Dollina Dodani
301305914

Project 2 - Report

1. Models

- **KNN-Naive**

  knnfit.1.2 <- knn(train=X.train.scaled, test=X.valid.scaled, cl=Y.train, k=1)

- **KNN min CV**

  Tried k values from 1-200 using a function
  knncv.fit <- knn.cv(train=X.train.scaled, cl=Y.train, k=x)
  misclass.2.knnmin <- mean(ifelse(knnfitmin.2 == Y.valid, yes=0, no=1))

- **KNN 1SE**

  Got SE using
  serule = max(which(mis<mis[mink]+mis.se[mink]))
  knnfitse.2 <- knn(train=X.train.scaled, test=X.valid.scaled, cl=Y.train, k=serule)
  where mink is the number of neighbours with minimum misclassification rate

- **Multinom**

  mod.fit <- multinom(data=train, formula=Y ~ ., trace=TRUE, maxit=600). Maximum iterations was set at 600.

- **glmnet**

  logit.fit <- glmnet(x=as.matrix(train[,-17]), y=train[,17], family="multinomial")

- **glenet cv**

  logit.cv <- cv.glmnet(x=as.matrix(train[,-17]), y=train[,17], family="multinomial")
  lambda.min = logit.cv$lambda.min
  lascv.pred.min <- predict(object=logit.cv, type="class", s=lambda.min,
  newx=as.matrix(valid[,-17]))

- **glenet 1SE**

  logit.cv <- cv.glmnet(x=as.matrix(train[,-17]), y=train[,17], family="multinomial")
  lambda.1se = logit.cv$lambda.1se
  lascv.pred.se <- predict(logit.cv, type="class", s=lambda.1se,
  newx=as.matrix(valid[,-17]))

- **LDA**

  fit.lda = lda(X.train.DA, Y.train)

- **QDA**

  fit.qda = qda(X.train.DA, Y.train)

- **GAM**

```
gam.m <- gam(data=data.train1, list(Y0
                        ~s(X1) + s(X2) + s(X3) + s(X4) +
                         s(X5) + s(X6) + s(X7) + s(X8) +
                        s(X9) + s(X10) + s(X11) + s(X12) +
                       s(X13) + s(X14) + s(X15) + s(X16),
                      ~s(X1) + s(X2) + s(X3) + s(X4) +
                       s(X5) + s(X6) + s(X7) + s(X8) +
                      s(X9) + s(X10) + s(X11) + s(X12) +
                     s(X13) + s(X14) + s(X15) + s(X16),
                    ~s(X1) + s(X2) + s(X3) + s(X4) +
                     s(X5) + s(X6) + s(X7) + s(X8) +
                    s(X9) + s(X10) + s(X11) + s(X12) +
```

```
    s(X13) + s(X14) + s(X15) + s(X16),
  ~s(X1) + s(X2) + s(X3) + s(X4) +
   s(X5) + s(X6) + s(X7) + s(X8) +
   s(X9) + s(X10) + s(X11) + s(X12) +
  s(X13) + s(X14) + s(X15) + s(X16)),
 family=multinom(K=4))
```

- **Naive Bayes kernel=FALSE**

```
NBn <- NaiveBayes(x=data.train[,-1], grouping=data.train[,1], usekernel=FALSE)
```

- **Naive Bayes + PCA - Normal Distribution**

```
pc <-  prcomp(x=data.train[,-1], scale.=TRUE)
xi.1 <- data.frame(pc$x, Y = as.factor(data.train$Y))
xi.2 <- data.frame(predict(pc, newdata=data.valid), Y = as.factor(data.valid$Y))
NBn.pc <- NaiveBayes(x=xi.1[,-17], grouping=xi.1[,17], usekernel=FALSE)
```

- **Naive Bayes + PCA + kernel**

```
NBk.pc <- NaiveBayes(x=xi.1[,-17], grouping=xi.1[,17], usekernel=TRUE)
```

- **Classification Trees - Default - Full**

```
wh.tree <- rpart(data=data.train, Y ~ ., method="class", cp=0)
```

