Homework 4 Solution

As preparation, first run the preliminary code provided in hw4.pdf (omitted).

0.1 Q1.

```
getConfusionMatrix = function(y,phat,th){
  TP <- length(which(phat > th & y=="1"))
  TN <- length(which(phat < th & y=="0"))
  FP <- length(which(phat > th & y=="0"))
  FN <- length(which(phat < th & y=="1"))
  conf.mat <- matrix(c(TP, FN, FP, TN), nrow=2, ncol=2,
                     dimnames=list(c("pos","neg"),c("Y","N")))
  return(conf.mat)
}

(cfm.lgfit <- getConfusionMatrix(df.test$Retained.in.2012., phat.lgfit.selected, 0.6072))
```

```
##       Y   N
## pos 244  56
## neg  60 140
```

```
(cfm.l1 <- getConfusionMatrix(df.test$Retained.in.2012., phat.l1.lgfit, 0.6072))
```

```
##       Y   N
## pos 252  56
## neg  52 140
```

Both models are generally able to correctly classify the samples, with a misclassification of around 22%. The Lasso has less false negatives than logistic regression, and a lower misspecification rate.

0.2 Q2.

```
library(ROCR)

getROC <- function(y, phat, title="", add=F, col=4){
  pred <- prediction(phat, y)
  # ROC curve
  perf <- performance(pred, measure = "tpr", x.measure = "fpr")
  plot(perf, add=add, col=col, main=paste("ROC", title), lwd=2)
  abline(0,1,lty=2)
  # AUC
```

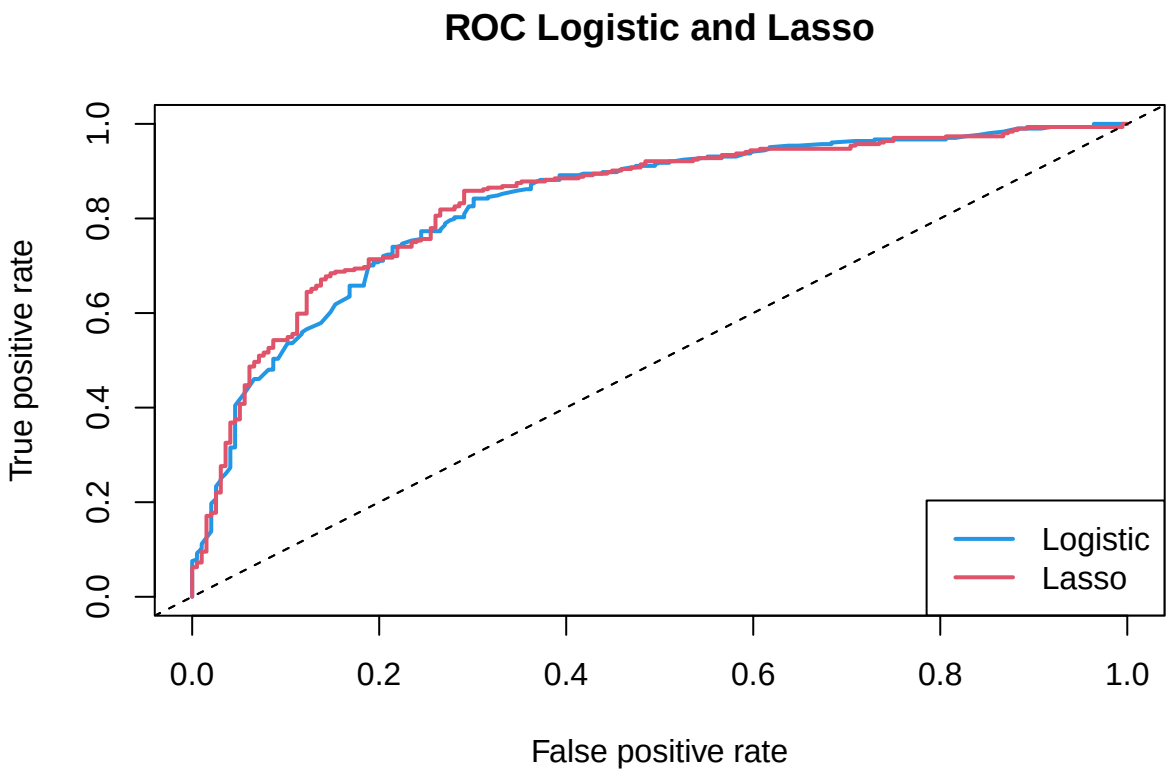
```
  perf.auc <- performance(pred, measure = "auc")
  auc <- perf.auc@y.values[[1]]

  return(auc)
}


# ROC for logistic regression
getROC(df.test$Retained.in.2012., phat.lgfit.selected, "Logistic and Lasso", col=4)
```

```
## [1] 0.8306089
```

```
# ROC for lasso
getROC(df.test$Retained.in.2012., phat.l1.lgfit, add=T, col=2)
```

```
## [1] 0.8374564
```

```
legend("bottomright", legend=c("Logistic","Lasso"), col=c(4,2), lwd=c(2,2))
```

