Final Project - Bank Dataset

Aaron Abromowitz, Stephanie Duarte, Dammy Owolabi

2024-04-20

# Before Part 2

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.1 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

# Pull in data  
data<-read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/bank-additional-full.csv',stringsAsFactors = T, sep=";")  
  
# Set levels to use for later  
data$y <- relevel(data$y, ref="yes")  
data$month <- factor(data$month, levels=c('mar','apr','may','jun','jul','aug','sep','oct','nov','dec'))  
data$day\_of\_week <- factor(data$day\_of\_week, levels=c('mon','tue','wed','thu','fri'))  
  
# Duration was removed since the dataset explanation file said that it was created after y variable was known, so shouldn't be used for prediction.  
data$duration <- c()  
data$default <- c()  
  
# Create the train and test split  
train\_perc <- .8  
set.seed(1234)  
train\_indices <- sample(nrow(data), floor(train\_perc \* nrow(data)))  
train\_data <- data[train\_indices, ]  
nrow(train\_data)

## [1] 32950

test\_data <- data[-train\_indices, ]   
nrow(test\_data)

## [1] 8238

#GLMNET Model

library(readr)  
library(GGally)

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(ggcorrplot)  
# Prepare the matrix of predictors  
x <- model.matrix(~ . - 1 - y, data = train\_data) # Excludes the intercept and the response variable  
  
# Define the trainControl with classProbs enabled  
fitControl <- trainControl(method = "cv",   
 number = 10,   
 classProbs = TRUE, # Enable class probability predictions  
 summaryFunction = twoClassSummary) # Use a summary function for classification  
  
# Run the glmnet model  
set.seed(1234) # for reproducibility  
glmnet\_fit <- train(x, y = train\_data$y,   
 method = "glmnet",  
 trControl = fitControl,  
 tuneLength = 10, # Number of lambda values to test  
 metric = "ROC") # Optimize the model based on ROC curve  
  
# View the best model's lambda value and corresponding coefficients  
best\_lambda <- glmnet\_fit$bestTune$lambda  
coef(glmnet\_fit$finalModel, s = best\_lambda)

## 52 x 1 sparse Matrix of class "dgCMatrix"  
## s1  
## (Intercept) 1.271357e+02  
## age 6.288258e-04  
## jobadmin. -5.537285e-02  
## jobblue-collar 1.042755e-01  
## jobentrepreneur -3.234962e-03  
## jobhousemaid -1.306103e-02  
## jobmanagement .   
## jobretired -3.078555e-01  
## jobself-employed 3.618493e-02  
## jobservices 5.156474e-02  
## jobstudent -2.873845e-01  
## jobtechnician .   
## jobunemployed 5.206854e-02  
## jobunknown 7.563443e-02  
## maritalmarried -2.397840e-02  
## maritalsingle -2.912019e-02  
## maritalunknown -4.058666e-01  
## educationbasic.6y .   
## educationbasic.9y 2.706680e-02  
## educationhigh.school -1.043184e-02  
## educationilliterate -3.876507e-01  
## educationprofessional.course -6.937688e-02  
## educationuniversity.degree -1.123063e-01  
## educationunknown -6.533999e-02  
## housingunknown 8.105645e-02  
## housingyes 3.384646e-02  
## loanunknown 1.362202e-02  
## loanyes 4.709627e-02  
## contacttelephone 6.154443e-01  
## monthapr 1.206473e+00  
## monthmay 1.744218e+00  
## monthjun 1.548731e+00  
## monthjul 1.116250e+00  
## monthaug 8.789533e-01  
## monthsep 1.219657e+00  
## monthoct 1.305819e+00  
## monthnov 1.619636e+00  
## monthdec 7.376210e-01  
## day\_of\_weektue -2.501461e-01  
## day\_of\_weekwed -3.222477e-01  
## day\_of\_weekthu -2.604626e-01  
## day\_of\_weekfri -2.037529e-01  
## campaign 4.271160e-02  
## pdays 9.320162e-04  
## previous 6.642481e-02  
## poutcomenonexistent -3.905654e-01  
## poutcomesuccess -9.262589e-01  
## emp.var.rate 1.023679e+00  
## cons.price.idx -1.356221e+00  
## cons.conf.idx -2.066279e-02  
## euribor3m -1.918785e-01  
## nr.employed .

There definitely is a change in the data over time. We may re-visit this later on.

# Objective 1: Simple Logistic Regerssion Model

We used a combination of forward and backward selection to determine a simple logistic regression model that performed well on the data. We did this by adding each variable to a model, and then doing Cross Validation to test the out of sample data on AUROC (Area under the ROC curve). We used 10 folds. After we chose the first variable, we then did this to add more variables to see if those increased the AUROC. We also tried removing variables (once we got three of them) to see if that improved the score. Below, I included example code for this logic. It takes several minutes to run though, so it is not set to evaluate the code. One caveat is that since the Default variable only has 2 Yes values, it can cause errors sometimes during the cross validation. This happens when the training data (a randomly chosen 90%) doesn’t have either of those values, and the test data (the other 10%) has both of them. This only happens 1% of the time, but when you are running 100s of tests, this happens regularly. And since Default didn’t increase the AUROC much anyways, we decided to drop the variable.

train\_data$default <- c()

# Forward Selection Example  
set.seed(21)  
vars <- names(train\_data)  
vars <- vars[vars!="y"]  
num\_vars <- length(vars)  
var\_aucs <- data.frame("vars" = vars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- vars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}  
  
# Backward selection example  
set.seed(24)  
start\_form\_str <- 'y ~ nr.employed + month + poutcome'  
vars <- c('nr.employed','month','poutcome')  
num\_vars <- length(vars)  
var\_aucs <- data.frame("vars" = vars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- vars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste(start\_form\_str," -",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

After we did this until the AUROC didn’t increase any more, the resulting model was y ~ month + poutcome + emp.var.rate + euribor3m + contact + cons.price.idx. We then checked the p values and VIR to see if it made sense to keep all of those variables.

library(car)

## Loading required package: carData

##   
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following object is masked from 'package:purrr':  
##   
## some

model <- glm(y ~ month + poutcome + emp.var.rate + euribor3m + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + euribor3m +   
## contact + cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7180 0.2565 0.3225 0.3652 2.0610   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 165.34420 9.72777 16.997 < 2e-16 \*\*\*  
## monthapr 1.48149 0.11493 12.891 < 2e-16 \*\*\*  
## monthmay 1.95366 0.10893 17.936 < 2e-16 \*\*\*  
## monthjun 1.99307 0.13976 14.261 < 2e-16 \*\*\*  
## monthjul 1.34672 0.12631 10.662 < 2e-16 \*\*\*  
## monthaug 0.77035 0.11695 6.587 4.48e-11 \*\*\*  
## monthsep 1.23224 0.14671 8.399 < 2e-16 \*\*\*  
## monthoct 1.48029 0.15076 9.819 < 2e-16 \*\*\*  
## monthnov 1.93888 0.13899 13.950 < 2e-16 \*\*\*  
## monthdec 0.88523 0.21637 4.091 4.29e-05 \*\*\*  
## poutcomenonexistent -0.41002 0.06036 -6.793 1.10e-11 \*\*\*  
## poutcomesuccess -1.81692 0.08686 -20.919 < 2e-16 \*\*\*  
## emp.var.rate 1.50678 0.10080 14.948 < 2e-16 \*\*\*  
## euribor3m -0.53296 0.07687 -6.933 4.12e-12 \*\*\*  
## contacttelephone 0.63484 0.06856 9.260 < 2e-16 \*\*\*  
## cons.price.idx -1.73621 0.10191 -17.037 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18332 on 32934 degrees of freedom  
## AIC: 18364  
##   
## Number of Fisher Scoring iterations: 6

vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## month 6.253389 9 1.107207  
## poutcome 1.288981 2 1.065520  
## emp.var.rate 76.465448 1 8.744452  
## euribor3m 51.138803 1 7.151140  
## contact 1.990009 1 1.410677  
## cons.price.idx 11.218806 1 3.349449

The p values were all significant at the 0.05 level, but there was a high amount of correlation between emp.var.rate and euribor3m. So we removed euribor3m to see if that didn’t make the model too much worse.

library(caret)  
library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

set.seed(80)  
form <- as.formula('y ~ month + poutcome + emp.var.rate + contact + cons.price.idx')  
num\_folds <- 10  
folds <- createFolds(train\_data$y, k = num\_folds)  
accuracy\_scores <- numeric(num\_folds)  
auc\_scores <- numeric(num\_folds)  
for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
}  
mean(auc\_scores)

