HW 02

Samuel Swain

2022-11-11

Question 1

Read Data

```
df <- read.csv(file = 'gradAdmit.csv', header = T)
```

1(a)

```
set.seed(400)
n = nrow(df)

# Get 20% for test set
sample = sample.int(n = n, size = floor(.2*n), replace = F)
train_wrong_index = df[-sample,]
test = df[sample,]
train = train_wrong_index
rownames(train) = 1:nrow(train_wrong_index)
```

1(b)

```
# Libraries
library(e1071)
library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

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

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(hash)
```

```
## hash-2.2.6.2 provided by Decision Patterns
```

Cross validation function

```
GridSearch_Custom <- function(nfolds = 5,
                              kernel = "radial",
                              cost = 1,
                              degree = 3,
                              gamma = 1/3,
                              coef0 = 0) {


  # Get folds
  folds = createFolds(1:nrow(train),
                      k = nfolds)

  temp_acc_list = c()
  temp_auc_list = c()

  for (i in 1:nfolds){
    # Get data for folds
    training = train[-folds[[i]],]
    validation = df[folds[[i]],]

    # Train temp SVM
    svm_temp <- svm(factor(admit) ~ gre + gpa + rank,
                    data = training,
                    scale = T,
                    probability = T,
                    # default radial
                    kernel = kernel,
                    # cost of constraints violation (default: 1)
                    cost = cost,
                    # needed for: polynomial (default: 3)
                    degree = degree,
                    # needed for all except linear (default: 1/(data dimension))
                    gamma = gamma,
                    # needed for: polynomial, sigmoid (default: 0)
                    coef0 = coef0
          )

    # Test temp SVM
    pred_temp <- predict(svm_temp,
                         validation,
                         decision.values = F,
                         probability = F)

    # Get temp accuracy and AUC
    acc_temp <- mean(pred_temp==validation$admit)
    roc_obj <- roc(as.numeric(validation$admit),
                   as.numeric(pred_temp),
                   levels = c(0, 1),
                   direction = "<")
    auc_temp <- auc(roc_obj)[1]

    # Print temp values
    # cat("Acc: ", acc_temp, "\n")
    # cat("AUC: ", auc_temp, "\n")
    # print("-----------------")

    #temp_acc_list = append(x = temp_acc_list, values = acc_temp)
    temp_auc_list = append(x = temp_auc_list, values = auc_temp)
  }
  return(mean(temp_auc_list))
}
```

```
# Set parameters
nfolds = 5
kernels = c('linear', 'polynomial', 'radial', 'sigmoid')
cost <- c(0.001, 0.1, 1)
gamma <- c(0.001, 0.1, 1)
coef0 <- c(1, 10, 100)
degree <- c(3, 4, 5)

# Set parameters testing set
# cost <- c(1)
# degree <- c(1)
# gamma <- c(1)
# coef0 <- c(1)

# Init best parameter storage
best_params = hash()
best_params[["k"]] <- 'radial'
best_params[["c"]] <- 1
best_params[["g"]] <- 1/3
best_params[["c0"]] <- 0
best_params[["d"]] <- 3
best_params[["auc"]] <- 0

for (k in kernels) {
  for (c in cost) {
    if (k == 'linear') {
      auc <- GridSearch_Custom(nfolds = nfolds,
                               kernel = k,
                               cost = c)
      if (auc > best_params[["auc"]]) {
        best_params[["k"]] <- k
        best_params[["c"]] <- c
        best_params[["g"]] <- 1/3
        best_params[["c0"]] <- 0
        best_params[["d"]] <- 3
        best_params[["auc"]] <- auc
      }
    } else {
      for (g in gamma) {
        if (k == 'radial') {
          auc <- GridSearch_Custom(nfolds = nfolds,
                                   kernel = k,
                                   cost = c,
                                   gamma = g)
          if (auc > best_params[["auc"]]) {
            best_params[["k"]] <- k
            best_params[["c"]] <- c
            best_params[["g"]] <- g
            best_params[["c0"]] <- 0
            best_params[["d"]] <- 3
            best_params[["auc"]] <- auc
          }
        } else {
          for (c0 in coef0) {
            if (k == 'sigmoid') {
              auc <- GridSearch_Custom(nfolds = nfolds,
                                       kernel = k,
                                       cost = c,
                                       gamma = g,
                                       coef0 = c0)
              if (auc > best_params[["auc"]]) {
                best_params[["k"]] <- k
                best_params[["c"]] <- c
                best_params[["g"]] <- g
                best_params[["c0"]] <- c0
                best_params[["d"]] <- 3
                best_params[["auc"]] <- auc
              }
            } else {
              for (d in degree) {
                auc <- GridSearch_Custom(nfolds = nfolds,
                                         kernel = k,
                                         cost = c,
                                         gamma = g,
                                         coef0 = c0,
                                         degree = d)
                if (auc > best_params[["auc"]]) {
                  best_params[["k"]] <- k
                  best_params[["c"]] <- c
                  best_params[["g"]] <- g
                  best_params[["c0"]] <- c0
                  best_params[["d"]] <- d
                  best_params[["auc"]] <- auc
                }
              }
            }
          }
        }
      }
    }
  }
}
```

The model with kernel polynomial performed the best. The hyper parameters are as follows:
- cost: 1
- gamma: 1
- coef0: 1
- degree: 5

1(c)

```
# Train final SVM
svm_final <- svm(factor(admit) ~ gre + gpa + rank,
                 data = train,
                 scale = T,
                 probability = T,
                 kernel = best_params[["k"]],
                 cost = best_params[["c"]],
                 degree = best_params[["d"]],
                 gamma = best_params[["g"]],
                 coef0 = best_params[["c0"]]
                 )

# Get final prediction
pred_final <- predict(svm_final,
                      test,
                      decision.values = F,
                      probability = F)

# Get accuracy of final model
acc_final <- mean(pred_final==test$admit)
print(acc_final)
```

```
## [1] 0.6125
```

The final accuracy of my model is 0.6125