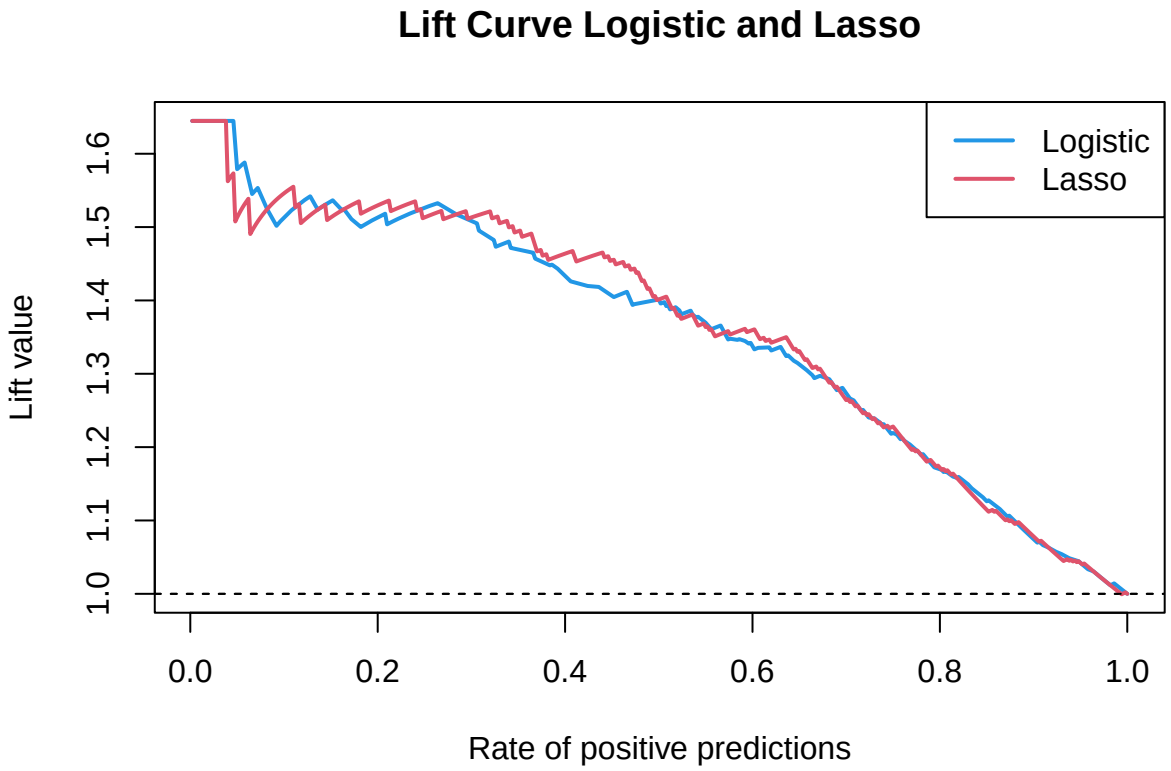


Both ROC curves are "very good" in terms of AUC, while lasso performs better between the two models.

0.3 Q3.

```
getLift <- function(y, phat, title="", add=F, col=4){
  pred = prediction(phat, y)
```

```
  perf = performance(pred, measure = "lift", x.measure = "rpp")
  plot(perf, add=add, col=col, main=paste("Lift Curve",title), lwd=2)
  abline(h=1, lty=2)
}


# Logistic
getLift(df.test$Retained.in.2012., phat.lgfit.selected, "Logistic and Lasso", col=4)
# Lasso
getLift(df.test$Retained.in.2012., phat.l1.lgfit, col=2, add=T)
legend("topright", legend=c("Logistic","Lasso"), col=c(4,2), lwd=c(2,2))
```



0.4 Q4.

```
ProfitCurve <- function(y, phat, benefitTP=60, benefitFN=0,
                        benefitFP=-40, benefitTN=0, title="", add=F, col=4){
  n <- length(y)
  df <- data.frame(y, phat)
  df <- df[order(df[,2], decreasing = T),]
  TP <- 0; FP <- 0; FN <- table(y)[2]; TN <- table(y)[1]

  ratio.vec <- seq(0,n)/n
  profit.vec <- rep(0,n+1)
  profit.vec[1] <- FN * benefitFN + TN * benefitTN

  for(k in 1:n){
```

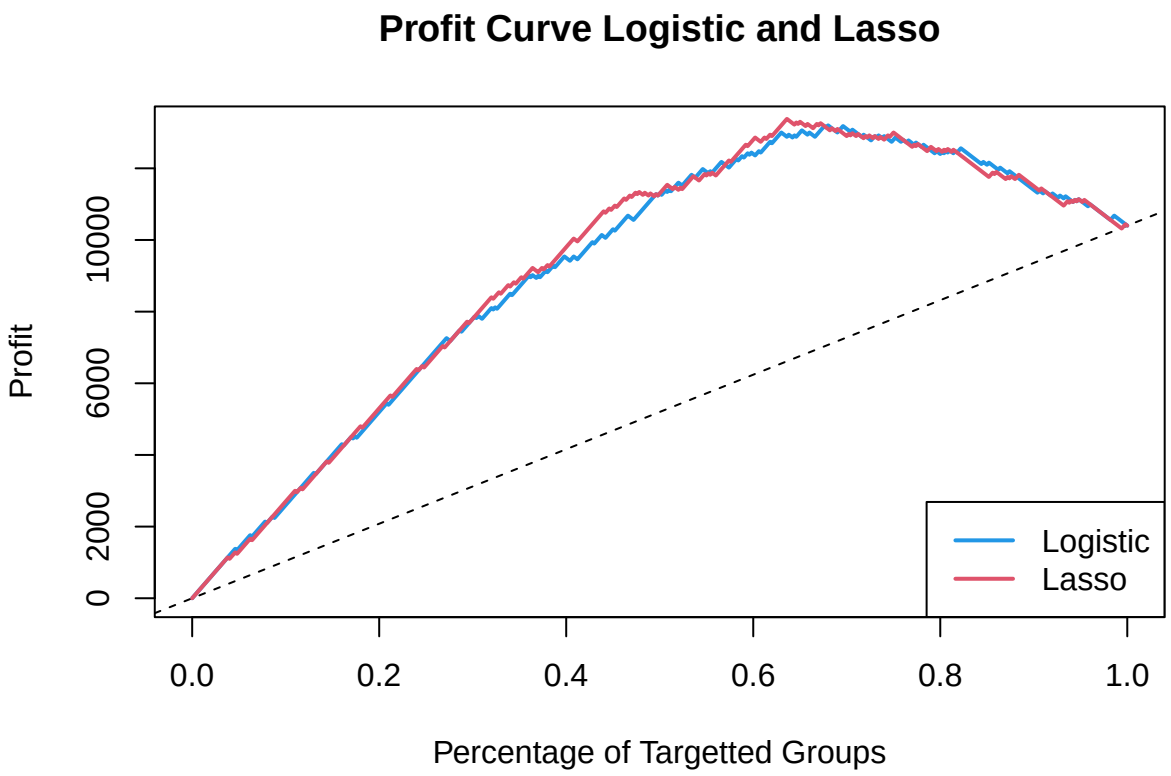
```
    if(df[k,1]=="1"){TP <- TP + 1; FN <- FN - 1}
    else{FP <- FP + 1; TN <- TN - 1}
    profit.vec[k+1] <- TP*benefitTP + FP*benefitFP + FN*benefitFN + TN*benefitTN
  }

  kmax <- which.max(profit.vec)-1
  maxprofit <- max(profit.vec)

  if(add==F){
    plot(ratio.vec, profit.vec, type="l", lwd=2, col=col,
       main=paste("Profit Curve", title),
       xlab="Percentage of Targetted Groups", ylab="Profit")
    abline(b=(profit.vec[n+1]-profit.vec[1]), a=profit.vec[1], lty=2) #Random guess
  }else{
    lines(ratio.vec, profit.vec, type="l", lwd=2, col=col)
  }

  return(list(max.k=kmax, max.profit=maxprofit))
}


# Logistic Regression
ProfitCurve(df.test$Retained.in.2012., phat.lgfit.selected, title="Logistic and Lasso")
```

```
## $max.k
## [1] 340
##
## $max.profit
## [1] 13200
```

```
# Lasso
ProfitCurve(df.test$Retained.in.2012., phat.l1.lgfit, add=T, col=2)
```

```
## $max.k
## [1] 318
##
## $max.profit
## [1] 13380
```

```
legend("bottomright", legend=c("Logistic","Lasso"), col=c(4,2), lwd=c(2,2))
```