## [1] 0.7917643

It wasn’t too much worse. And removing the correlation between those two variables makes the model easier to interpret.

model <- glm(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + contact +   
## cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7139 0.2257 0.3265 0.3616 1.9460   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 108.53001 5.14668 21.087 < 2e-16 \*\*\*  
## monthapr 1.36684 0.11411 11.979 < 2e-16 \*\*\*  
## monthmay 1.88593 0.10846 17.388 < 2e-16 \*\*\*  
## monthjun 1.47245 0.11859 12.416 < 2e-16 \*\*\*  
## monthjul 1.02367 0.11750 8.712 < 2e-16 \*\*\*  
## monthaug 0.69115 0.11645 5.935 2.94e-09 \*\*\*  
## monthsep 1.01545 0.14351 7.076 1.49e-12 \*\*\*  
## monthoct 1.03319 0.13677 7.554 4.21e-14 \*\*\*  
## monthnov 1.44456 0.11940 12.098 < 2e-16 \*\*\*  
## monthdec 0.55440 0.21012 2.638 0.00833 \*\*   
## poutcomenonexistent -0.43221 0.05976 -7.232 4.75e-13 \*\*\*  
## poutcomesuccess -1.81937 0.08629 -21.083 < 2e-16 \*\*\*  
## emp.var.rate 0.82587 0.02213 37.325 < 2e-16 \*\*\*  
## contacttelephone 0.43524 0.06009 7.244 4.37e-13 \*\*\*  
## cons.price.idx -1.14550 0.05491 -20.863 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18379 on 32935 degrees of freedom  
## AIC: 18409  
##   
## Number of Fisher Scoring iterations: 6

vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## month 2.394736 9 1.049711  
## poutcome 1.274785 2 1.062574  
## emp.var.rate 3.656646 1 1.912236  
## contact 1.506607 1 1.227439  
## cons.price.idx 3.284604 1 1.812348

Now the VIFs are much more resonable without euribor3m. So we chose the simpler model for our Simple Logistic Regression Model.

Now that we have a model, we look at the model coefficients for interpretations.

train\_data$y <- relevel(train\_data$y, ref="no")  
mod <- glm(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx, data = train\_data, family = "binomial")  
summary(mod)

##   
## Call:  
## glm(formula = y ~ month + poutcome + emp.var.rate + contact +   
## cons.price.idx, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.9460 -0.3616 -0.3265 -0.2257 2.7139   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -108.53001 5.14668 -21.087 < 2e-16 \*\*\*  
## monthapr -1.36684 0.11411 -11.979 < 2e-16 \*\*\*  
## monthmay -1.88593 0.10846 -17.388 < 2e-16 \*\*\*  
## monthjun -1.47245 0.11859 -12.416 < 2e-16 \*\*\*  
## monthjul -1.02367 0.11750 -8.712 < 2e-16 \*\*\*  
## monthaug -0.69115 0.11645 -5.935 2.94e-09 \*\*\*  
## monthsep -1.01545 0.14351 -7.076 1.49e-12 \*\*\*  
## monthoct -1.03319 0.13677 -7.554 4.21e-14 \*\*\*  
## monthnov -1.44456 0.11940 -12.098 < 2e-16 \*\*\*  
## monthdec -0.55440 0.21012 -2.638 0.00833 \*\*   
## poutcomenonexistent 0.43221 0.05976 7.232 4.75e-13 \*\*\*  
## poutcomesuccess 1.81937 0.08629 21.083 < 2e-16 \*\*\*  
## emp.var.rate -0.82587 0.02213 -37.325 < 2e-16 \*\*\*  
## contacttelephone -0.43524 0.06009 -7.244 4.37e-13 \*\*\*  
## cons.price.idx 1.14550 0.05491 20.863 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 18379 on 32935 degrees of freedom  
## AIC: 18409  
##   
## Number of Fisher Scoring iterations: 6

odds\_ratio <- exp(mod$coefficients)  
confident\_interval <- exp(confint(mod))

## Waiting for profiling to be done...

train\_data$y <- relevel(train\_data$y, ref="yes")

Interpretations (interpreting individually):

Monthapr The odds of getting the clients subscribing to a term deposit in the month of April is 0.255 times lower than that of subscribing in the month of March with a CI of (0.204, 0.319)

Monthmay The odds of getting the clients subscribing to a term deposit in the month of May is 0.152 times lower than that of subscribing in the month of March with a CI of (0.123, 0.188)

Monthjun The odds of getting the clients subscribing to a term deposit in the month of June is 0.229 times lower than that of subscribing in the month of March with a CI of (0.182, 0.289).

Monthjul The odds of getting the clients subscribing to a term deposit in the month of July is 0.359 times lower than that of subscribing in the month of March with a CI of (0.285, 0.452).

Monthaug The odds of getting the clients subscribing to a term deposit in the month of August is 0.500 times lower than that of subscribing in the month of March with a CI of (0.399, 0.629)

Monthsep The odds of getting the clients subscribing to a term deposit in the month of September is 0.362 times lower than that of subscribing in the month of March with a CI of (0.273, 0.479)

Monthoct The odds of getting the clients subscribing to a term deposit in the month of October is 0.356 times lower than that of subscribing in the month of March with a CI of (0.272, 0.465)

Monthnov The odds of getting the clients subscribing to a term deposit in the month of November is 0.236 times lower than that of subscribing in the month of March with a CI of (0.187, 0.298)

Monthdec The odds of getting the clients subscribing to a term deposit in the month of December is 0.574 times lower than that of subscribing in the month of March with a CI of (0.380, 0.868)

poutcomenonexistent The odds of getting the clients subscribing to a term deposit based on a nonexistent outcome of the previous campaign is 1.54 times higher than that of a failed outcome with a CI of (1.37, 1.73)

poutcomesuccess The odds of getting the clients subscribing to a term deposit based on a succesful outcome of the previous campaign is 6.17 times higher than that of a failed outcome with a CI of (5.21, 7.31)

emp.var.rate For every 1 unit increase in customer subscription to a term deposit, the odds of the customer subscribing based on the employment variation rate decreases by 56.2% with a CI of (58.1%, 54.3%)

contacttelephone The odds of getting the clients subscribing to a term deposit based on the contact communication type is 0.647 times lower than that of subscribing by cell phone with a CI of (0.575, 0.728)

cons,price.idx For every 1 unit increase in customer subscription to a term deposit, the odds of the customer subscribing based on the consumers price index increases by a factor of 3.14 with a CI of (2.82, 3.50)

# Objective 2: Complex Logistic Regerssion Model

For the second model, we looked into adding polynomial terms and/or interaction terms to the regression model.

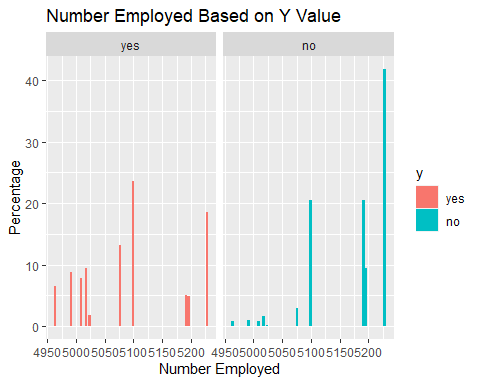
## Polynomial Terms

To see if it made sense to add some polynomial terms we looked at what happened with adding those for Number of Employees, since that was the first variable added during Forward Selection (although it got removed later).

library(tidyverse)  
  
# Make a nr.employed^2 variable  
train\_data$ne2 = (train\_data$nr.employed)^2  
  
# Plot nr.employed  
summary <- train\_data %>%  
 group\_by(nr.employed,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'nr.employed'. You can override using the  
## `.groups` argument.

