Lasso Regularization , Logistic Regression, and Random Forest

Mason Del Rio

11/20/2020

Logistic Regression

We are working with the "bank-additional-full.csv" dataset. In order to select which features matter the most, we will use Lasso Regularization on our Logistic Regression Model.

The equation for Logistic Regression is :

$P = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$

Where P is our target variable which varies from 0 to 1, and the β_0, β_1, and X are the independent Variables.

Or similarly, $log(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 ... \beta_k X_k$ Where $log(\frac{p_i}{1-p_i})$ is the Log Likelihood.

We first index the data into a Training Set and a Testing Set using the sample function.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
library(ggplot2)
library(data.table)
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(class)
library(glmnet)

set.seed(1)
#Load Data
setwd("/Users/masondelrio/Desktop/Fall 2020 Files/Fall 2020 Handouts/STA 141A/STA 141A Datasets")
bank.data = read.csv("bank-additional-full.csv", sep = ";")
index = sample(1:nrow(bank.data), 0.7*nrow(bank.data))
train = bank.data[index,]
test = bank.data[-index,]
dim(train)
```

```
## [1] 28831    21
```

```
dim(test)
```

```
## [1] 12357    21
```

As we can see, 70% of our data was split for indexing to our training and testing data.

$28831/41188 = 0.69999854$

We now have our data split up, randomly, most importantly, and we can fit our logistic model to the training data using the glm() function.

```
glm_model <- glm(y~., family = binomial, data = train)
glm_model
```

```
##
## Call:  glm(formula = y ~ ., family = binomial, data = train)
##
## Coefficients:
##                   (Intercept)                           age
##                     -2.592e+02                    -5.202e-04
##                jobblue-collar                jobentrepreneur
##                     -2.230e-01                    -1.544e-01
##                  jobhousemaid                  jobmanagement
##                     -1.330e-01                    -3.910e-02
##                     jobretired               jobself-employed
##                      1.840e-01                    -1.270e-01
##                    jobservices                     jobstudent
##                     -1.110e-01                     1.575e-01
##                  jobtechnician                  jobunemployed
##                     -3.973e-03                    -1.252e-01
##                     jobunknown                 maritalmarried
##                     -2.090e-01                     3.441e-02
##                  maritalsingle                 maritalunknown
##                      4.207e-02                     1.514e-01
##             educationbasic.6y             educationbasic.9y
##                     -1.325e-02                    -1.003e-02
##          educationhigh.school           educationilliterate
##                      3.500e-02                     1.484e+00
## educationprofessional.course    educationuniversity.degree
##                      4.533e-02                     1.207e-01
##              educationunknown                defaultunknown
##                      1.078e-01                    -3.288e-01
##                    defaultyes                housingunknown
##                     -7.295e+00                    -6.708e-02
##                    housingyes                   loanunknown
##                     -1.777e-02                            NA
##                       loanyes               contacttelephone
##                     -8.333e-02                    -7.356e-01
##                      monthaug                      monthdec
##                      7.636e-01                     4.606e-01
##                      monthjul                      monthjun
##                      5.486e-02                    -5.818e-01
##                      monthmar                      monthmay
##                      1.992e+00                    -4.789e-01
##                      monthnov                      monthoct
##                     -5.174e-01                     1.576e-01
```

```
##                     monthsep                day_of_weekmon
##                    4.115e-01                    -1.396e-01
##                day_of_weekthu                day_of_weektue
##                    2.131e-02                     8.368e-02
##                day_of_weekwed                     duration
##                    1.997e-01                     4.651e-03
##                     campaign                         pdays
##                   -3.827e-02                    -1.272e-03
##                     previous           poutcomenonexistent
##                   -1.137e-01                     3.566e-01
##               poutcomesuccess                  emp.var.rate
##                    6.784e-01                    -1.777e+00
##                cons.price.idx                 cons.conf.idx
##                    2.340e+00                     3.016e-02
##                    euribor3m                   nr.employed
##                    2.543e-01                     7.329e-03
##
## Degrees of Freedom: 28830 Total (i.e. Null);  28778 Residual
## Null Deviance:       20360
## Residual Deviance: 12010     AIC: 12110
```

In order to see how accurate our model is, we need to set up a confusion matrix. We use the test data that was separate from the training model in order to simulate some form of real world implementation and feed it into the predict.glm() function.