Now assume the cost is \$65 instead of \$40, while the benefit is still \$100:

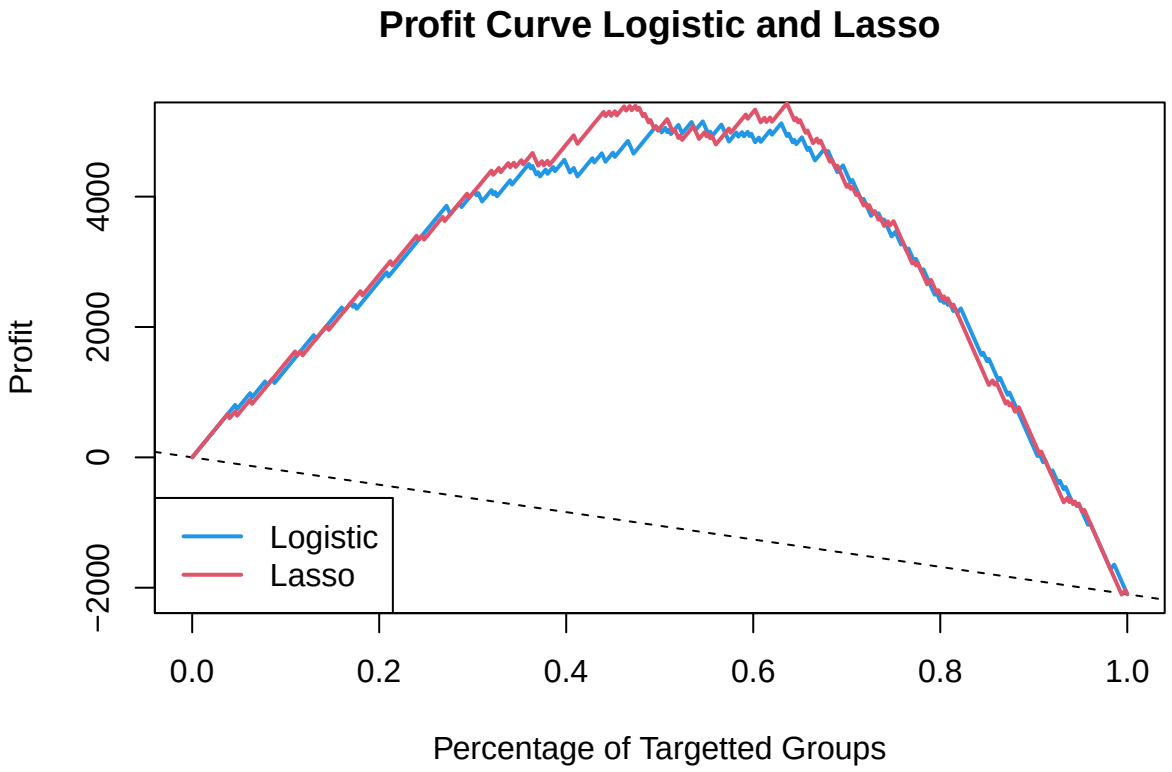
```
# Logistic Regression
ProfitCurve(df.test$Retained.in.2012., phat.lgfit.selected, 35,0,-65,0, title="Logistic and Lasso")
```

```
## $max.k
## [1] 273
##
## $max.profit
## [1] 5155
```

```
# Lasso
ProfitCurve(df.test$Retained.in.2012., phat.l1.lgfit, 35,0,-65,0, add=T, col=2)
```

```
## $max.k
## [1] 318
##
## $max.profit
## [1] 5430
```

```
legend("bottomleft", legend=c("Logistic","Lasso"), col=c(4,2), lwd=c(2,2))
```



When the ratio between the benefit and cost decreases, the optimal number of targetted groups decreases, and the maximal profit also decreases.