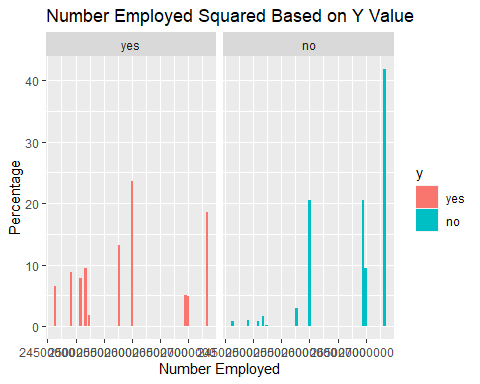
summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=nr.employed,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Number Employed') + ggtitle('Number Employed Based on Y Value')



# Plot nr.employed^2  
summary <- train\_data %>%  
 group\_by(ne2,y) %>%  
 summarize(count=n())

## `summarise()` has grouped output by 'ne2'. You can override using the `.groups`  
## argument.

summary$perc <- 0  
summary$perc[summary$y == 'no'] <- summary$count[summary$y == 'no'] / nrow(train\_data[train\_data$y == 'no',]) \* 100  
summary$perc[summary$y == 'yes'] <- summary$count[summary$y == 'yes'] / nrow(train\_data[train\_data$y == 'yes',]) \* 100  
summary %>% ggplot(aes(x=ne2,y=perc,fill=y)) + geom\_bar(stat="identity") + facet\_wrap(~y) +   
 ylab('Percentage') + xlab('Number Employed') + ggtitle('Number Employed Squared Based on Y Value')

 These plots look very hard to distinguish. So instead of plotting polynomial terms, we tried creating simple models and then calculating out of sample AUC. If this increases as polynomial degree increases, then it could make sense to include polynomial terms.

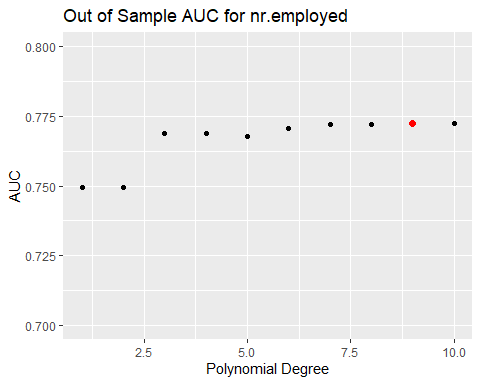
# Maybe just plot the improvement of out of sample AUC  
set.seed(120)  
vars <- 'nr.employed'  
allVars <- vars  
num\_poly <- 10  
for (i in 1:length(vars)){  
 for (j in 2:num\_poly){  
 if (class(train\_data[,vars[i]]) != "factor") {  
 allVars <- c(allVars,paste('poly(',vars[i],',',j,')',sep=""))  
 }  
 }  
}  
num\_vars <- length(allVars)  
var\_aucs <- data.frame("vars" = allVars)  
num\_folds <- 10  
for (j in 1:num\_vars) {  
 var <- allVars[j]  
 print(var)  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

## [1] "nr.employed"  
## [1] "poly(nr.employed,2)"  
## [1] "poly(nr.employed,3)"  
## [1] "poly(nr.employed,4)"  
## [1] "poly(nr.employed,5)"  
## [1] "poly(nr.employed,6)"  
## [1] "poly(nr.employed,7)"  
## [1] "poly(nr.employed,8)"  
## [1] "poly(nr.employed,9)"  
## [1] "poly(nr.employed,10)"

# Get max val  
maxAUC <- max(var\_aucs$auc, na.rm = TRUE)  
maxDeg <- which.max(var\_aucs$auc)  
  
# Looking at the p values for the highest degree model  
model <- glm(form, data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = form, family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6432 0.2485 0.3321 0.3578 1.2860   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 2.44760 0.02338 104.676 < 2e-16 \*\*\*  
## poly(nr.employed, 10)1 156.34441 3.19323 48.961 < 2e-16 \*\*\*  
## poly(nr.employed, 10)2 -44.45416 3.14126 -14.152 < 2e-16 \*\*\*  
## poly(nr.employed, 10)3 -51.90203 4.32051 -12.013 < 2e-16 \*\*\*  
## poly(nr.employed, 10)4 4.92844 2.83743 1.737 0.082398 .   
## poly(nr.employed, 10)5 26.51619 2.34879 11.289 < 2e-16 \*\*\*  
## poly(nr.employed, 10)6 30.20002 3.42973 8.805 < 2e-16 \*\*\*  
## poly(nr.employed, 10)7 8.50557 3.68814 2.306 0.021100 \*   
## poly(nr.employed, 10)8 -2.79958 2.14882 -1.303 0.192628   
## poly(nr.employed, 10)9 11.69125 3.00755 3.887 0.000101 \*\*\*  
## poly(nr.employed, 10)10 -7.20951 2.02859 -3.554 0.000379 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 19131 on 32939 degrees of freedom  
## AIC: 19153  
##   
## Number of Fisher Scoring iterations: 6

# Plot the AUC improving  
var\_aucs %>% ggplot(aes(x=1:10, y=auc)) + geom\_point() + ylim(c(0.7,0.8)) +   
 ylab('AUC') + xlab('Polynomial Degree') + ggtitle('Out of Sample AUC for nr.employed') +   
 geom\_point(data = data.frame(x = maxDeg, y = maxAUC),   
 aes(x = x, y = y), size = 2, color = "red", fill = "red", shape = 21)

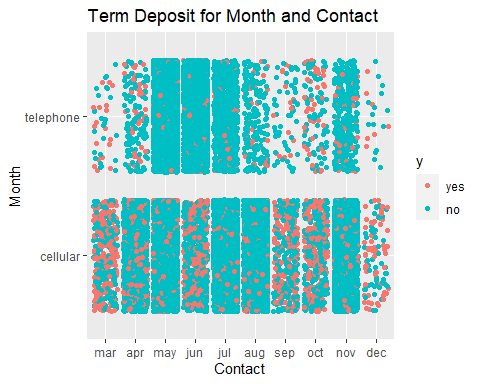
 You can see from the plot that there is a pretty substantial increase in performance after 4 polynomial terms. It levels off after that, but the maximum AUC is technically at n = 9. And even the model for n = 10 shows that many of the higher degree terms are statistically significant, including the 10th term.

We will proceed with trying forward selection using polynomial terms. Using up to the 10th degree term seems like it should be sufficient.

## Variable Interactions

Variable interactions are often present that affect numeric response variables. Usually to show that, you can plot. To see if there are possible interactions, let’s try with a coloring a couple of interactions.

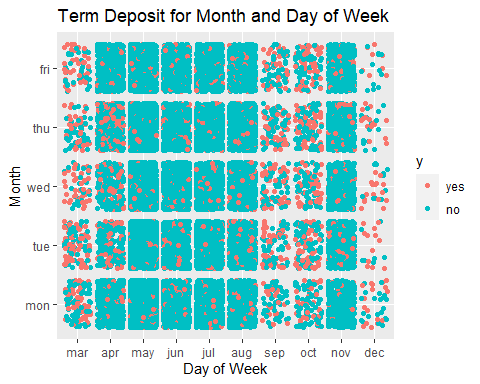
# Try plotting month by contact  
train\_data %>% ggplot(aes(x=month,y=contact,color=y)) + geom\_jitter() +   
 ylab('Month') + xlab('Contact') + ggtitle('Term Deposit for Month and Contact')



# Looking at model  
model <- glm('y ~ month\*contact', data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = "y ~ month\*contact", family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6172 0.2573 0.4524 0.4794 1.2668   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -0.09097 0.10061 -0.904 0.36588   
## monthapr 1.44852 0.11528 12.565 < 2e-16 \*\*\*  
## monthmay 2.19672 0.11155 19.692 < 2e-16 \*\*\*  
## monthjun 0.37338 0.12758 2.927 0.00343 \*\*   
## monthjul 2.31893 0.11155 20.788 < 2e-16 \*\*\*  
## monthaug 2.19520 0.11095 19.786 < 2e-16 \*\*\*  
## monthsep 0.16392 0.14336 1.143 0.25287   
## monthoct 0.31811 0.13822 2.301 0.02137 \*   
## monthnov 2.27145 0.11756 19.322 < 2e-16 \*\*\*  
## monthdec -0.11667 0.21208 -0.550 0.58224   
## contacttelephone 0.36541 0.32055 1.140 0.25431   
## monthapr:contacttelephone -0.57305 0.37493 -1.528 0.12640   
## monthmay:contacttelephone 0.92049 0.33152 2.777 0.00549 \*\*   
## monthjun:contacttelephone 2.34176 0.33919 6.904 5.06e-12 \*\*\*  
## monthjul:contacttelephone 0.39625 0.36151 1.096 0.27305   
## monthaug:contacttelephone -0.59236 0.38226 -1.550 0.12123   
## monthsep:contacttelephone 0.70108 0.44236 1.585 0.11300   
## monthoct:contacttelephone -0.03293 0.38399 -0.086 0.93166   
## monthnov:contacttelephone -0.50233 0.36799 -1.365 0.17223   
## monthdec:contacttelephone 0.60437 0.58918 1.026 0.30499   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 20600 on 32930 degrees of freedom  
## AIC: 20640  
##   
## Number of Fisher Scoring iterations: 6