- **Classification Tree - pruning - Min CV**

```
cpt = wh.tree$cptable
minrow <- which.min(cpt[,4])
cplow.min <- cpt[minrow,1]
cpup.min <- ifelse(minrow==1, yes=1, no=cpt[minrow-1,1])
cp.min <- sqrt(cplow.min*cpup.min)
wh.prune.cv.min <- prune(wh.tree, cp=cp.min)
```

- **Classification Tree - pruning 1SE**

```
se.row <- min(which(cpt[,4] < cpt[minrow,4]+cpt[minrow,5]))
cplow.1se <- cpt[se.row,1]
cpup.1se <- ifelse(se.row==1, yes=1, no=cpt[se.row-1,1])
cp.1se <- sqrt(cplow.1se*cpup.1se)
wh.prune.cv.1se <- prune(wh.tree, cp=cp.1se)
```

- **Random Forest - Default**

```
wh.rf <- randomForest(data=data.train, Y~., importance=TRUE, keep.forest=TRUE)
```

- **Random Forest - Tuned**

```
h.rfm <- randomForest(data=data.train, Y~., mtry=m, nodesize=ns)
m loop: 1:16
node size loop: c(2,3,4,5,6,7,8)
```

- **SVM - Default**

```
svm.1.1 <- svm(data=data.train, Y ~ ., kernel="radial", gamma=1, cost=1, cross=10)
```

- **SVM - Tuned with caret**

```
tuned.nnet <- train(x=data.train[,-1], y=data.train$Y, method="svmRadial",
      preProcess=c("center","scale"), trace=FALSE,
      tuneGrid=parmgrid, trControl = trcon)
vm.wh.tun <- svm(data=data.train, Y ~ ., kernel="radial",
     gamma=sigma, cost=C)
```

- **NNet - Default**

```
nn.1.0 <- nnet(x=X.train.scaled.NN, y=y.train.NN, size=1, maxit=1000,
              softmax=TRUE)
```

- **NNet - Tuned**

```
nn <- nnet(y=y.1, x=x.1, size=s, decay=d, maxit=2000, softmax=TRUE, trace=FALSE)
```

- **Side Note:** Did not fit boosting since it is only for classification problems with K<2 and did not fit GAM due to runtime/memory issues.

2. I used 10-fold cross-validation on the data. I first split the data (P2Data2020.csv) into two sets: training and validation using a randomized seed (set.seed(46685327, kind="Mersenne-Twister"). I then generated two folds for this seed to get the training and validation set. I tested each model using the training data and then predicted values using the test data. From the predicted values I calculated the misclassification rate (for each process in each fold) using: mean(ifelse(model == Y.valid, yes=0, no=1)). I then selected the model with the lowest mean misclassification rate. This was done using box plots and mean function in R.

3. Models that needed tuning:

a) Some methods like KNN, glmnet and classification trees were tuned with basic approaches (using min CV and 1SE). Other methods that required tuning utilized the carat package and nested cross validation that looped over multiple options for the parameters. For each combination, the misclassification rate was saved and for each fold, the best option was selected based on the internal CV (the one that had minimum misclassification rate). The model (with those parameters options) was then fitted in the origin 10-fold cross validation. Exact details for each method that was tuned is described below.

b)

KNN: The number of neighbours (k) tested were 1:200. For each value of k, the misclassification rate was found using:

```
mean(ifelse(knnfit.1.2 == Y.valid, yes=0, no=1))
```

The value of K with teh minimum CV error was found next and that was used to predict KNN min CV.

For 1SE, the largest the k within the 1 SE of minimum CV using:

```
mis.se <- sqrt(mis*(1-mis)/nrow(data.valid)) #SE of misclass rates
serule = max(which(mis<mis[mink]+mis.se[mink]))
```

where mink is the value of k were the cv error is minimum.

glmnet: The inbuilt cv function from the glmnet package was used (cv.glmnet). Lambda was then tuned using the returned model. Lambda min and lambda 1se were found using:

```
lambda.min = logit.cv$lambda.min
lambda.1se = logit.cv$lambda.1se
```

After getting these values, the model was model using the respective values of s as shown in question 1.

