Project 1

Jacob Cadena

9/19/2018

Heart Disease Analysis

Studying the heart disease dataset, there were several interesting insights that came from the models created. These insights led to improvements on each model, such as the relevance of the old peak variable.

```r
# Import libraries for data tools
library(tidyverse)
library(caret)
library(e1071)
library(GGally)

# Clearing the screen and work environment
rm(list = ls())
cat("\014")
```

```r
# Set the working directory
setwd("/Users/jcadena/Documents/Semesters/Fall 2018/Project-1")
```

```r
# Importing each processed dataset into list to aggregate

col.names <- c("age","sex", "cp","trestbps", "chol",
               "fbs", "restecg", "thalach", "exang",
               "oldpeak", "slope", "ca", "thal", "num")


data.directory <- "./heart-disease/"
list.of.files <- paste0(data.directory,"processed.", c("cleveland",
                                                       "hungarian",
                                                       "switzerland",
                                                       "va"), ".data")

list.of.datasets <- lapply(X = list.of.files,
                           FUN = read.csv,
                           header = FALSE,
                           col.names = col.names,
                           na.strings = c("?","", " "))


# Aggregate the data from the list of the datasets
dataset <- list.of.datasets[1] %>%
  bind_rows(list.of.datasets[2]) %>%
  bind_rows(list.of.datasets[3]) %>%
  bind_rows(list.of.datasets[4])
```

Preprocessing

The preprocessing stage proved to be a very important step in predicting heart disease. After changing the categoircal variables to factors, one hot encoding was implemented on the chest pain type variable. That seperated the variable into binary variables, allowing the model to grasp the essence of each type of chest pain type. In other words, rather than having one variable with four categories, the chest pain variable

```
# Cleaning the dataset for anaylsis

# Change categorical variables to factors
col.names.to.factor <- c("sex", "cp", "fbs", "restecg", "exang", "slope", "num")
dataset[,col.names.to.factor] <- lapply(dataset[,col.names.to.factor], factor)

dummies <- dummyVars(formula = ~ cp, data = dataset)
one.hot.encoders<- data.frame(predict(dummies, newdata = dataset))
one.hot.encoders <- one.hot.encoders[-4]

dataset <- dataset %>%
  select(-cp) %>%
  bind_cols(one.hot.encoders)

# Change cardinality in the num column
levels(dataset$num)[2:5] <- "1"
print(levels(dataset$num))
```

```
## [1] "0" "1"
```

```
# Remove features from the dataset if 30% of a column are NA's
dataset <- dataset[, colSums(is.na(dataset)) < floor(.3*nrow(dataset))]

print(colSums(is.na(dataset)))
```

```
##      age      sex trestbps     chol      fbs  restecg  thalach    exang
##        0        0       59       30       90        2       55       55
##  oldpeak      num     cp.1     cp.2     cp.3
##       62        0        0        0        0
```

```
missing.value.strategy <- function(dataset, str = "naive"){
  if(str == "naive")
    return(dataset %>% drop_na())
}

dataset <- missing.value.strategy(dataset)

print(colSums(is.na(dataset)))
```

```
##      age      sex trestbps     chol      fbs  restecg  thalach    exang
##        0        0        0        0        0        0        0        0
##  oldpeak      num     cp.1     cp.2     cp.3
##        0        0        0        0        0
```

After creating the dataset obtained my transforming the data, it was split into a training set and test set, which later goes through k-fold cross validation. The data partioning using the the sample function in base R really didn't address the problem of covariate shift. A solution was to utilize the createDataPartition in the caret package, which seperate data with equal distributions.

```r
# Split dataset in training/validation/test sets
set.seed(3123)
n = which(colnames(dataset) == "num")
indices <- createDataPartition(y = dataset$num, p=0.7, list = FALSE)
training.set <- dataset[indices,]
test.set <- dataset[-indices,]
```

That included normalizing variables that have different magnitudes of the mean and standard deviation for faster convergence in training. Utlizing the normalizing strategies such as mean normalization or min max normlization, did not really speed up training for any for the models attempted, which may be due to the fact that there are not many observations.

```r
# If mean or standard deviation is too large in comparision to other
# variables apply feature scaling

minmax.normalization <- function(x){
  return((x - min(x))/(max(x) - min(x)))
}

mean.normalization <- function(x){
  return((x - mean(x,na.rm = TRUE))/sd(x,na.rm = TRUE))
}

continous.vars <- c("age","trestbps","chol","thalach","oldpeak")
training.set[,continous.vars] <- sapply(training.set[,continous.vars], mean.normalization)
test.set[,continous.vars] <- sapply(test.set[,continous.vars], mean.normalization)


cor(dataset[,continous.vars])
```

```
##                  age    trestbps        chol    thalach     oldpeak
## age       1.00000000  0.25181660 -0.06841944 -0.3670200  0.25174112
## trestbps  0.25181660  1.00000000  0.06075887 -0.1215939  0.18202690
## chol     -0.06841944  0.06075887  1.00000000  0.1962806  0.06065968
## thalach  -0.36701996 -0.12159392  0.19628057  1.0000000 -0.18186697
## oldpeak   0.25174112  0.18202690  0.06065968 -0.1818670  1.00000000
```

```r
# Clean up space in memory by deleting variables not used
rm(list = c("list.of.datasets", "list.of.files", "data.directory",
            "col.names.to.factor","indices", "one.hot.encoders",
            "missing.value.strategy", "mean.normalization", "minmax.normalization",
            "dummies", "col.names"))
```