# Try plotting month by day of week  
train\_data %>% ggplot(aes(x=month,y=day\_of\_week,color=y)) + geom\_jitter() +   
 ylab('Month') + xlab('Day of Week') + ggtitle('Term Deposit for Month and Day of Week')



# Looking at model  
model <- glm('y ~ month\*day\_of\_week', data = train\_data, family = "binomial")  
summary(model)

##   
## Call:  
## glm(formula = "y ~ month\*day\_of\_week", family = "binomial", data = train\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.4220 0.3623 0.4223 0.4727 1.4132   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.312872 0.187972 1.664 0.096020 .   
## monthapr 1.827194 0.233674 7.819 5.31e-15 \*\*\*  
## monthmay 2.278239 0.206389 11.039 < 2e-16 \*\*\*  
## monthjun 1.730202 0.213119 8.118 4.72e-16 \*\*\*  
## monthjul 2.162129 0.216775 9.974 < 2e-16 \*\*\*  
## monthaug 1.979662 0.218099 9.077 < 2e-16 \*\*\*  
## monthsep 0.337715 0.314228 1.075 0.282489   
## monthoct 0.435845 0.278242 1.566 0.117249   
## monthnov 1.977134 0.234519 8.431 < 2e-16 \*\*\*  
## monthdec -0.158722 0.372167 -0.426 0.669758   
## day\_of\_weektue -0.696831 0.266651 -2.613 0.008968 \*\*   
## day\_of\_weekwed -0.772405 0.321428 -2.403 0.016259 \*   
## day\_of\_weekthu -0.055043 0.295758 -0.186 0.852359   
## day\_of\_weekfri -0.430655 0.307183 -1.402 0.160930   
## monthapr:day\_of\_weektue -0.659116 0.336886 -1.956 0.050407 .   
## monthmay:day\_of\_weektue 0.983982 0.295241 3.333 0.000860 \*\*\*  
## monthjun:day\_of\_weektue 0.506141 0.303830 1.666 0.095739 .   
## monthjul:day\_of\_weektue 0.521733 0.304277 1.715 0.086407 .   
## monthaug:day\_of\_weektue 0.427297 0.304207 1.405 0.160132   
## monthsep:day\_of\_weektue 0.112202 0.422507 0.266 0.790577   
## monthoct:day\_of\_weektue 0.148785 0.383220 0.388 0.697832   
## monthnov:day\_of\_weektue 0.496488 0.325825 1.524 0.127562   
## monthdec:day\_of\_weektue 0.003684 0.632825 0.006 0.995355   
## monthapr:day\_of\_weekwed -0.514429 0.377348 -1.363 0.172796   
## monthmay:day\_of\_weekwed 0.872195 0.343141 2.542 0.011028 \*   
## monthjun:day\_of\_weekwed 0.797344 0.355250 2.244 0.024803 \*   
## monthjul:day\_of\_weekwed 0.608365 0.353901 1.719 0.085610 .   
## monthaug:day\_of\_weekwed 0.493582 0.354500 1.392 0.163821   
## monthsep:day\_of\_weekwed 0.041774 0.454737 0.092 0.926805   
## monthoct:day\_of\_weekwed 0.233983 0.428858 0.546 0.585344   
## monthnov:day\_of\_weekwed 0.670189 0.372352 1.800 0.071879 .   
## monthdec:day\_of\_weekwed 0.243561 0.599916 0.406 0.684749   
## monthapr:day\_of\_weekthu -1.284675 0.338186 -3.799 0.000145 \*\*\*  
## monthmay:day\_of\_weekthu 0.193431 0.321553 0.602 0.547473   
## monthjun:day\_of\_weekthu 0.282541 0.335666 0.842 0.399938   
## monthjul:day\_of\_weekthu -0.028636 0.330096 -0.087 0.930869   
## monthaug:day\_of\_weekthu -0.018694 0.331765 -0.056 0.955064   
## monthsep:day\_of\_weekthu -0.470381 0.438986 -1.072 0.283936   
## monthoct:day\_of\_weekthu -0.442359 0.401626 -1.101 0.270714   
## monthnov:day\_of\_weekthu -0.033120 0.350090 -0.095 0.924630   
## monthdec:day\_of\_weekthu -0.159732 0.558543 -0.286 0.774894   
## monthapr:day\_of\_weekfri 0.426120 0.367607 1.159 0.246385   
## monthmay:day\_of\_weekfri 0.449520 0.329336 1.365 0.172275   
## monthjun:day\_of\_weekfri 0.759834 0.343928 2.209 0.027155 \*   
## monthjul:day\_of\_weekfri -0.007922 0.343393 -0.023 0.981596   
## monthaug:day\_of\_weekfri 0.061498 0.342213 0.180 0.857382   
## monthsep:day\_of\_weekfri 0.239600 0.450654 0.532 0.594953   
## monthoct:day\_of\_weekfri -0.208862 0.415835 -0.502 0.615476   
## monthnov:day\_of\_weekfri 0.209000 0.361374 0.578 0.563030   
## monthdec:day\_of\_weekfri 0.681970 0.637079 1.070 0.284411   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 23269 on 32949 degrees of freedom  
## Residual deviance: 21314 on 32900 degrees of freedom  
## AIC: 21414  
##   
## Number of Fisher Scoring iterations: 5

It looks like there is some promise for variable interactions. When looking at Month vs Day of Week, certain months have days that have a different distribution of yes and no. And certain days have months that have a different distribution. Some of the interaction terms also have significant p values. We will also include interaction terms in the Forward Selection, along with the polynomial terms. Below is some example code (not being evaluate, because it takes over an hour to finish) of adding all the single variables, polynomial variables, and interaction terms to the model.

set.seed(70)  
vars <- colnames(train\_data)  
vars <- vars[vars!="y"]  
allVars <- vars  
num\_poly <- 10  
for (i in 1:length(vars)){  
 for (j in 2:num\_poly){  
 if (class(train\_data[,vars[i]]) != "factor") {  
 allVars <- c(allVars,paste('poly(',vars[i],',',j,')',sep=""))  
 }  
 }  
}  
for (i in 1:length(vars)){  
 for (j in 1:i){  
 if(vars[i]!=vars[j]) {  
 allVars <- c(allVars,paste(vars[i],'\*',vars[j],sep=''))  
 }  
 }  
}  
# These variables need to be removed so the code doesn't error out  
allVars <- allVars[allVars!="poly(pdays,6)"]   
allVars <- allVars[allVars!="poly(pdays,7)"]  
allVars <- allVars[allVars!="poly(pdays,8)"]  
allVars <- allVars[allVars!="poly(pdays,9)"]  
allVars <- allVars[allVars!="poly(pdays,10)"]  
allVars <- allVars[allVars!="poly(previous,7)"]  
allVars <- allVars[allVars!="poly(previous,8)"]  
allVars <- allVars[allVars!="poly(previous,9)"]  
allVars <- allVars[allVars!="poly(previous,10)"]  
allVars <- allVars[allVars!="poly(emp.var.rate,10)"]  
var\_aucs <- data.frame("vars" = allVars)  
num\_vars <- length(allVars)  
num\_folds <- 10  
start\_num <- 124  
for (j in start\_num:num\_vars) {  
 var <- allVars[j]  
 print(paste(j,'/',num\_vars,': ',var,sep=''))  
 folds <- createFolds(train\_data$y, k = num\_folds)  
 auc\_scores <- numeric(num\_folds)  
 for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 form <- as.formula(paste("y ~ ",var,sep=""))  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
 }  
 var\_aucs$auc[var\_aucs$var == var] <- mean(auc\_scores)  
}

After several iterations of this (plus Backwards Selection), we arrived at the final model: y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week \* month + month \* contact + cons.conf.idx \* housing + poutcome \* previous + poly(campaign,5) + poly(euribor3m,8) + campaign \* month + cons.conf.idx \* age + poly(previous,6) + campaign \* contact + poly(age,3).

## Adding PCA

Since our earlier PCA analysis looked promising, I tried adding PC1 to the model.

