increase dimensionality of the data. Thus, since there aren't too many features and dimensionality in this dataset, I would expect this linear kernel to not be as accurate as the Gaussian kernel, but it doesn't hurt to try.

The only tuning parameter needed for the linear kernel was the variable C, and I set up the classifier to cycle through C equaling 1 to 5. The highest average accuracy was when C = 1, so that set of parameters was used for the test set. The build time of the linear kernel took an average of 60.1 seconds across the different parameter sets. The classifier produced an accuracy of 67.4% on the test set with a 95% confidence interval of (64.0%, 70.6%).

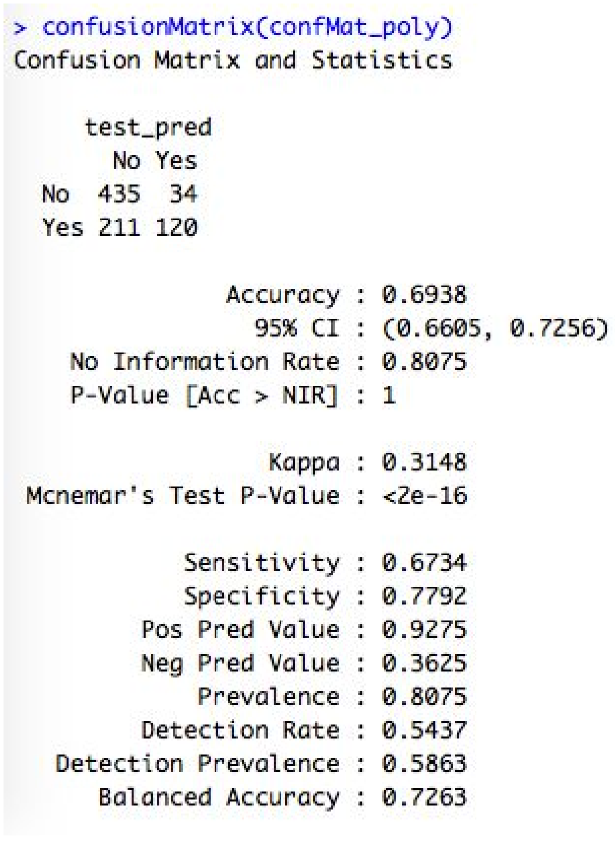


```
> confusionMatrix(confMat_lin)
Confusion Matrix and Statistics

     test_pred
       No Yes
  No  428  41
  Yes 220 111

               Accuracy : 0.6738
                 95% CI : (0.64, 0.7062)
    No Information Rate : 0.81
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2694
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6605
            Specificity : 0.7303
         Pos Pred Value : 0.9126
         Neg Pred Value : 0.3353
             Prevalence : 0.8100
         Detection Rate : 0.5350
   Detection Prevalence : 0.5863
      Balanced Accuracy : 0.6954
```

Figure 9. The confusion matrix of the linear kernel support vector machine.

This seemed to be much lower accuracy than the Gaussian kernel, but I had to prove it with a significance test. When conducting a t-test, I got a p-value of 0.003. Using an alpha = 0.05, I was able to reject the null hypothesis and accept that the Gaussian kernel is more accurate than the linear kernel. Thus, the Gaussian kernel is what we'll be comparing with other models going further.

The last support vector machine kernel I wanted to test was the polynomial kernel. The polynomial kernel in R has three parameters: sigma, C, and degree. The degree parameter refers to the degree of the polynomial equation that the hyperplane to separate the classes is drawn at. I set up the degree parameter to range from 2, 3, to 4, the C variable to vary from 1 to 3, and the sigma variable from 0.1, 0.15, 0.2. The build time of the polynomial kernel was much longer than the other two kernels, averaging 128.3 across the different parameter combinations. The parameter set with the highest accuracy was degree = 2, C = 1, and sigma = 0.1. When using this polynomial kernel with these parameters on the testing set, the classifier registered a 69.4% accuracy with a 95% confidence of (66.1%, 72.6%).



```
> confusionMatrix(confMat_poly)
Confusion Matrix and Statistics

     test_pred
       No Yes
  No  435  34
  Yes 211 120

               Accuracy : 0.6938
                 95% CI : (0.6605, 0.7256)
    No Information Rate : 0.8075
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3148
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6734
            Specificity : 0.7792
         Pos Pred Value : 0.9275
         Neg Pred Value : 0.3625
             Prevalence : 0.8075
         Detection Rate : 0.5437
   Detection Prevalence : 0.5863
      Balanced Accuracy : 0.7263
```

Figure 10. The confusion matrix of the polynomial kernel support vector machine.