```
glm_prob <- predict.glm(glm_model, test, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
glm_predict<- rep("no", nrow(test))
glm_predict[glm_prob>.5] <- "yes"
table(pred=glm_predict, true = test$y)
```

```
##      true
## pred     no   yes
##   no  10705   793
##   yes   274   585
```

```
mean(glm_predict == test$y)
```

```
## [1] 0.9136522
```

The Logistic Regression model works fairly well for the test data, with 91.3% accuracy. This is a model that works well with the data that was randomized and set once, for training and testing purposes. However, we must use K-Fold Cross Validation in order to test the model's ability to predict outside data not used in the process of creating this model.

Logistic Regression with K-Fold Cross Validation

We will use the caret library to implement cross validation. Below, we will again create a training and testing set for the data. However, this time the K-Fold Cross Validation will randomly select portions of the training data 10 times.

```
bank = read.csv("bank-additional-full.csv", sep = ";")
#Install caret function

library(caret)
```

```
## Loading required package: lattice
```

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#Partition Data: Create index matrix of selected values
set.seed(1)


# Create index matrix
bank = na.omit(bank)
# data preparation
bank <- bank %>% mutate(marital = ifelse(as.character(marital) =="married", 1, 0))
bank <- bank %>% mutate(housing = ifelse(as.character(housing) =="yes", 1, 0))
bank <- bank %>% mutate(loan = ifelse(as.character(loan) =="yes", 1, 0))
bank <- bank %>% mutate(y = ifelse(as.character(y) =="yes", 1, 0))
bank$job <- as.numeric(factor(bank$job, levels = as.character(unique(bank$job))))
bank$education <- as.numeric(factor(bank$education, levels = as.character(unique(bank$education))))
bank$default <- as.numeric(factor(bank$default, levels = as.character(unique(bank$default))))
bank$contact <- as.numeric(factor(bank$contact, levels = as.character(unique(bank$contact))))
bank$month <- as.numeric(factor(bank$month, levels = as.character(unique(bank$month))))
bank$day_of_week <- as.numeric(factor(bank$day_of_week, levels = as.character(unique(bank$day_of_week))))
bank$poutcome <- as.numeric(factor(bank$poutcome, levels = as.character(unique(bank$poutcome))))

#Don't want list, we get a matrix.
index <- createDataPartition(bank$y, p = .7, list = FALSE, times = 1)

#Convert Dataframe bank to actual df
bank <- as.data.frame(bank)
# create train dataframe and test dataframe
train_df <- bank[index,]
test_df <- bank[-index,]

#Relabel values of y to factor (1 = yes, 0 = no), because Logistic Regression needs factors for the out
train_df$y[train_df$y == 1] <- "yes"
train_df$y[train_df$y == 0] <- "no"

test_df$y[test_df$y == 1] <- "yes"
test_df$y[test_df$y == 0] <- "no"

#Convert outcome variable to type factor with function as.factor()
```

```r
train_df$y <- as.factor(train_df$y)
test_df$y <- as.factor(test_df$y)

#Specify type of method, in this case cross validation, and number of folds will be 10.
control <- trainControl(method = "cv", number = 10, savePredictions = "all",
                        classProbs = TRUE)
#Set random seed for folds
set.seed(1)

#Specify logistic regression model with method 'glm'.

model1 <- train(y ~., data = train_df, method = 'glm', family = binomial,
                trControl = control)
```

After the cross validation is done, we need to see how well our improved model performs on test data. We use the predict() function to store the predictions generated by our model in predictions.

```r
#Predict outcome using trained model with test data.
predictions <- predict(model1, newdata = test_df)
```