# PCA  
df.numeric <- train\_data[ , sapply(train\_data, is.numeric)]  
pc.result<-prcomp(df.numeric,scale.=TRUE)  
pc.scores<-pc.result$x  
pc.scores<-data.frame(pc.scores)  
  
# Trying out adding PC1  
train\_data$PC1 <- pc.scores$PC1  
set.seed(134)  
form <- as.formula('y ~ PC1 + poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3)')  
num\_folds <- 10  
folds <- createFolds(train\_data$y, k = num\_folds)  
accuracy\_scores <- numeric(num\_folds)  
auc\_scores <- numeric(num\_folds)  
for (i in 1:num\_folds) {  
 train\_indices <- unlist(folds[-i])  
 test\_indices <- unlist(folds[i])  
 train <- train\_data[train\_indices, ]  
 test <- train\_data[test\_indices, ]  
 model <- glm(form, data = train, family = "binomial")  
 predictions <- predict(model, newdata = test, type = "response")  
 roc <- roc(response=test$y,predictor=predictions,levels=c("no", "yes"),direction = ">")  
 auc\_scores[i] <- auc(roc)  
}  
mean(auc\_scores)

## [1] 0.8026035

The AUC value was slightly worse than it was when PC1 wasn’t there. So we’ll just use the formula derived above.

## Support Vector Machine (SVM)

For a non-parametric model, we tried Support Vector Machines (SVM). These are a type of non-parametric model that try to form a line, plane, hyper-plan between datapoints.

They had three important hyper parameters: the kernal, gamma, and the cost. The kernal would be the type of fit. Possible values are linear, polynomial, radial, etc. Gamma sets the curvature of the separating hyper-plane. A higher Gamma values means more curvature. The Cost parameter sets how much error points it wants to classify on. The higher the Cost, the worse it predicts on the training set errors (and hope isn’t overfitting).

Here is some example code for training the SVM. It takes over half an hour to train even fairly simple models, so it isn’t set to execute.

form <- as.formula("y ~ euribor3m + month + poutcome + contact + cons.conf.idx + campaign + previous + age + housing + day\_of\_week")  
svm\_model <- svm(form, data = train\_data, kernel = "radial", gamma = 1, cost = 1, probability = TRUE, decision.values = TRUE)  
predictions <- predict(svm\_model, newdata = train\_data, probability = TRUE, decision.values = TRUE)  
probs <- attr(predictions,"probabilities")[,'yes']  
predicted\_classes <- ifelse(probs > .083, 'yes','no')   
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))  
confusion\_matrix # PPV = 0.69, Sens = 0.62  
confusion\_matrix$byClass['F1'] # 0.65  
roc <- roc(response=train\_data$y,predictor=probs,levels=c("yes", "no"),direction = ">")  
auc(roc) # 0.8513  
plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for SVM', line = 3)

And here is the sample code for testing.

predictions <- predict(svm\_model, newdata = test\_data, probability = TRUE)  
probs <- attr(predictions,"probabilities")[,'yes']  
predicted\_classes <- ifelse(probs > .083, 'yes','no')   
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))  
confusion\_matrix # Sensitivity = 0.5236, Specificity = 0.8705, PPV = 0.3345, NPV = 0.9363  
confusion\_matrix$byClass['F1'] # 0.4082157  
roc <- roc(response=test\_data$y,predictor=probs,levels=c("yes", "no"),direction = ">")  
auc(roc) # 0.6979

# Calculating Metrics

We are calculating Sensitivity (Correctly Predicted Positive / All Positive), Specificity (Correctly Predicted Negative / All Negative), Prevalence (All Positive / All Observations), PPV (Correctly Predicted Positive / Predicted Positive), NPV (Correctly Predicted Negative / Predicted Negative), and AUROC (Area under the ROC Curve).

In addition, we are also calculating the F1 score. This is equal to 2 \* Sensitivity \* PPV / (Sensitivity + PPV). Since it is more important that we do a good job of predicting whether people will get a term deposit, and the F1 score is a nice metric to make sure that there is a good balance between Sensitivity and PPV, we decided to track this metric as well.

## Simple Logisitc Regression Model

First we determined the threshold to use for the Simple Logistic Regression Model in order to maximize the F1 metric.

# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 0  
metrics$specificity <- 0  
metrics$ppv <- 0  
metrics$npv <- 0  
metrics$accuracy <- 0  
metrics$f1 <- 0  
form <- as.formula(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx)  
model <- glm(form, data = train\_data, family = "binomial")  
predicted <- predict(model, newdata = train\_data, type = "response")

for (i in 1:num\_thresh){  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted > metrics$thresh[i], 'no','yes')  
 confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))  
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

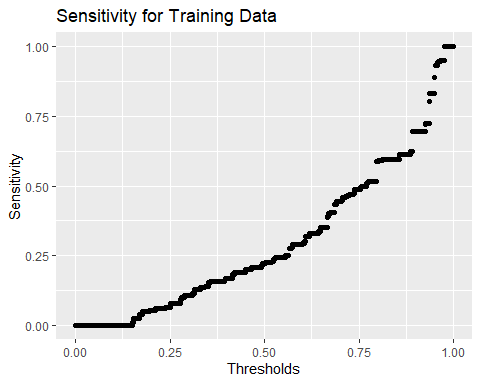
# Get threshold value that maximizes F1  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_simple.csv')  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1

## [1] 0.4798184

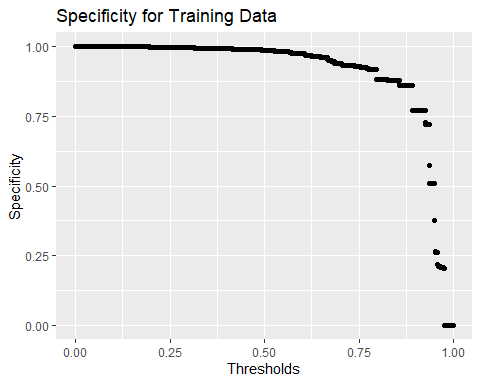
theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1

## [1] 0.7703

# Plots  
metrics %>% ggplot(aes(x = thresh, y = sensitivity)) + geom\_point() +   
 ylab('Sensitivity') + xlab('Thresholds') + ggtitle('Sensitivity for Training Data')

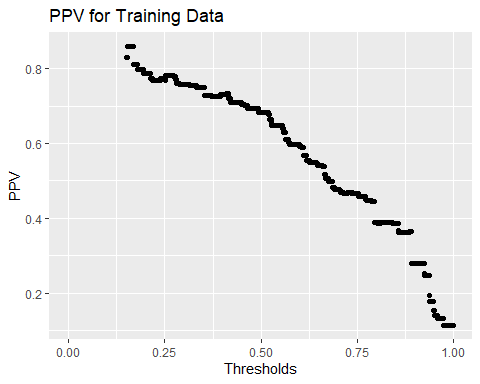


metrics %>% ggplot(aes(x = thresh, y = specificity)) + geom\_point() +   
 ylab('Specificity') + xlab('Thresholds') + ggtitle('Specificity for Training Data')



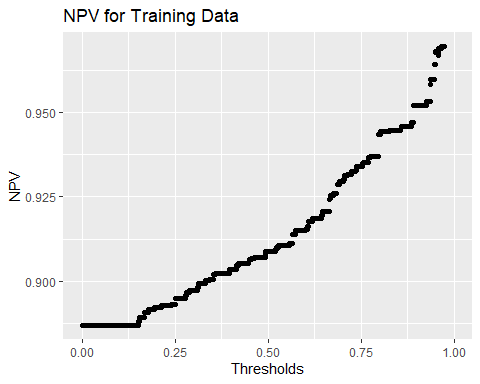
metrics %>% ggplot(aes(x = thresh, y = ppv)) + geom\_point() +   
 ylab('PPV') + xlab('Thresholds') + ggtitle('PPV for Training Data')

## Warning: Removed 1506 rows containing missing values (`geom\_point()`).



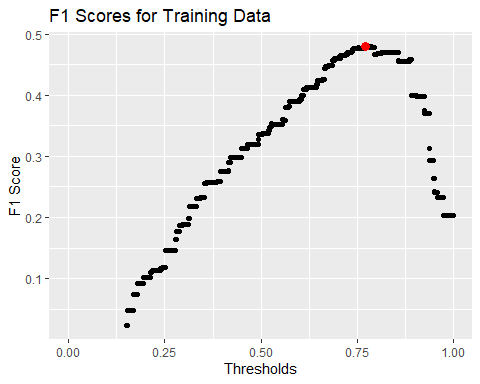
metrics %>% ggplot(aes(x = thresh, y = npv)) + geom\_point() +   
 ylab('NPV') + xlab('Thresholds') + ggtitle('NPV for Training Data')

## Warning: Removed 252 rows containing missing values (`geom\_point()`).



metrics %>% ggplot(aes(x = thresh, y = f1)) + geom\_point() +   
 ylab('F1 Score') + xlab('Thresholds') + ggtitle('F1 Scores for Training Data') +   
 geom\_point(data = data.frame(x = theshF1, y = maxF1), aes(x = x, y = y), size = 3, color = "red", fill = "red", shape = 21)

## Warning: Removed 1506 rows containing missing values (`geom\_point()`).



# Get Confusion Matrix for threshold value  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1902 2297  
## no 1827 26924  
##   
## Accuracy : 0.8748   
## 95% CI : (0.8712, 0.8784)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.409   
##   
## Mcnemar's Test P-Value : 2.81e-13   
##   
## Sensitivity : 0.51006   
## Specificity : 0.92139   
## Pos Pred Value : 0.45296   
## Neg Pred Value : 0.93645   
## Prevalence : 0.11317   
## Detection Rate : 0.05772   
## Detection Prevalence : 0.12744   
## Balanced Accuracy : 0.71572   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.4798184

After we got the thresholds to use, then we calculated the metrics on the test dataset.