0.5 Q5

The answer consists of two steps: first we compute their respective estimated probabilities, then we plot and compare the curves.

```
get.deviance = function(y,phat,wht=1e-7) {
  if(is.factor(y)) y = as.numeric(y)-1
  phat = (1-wht)*phat + wht*.5
  py = ifelse(y==1, phat, 1-phat)
  return(-2*sum(log(py)))
}
```

0.5.1 Step 1. computing phat

```
##########
## Tree
##########

library(rpart)
library(rpart.plot)

get.phat.tree <- function(df.train, df.test){

  hyper_grid_tree <- expand.grid(
```

```
    minsplit = c(2,5,10),
    cp = c(0.0001, 0.00001),
    xval = c(5,10)
  )

  phat.tree.matrix <- matrix(0.0, nrow(df.test), nrow(hyper_grid_tree))

  for(j in 1:nrow(hyper_grid_tree)){
    big.tree <- rpart(Retained.in.2012. ~ ., data=df.train,
                      control=rpart.control(
                        minsplit=hyper_grid_tree$minsplit[j],
                        cp=hyper_grid_tree$cp[j],
                        xval=hyper_grid_tree$xval[j]
                      ))
    cptable <- printcp(big.tree)
    bestcp <- cptable[which.min(cptable[,"xerror"]), "CP"]
    best.tree <- prune(big.tree, cp=bestcp)
    phat <- predict(best.tree, df.test)[,"1"]
    phat.tree.matrix[,j] <- phat
  }

  y <- as.numeric(df.test$Retained.in.2012.) - 1
  dev <- rep(0, nrow(hyper_grid_tree))
  for(j in 1:nrow(hyper_grid_tree)){
    dev[j] <- get.deviance(y, phat.tree.matrix[j])
  }
  j.best <- which.min(dev)
  phat.best <- phat.tree.matrix[,j.best]
  return(phat.best)
}

phat.tree <- get.phat.tree(df.train, df.test)
```

```
##########
## Rdm Frst
##########

library(ranger)

get.phat.rf <- function(df.train, df.test){

  p <- ncol(df.train)-1

  hyper_grid_rf <- expand.grid(
    mtry       = c(p,ceiling(sqrt(p))),
    node_size  = c(5, 10, 20))

  phat.rf.matrix <- matrix(0.0, nrow(df.test), nrow(hyper_grid_rf))

  for(j in 1:nrow(hyper_grid_rf)){
    # Train model
    rf.model <- ranger(
      formula = Retained.in.2012. ~ .,
```

```
      data = df.train,
      num.trees = 250,
      mtry = hyper_grid_rf$mtry[j],
      min.node.size = hyper_grid_rf$node_size[j],
      probability = T,
      seed = 233
    )
    phat <- predict(rf.model, data = df.test)$predictions[,2]
    phat.rf.matrix[,j] <- phat
  }

  y <- as.numeric(df.test$Retained.in.2012.) - 1
  dev <- rep(0, nrow(hyper_grid_rf))
  for(j in 1:nrow(hyper_grid_rf)){
    dev[j] <- get.deviance(y,phat.rf.matrix[j])
  }
  j.best <- which.min(dev)
  phat.best <- phat.rf.matrix[,j.best]
  return(phat.best)
}

phat.rf <- get.phat.rf(df.train, df.test)
```

```
##########
## Boosting (takes a long time)
##########

library(xgboost)
```

```
##
## Attaching package: 'xgboost'

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

```
library(caret)

get.phat.xgb <- function(df.train, df.test){
  # First put our data into a suitable form, i.e. dummify and partition
  df.tr.in <- df.train[,-55]
  dmy <- dummyVars(" ~ .", data = df.tr.in)
  trsf <- data.frame(predict(dmy, newdata = df.tr.in))
  X.train <- as.matrix(trsf)
  Y.train <- as.numeric(df.train[,55])-1

  df.te.in <- df.test[,-55]
  dmy <- dummyVars(" ~ .", data = df.te.in)
  tesf <- data.frame(predict(dmy, newdata = df.te.in))
  X.test <- as.matrix(tesf)
  Y.test <- as.numeric(df.test[,55])-1
```

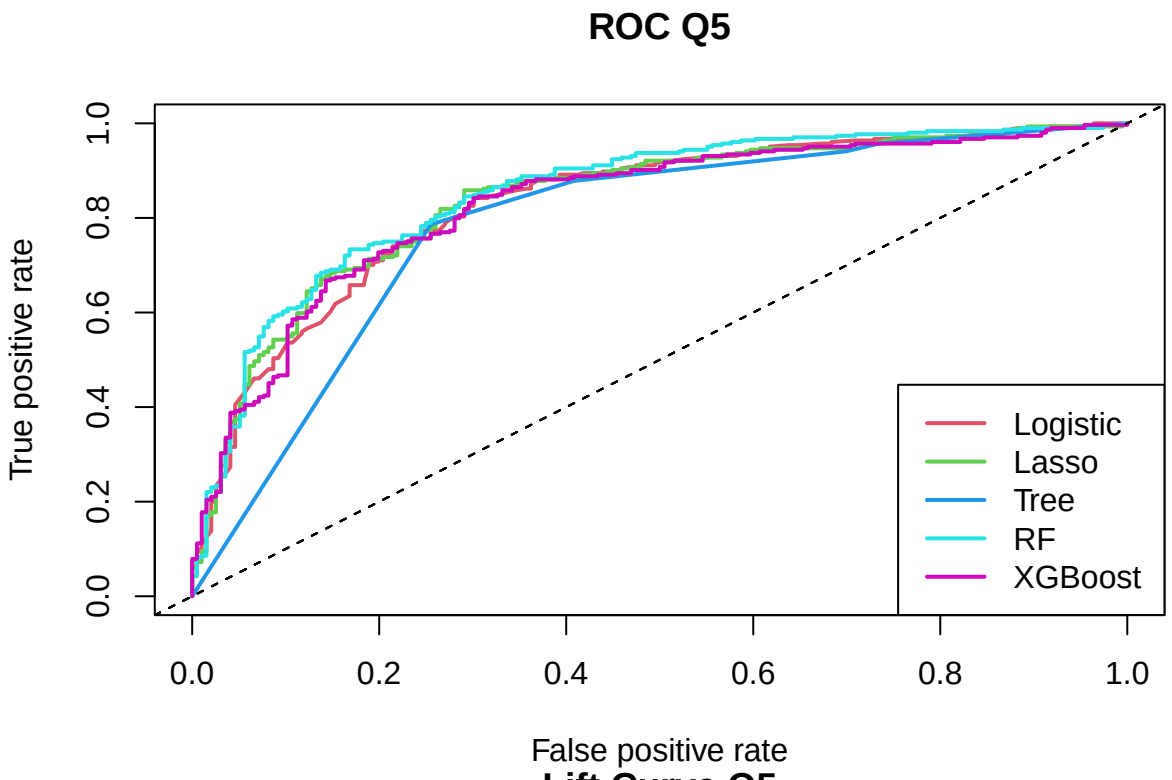
```
  # Hypergrid of XGBoost parameters
  hyper_grid_xgb <- expand.grid(
    shrinkage =c(.01, .1), ## controls the learning rate
    interaction.depth =c(1, 2, 4), ## tree depth
    nrounds =c(1000, 3000) ## number of trees
  )
  set.seed(233)
  phat.xgb.matrix <- matrix(0.0, nrow(df.test), nrow(hyper_grid_xgb))