Now, we use the confusionMatrix() function from the caret package to generate our accuracy.

```r
confusionMatrix(data = predictions, test_df$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  10696   833
##        yes   282   545
##
##                Accuracy : 0.9098
##                  95% CI : (0.9046, 0.9148)
##     No Information Rate : 0.8885
##     P-Value [Acc > NIR] : 5.554e-15
##
##                   Kappa : 0.4482
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9743
##             Specificity : 0.3955
##          Pos Pred Value : 0.9277
##          Neg Pred Value : 0.6590
##              Prevalence : 0.8885
##          Detection Rate : 0.8657
##    Detection Prevalence : 0.9331
##       Balanced Accuracy : 0.6849
##
##        'Positive' Class : no
##
```

As we can see, the Logisitic Regression Model with and without cross validation has ~90-91% accuracy, but we need to add some form of bias because this model is overfit to this certain dataset. If we wanted to generalize this data to other banks in Portugal, we would need to compensate for the overfittness of this model to this dataset.

This is where Lasso Regularization comes in.

Lasso Regularization

Lasso Regularization for Logistic Regression is done by minimizing this Cost Function:

$$J(\Theta) = \frac{1}{m}\Sigma_{i=1}^{m}[-y^i log(h_\Theta(x^i)) - (1-y^i)log(1-(h_\Theta(x^i))] + \frac{\lambda}{2m}\Sigma_{j=1}^{n}|\Theta_j|$$

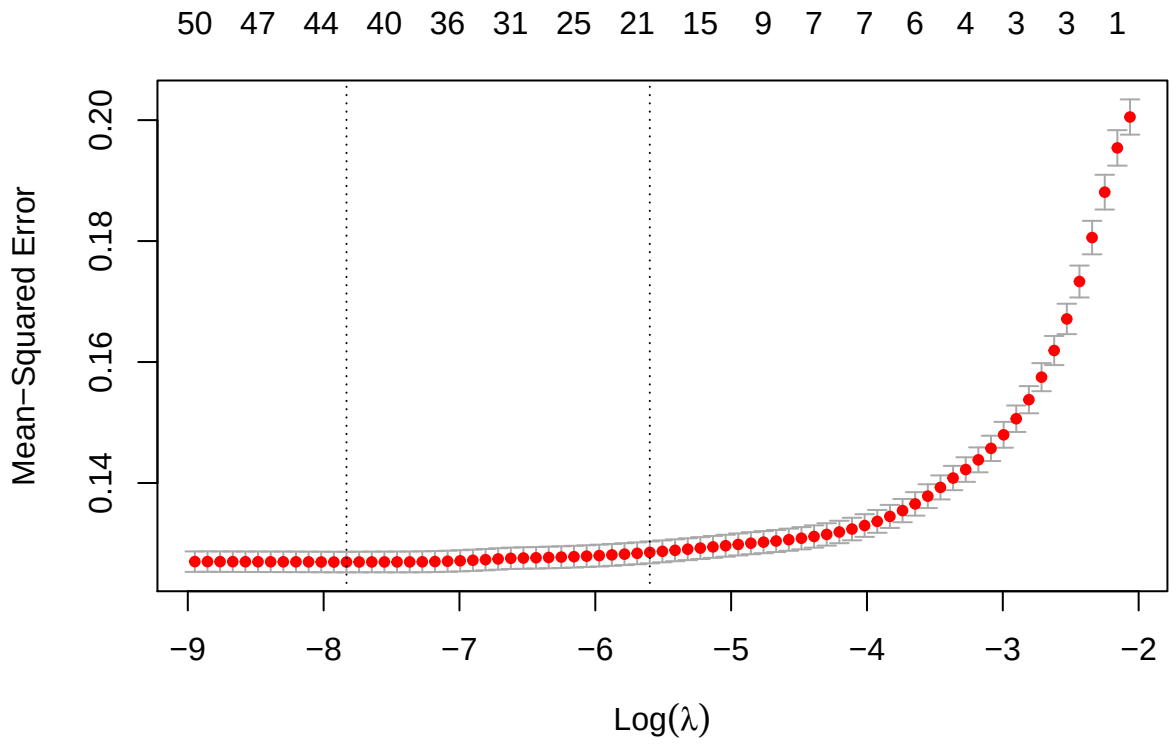
First, we build a matrix for all the features of the dataset, which we'll call x, and a vector for the response variable, we'll call y, which consists of 1's and 0's for Yes and No.

```
x <- model.matrix(y~., train)

y<- ifelse(train$y == "yes", 1,0)
```