# Test data  
predicted <- predict(model, newdata = test\_data, type = "response")  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 449 571  
## no 462 6756  
##   
## Accuracy : 0.8746   
## 95% CI : (0.8673, 0.8817)  
## No Information Rate : 0.8894   
## P-Value [Acc > NIR] : 0.9999882   
##   
## Kappa : 0.3943   
##   
## Mcnemar's Test P-Value : 0.0007787   
##   
## Sensitivity : 0.4929   
## Specificity : 0.9221   
## Pos Pred Value : 0.4402   
## Neg Pred Value : 0.9360   
## Prevalence : 0.1106   
## Detection Rate : 0.0545   
## Detection Prevalence : 0.1238   
## Balanced Accuracy : 0.7075   
##   
## 'Positive' Class : yes   
##

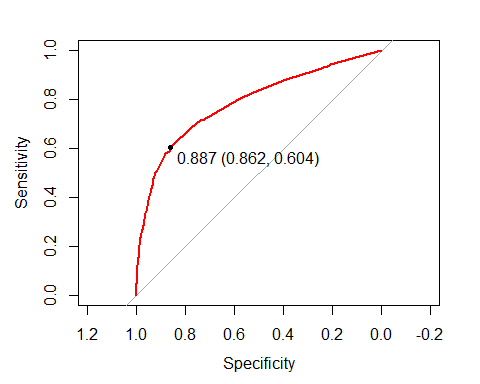
confusion\_matrix$byClass['F1']

## F1   
## 0.465044

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc)

## Area under the curve: 0.7868

plot(roc,print.thres="best",col="red")

 This code takes a while to run, so we won’t include the code to maximize the threshold value going forward. Given the thresholds though, here is the code to get the metrics for the complicated logistic regression model.

## Complex Logistic Regression

# Get threshold value that maximizes F1  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_complex\_logistic.csv')  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1

## [1] 0.5073269

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1

## [1] 0.7354

# Get Confusion Matrix for threshold value  
form <- as.formula(y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
predicted <- predict(model, newdata = train\_data, type = "response")

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

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 2008 2179  
## no 1721 27042  
##   
## Accuracy : 0.8816   
## 95% CI : (0.8781, 0.8851)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 0.9985   
##   
## Kappa : 0.4403   
##   
## Mcnemar's Test P-Value : 2.52e-13   
##   
## Sensitivity : 0.53848   
## Specificity : 0.92543   
## Pos Pred Value : 0.47958   
## Neg Pred Value : 0.94017   
## Prevalence : 0.11317   
## Detection Rate : 0.06094   
## Detection Prevalence : 0.12707   
## Balanced Accuracy : 0.73196   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1']

## F1   
## 0.5073269

# Test data  
predicted <- predict(model, newdata = test\_data, type = "response")

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

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 481 550  
## no 430 6777  
##   
## Accuracy : 0.881   
## 95% CI : (0.8739, 0.888)  
## No Information Rate : 0.8894   
## P-Value [Acc > NIR] : 0.9922347   
##   
## Kappa : 0.4282   
##   
## Mcnemar's Test P-Value : 0.0001439   
##   
## Sensitivity : 0.52799   
## Specificity : 0.92494   
## Pos Pred Value : 0.46654   
## Neg Pred Value : 0.94034   
## Prevalence : 0.11059   
## Detection Rate : 0.05839   
## Detection Prevalence : 0.12515   
## Balanced Accuracy : 0.72646   
##   
## 'Positive' Class : yes   
##

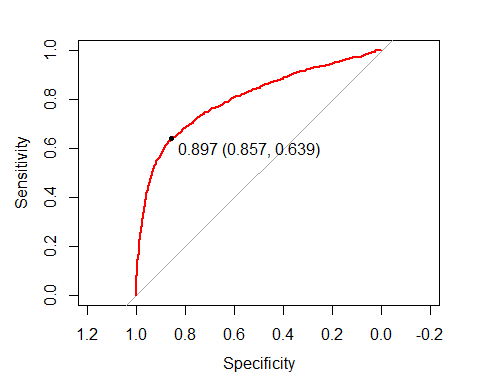
confusion\_matrix$byClass['F1']

## F1   
## 0.4953656

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc)

## Area under the curve: 0.8013

plot(roc,print.thres="best",col="red")



## Comparing old vs new data

We noticed as part of EDA that the newer data (data was in order of when it was received) had a different Yes/No distribution than older data. We thought it would be interesting to see how training on old data and testing on newer data would perform.

# Simple logistic model  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_simple\_date.csv')  
  
# Train simple model  
form <- as.formula(y ~ month + poutcome + emp.var.rate + contact + cons.price.idx)  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Get threshold value that maximizes F1  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1 # 0.2623695

## [1] 0.2623695

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.8486

## [1] 0.8486

# Try filtering data to get it to work  
test\_data <- test\_data[test\_data$month != "sep",]  
  
# Get the confusion matrix  
predicted <- predict(model, newdata = test\_data, type = "response")  
predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5661, Specificity = 0.7740, PPV = 0.5151, NPV = 0.8079, Prevalence = 0.2979

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 474 819  
## no 382 6443  
##   
## Accuracy : 0.8521   
## 95% CI : (0.8441, 0.8597)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3599   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.55374   
## Specificity : 0.88722   
## Pos Pred Value : 0.36659   
## Neg Pred Value : 0.94403   
## Prevalence : 0.10544   
## Detection Rate : 0.05839   
## Detection Prevalence : 0.15928   
## Balanced Accuracy : 0.72048   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.5394243

## F1   
## 0.4411354

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc) # 0.7095

## Area under the curve: 0.7788

# Complex model  
metrics <- read.csv('https://raw.githubusercontent.com/stedua22/6372-Project-2/main/metrics\_complex\_logistic\_date.csv')  
  
# Train simple model  
form <- as.formula(y ~ poly(cons.conf.idx,10) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,6) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Error about poly(cons.conf.idx,10) having too high of a degree  
form <- as.formula(y ~ poly(cons.conf.idx,9) + pdays + day\_of\_week\*month + month\*contact + cons.conf.idx\*housing + poutcome\*previous + poly(campaign,5) + poly(euribor3m,8) + campaign\*month + cons.conf.idx\*age + poly(previous,3) + campaign\*contact + poly(age,3))  
model <- glm(form, data = train\_data, family = "binomial")  
  
# Get threshold value that maximizes F1  
maxF1 <- max(metrics$f1, na.rm = TRUE)  
maxF1 # 0.2431846

## [1] 0.2431846

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.7308

## [1] 0.7308

# Try filtering data to get it to work  
test\_data <- test\_data[test\_data$month != "sep",]  
  
# Get the confusion matrix  
predicted <- predict(model, newdata = test\_data, type = "response")

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

predicted\_classes <- ifelse(predicted > theshF1, 'no','yes')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.6287, Specificity = 0.6716, PPV = 0.4482, NPV = 0.8100, Prevalence = 0.2979

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 430 513  
## no 426 6749  
##   
## Accuracy : 0.8843   
## 95% CI : (0.8772, 0.8912)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.998562   
##   
## Kappa : 0.4132   
##   
## Mcnemar's Test P-Value : 0.005008   
##   
## Sensitivity : 0.50234   
## Specificity : 0.92936   
## Pos Pred Value : 0.45599   
## Neg Pred Value : 0.94063   
## Prevalence : 0.10544   
## Detection Rate : 0.05297   
## Detection Prevalence : 0.11616   
## Balanced Accuracy : 0.71585   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.5233236

## F1   
## 0.4780434

# AUC  
roc <- roc(response=test\_data$y,predictor=predicted,levels=c("no", "yes"),direction = ">")  
auc(roc) # 0.6502

## Area under the curve: 0.7936

We compared the simple model and complex model to how they did previously. To even be able to test against the newer data, we first had to filter out month = Sep, since that didn’t exist in the training data. Also, we had to lower the order of some of the polynomials for the complex model.