Classification Trees: The default tree was built using cp value of 0. The cptable was then used to get values of geometric mean of cp values at minimum error and one step up. This was used for the CV min tree and the SE values were calculated similarly as shown below.

```
cplow.min <- cpt[minrow,1]
cpup.min <- ifelse(minrow==1, yes=1, no=cpt[minrow-1,1])
cp.min <- sqrt(cplow.min*cpup.min)
cplow.1se <- cpt[se.row,1]
cpup.1se <- ifelse(se.row==1, yes=1, no=cpt[se.row-1,1])
cp.1se <- sqrt(cplow.1se*cpup.1se)
```

The pruned tree was then used to to predict values as shown above

Random Forest: Initially the random forest model was built using default values. To improve results, the mtry was then tested for 1:16 and node sizes for c(2,3,4,5,6,7,8). This list was compiled after testing for node sizes c(1,3,5,8,10,12) (boxplots were used to analyze this first run). Each combination was then fit 5 times in the origin 10 fold model. At every iteration of the 5 restatement the misclassification rate was saved and then the combination with the lowest rate (this was found by ordering the list in which all the rates were saved) was then fit in the original 10-fold cross validation.

SVM: Just like random forests, SVM was also initially fit to its default parameters. The model was then improved by evaluating gamma a cost at C=10^c(0:5) and sigma=10^(-c(5:0))

respectively. The grid was then used in a training model from the caret package as shown below.

```
tuned.nnet <- train(x=data.train[,-1], y=data.train$Y, method="svmRadial",
        preProcess=c("center","scale"), trace=FALSE,
        tuneGrid=parmgrid, trControl = trcon)
```

The internal method used here was repeated cross validation with 10 folds and 2 repeats. Boxplots were used to analyze the first run and finalize the above mentioned grid.