In Lasso Regularization, there is a tuning paramater called Lambda in which we need to find the minimal value of, through cross validation. The minimum lambda will give us the most parsimonious model, but it will also generate a model which generally overfits. So we will pick the lambda which falls one standard error away from the minimum value in order to choose the simplest model without overfitting.

```
#Cross Validation for lambda parameter
cv.out <- cv.glmnet(x,y,alpha=1, family="binomial", type.measure = "mse")
#Show log plot
plot(cv.out)
```




```
#Minimum lambda
lambda_min <- cv.out$lambda.min
#Lambda value that gives simplest model but also lies within one standard error of the optimal value of
lambda_1se <- cv.out$lambda.1se
```

From this graph, we see that the lambda value between -8 and -7 will give us the most minimal model, but one standard error away from that value will give us the lambda value near -5, which will give us the minimal

model with the least overfitting.

```
lambda_min
```

```
## [1] 0.0003969794
```

```
#Lambda value that gives simplest model but also lies within one standard error of the optimal value of
lambda_1se
```

```
## [1] 0.003702243
```

```
#Show coefficients of lassomodel
coef(cv.out, s = lambda_1se)
```

```
## 55 x 1 sparse Matrix of class "dgCMatrix"
##                                        1
## (Intercept)                 46.3112124489
## (Intercept)                  .
## age                          .
## jobblue-collar              -0.1211438092
## jobentrepreneur              .
## jobhousemaid                 .
## jobmanagement                .
## jobretired                   0.0937744287
## jobself-employed             .
## jobservices                  .
## jobstudent                   0.0950202311
## jobtechnician                .
## jobunemployed                .
## jobunknown                   .
## maritalmarried               .
## maritalsingle                .
## maritalunknown               .
## educationbasic.6y            .
## educationbasic.9y            .
## educationhigh.school         .
## educationilliterate          .
## educationprofessional.course .
## educationuniversity.degree   0.0167761111
## educationunknown             .
## defaultunknown              -0.1761173523
## defaultyes                   .
## housingunknown               .
## housingyes                   .
## loanunknown                  .
## loanyes                      .
## contacttelephone            -0.1618307419
## monthaug                     .
## monthdec                     .
## monthjul                     .
## monthjun                     0.0517527751
## monthmar                     1.0229018486
## monthmay                    -0.8062421344
## monthnov                    -0.2427831094
## monthoct                     .
## monthsep                    -0.0338662302
```

```
## day_of_weekmon               -0.0492829737
## day_of_weekthu                .
## day_of_weektue                .
## day_of_weekwed                0.0221837727
## duration                      0.0042790778
## campaign                      .
## pdays                        -0.0009976067
## previous                      .
## poutcomenonexistent           0.1642595712
## poutcomesuccess               0.6431301954
## emp.var.rate                 -0.1796682179
## cons.price.idx                .
## cons.conf.idx                 0.0098710133
## euribor3m                     .
## nr.employed                  -0.0094368591
```

From the list of coefficients for the lasso model, we see that it selected 12 features that were the most significant on the model.

We now use our test data to see how accurate our model is.

```
x_test <- model.matrix(y~., test)
lasso_prob <- predict(cv.out, newx = x_test, s = lambda_1se, type = "response")
lasso_predict<- rep("no", nrow(test))
lasso_predict[lasso_prob>.5]<- "yes"
#confusion matrix
table(pred= lasso_predict, true = test$y)
```

```
##      true
## pred     no   yes
##   no  10728   868
##   yes   251   510
```

```
mean(lasso_predict == test$y)
```

```
## [1] 0.909444
```

As we see, the accuracy of this model has decreased because we introduced some bias with the Lasso Regularization method. In order for this model to be applied to other data sets, we will sacrifice accuracy for the sake of not overfitting the model.

Random Forest

Another method we can use on this data is the Random Forest Method.

