My homework

John Doe

June 15, 2015

## STAT LEARNING 2  
## HOMEWORK 4, Problem 6

# Libraries needed  
library(caret)  
library(magrittr)  
library("e1071")  
  
# Set working directory  
setwd("C:/Users/jrdha/OneDrive/Desktop/USU\_Fa2018/Moon\_\_SLDM2/hw4/Problem6")  
  
  
  
  
  
#===============================================================================  
#==== INITIAL STEPS ============================================================  
#===============================================================================  
  
# Read in the data  
parkData <- read.csv("parkinsonsData.csv", header = TRUE)  
  
# Subset, get rid of first column  
parkData <- parkData[ , -1]  
  
# Split into training and test data sets. Going with the typical 80/20 split.  
lengthTrain <- round(nrow(parkData) \* 0.8)  
lengthTest <- nrow(parkData) - lengthTrain  
  
# Set seed  
set.seed(1234)  
  
# Make indices for training data subsetting  
train\_ind <- sample(seq\_len(nrow(parkData)), size = lengthTrain)  
  
# Split up the data  
trainData <- parkData[train\_ind, ]  
testData <- parkData[-train\_ind, ]  
  
# Make the response a factor (rather than integer) so the algorithms perform  
# classification instead of regression.  
trainData$status <- as.factor(trainData$status)  
  
  
  
  
  
  
  
  
  
#===============================================================================  
#==== LOGISTIC REGRESSION ======================================================  
#===============================================================================  
  
# This specifies that we'll be doing 10-fold crossvalidation  
ctrl <- trainControl(method = "repeatedcv",  
 number = 10,  
 savePredictions = TRUE)  
  
# Train the model  
mod\_fit <- train(status ~.,  
 data = trainData,  
 method= "glm",  
 family= binomial(),  
 trControl = ctrl,  
 tuneLength = 10)  
  
# This gives the 10-fold crossvalidated training error  
trainError\_logReg <- 1 - mod\_fit$results$Accuracy  
trainError\_logReg

## [1] 0.1533929

# Generates the predictions for our testData  
testData$pred = predict(mod\_fit, newdata=testData)  
  
# Generates a column that says whether or not a test observation was correctly  
# classified (1==correct, 0==incorrect)  
testData$correct <- with(testData, ifelse(status == pred, 1, 0))  
  
# Generate the test error  
totalCorrect <- sum(testData$correct)  
testError\_logReg <- 1 - (totalCorrect / lengthTest)  
testError\_logReg

## [1] 0.1794872

# RETURNS A TEST ERROR RATE OF 0.1794872, WHICH IS PRECISELY 7  
# MISCLASSIFICATIONS. WE WILL BEAT THIS WITH BOTH KERNEL-VERSIONS OF SVM.  
  
# Remove the "correct" and "pred" columns so that we'll have the original data   
# to work with for the SVM portion  
testData <- subset(testData, select = -c(correct, pred))  
  
  
  
  
  
  
  
  
  
  
  
#===============================================================================  
#==== SVM ======================================================================  
#===============================================================================  
  
  
  
#========== TUNING LINEAR-KERNEL MODEL =========================================  
#===============================================================================  
  
  
# Tuning LINEAR KERNEL: Round 1 (used this to get a set of values that did well)  
# Ran it by changing cost = 1.digit\*10^(-3:3) where I set digit = 0,1,2,....,9  
set.seed(2345)  
svm\_tune\_linear <- tune.svm(status~.,  
 data = trainData,  
 kernel = "linear",   
 cost = 1.9\*10^(-3:3))  
print(svm\_tune\_linear)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1.9  
##   
## - best performance: 0.14125

#====== RESULTS ================================================================  
# Parameter tuning of 'svm':  
# - sampling method: 10-fold cross validation  
# - best parameters:  
# cost  
# 1.9  
# - best performance: 0.14125   
  
  
  
# Having found 1.9 to be the best value from above (by trying 1.0, 1.1, 1.2,...  
# ...,1.9) and seeing that most of these values were around 1.5 - 2.0, I decided  
# to just look at values between 1 and 2.  
# Tuning LINEAR KERNEL: Round 2 (looking only at values between 1 and 2.1)  
set.seed(2345)  
svm\_tune\_linear <- tune.svm(status~.,  
 data = trainData,  
 kernel = "linear",   
 cost = seq(1,2.1,0.01))  
print(svm\_tune\_linear)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1.52  
##   
## - best performance: 0.14125