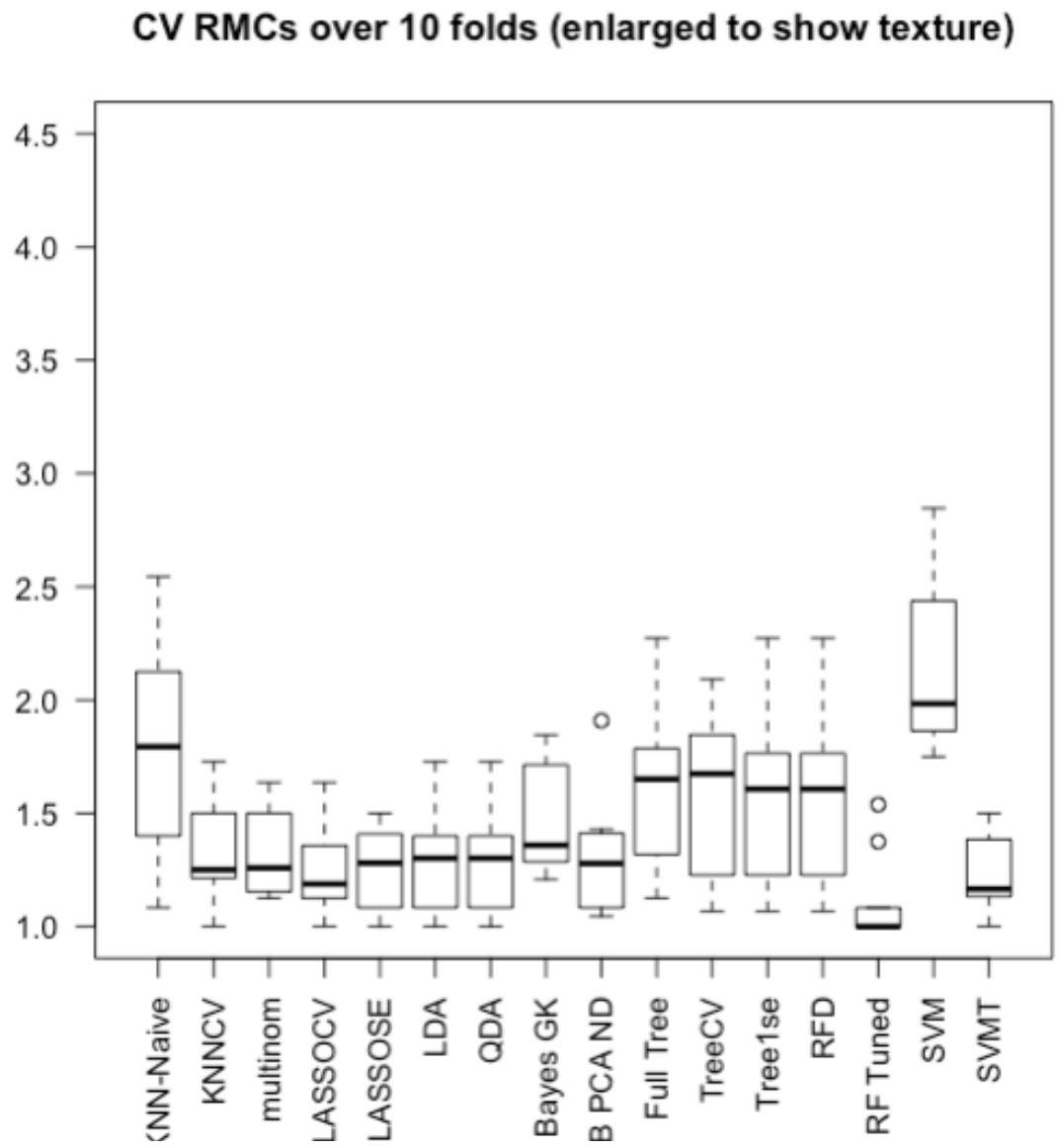
For each resample, the misclassification for each combination in the grid was saved in a matrix. Based on the mean classification rate from each fold and resample, the the combination with the minimum error rate was used to fit a model in the original 10-fold CV

NNet: Just like the above two methods, first a neural net was fitted using default parameters. Then an initial grid was formed using node sizes of c(1,3,5,7,10) and decay values of c(0.0001, 0.001,0.01, 0.1, 1). After analyzing this combinations using box plots and median misclassification rates the following grid was curated: node sizes of c(1,3,4,5,6,7,10) and decay values of c(0.0001, 0.001, 0.002, 0.003).

For neural nets, a manual cross validation with multiple restarts (a seed was set and the original training dat was further split into training and validation data) was used (10 fold and 2 repeats) and each net was fitted 10 times. (No caret package was used). Within each internal fold, the misclassification error (found as shown below) was saved in a matrix. The best combination was the one that resulted in the lowest mean misclassification rate. This model with those combinations was then fit in the original 10 fold CV using the original training and validation data.

```
nn <- nnet(y=y.1, x=x.1, size=s, decay=d, maxit=2000, softmax=TRUE, trace=FALSE)
Pi <- predict(nn, newdata=x.1, type ="class")
Mi <- mean(Pi != as.factor(data.train[foldsNN[,r]!=v,1]))
```

4. At the end of the original 10-fold cv a matrix of all the misclassification rates was collected and a box plot was created. This helped me finalize my final model. Neural nets is not in the figure since it was processed separately using the same folds and was later eliminated from analysis due to the immense tuning required and runtime issues. Functions that needed the variables to scaled was done as required (those functions were copied from lecture/tutorial videos).