  for(j in 1:nrow(hyper_grid_xgb)){
    params <- list(
      eta       = hyper_grid_xgb$shrinkage[j],
      max_depth = hyper_grid_xgb$interaction.depth[j]
    )
    xgb.model <- xgboost(
      data      = X.train,
      label     = Y.train,
      params    = params,
      nrounds   = hyper_grid_xgb$nrounds[j],
      objective = "binary:logistic",
      verbose   = 0,
      verbosity = 0
    )
    phat <- predict(xgb.model, X.test)
    phat.xgb.matrix[,j] <- phat
  }

  y <- as.numeric(df.test$Retained.in.2012.) - 1
  dev <- rep(0, nrow(hyper_grid_xgb))
  for(j in 1:nrow(hyper_grid_xgb)){
    dev[j] <- get.deviance(y,phat.xgb.matrix[j])
  }
  j.best <- which.min(dev)
  phat.best <- phat.xgb.matrix[,j.best]
  return(phat.best)
}

phat.xgb <- get.phat.xgb(df.train, df.test)
```

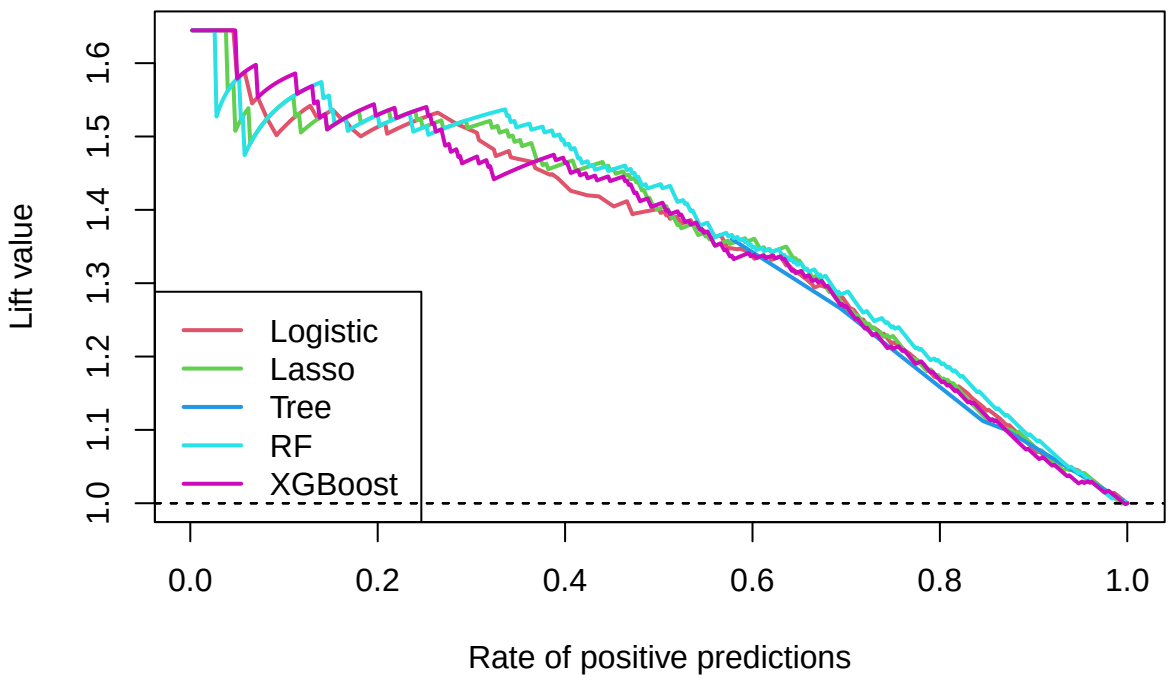
0.5.2 Step 2. Evaluating the estimated probabilities

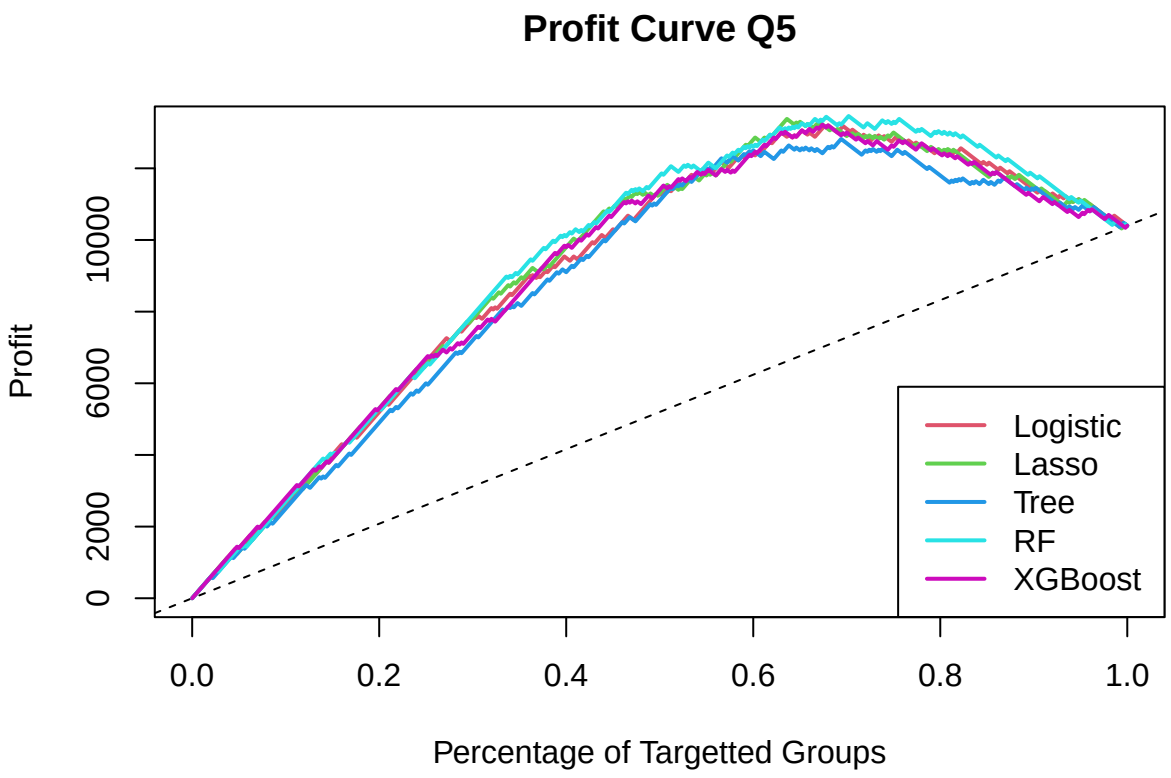
```
# Function to evaluate the result
# Inputs:
#   - y: vector of true labels
#   - phat.list: a list of phat vectors
#   - vec.method: vector of names of methods, for labeling the legends