#========= RESULTS =============================================================  
# IRONICALLY, THIS COMES UP WITH THE SAME CROSSVALIDATED ACCURACY AS FOR C=1.9,  
# SO WE SHOULDN'T EXPECT BETTER RESULTS FOR OUR TEST ERROR BY USING C=1.52 OVER  
# C=1.9.  
# Parameter tuning of 'svm':  
# - sampling method: 10-fold cross validation   
# - best parameters:  
# cost  
# 1.52  
# - best performance: 0.14125   
  
  
# THIS FITS A FINAL SVM MODEL, USING THE CROSSVALIDATED TUNING PARAMETER 1.52  
svm\_model <- svm(status ~., trainData, kernel = "linear", cost = 1.52)  
testData$pred <- predict(svm\_model, testData)  
testData$correct <- with(testData, ifelse(testData$status == pred, 1, 0))  
# Generate the test error  
# Total number of correct classifications  
totalCorrect <- sum(testData$correct)  
testError\_SVM <- 1 - (totalCorrect / lengthTest)  
testError\_SVM

## [1] 0.05128205

# Remove the "correct" column for the SVM portion  
testData <- subset(testData, select = -c(correct, pred))  
#=========== RESULTS ===========================================================  
# We get the same error rate (0.05128205) as we did by using C=1.9. However,  
# this makes sense, as we're already correctly classifying 37/39 values  
# correctly with C=1.9, so there's very little room for improvement. Using  
#C=1.52 gives just as good of results.  
  
  
  
# ADDITIONAL TUNING: WIDENING THE WINDOW, LOOKING BETWEEN 0.1-19  
# ONCE AGAIN, WE GET A COST IN THE SWEET SPOT OF 1.5 - 1.9, RETURNING 1.6, AND  
# THE IDENTICAL 0.14125 CROSS-VALIDATED ERROR RATE  
set.seed(2345)  
svm\_tune\_linear <- tune.svm(status~.,  
 data = trainData,  
 kernel = "linear",   
 cost = seq(0.1,19,0.1))  
print(svm\_tune\_linear)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1.6  
##   
## - best performance: 0.14125

# Parameter tuning of 'svm':  
# - sampling method: 10-fold cross validation   
# - best parameters:  
# cost  
# 1.6  
# - best performance: 0.14125  
  
  
  
# THIS FITS A FINAL SVM MODEL, USING THE CROSSVALIDATED TUNING PARAMETER 1.6  
svm\_model <- svm(status ~., trainData, kernel = "linear", cost = 1.6)  
testData$pred <- predict(svm\_model, testData)  
testData$correct <- with(testData, ifelse(testData$status == pred, 1, 0))  
# Generate the test error  
# Total number of correct classifications  
totalCorrect <- sum(testData$correct)  
testError\_SVM <- 1 - (totalCorrect / lengthTest)  
testError\_SVM

## [1] 0.05128205

# Remove the "correct" column for the SVM portion  
testData <- subset(testData, select = -c(correct, pred))  
  
  
  
# COMMON TEST ERROR RATES:  
# 0.05128205 for C values in the range of about 1.5 - 1.9  
# 0.07692308 for other C values  
# 0.1025641 for other C values  
# THIS MAKES SENSE: our test data set has only 39 observations. Therefore, we   
# can only achieve a certain level of granularity with our test error. The   
# best-tuned values for C miss only 2/39 classifications.  
  
  
  
  
  
  
  
  
  
#========== TUNING GAUSSIAN-KERNEL MODEL =======================================  
#===============================================================================  
  
# INITIAL PASS (leave base values at 10)  
set.seed(2345)  
svm\_tune\_gaussian <- tune.svm(status~., data = trainData,  
 cost = 10^(-3:7),  
 gamma = 10^(-5:2))  
print(svm\_tune\_gaussian)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## gamma cost  
## 0.1 10  
##   
## - best performance: 0.03208333

# Gives cost = 10 and gamma = 0.1 as best values, CV error of 0.03208333  
  
  
  
