#

# -------------------------- Week 9 Synchronous Code: SVM -----------------------------

# install.packages("kernlab")

# library("kernlab")

# library(e1071)

# library(ggplot2)

# library(gridExtra)

# Packages: kernlab, e1071, gridExtra, ggplot2, caret

#specify the packages of interest

packages=c("kernlab","e1071","gridExtra","ggplot2", "caret")

#use this function to check if each package is on the local machine

#if a package is installed, it will be loaded

#if any are not, the missing package(s) will be installed and loaded

package.check <- lapply(packages, FUN = function(x) {

if (!require(x, character.only = TRUE)) {

install.packages(x, dependencies = TRUE)

library(x, character.only = TRUE)

}

})

#verify they are loaded

search()

# --------------------------------------------------------------------

# Step 1: Load the data

air <- data.frame(airquality)

# find which columns in the dataframe contain NAs.

colnames(air)[colSums(is.na(air)) > 0]

# find the NAs in column "Ozone" and replace them by the mean value of this column

air$Ozone[is.na(air$Ozone)] <- mean(air$Ozone, na.rm=TRUE)

# find the NAs in column "Solar.R" and replace those NAs by the mean value of this column

air$Solar.R[is.na(air$Solar.R)] <- mean(air$Solar.R, na.rm=TRUE)

# --------------------------------------------------------------------

# Step 2: Create train and test data sets

# create a list of random index for air data and store the index in a variable called "ranIndex"

#

dim(air)

air[1:5,]

randIndex <- sample(1:dim(air)[1])

head(randIndex)

length(randIndex)

air[148,]

air[45,]

#

# # In order to split data, create a 2/3 cutpoint and round the number

cutpoint2\_3 <- floor(2\*dim(air)[1]/3)

# check the 2/3 cutpoint

cutpoint2\_3

#

# create train data set, which contains the first 2/3 of overall data

#

trainData <- air[randIndex[1:cutpoint2\_3],]

dim(trainData)

head(trainData)

#

# create test data, which contains the left 1/3 of the overall data

#

testData <- air[randIndex[(cutpoint2\_3+1):dim(air)[1]],]

dim(testData) # check test data set

head(trainData)

# --------------------------------------------------------------------

# Step 3: Build a Model using KSVM & visualize the results

# 1) Build a model to predict Ozone and name it "svmOutput"

# This is the Training step

#

svmOutput <- ksvm(Ozone~., # set "Ozone" as the target predicting variable; "." means use all other variables to predict "Ozone"

data = trainData, # specify the data to use in the analysis

kernel = "rbfdot", # kernel function that projects the low dimensional problem into higher dimensional space

kpar = "automatic",# kpar refer to parameters that can be used to control the radial function kernel(rbfdot)

C = 10, # C refers to "Cost of Constrains"

cross = 10, # use 10 fold cross validation in this model

prob.model = TRUE # use probability model in this model

)

# check the model

#

svmOutput

# 2) Test the model with the testData data set

#

svmPred <- predict(svmOutput, # use the built model "svmOutput" to predict

testData, # use testData to generate predictions

type = "votes" # request "votes" from the prediction process

)

str(svmPred)

#

# create a comparison dataframe that contains the exact "Ozone" value and the predicted "Ozone" value

# use for RMSE calc

#

compTable <- data.frame(testData[,1], svmPred[,1])

# change the column names to "test" and "Pred"

colnames(compTable) <- c("test","Pred")

#

# compute the Root Mean Squared Error

#

sqrt(mean((compTable$test-compTable$Pred)^2)) #A smaller value indicates better model performance.

# 3) Plot the results

# library(ggplot2)

# compute absolute error for each case

compTable$error <- abs(compTable$test - compTable$Pred)

# create a new dataframe contains error, tempreture and wind

svmPlot <- data.frame(compTable$error, testData$Temp, testData$Wind)

# assign column names

colnames(svmPlot) <- c("error","Temp","Wind")

# polt result using ggplot, setting "Temp" as x-axis and "Wind" as y-axis

ggplot(svmPlot, aes(x=Temp,y=Wind)) +

# use point size and color shade to illustrate how big is the error

geom\_point(aes(size=error, color=error))+

ggtitle("ksvm")

# --------------------------------------------------------------------

# Step 4: Create a "goodOzone" variable

# calculate average Ozone

meanOzone <- mean(air$Ozone,na.rm=TRUE)

# create a new variable named "goodOzone" in train data set

# goodOzone = 0 if Ozone is below average Ozone

# googOzone = 1 if Ozone is eaqual or above the average ozone

trainData$goodOzone <- ifelse(trainData$Ozone<meanOzone, 0, 1)

# do the same thing for test dataset

testData$goodOzone <- ifelse(testData$Ozone<meanOzone, 0, 1)

# remove "Ozone" from train data

trainData <- trainData[,-1]

# remove "Ozone" from test data

testData <- testData[,-1]

# --------------------------------------------------------------------

# Step 5: See if we can do a better job predicting 'good' and 'bad' days

# convert "goodOzone" in train data from numeric to factor

trainData$goodOzone <- as.factor(trainData$goodOzone)

# convert "goodOzone" in test data from numeric to factor

testData$goodOzone <- as.factor(testData$goodOzone)

# 1) Build a model

# build a model using ksvm function,and use all other variables to predict

svmGood <- ksvm(goodOzone~., # set "Ozone" as target variable; "." means use all other variables to predict "Ozone"

data=trainData, # specify the data to use in the analysis

kernel="rbfdot", # kernel function that projects the low dimensional problem into higher dimensional space

kpar="automatic",# kpar refer to parameters that can be used to control the radial function kernel(rbfdot)

C=10, # C refers to "Cost of Constrains"

cross=10, # use 10 fold cross validation in this model

prob.model=TRUE # use probability model in this model

)

# check the model

svmGood

# 2) Test the model

goodPred <- predict(svmGood, # use model "svmGood" to predict

testData # use testData to do the test

)

# create a dataframe that contains the exact "goodOzone" value and the predicted "goodOzone"

compGood1 <- data.frame(testData[,6], goodPred)

# change column names

colnames(compGood1) <- c("test","Pred")

# Compute the percentage of correct cases

perc\_ksvm <- length(which(compGood1$test==compGood1$Pred))/dim(compGood1)[1]

perc\_ksvm

# Confusion Matrix

#

results <- table(test=compGood1$test, pred=compGood1$Pred)

print(results)

# pred

# test 0 1

# 0 19 7 # read horizontal,, 0 class, 19 identified correctly, 7 incorrectly

# 1 7 18 # 1 class, 7 identified incorrectly, 18 correctly

# 3) Plot the results.

# determine the prediction is "correct" or "wrong" for each case

compGood1$correct <- ifelse(compGood1$test==compGood1$Pred,"correct","wrong")

# create a new dataframe contains correct, tempreture and wind, and goodZone

Plot\_ksvm <- data.frame(compGood1$correct,testData$Temp,testData$Wind,testData$goodOzone,compGood1$Pred)

# change column names

colnames(Plot\_ksvm) <- c("correct","Temp","Wind","goodOzone","Predict")

# polt result using ggplot

# size representing correct/wrong; color representing actual good/bad day; shape representing predicted good/bad day.

ggplot(Plot\_ksvm, aes(x=Temp,y=Wind)) +

geom\_point(aes(size=correct,color=goodOzone,shape = Predict))+

ggtitle("ksvm - good/bad ozone")

#

# caret

#

x<-train(Ozone~.,data = trainData,

method="svmRadial",

preProc=c("center","scale"),

tuneLength=14,

trControl = trainControl(method="cv"))

x