I conducted a t-test to compare the accuracies of the Gaussian kernel and the polynomial kernel. The t-test generated a p value of 0.038, and using an alpha = 0.05, I was able to reject the null hypothesis and accept that the accuracies are different. Thus, I chose the Gaussian kernel as the best SVM classifier moving forward because it obtained the highest prediction accuracy on the test set.

For the last part of my analysis, I wanted to compare the differences between the Gaussian kernel support vector machine and the random forest. Overall, ensemble methods like random forests are usually more accurate than single classifiers like SVM's because they're more robust in capturing the different dimensions of a dataset. Random forests are also much easier to show which variables are important to classifying a success or fail. While scrutability is not the most important metric for this case study, it is something that could separate two classifiers if their accuracies are very similar to each other. The table below shows the differences between the Gaussian kernel SVM and the random forest.

Model Comparison Metrics

Classifier	Accuracy	95% Conf Int	Scrutability	Time to build
Random Forest	73.6%	(70.4%, 76.7%)	Good	5.6 seconds
Gaussian SVM	73.0%	(69.8%, 76.1%)	Bad	36.7 seconds

Using a t-test, I got a p-value = 0.87, which means I failed to reject the null hypothesis. Thus, I could not accept the accuracies between the two classifiers are different. However, there are other non-performance metrics like how scrutable the model is and how long it takes to build. These two non-performance metrics make the random forest more desirable for the final model. Therefore, I selected the random forest as the final model for this study.

Conclusion

The goal of this case study was to find the most accurate classifier with an emphasis on support vector machines. While the support vector machine with the Gaussian kernel had a very high accuracy compared to the other kernels, it still was unable to beat the random forest's accuracy. On top of that, the random forest is much easier to show which features are important for calling different customers and the time to build the random forest was much quicker, so the random forest's model was selected over the support vector machines.

The most important features in the random forest were Balance, Age, Job, and Month. Customers with more money tend to buy insurance and the tails of the age distribution are also more likely to buy. Jobs like students and retirees and students are most likely to buy insurance. Finally, October, September, December, and February saw the most success for calls. It was unfortunate that the binned features created in the data cleaning sections didn't show to be as important as the numerical features. But as explained earlier, this was probably because decision trees already place numerical features like salary and age in bins when traversing down the algorithm.

If I were to re-attempt this project, I might want to try a different metric to compare the models against. Because only 40% of customers actually buy insurance, it may be better to look at success probabilities instead of classifying each customer as a success or fail. In this situation, I would use the quadratic loss function to compare sets of models instead of accuracy. This could be a more robust way to find the customers that are extremely likely to buy (about 90% probability) over contacting customers that are somewhat likely to buy (about 55% probability).

Overall, I’m satisfied with these results, and I feel using this random forest model will help this company’s success in selling car insurance.