# LEFT COST=1.6, SINCE THIS WAS THE BEST VALUE FROM LINEAR-KERNEL SVM  
# THIS GIVES POORER RESULTS THAN WHEN WE LET THE COST BE BIGGER (and vary from  
# c=1.6, see below)  
# CHANGE THE BASE VALUES FOR GAMMA, TRYING 1.0, 1.1, 1.2,....,1.9  
set.seed(2345)  
svm\_tune\_gaussian <- tune.svm(status~., data = trainData,  
 cost = 1.6,  
 gamma = 1.9\*10^(-3:2))  
print(svm\_tune\_gaussian)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## gamma cost  
## 0.19 1.6  
##   
## - best performance: 0.06375

# Gives cost = 1.6 and gamma = 0.19 as best values with CV error: 0.06375  
  
  
  
# MANUAL GRID SEARCH, TRYING ALL 100 POSSIBLE COMBINATIONS OF (1.digit,1.digit)  
# for GAMMA and COST  
# CHANGE THE BASE VALUES FOR GAMMA, TRYING 1.0, 1.1, 1.2,....,1.9  
# CHANGE THE BASE VALUES FOR COST, TRYING 1.0, 1.1, 1.2,....,1.9  
set.seed(2345)  
svm\_tune\_gaussian <- tune.svm(status~., data = trainData,  
 cost = 1.0\*10^(-2:2),  
 gamma = seq(0.1,0.19,0.01))  
print(svm\_tune\_gaussian)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## gamma cost  
## 0.11 10  
##   
## - best performance: 0.02583333

# for cost=1.0^power, was consistently picking cost=10, and gamma=0.1digit,  
# so I set gamma to only be values of 0.1, 0.11, 0.12,....,0.19, and then tried  
# different values for cost (changing base to be 1.0, 1.1, 1.2,...,1.9)  
# For these restricted gamma, it was consistenly using gamma=0.16, and then cost  
# of 12, 13, 14, and giving CV error of 0.03833333.  
# However, the best combination turned out to be cost=10, gamma=0.11, giving a  
# CV error of 0.02583333!  
  
  
  
# TRYING TO NARROW DOWN EVEN FURTHER: IF WE KNOW THAT COST=10 AND GAMMA=0.11  
# GIVES THE BEST CV ERROR, LET'S TRY OTHER VALUES CLOSE TO THESE ONES.  
# I made the steps of the sequences more granular so we could try more values  
# and potentially find better ones.  
set.seed(2345)  
svm\_tune\_gaussian <- tune.svm(status~., data = trainData,  
 cost = seq(5,15,0.5),  
 gamma = seq(0.1,0.19,0.005))  
print(svm\_tune\_gaussian)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## gamma cost  
## 0.105 8  
##   
## - best performance: 0.02583333

# HERE'S WHAT IS RETURNED. As we can see, we're still getting the same CV error  
# RATE (0.02583333) WE WERE FOR GAMMA=0.11 AND COST=10.  
# SO I THINK WE CAN SAFELY CONCLUDE THAT A COST IN 8-10 IS GOOD, AND TUNED  
# GAMMA OF EITHER 0.105 OR 0.11 IS GOING TO BE OUR BEST BEST FOR THE   
# GAUSSIAN-KERNEL SVM  
# Parameter tuning of 'svm':  
# - sampling method: 10-fold cross validation   
# - best parameters:  
# gamma cost  
# 0.105 8  
# - best performance: 0.02583333   
  
  
  
  
  
  
  
  
# THIS FITS A FINAL SVM MODEL, USING THE CROSSVALIDATED TUNING PARAMETERs  
# of C = (range of 8,9,10), and gamma = 0.105 or 0.11  
costVal = 8  
gammaVal = 0.105  
svm\_model <- svm(status ~., trainData, kernel = "radial",  
 cost = costVal, gamma = gammaVal)  
testData$pred <- predict(svm\_model, testData)  
testData$correct <- with(testData, ifelse(testData$status == pred, 1, 0))  
# Generate the test error  
# Total number of correct classifications  
totalCorrect <- sum(testData$correct)  
testError\_SVM <- 1 - (totalCorrect / lengthTest)  
testError\_SVM

## [1] 0.07692308

# Remove the "correct" column for the SVM portion  
testData <- subset(testData, select = -c(correct, pred))  
#========= RESULTS =============================================================  
# All different values here give error rate of 0.07692308, which is   
# misclassifying precisely 3/39 of our test set, so 1/39 worse than for the   
# linear kernel. Some of this is certainly due to there only being so much room  
# to improve the error rate, since we can only pick off 1 or 2   
# misclassifications in our test set.