Using Random Forest gives us a vast amount of decision trees in which a model is made from the culmination of them. Using the ranodmForest library package, we can create the model in R. We first subset our data just like we previously did for the Logistic Regression model, with 70% of the data lumped into a Training Set, and 30% of our data lumped into the Test set.

```
library(randomForest)   # random forest methodology
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

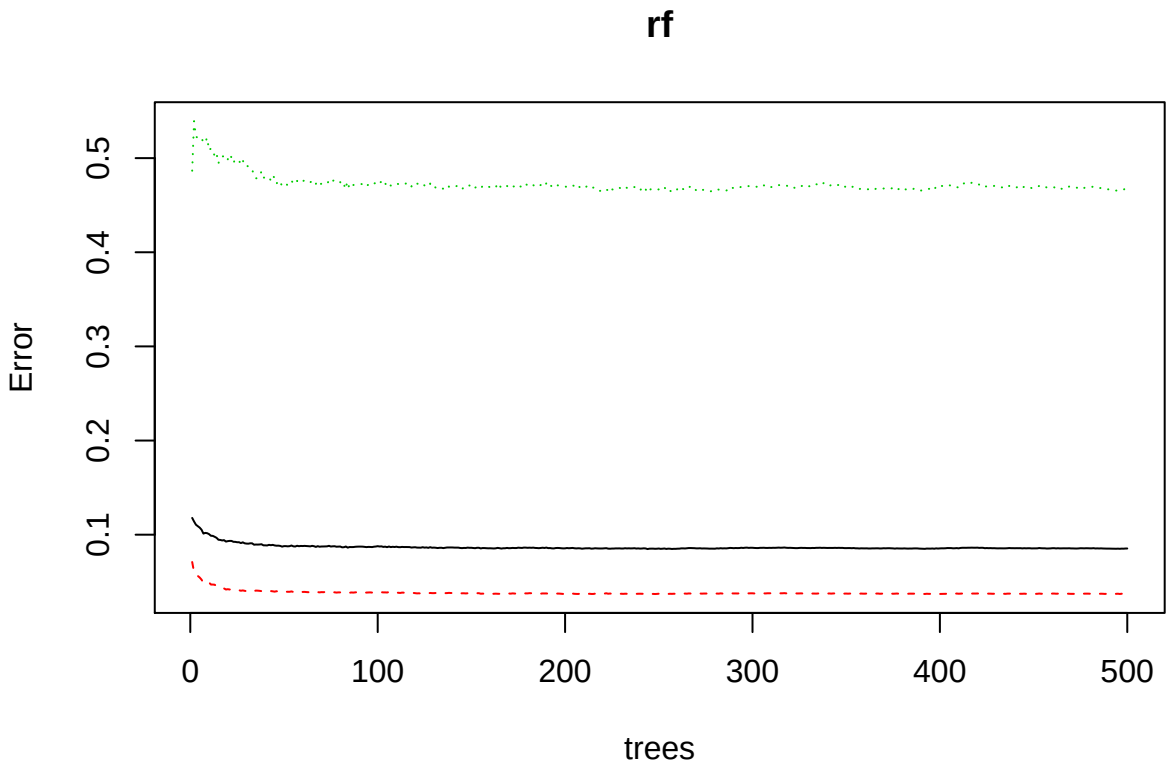
```
bank.data = na.omit(bank.data)
bank.data$y = as.factor(bank.data$y)
dataset.size = floor(nrow(bank.data)*0.70)
index <- sample(1:nrow(bank.data),size= dataset.size)
training <- bank.data[index,]
test <- bank.data[-index,]
```

We then use the function "randomForest()" and train it with the previously created training set, which will create our random forest model to use for predictions.