Code

Setup.R

```r
library(readr)
library(lubridate)
library(caret)

# Load the dataset
carInsurance <- read_csv("~/Desktop/carinsurance/carInsurance_train.csv")

# Convert all the binary/categorical variables to factors
carInsurance$Id = NULL
carInsurance$Marital = as.factor(carInsurance$Marital)
carInsurance$Education = ifelse(is.na(carInsurance$Education), "N/A",
                                      carInsurance$Education)
carInsurance$Education = as.factor(carInsurance$Education)
carInsurance$Communication = ifelse(is.na(carInsurance$Communication), "N/A",
                                      carInsurance$Communication)
carInsurance$Communication = as.factor(carInsurance$Communication)
carInsurance$Job = ifelse(is.na(carInsurance$Job), "N/A",
                                          carInsurance$Job)
carInsurance$Job = as.factor(carInsurance$Job)
carInsurance$Default = as.factor(carInsurance$Default)
carInsurance$HHInsurance = as.factor(carInsurance$HHInsurance)
carInsurance$CarLoan = as.factor(carInsurance$CarLoan)

# Switch PrevOutcome to PrevAttemptOutcome, convert it to a fctor
colnames(carInsurance)[15] = "PrevAttemptOutcome"
carInsurance$PrevAttemptOutcome = ifelse(is.na(carInsurance$PrevAttemptOutcome), "N/A",
                                          carInsurance$PrevAttemptOutcome)
carInsurance$PrevAttemptOutcome = as.factor(carInsurance$PrevAttemptOutcome)

# Get the time the call started and ended
start =  as.POSIXct(carInsurance$CallStart, format = "%H:%M:%S")
end = as.POSIXct(carInsurance$CallEnd, format = "%H:%M:%S")
carInsurance$CallStart = NULL
carInsurance$CallEnd = NULL
carInsurance$CallDuration = round(as.numeric(end-start)/60, 2)

# Bin the ages
carInsurance$AgeBin = "College-Aged"
carInsurance$AgeBin = ifelse(carInsurance$Age > 23 & carInsurance$Age < 30, "Out-of-College",
carInsurance$AgeBin)
carInsurance$AgeBin = ifelse(carInsurance$Age >= 30 & carInsurance$Age < 60, "Middle-Aged",
carInsurance$AgeBin)
carInsurance$AgeBin = ifelse(carInsurance$Age >= 60, "Senior", carInsurance$AgeBin)
carInsurance$AgeBin = as.factor(carInsurance$AgeBin)
```

```
# Get the month index
carInsurance$LastContactMonth = paste0(toupper(substr(carInsurance$LastContactMonth, 1, 1)),
substr(carInsurance$LastContactMonth, 2, 3))
carInsurance$Month = match(carInsurance$LastContactMonth,month.abb)
carInsurance$CallDate <- as.Date(with(carInsurance, paste('2016',carInsurance$Month,
carInsurance$LastContactDay, sep="-")), "%Y-%m-%d")

# Bin the months together
carInsurance$MonthBin = ifelse(carInsurance$Month > 6, "EndOfYear", "BeginningOfYear")
carInsurance$MonthBin = as.factor(carInsurance$MonthBin)

# Create a season variable based on the month index
carInsurance$Season = "Fall"
carInsurance$Season = ifelse(carInsurance$Month > 11 | carInsurance$Month <= 2, "Winter",
carInsurance$Season)
carInsurance$Season = ifelse(carInsurance$Month > 2 & carInsurance$Month <= 5, "Spring",
carInsurance$Season)
carInsurance$Season = ifelse(carInsurance$Month > 5 & carInsurance$Month <= 8, "Summer",
carInsurance$Season)
carInsurance$Season = as.factor(carInsurance$Season)

# Bin the different bank account balances
carInsurance$BalanceBin = "Low"
carInsurance$BalanceBin = ifelse(carInsurance$Balance < 0, "Negative",
carInsurance$BalanceBin)
carInsurance$BalanceBin = ifelse(carInsurance$Balance > 250, "Normal",
carInsurance$BalanceBin)
carInsurance$BalanceBin = ifelse(carInsurance$Balance > 2000, "High", carInsurance$BalanceBin)
carInsurance$BalanceBin = ifelse(carInsurance$Balance > 10000, "Very High",
carInsurance$BalanceBin)
carInsurance$BalanceBin = as.factor(carInsurance$BalanceBin)

# Remove some of the variables, conert month index to a factor
carInsurance$Month = as.factor(carInsurance$Month)
carInsurance$CallDate = NULL
carInsurance$LastContactDay = NULL
carInsurance$LastContactMonth = NULL
carInsurance$CallDuration = NULL

# Convert the response variable into a factor
carInsurance$Response = factor(carInsurance$CarInsurance)
levels(carInsurance$Response) = c("No", "Yes")
carInsurance$CarInsurance = NULL

# Create a test/training set, along with a control variable for the train funcitons.
set.seed(123)
train_index = sample(1:nrow(carInsurance), size = 0.8 * nrow(carInsurance))

train = carInsurance[train_index, ]
```

```
test = carInsurance[-train_index, ]

train_control <- trainControl(method="cv", repeats = 3, number=10, savePredictions = TRUE,
classProbs = TRUE)
```