5. The best models were the tuned random forest with mtry=3 and node size=4.

```
set.seed(46685327, kind="Mersenne-Twister")

library(FNN)
library(tidyverse)
library(pls)
library(nnet)
library(car)
library(glmnet)
library(MASS)
#library(mgcv)
library(klaR)
library(rpart)
library(randomForest)
library(e1071)
library(caret)
library(nnet)
library(foreach)
library(doSNOW)
library("doParallel")

data_raw <- read.csv("P2Data2020.csv")
test_raw <- read.csv("P2Data2020testX.csv")
data <- na.omit(data_raw[, c(1:17)])
test <- na.omit(test_raw[, c(1:16)])
data$Y <- as.factor(data$Y)

scale.1 <- function(x1,x2){
  for(col in 1:ncol(x1)){
    a <- mean(x2[,col])
    b <- sd(x2[,col])
    x1[,col] <- (x1[,col]-a)/b
  }
  x1
}
rescale <- function(x1,x2){
  for(col in 1:ncol(x1)){
    a <- min(x2[,col])
    b <- max(x2[,col])
    x1[,col] <- (x1[,col]-a)/(b-a)
  }
  x1
}

### Rescale x1 using the means and SDs of x2
scale.3 <- function(x1,x2){
  for(col in 1:ncol(x1)){
    a <- mean(x2[,col])
```

```
    b <- sd(x2[,col])
    x1[,col] <- (x1[,col]-a)/b
  }
  x1
}


get.folds = function(n, K) {
  ### Get the appropriate number of fold labels
  n.fold = ceiling(n / K) # Number of observations per fold (rounded up)
  fold.ids.raw = rep(1:K, times = n.fold) # Generate extra labels
  fold.ids = fold.ids.raw[1:n] # Keep only the correct number of labels

  ### Shuffle the fold labels
  folds.rand = fold.ids[sample.int(n)]

  return(folds.rand)
}

### Number of folds
K = 10
### Construct folds
n = nrow(data) # Sample size
folds = get.folds(n, K)
all.models = c("RF" )
all.MC = array(0, dim = c(K, length(all.models)))
colnames(all.MC) = all.models
for(i in 1:K){
  print(paste0(i, " CofC ", K))
  data.train = data[folds != i,]
  data.valid = data[folds == i,]
  n.train = nrow(data.train)
  X.train = data.train[, -1]
  X.valid = data.valid[, -1]

  ### Get response vectors
  Y.train = data.train$Y
  Y.valid = data.valid$Y

  wh.rf.tun <- randomForest(data=data.train, Y~., mtry=3, nodesize=4,
                            importance=TRUE, keep.forest=TRUE)
  pred.rf.train.tun <- predict(wh.rf.tun, newdata=data.train, type="response")
  pred.rf.test.tun <- predict(wh.rf.tun, newdata=data.valid, type="response")
  pred.rf.vtrain.tun <- predict(wh.rf.tun, newdata=data.train, type="vote")
  pred.rf.vtest.tun <- predict(wh.rf.tun, newdata=data.valid, type="vote")
  misclass.train.rf.tun <- mean(ifelse(pred.rf.train.tun == data.train$Y, yes=0, no=1))
  misclass.test.rf.tun <- mean(ifelse(pred.rf.test.tun == data.valid$Y, yes=0, no=1))
  all.MC[i, "RF"] <- misclass.test.rf.tun
}
```

```
train <- randomForest(data=data.train, Y~., mtry=3, nodesize=4,
                      importance=TRUE, keep.forest=TRUE)
validate <- predict(train, newdata=X.valid, type="response")
(misclass.test <- mean(ifelse(validate == Y.valid, yes=0, no=1)))
table(data.valid$Y,validate,  dnn=c("Observed","Predicted"))

pred.rf.test.tun <- predict(train, newdata=test, type="response")
pred.rf.test.tun
write.table(pred.rf.test.tun, "/Users/dollina/Desktop/Submit1.csv", sep = ",",row.names = FALSE,
col.names=FALSE)
```

6) Test error: 0.26

Confusion matrix:

```
> table(data.valid$Y,validate,  dnn=c("Observed","Predicted"))
        Predicted
Observed  A  B  C  D  E
       A 13  7  0  1  0
       B  4 14  0  6  0
       C  0  1 18  1  0
       D  0  4  0 16  0
       E  0  1  1  0 13
```