```
rf <- randomForest(y~., data = training, importance = T)
```

After this is made, we can plot to see how many trees are necessary for our model

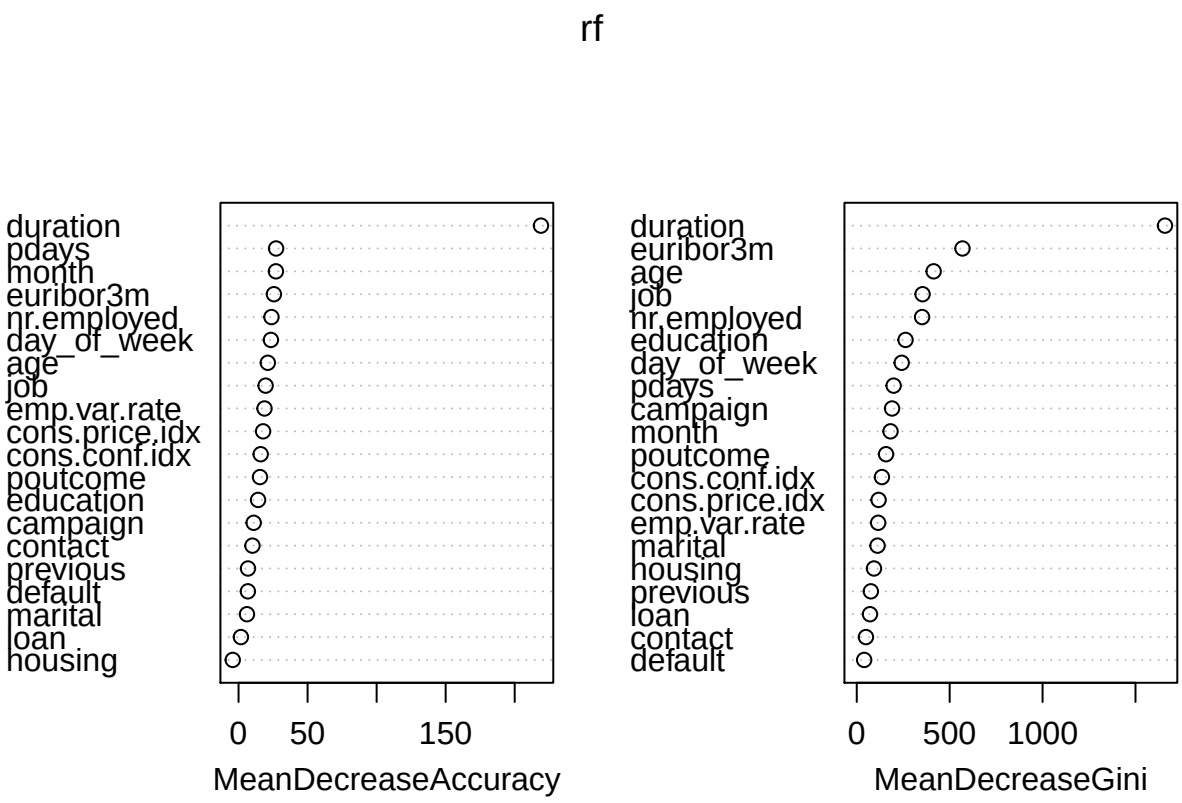
```
plot(rf)
```



This graph shows us that around 100-200 trees is where our model has minimum error

Now, let's confirm the variables of importance for the dataset using the varImpPlot() function

```
varImpPlot(rf)
```



As we can see, duration ranks number 1 most important variable in this dataset, which from eyeballing the data, we see the strong relationship between a phone calls duration and the result of a term deposit being made. This graph confirms this notion.

Now, let's see how accurate this model is after feeding the model our test data. We will use the caret package to create the confusion matrix.

```
library(caret)
rftest <- predict(rf, newdata = test)
confusionMatrix(data = rftest, test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  10587   708
##        yes   352   710
##
##                Accuracy : 0.9142
##                  95% CI : (0.9091, 0.9191)
##     No Information Rate : 0.8852
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.526
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9678
```

```
##             Specificity : 0.5007
##          Pos Pred Value : 0.9373
##          Neg Pred Value : 0.6685
##              Prevalence : 0.8852
##          Detection Rate : 0.8568
##    Detection Prevalence : 0.9141
##       Balanced Accuracy : 0.7343
##
##        'Positive' Class : no
##
```

From this confusion matrix, we can see that the accuracy was at 91.34%, and the Kappa value was at .52, which makes for a moderate model. This model gives us around the same accuracy as the logistic regression model.