evaluate <- function(y, phat.list, vec.method, title="Q5"){
  m <- length(phat.list)
  vec.auc <- rep(0,m)
  vec.k <- rep(0,m)
  vec.profit <- rep(0,m)
```

```
  # Figure 1. ROC
  vec.auc[1] <- getROC(y, phat.list[[1]], title=title, col=2)
  for(j in 2:m){ # for each method
    vec.auc[j] <- getROC(y, phat.list[[j]], col=j+1, add=T)
  }
  legend("bottomright", vec.method, col=(2:(m+1)), lwd=2)

  # Figure 2. Lift Curve
  getLift(y, phat.list[[1]], title=title, col=2)
  for(j in 2:m){
    getLift(y, phat.list[[j]], col=j+1, add=T)
  }
  legend("bottomleft", vec.method, col=(2:(m+1)), lwd=2)

  # Figure 3. Profit Curve
  pc <- ProfitCurve(y, phat.list[[1]], title=title, col=2)
  vec.k[1] <- pc$max.k; vec.profit[1] <- pc$max.profit
  for(j in 2:m){
    pc <- ProfitCurve(y, phat.list[[j]], col=j+1, add=T)
    vec.k[j] <- pc$max.k; vec.profit[j] <- pc$max.profit
  }
  legend("bottomright", vec.method, col=(2:(m+1)), lwd=2)

  # Printed results
  cat("====================================\n")
  cat(paste("     ", title,"Quantitive Results \n"))
  cat("====================================\n")
  cat("AUC:\n")
  for(j in 1:m){
    cat(paste("   *",vec.method[j],"---", round(vec.auc[j],5),"\n"))
  }
  cat("------------------------------------\n")
  cat("Optimal number of targetted groups:\n")
  for(j in 1:m){
    cat(paste("   *",vec.method[j],"---", vec.k[j],"\n"))
  }
  cat("------------------------------------\n")
  cat("Optimal profit:\n")
  for(j in 1:m){
    cat(paste("   *",vec.method[j],"---", vec.profit[j],"\n"))
  }
  cat("====================================\n")
}
```

```
phat.list <- list(phat.lgfit.selected, phat.l1.lgfit, phat.tree, phat.rf, phat.xgb)
vec.method <- c("Logistic",
                "Lasso   ",
                "Tree    ",
                "RF      ",
                "XGBoost ")
evaluate(df.test$Retained.in.2012., phat.list, vec.method)
```


ROC Q5
True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
Logistic
Lasso
Tree
RF
XGBoost


Lift Curve Q5
Lift value
1.0
1.1
1.2
1.3
1.4
1.5
1.6
Rate of positive predictions
Logistic
Lasso
Tree
RF
XGBoost




```
## ====================================
##       Q5 Quantitive Results
## ====================================
## AUC:
##    * Logistic --- 0.83061
##    * Lasso    --- 0.83746
##    * Tree     --- 0.7857
##    * RF       --- 0.85182
##    * XGBoost  --- 0.82942
## ------------------------------------
## Optimal number of targetted groups:
##    * Logistic --- 340
##    * Lasso    --- 318
##    * Tree     --- 347
##    * RF       --- 351
##    * XGBoost  --- 337
## ------------------------------------
## Optimal profit:
##    * Logistic --- 13200
##    * Lasso    --- 13380
##    * Tree     --- 12820
##    * RF       --- 13460
##    * XGBoost  --- 13220
## ====================================
```

Random forest performs better than logistic regression in terms of AUC and optimal profit. XGBoost has a similar performance, while classification tree does not deliver a very good result.

0.6 Q6.

We merge the datasets, preprocess it following the same scheme above, and separate it into training and test set.