Ttest.R

```
# train_control, cross validation from caret library that will be used for the following
models
train_control <- trainControl(method="cv", number=10)

# User defined function to get the t-value
t_test = function(a1, a2, s1, s2, N) {
  t = (as.numeric(a1) - as.numeric(a2))/(sqrt((s1^2)/N+(s2^2)/N))
  return(t)
}
```

SVMRBF.R

```
# Create a tuning grid
tune_grid <- expand.grid(sigma = c(.01, .015, 0.2),
                         C = seq(1, 3, 1))

# Build the SVM model
start = proc.time()
svm_fit1 = train(Response ~ ., data = train,
                 method = "svmRadial",
                 preProc = c("center","scale"),
                 trControl = train_control,
                 tuneGrid = tune_grid)
svm_fit1
proc.time() - start

# Get the accuracy and sd metrics
test_pred_1 = predict(svm_fit1, test)
confMat_1 <- table(test$Response, test_pred_1)
a1 = sum(diag(confMat_1))/sum(confMat_1)
confusionMatrix(confMat_1)

s1 = sqrt(sum((as.numeric(test_pred_1)-1 - mean(as.numeric(test_pred_1)-1))^2)/(nrow(test) -
1))
```

SVMLinear.R

```
# Create a tuning grid
tune_grid_linear <- expand.grid(C = seq(1, 5, 1))

# Build the linear SVM model
start = proc.time()
```

```r
svm_fit_linear = train(Response ~ ., data = train,
                 method = "svmLinear",
                 preProc = c("center","scale"),
                 trControl = train_control,
                 tuneGrid = tune_grid_linear)
svm_fit_linear
proc.time() - start

# Get the model metrics of the test set
start = proc.time()
test_pred = predict(svm_fit_linear, test)
proc.time() - start

confMat_lin <- table(test$Response, test_pred)
a2 = sum(diag(confMat_lin))/sum(confMat_lin)

s2 = sqrt(sum((as.numeric(test_pred)-1 - mean(as.numeric(test_pred)-1))^2)/(nrow(test) - 1))

# 300.236 -> 60.1 sec
```

SVMPoly.R

```r
# Tuning grid for polynomial SVM's
tune_grid_poly <- expand.grid(degree = c(2, 3),
                         scale = c(0.1, 0.3),
                         C = seq(1, 2, 1))

# Build the SVM polynomial kernel
start = proc.time()
svm_fit_poly = train(Response ~ ., data = train,
                     method = "svmPoly",
                     preProc = c("center","scale"),
                     trControl = train_control,
                     tuneGrid = tune_grid_poly)
svm_fit_poly
proc.time() - start

# Get the metrics for the polynomial kernel
start = proc.time()
test_pred = predict(svm_fit_poly, test)
proc.time() - start

confMat_poly <- table(test$Response, test_pred)
a3 = sum(diag(confMat_poly))/sum(confMat_poly)
s3 = sqrt(sum((as.numeric(test_pred)-1 - mean(as.numeric(test_pred)-1))^2)/(nrow(test) - 1))

#1025.917 -> 128.25
```

RandomForest.R

```r
library(randomForest)

# Build the random forest model using the train set
start = proc.time()
rf = randomForest(Response ~., data = train)
proc.time() - start

# Predict the results and calculate the accuracy/sd of the test set.
pred_rf = predict(rf, test)
confMat_rf <- table(test$Response, predict(rf, test))

a_rf = sum(diag(confMat_rf))/sum(confMat_rf)
s_rf = sqrt(sum((as.numeric(pred_rf)-1 - mean(as.numeric(pred_rf)-1))^2)/(nrow(test) - 1))
confusionMatrix(confMat_rf)

varImpPlot(rf)
```