The training metrics were poor, with an F1 score of around .25 for both models. However, the F1 scores for the test data were both above 0.5. The extra Yes results in the data seemed to help out. However, the AUC scores were worse, as well as the Specificity and NPV. This makes some sense, since there were less No results.

#QDA/LDA Model

library(caret) # CreateFolds  
library(pROC)  
library(car) # VIF  
library(tidyverse)  
  
  
#LDA Model--- simple model month + poutcome + emp.var.rate + contact + cons.price.idx  
  
# Convert the binary outcome to a factor  
train\_data$y <- as.factor(train\_data$y)  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ month + poutcome + emp.var.rate + contact + cons.price.idx,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4847

## [1] 0.4847892

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1313

## [1] 0.1313

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5159, Specificity = 0.9179, PPV = 0.4388, NPV = 0.9385, Prevalence = 0.11059

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 415 536  
## no 441 6726  
##   
## Accuracy : 0.8797   
## 95% CI : (0.8724, 0.8867)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.999992   
##   
## Kappa : 0.3918   
##   
## Mcnemar's Test P-Value : 0.002636   
##   
## Sensitivity : 0.48481   
## Specificity : 0.92619   
## Pos Pred Value : 0.43638   
## Neg Pred Value : 0.93847   
## Prevalence : 0.10544   
## Detection Rate : 0.05112   
## Detection Prevalence : 0.11715   
## Balanced Accuracy : 0.70550   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4743

## F1   
## 0.4593248

#AUC  
  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7846

## Area under the curve: 0.7768

#LDA Model--- using numeric variables only campaign + pdays + previous + emp.var.rate + cons.price.idx + euribor3m + nr.employed  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ campaign + pdays + previous + emp.var.rate + cons.price.idx + euribor3m + nr.employed,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE)   
maxF1 #

## [1] 0.4744277

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 #

## [1] 0.1082

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 410 563  
## no 446 6699  
##   
## Accuracy : 0.8757   
## 95% CI : (0.8683, 0.8828)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1.0000000   
##   
## Kappa : 0.3786   
##   
## Mcnemar's Test P-Value : 0.0002604   
##   
## Sensitivity : 0.47897   
## Specificity : 0.92247   
## Pos Pred Value : 0.42138   
## Neg Pred Value : 0.93758   
## Prevalence : 0.10544   
## Detection Rate : 0.05051   
## Detection Prevalence : 0.11986   
## Balanced Accuracy : 0.70072   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4640

## F1   
## 0.4483324

#AUC  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc)

## Area under the curve: 0.7502

#LDA Model--- using numeric variables pdays + previous + emp.var.rate + cons.price.idx + nr.employed  
  
# Convert the binary outcome to a factor  
train\_data$y <- as.factor(train\_data$y)  
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ pdays + previous + emp.var.rate + cons.price.idx + nr.employed,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 #0.4744

## [1] 0.4647335

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1082

## [1] 0.1254

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.51043, Specificity = 0.9143, PPV = 0.4254, NPV = 0.9376, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 376 470  
## no 480 6792  
##   
## Accuracy : 0.883   
## 95% CI : (0.8758, 0.8899)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.9996   
##   
## Kappa : 0.3765   
##   
## Mcnemar's Test P-Value : 0.7703   
##   
## Sensitivity : 0.43925   
## Specificity : 0.93528   
## Pos Pred Value : 0.44444   
## Neg Pred Value : 0.93399   
## Prevalence : 0.10544   
## Detection Rate : 0.04632   
## Detection Prevalence : 0.10421   
## Balanced Accuracy : 0.68727   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4641

## F1   
## 0.4418331

#AUC  
  
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7599

## Area under the curve: 0.7439

#LDA Model--- using numeric variables emp.var.rate + cons.price.idx + euribor3m   
  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
  
lda.fit<-train(y~ emp.var.rate + cons.price.idx + euribor3m ,  
 data=train\_data,  
 method="lda",  
 trControl=fitControl,  
 metric="logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4561

## [1] 0.4560976

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.2306

## [1] 0.2306

# Test data  
predicted <- predict(lda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.4522, Specificity = 0.9335, PPV = 0.4583, NPV = 0.9320, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 357 422  
## no 499 6840  
##   
## Accuracy : 0.8865   
## 95% CI : (0.8794, 0.8934)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.99051   
##   
## Kappa : 0.3738   
##   
## Mcnemar's Test P-Value : 0.01227   
##   
## Sensitivity : 0.41706   
## Specificity : 0.94189   
## Pos Pred Value : 0.45828   
## Neg Pred Value : 0.93201   
## Prevalence : 0.10544   
## Detection Rate : 0.04398   
## Detection Prevalence : 0.09596   
## Balanced Accuracy : 0.67947   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4552

## F1   
## 0.4366972

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7506

## Area under the curve: 0.7414

# Training data  
predicted <- predict(lda.fit, newdata = train\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1683 1968  
## no 2046 27253  
##   
## Accuracy : 0.8782   
## 95% CI : (0.8746, 0.8817)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1.0000   
##   
## Kappa : 0.3875   
##   
## Mcnemar's Test P-Value : 0.2242   
##   
## Sensitivity : 0.45133   
## Specificity : 0.93265   
## Pos Pred Value : 0.46097   
## Neg Pred Value : 0.93017   
## Prevalence : 0.11317   
## Detection Rate : 0.05108   
## Detection Prevalence : 0.11080   
## Balanced Accuracy : 0.69199   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] #0.4561

## F1   
## 0.4560976

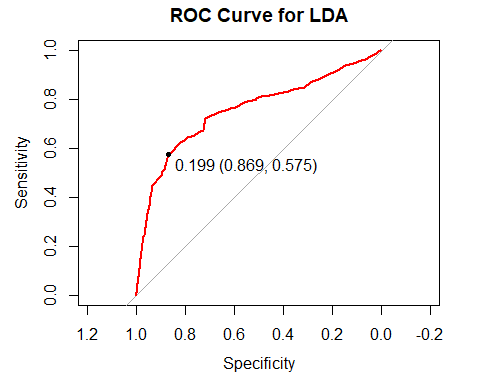
# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = train\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(train\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7555

## Area under the curve: 0.7555

plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for LDA', line = 3)



# QDA Model--simple model  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ month + poutcome + emp.var.rate + contact + cons.price.idx,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4692

## [1] 0.4692364

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.0227

## [1] 0.0227

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # Sensitivity = 0.5917, Specificity = 0.8781, PPV = 0.3764, NPV = 0.9453, Prevalence =0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 484 828  
## no 372 6434  
##   
## Accuracy : 0.8522   
## 95% CI : (0.8443, 0.8598)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3655   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.56542   
## Specificity : 0.88598   
## Pos Pred Value : 0.36890   
## Neg Pred Value : 0.94534   
## Prevalence : 0.10544   
## Detection Rate : 0.05962   
## Detection Prevalence : 0.16162   
## Balanced Accuracy : 0.72570   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4600

## F1   
## 0.4464945

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7811

## Area under the curve: 0.7741

# QDA --- using numeric variables pdays + previous + emp.var.rate + cons.price.idx + nr.employed  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ pdays + previous + emp.var.rate + cons.price.idx + nr.employed,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4382

## [1] 0.4381683

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.0635

## [1] 0.0635

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # accuracy = 0.8546, Sensitivity = 0.4951, Specificity = 0.8993, PPV = 0.3793, NPV = 0.9347, Prevalence =0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 396 673  
## no 460 6589  
##   
## Accuracy : 0.8604   
## 95% CI : (0.8527, 0.8679)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3334   
##   
## Mcnemar's Test P-Value : 3.01e-10   
##   
## Sensitivity : 0.46262   
## Specificity : 0.90733   
## Pos Pred Value : 0.37044   
## Neg Pred Value : 0.93474   
## Prevalence : 0.10544   
## Detection Rate : 0.04878   
## Detection Prevalence : 0.13168   
## Balanced Accuracy : 0.68497   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4295