```
STCdata_A <- read.csv('travelData.csv')
STCdata_B <- read.csv('travelData_supplement.csv')
STCdata_merged = merge(STCdata_A, STCdata_B, by = 'ID')
STCdata_merged <- STCdata_merged[,-1]
unique.per.column <- sapply( dplyr::select_if(STCdata_merged, is.numeric), n_distinct)
column.names.to.factor <- names(unique.per.column)[unique.per.column < 15]
STCdata_merged <- mutate_at(STCdata_merged, column.names.to.factor, as.factor)
date.columns = c('Departure.Date', 'Return.Date', 'Deposit.Date', 'Early.RPL', 'Latest.RPL',
                 'Initial.System.Date', 'FirstMeeting', 'LastMeeting')
STCdata_merged <- mutate_at(STCdata_merged, date.columns, function(x) as.Date(x, format = "%m/%d/%Y"))
STCdata_merged <- mutate_if(STCdata_merged, is.character, as.factor)
STCdata_merged <- fixNAs(STCdata_merged)
STCdata_merged <- combinerarecategories(STCdata_merged,10)
df2.train <- STCdata_merged[ inTrain,]
df2.test <- STCdata_merged[-inTrain,]
```

(1) Logistic Regression

```
lgfit2.null <- glm(Retained.in.2012.~ 1,
                   data=df2.train, family="binomial")
lgfit2.all <- glm(Retained.in.2012.~ .,
                   data=df2.train, family="binomial")
lgfit2.selected <- step(lgfit2.null, scope=formula(lgfit2.all),
                        direction="forward", k=log(nrow(df2.train)), trace=1)
summary(lgfit2.selected)
phat.lgfit2.selected <- predict(lgfit2.selected, newdata = df2.test, type = "response")
```

(2) Lasso

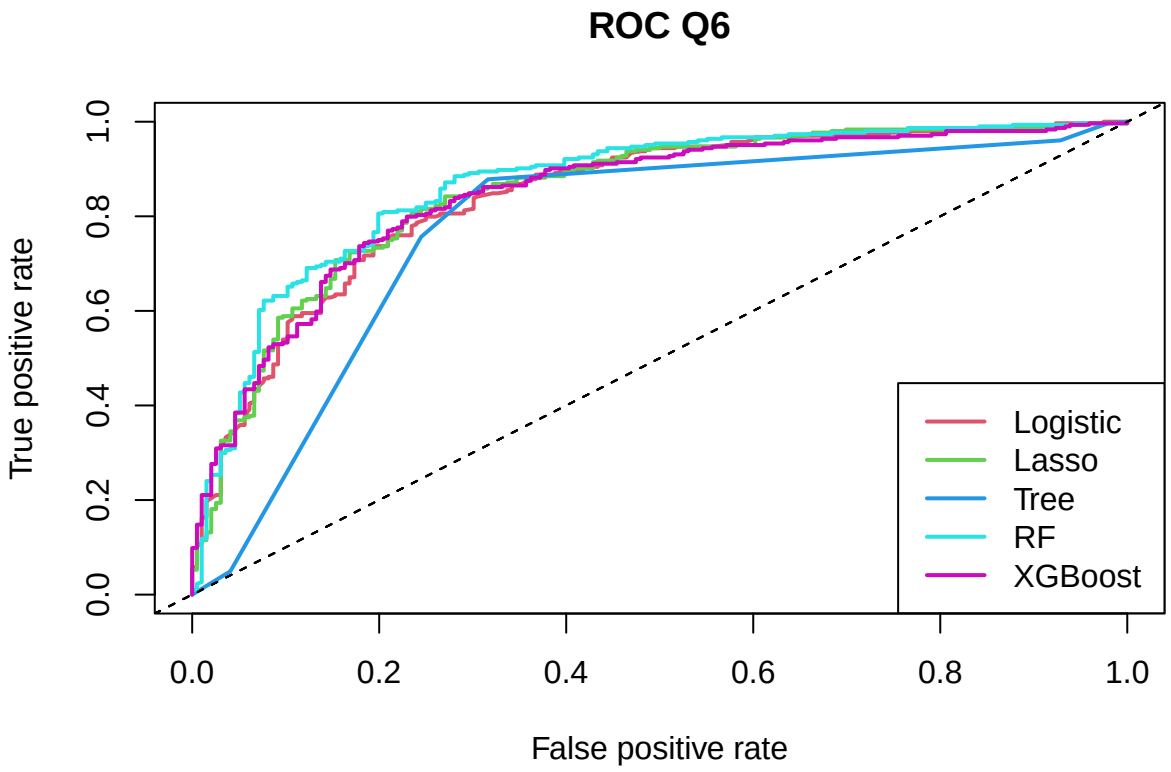
```
X2 <- model.matrix(formula(lgfit2.all), STCdata_merged)
#need to subtract the intercept
X2 <- X2[,-1]
X2.train = X2[ inTrain, ]
X2.test = X2[ -inTrain, ]
cv.l1.lgfit2 <- cv.glmnet(
  x = X2.train,
  y = df2.train$Retained.in.2012.,
  family = "binomial",
  alpha = 1,
  nfolds = 5
)
glmnet.fit2 <- cv.l1.lgfit2$glmnet.fit
phat.l1.lgfit2 <- predict(glmnet.fit2, newx = X2.test,
                          s = cv.l1.lgfit2$lambda.1se,
                          type = "response")
```

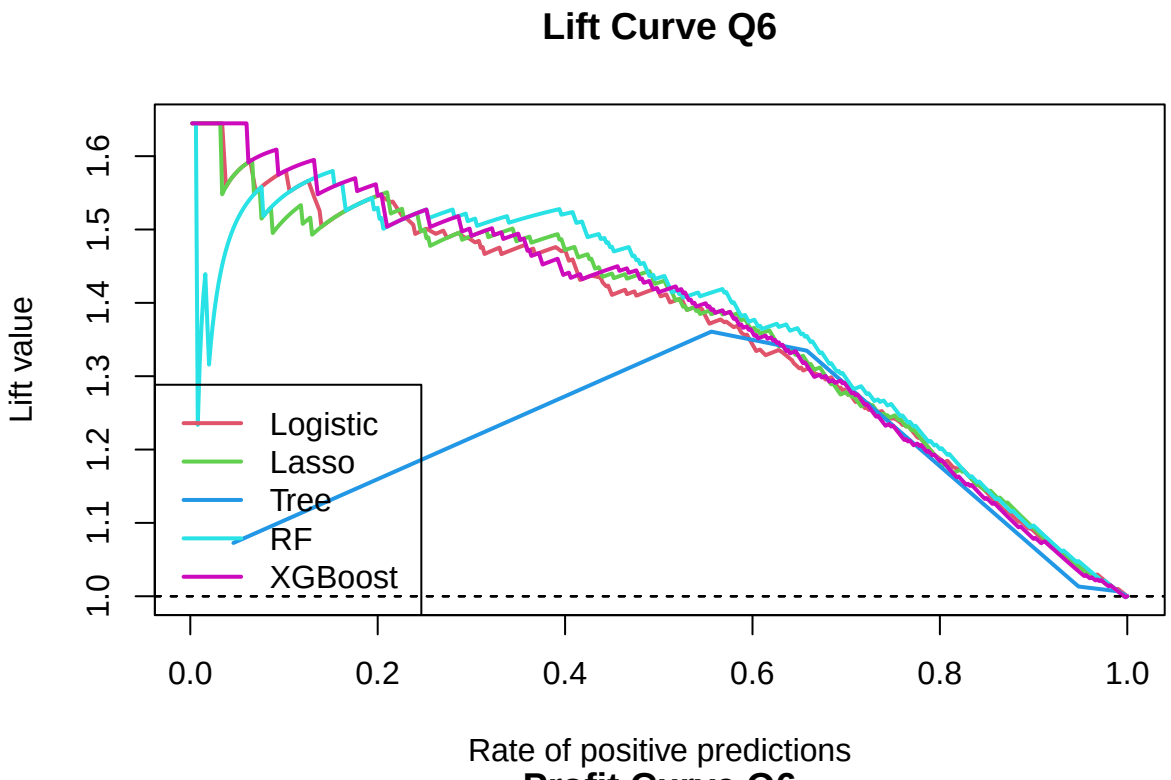
(3) Tree, RF and XGBoost

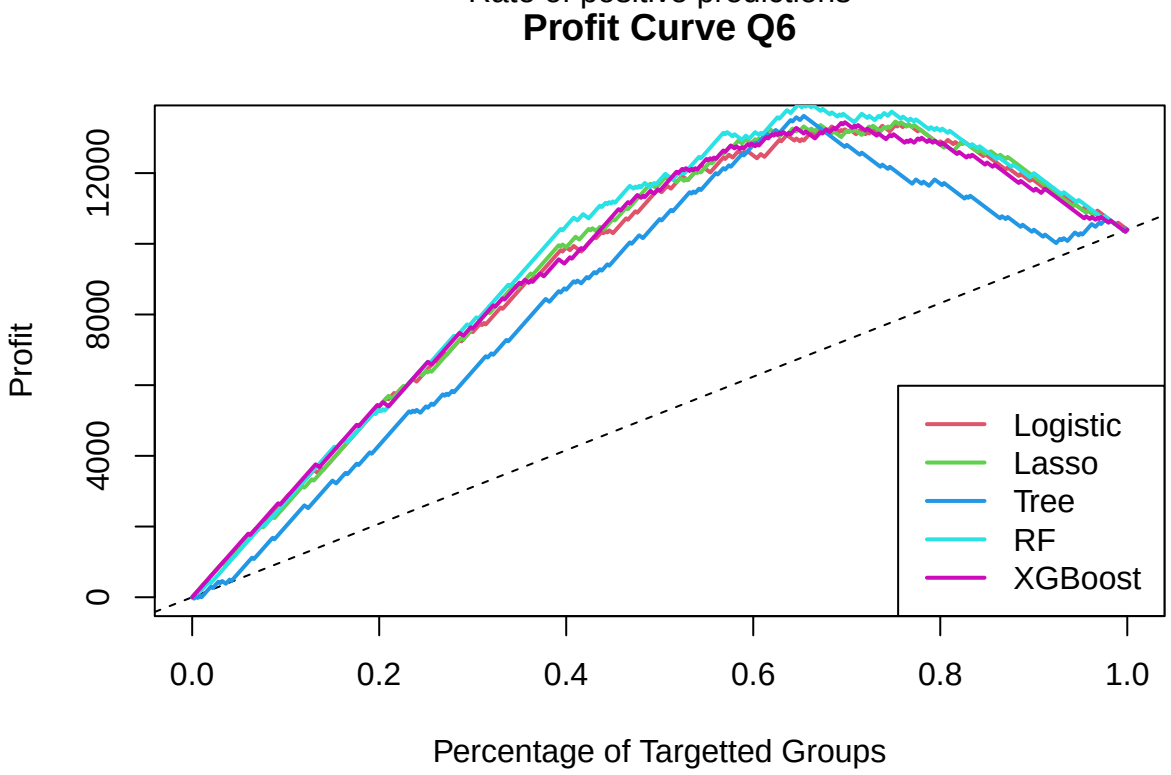
```
phat.tree2 <- get.phat.tree(df2.train, df2.test)
phat.rf2 <- get.phat.rf(df2.train, df2.test)
phat.xgb2 <- get.phat.xgb(df2.train, df2.test)
```

Finally we evaluate the results.