EDA.R

```r
library(corrplot)

# Get the correlation matrix
corrplot(cors, method = "ellipse")

# Find the distribution plots of the response variable
g1 = ggplot(carInsurance, aes(PrevAttemptOutcome, ..count..)) + geom_bar(aes(fill = Response),
position = "dodge") + labs(title = "PrevAttemptOutcome", fill = "Response")

g2 = ggplot(carInsurance, aes(Education, ..count..)) + geom_bar(aes(fill = Response), position
= "dodge") + labs(title = "Education", fill = "Response")

g3 = ggplot(carInsurance, aes(Default, ..count..)) + geom_bar(aes(fill = Response), position =
"dodge") + labs(title = "Default", fill = "Response")

g4 = ggplot(carInsurance, aes(HHInsurance, ..count..)) + geom_bar(aes(fill = Response),
position = "dodge") + labs(title = "Household Insurance", fill = "Response")

g5 = ggplot(carInsurance, aes(CarLoan, ..count..)) + geom_bar(aes(fill = Response), position =
"dodge") + labs(title = "Car Loan",  fill = "Response")

g6 = ggplot(carInsurance, aes(Communication, ..count..)) + geom_bar(aes(fill = Response),
position = "dodge") + labs(title = "Communication", fill = "Response")

grid.arrange(g2,g3,g4,g5,g6,g1,nrow = 3)
```

```
ggplot(carInsurance, aes(Job, ..count..)) + geom_bar(aes(fill = Response), position = "dodge")
+ labs(title = "Job", fill = "Response")

h1 = ggplot(carInsurance, aes(Balance)) + geom_density(alpha = 0.5, color = 'black', fill =
'red') + labs(title = "Balance", x = "Balance", y = "Count")

h2 = ggplot(carInsurance, aes(Age)) + geom_density(alpha = 0.5, color = 'black', fill = 'red')
+ labs(title = "Age", x = "Age", y = "Count")

grid.arrange(h1,h2,nrow = 2)

carInsurance$BalanceBin = factor(carInsurance$BalanceBin, levels = c('Negative', 'Low',
'Normal', 'High', 'Very High'))
g7 = ggplot(carInsurance, aes(BalanceBin, ..count..)) + geom_bar(aes(fill = Response),
position = "dodge") + labs(title = "Balance", fill = "Response")

carInsurance$AgeBin = factor(carInsurance$AgeBin, levels = c('College-Aged', 'Out-of-College',
'Middle-Aged', 'Senior'))
g8 = ggplot(carInsurance, aes(AgeBin, ..count..)) + geom_bar(aes(fill = Response), position =
"dodge") + labs(title = "Age", fill = "Response")

grid.arrange(g7,g8, nrow = 2)

ggplot(carInsurance, aes(Month, ..count..)) + geom_bar(aes(fill = Response), position =
"dodge") + labs(title = "Month", fill = "Response")

g9 = ggplot(carInsurance, aes(MonthBin, ..count..)) + geom_bar(aes(fill = Response), position
= "dodge") + labs(title = "Month", fill = "Response")

g10 = ggplot(carInsurance, aes(Season, ..count..)) + geom_bar(aes(fill = Response), position =
"dodge") + labs(title = "Season", fill = "Response")

grid.arrange(g9,g10, nrow = 2)

ggplot(carInsurance) + geom_point(aes(x = Age, y = CarInsurance, fill = CarInsurance)) +
labs(title = "Age", x = "Age", y = "Count")

ggplot(carInsurance, aes(Age)) + geom_density(aes(group = CarInsurance, fill =
as.factor(CarInsurance)), alpha = 0.3) + labs(title = "Age", x = "Age", y = "Count")
```

Works Cited

Lincoln, Melissa. “Cold-Calling reps are heating up sales with data science.” *Leadspace,* https://www.leadspace.com/cold-calling-reps-are-heating-up-sales-with-data-science-predictive-analytics/

“Mattersight.com” *NFL Play Data.* https://www.mattersight.com/

Bambrick, Noel. “Support Vector Machines: A simple explanation.” *KDD Nuggets*, https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html