## F1   
## 0.4114286

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7527

## Area under the curve: 0.7428

# QDA --- using numeric variables emp.var.rate + cons.price.idx + euribor3m  
  
fitControl<-trainControl(method="repeatedcv",number=5,repeats=1,classProbs=TRUE, summaryFunction=mnLogLoss)  
set.seed(1234)  
  
qda.fit <- train(y~ emp.var.rate + cons.price.idx + euribor3m,  
 data = train\_data,  
 method = "qda",  
 trControl = fitControl,  
 metric = "logLoss")  
  
# Get threshold  
metrics = data.frame(thresh=seq(0, 1, by = 0.0001))  
num\_thresh <- nrow(metrics)  
metrics$sensitivity <- 1  
metrics$specificity <- 1  
metrics$ppv <- 1  
metrics$npv <- 1  
metrics$accuracy <- 1  
metrics$f1 <- 1  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
for (i in 1:num\_thresh){  
 if(i %% 100 == 0) {  
 print(paste(i,'/',num\_thresh,sep=''))  
 }  
   
 # Confusion Matrix  
 predicted\_classes <- ifelse(predicted[, "yes"] > metrics$thresh[i], 'yes', 'no')  
 predicted\_classes\_factor <- factor(predicted\_classes, levels = levels(train\_data$y))  
 confusion\_matrix <- confusionMatrix(predicted\_classes\_factor, train\_data$y)  
   
   
 # Metrics  
 metrics$sensitivity[i] <- as.numeric(confusion\_matrix$byClass['Sensitivity'])  
 metrics$specificity[i] <- as.numeric(confusion\_matrix$byClass['Specificity'])  
 metrics$ppv[i] <- as.numeric(confusion\_matrix$byClass['Pos Pred Value'])  
 metrics$npv[i] <- as.numeric(confusion\_matrix$byClass['Neg Pred Value'])  
 metrics$accuracy[i] <- as.numeric(confusion\_matrix$overall['Accuracy'])  
 metrics$f1[i] <- as.numeric(confusion\_matrix$byClass['F1'])  
}

## [1] "100/10001"  
## [1] "200/10001"  
## [1] "300/10001"  
## [1] "400/10001"  
## [1] "500/10001"  
## [1] "600/10001"  
## [1] "700/10001"  
## [1] "800/10001"  
## [1] "900/10001"  
## [1] "1000/10001"  
## [1] "1100/10001"  
## [1] "1200/10001"  
## [1] "1300/10001"  
## [1] "1400/10001"  
## [1] "1500/10001"  
## [1] "1600/10001"  
## [1] "1700/10001"  
## [1] "1800/10001"  
## [1] "1900/10001"  
## [1] "2000/10001"  
## [1] "2100/10001"  
## [1] "2200/10001"  
## [1] "2300/10001"  
## [1] "2400/10001"  
## [1] "2500/10001"  
## [1] "2600/10001"  
## [1] "2700/10001"  
## [1] "2800/10001"  
## [1] "2900/10001"  
## [1] "3000/10001"  
## [1] "3100/10001"  
## [1] "3200/10001"  
## [1] "3300/10001"  
## [1] "3400/10001"  
## [1] "3500/10001"  
## [1] "3600/10001"  
## [1] "3700/10001"  
## [1] "3800/10001"  
## [1] "3900/10001"  
## [1] "4000/10001"  
## [1] "4100/10001"  
## [1] "4200/10001"  
## [1] "4300/10001"  
## [1] "4400/10001"  
## [1] "4500/10001"  
## [1] "4600/10001"  
## [1] "4700/10001"  
## [1] "4800/10001"  
## [1] "4900/10001"  
## [1] "5000/10001"  
## [1] "5100/10001"  
## [1] "5200/10001"  
## [1] "5300/10001"  
## [1] "5400/10001"  
## [1] "5500/10001"  
## [1] "5600/10001"  
## [1] "5700/10001"  
## [1] "5800/10001"  
## [1] "5900/10001"  
## [1] "6000/10001"  
## [1] "6100/10001"  
## [1] "6200/10001"  
## [1] "6300/10001"  
## [1] "6400/10001"  
## [1] "6500/10001"  
## [1] "6600/10001"  
## [1] "6700/10001"  
## [1] "6800/10001"  
## [1] "6900/10001"  
## [1] "7000/10001"  
## [1] "7100/10001"  
## [1] "7200/10001"  
## [1] "7300/10001"  
## [1] "7400/10001"  
## [1] "7500/10001"  
## [1] "7600/10001"  
## [1] "7700/10001"  
## [1] "7800/10001"  
## [1] "7900/10001"  
## [1] "8000/10001"  
## [1] "8100/10001"  
## [1] "8200/10001"  
## [1] "8300/10001"  
## [1] "8400/10001"  
## [1] "8500/10001"  
## [1] "8600/10001"  
## [1] "8700/10001"  
## [1] "8800/10001"  
## [1] "8900/10001"  
## [1] "9000/10001"  
## [1] "9100/10001"  
## [1] "9200/10001"  
## [1] "9300/10001"  
## [1] "9400/10001"  
## [1] "9500/10001"  
## [1] "9600/10001"  
## [1] "9700/10001"  
## [1] "9800/10001"  
## [1] "9900/10001"  
## [1] "10000/10001"

# Get threshold value that maximizes F1  
# Get F1 thresholds  
maxF1 <- max(metrics$f1, na.rm = TRUE) #   
maxF1 # 0.4681

## [1] 0.4680797

theshF1 <- metrics$thresh[which.max(metrics$f1)]   
theshF1 # 0.1264

## [1] 0.1264

# Test data  
predicted <- predict(qda.fit, newdata = test\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(test\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(test\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix # accuracy = 0.8765, Sensitivity = 0.4829, Specificity = 0.9255, PPV = 0.4463, NPV = 0.9351, Prevalence = 0.1106

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 385 481  
## no 471 6781  
##   
## Accuracy : 0.8827   
## 95% CI : (0.8755, 0.8897)  
## No Information Rate : 0.8946   
## P-Value [Acc > NIR] : 0.9997   
##   
## Kappa : 0.3816   
##   
## Mcnemar's Test P-Value : 0.7705   
##   
## Sensitivity : 0.44977   
## Specificity : 0.93376   
## Pos Pred Value : 0.44457   
## Neg Pred Value : 0.93505   
## Prevalence : 0.10544   
## Detection Rate : 0.04743   
## Detection Prevalence : 0.10668   
## Balanced Accuracy : 0.69177   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] # 0.4639

## F1   
## 0.4471545

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc.qda <- roc(response = test\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(test\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc.qda) # 0.7623

## Area under the curve: 0.754

# Training data  
predicted <- predict(qda.fit, newdata = train\_data, type = "prob")  
predicted\_classes <- ifelse(predicted[, "yes"] > theshF1, 'yes', 'no')  
confusion\_matrix <- confusionMatrix(as.factor(predicted\_classes), as.factor(train\_data$y))

## Warning in confusionMatrix.default(as.factor(predicted\_classes),  
## as.factor(train\_data$y)): Levels are not in the same order for reference and  
## data. Refactoring data to match.

confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 1833 2270  
## no 1896 26951  
##   
## Accuracy : 0.8736   
## 95% CI : (0.8699, 0.8771)  
## No Information Rate : 0.8868   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3965   
##   
## Mcnemar's Test P-Value : 7.517e-09   
##   
## Sensitivity : 0.49155   
## Specificity : 0.92232   
## Pos Pred Value : 0.44675   
## Neg Pred Value : 0.93427   
## Prevalence : 0.11317   
## Detection Rate : 0.05563   
## Detection Prevalence : 0.12452   
## Balanced Accuracy : 0.70693   
##   
## 'Positive' Class : yes   
##

confusion\_matrix$byClass['F1'] #0.4681

## F1   
## 0.4680797

# Assuming `predicted` contains the predicted probabilities for the 'yes' class  
roc <- roc(response = train\_data$y,   
 predictor = as.numeric(as.character(predicted[, "yes"])),  
 levels = rev(levels(train\_data$y))) # Ensure correct ordering of levels if needed

## Setting direction: controls < cases

# Print the AUROC  
auc(roc) # 0.7694

## Area under the curve: 0.7694

plot(roc,print.thres="best",col="red")  
title(main = 'ROC Curve for QDA', line = 3)