```
phat.list <- list(phat.lgfit2.selected, phat.l1.lgfit2, phat.tree2, phat.rf2, phat.xgb2)
vec.method <- c("Logistic",
                "Lasso   ",
                "Tree    ",
                "RF      ",
                "XGBoost ")
evaluate(df.test$Retained.in.2012., phat.list, vec.method, title="Q6")
```








```
## ===================================
##       Q6 Quantitive Results
## ===================================
## AUC:
##    * Logistic --- 0.84237
```

```
##    * Lasso    --- 0.84925
##    * Tree     --- 0.77484
##    * RF       --- 0.86657
##    * XGBoost  --- 0.84536
## ---------------------------------
## Optimal number of targetted groups:
##    * Logistic --- 378
##    * Lasso    --- 376
##    * Tree     --- 327
##    * RF       --- 324
##    * XGBoost  --- 349
## ---------------------------------
## Optimal profit:
##    * Logistic --- 13380
##    * Lasso    --- 13460
##    * Tree     --- 13620
##    * RF       --- 13940
##    * XGBoost  --- 13440
## =================================
```

For most of the models (except classification tree), the results have improved significantly after incorporating the NPS data, in terms of AUC and optimal profit.

In our complex problem, the classification tree is too simple for a satisfactory capacity of model complexity, therefore prone to underfitting. The more complex the model is, the worse prediction it gives.

The other models are more explanatory after introducing new variables, which yield better performance.