Assignment 3

Conor Kelly 18345361

Contents

Question 1

Data Preparation

To compare the different models I have divided the dataset into 75% training and validation with the remaining 25% to be used for testing. The data was randomly assigned to the two sets. I will use K-folds cross validation to compare the three competing models so I have left the training and validation as one set.

```
library(partykit)
library(rpart)
library(randomForest)
library(nnet)
library(adabag)
#importing data
data_mir_strawberry <- read.csv("C:/Users/conor/Downloads/data_mir_strawberry.csv")

#converting class labels to factors
data_mir_strawberry$class = factor(data_mir_strawberry$class)
#dividing the data into training validation and test sets.
set.seed(215454)
N<- nrow(data_mir_strawberry)

#3/4 of the data for training and validation and using remaining for testing will use k-fold cross
#validation on training and validation set for evaluating performance
train <- sample(1:N , 738)
test <- setdiff(1:N , train)
dat <- data_mir_strawberry[train, ]
dat_test <- data_mir_strawberry[test, ]
```

Support Vector Machine Hyperparameter Selection

For the support vector machine I will use Gaussian Radial Basis Function kernal. I will first tune the SVM and then use the optimal parameter for cross validation against the random forest and boosting classifiers. The tuning process will use a 5 fold cross validation procedure with each fold containing 147 observations.

```
#tuning the Support vector machine
library(kernlab)

X <- scale(dat[, 2:48])
Y <- dat[,1]

#values of Cost and sigma to test
C <-c(1,2,5,10,20)
sigma <-c(0.010,0.015,0.020,0.025,0.030)
grid <-expand.grid(C, sigma)
colnames(grid) <-c("C","sigma")

K <- 5
R <- 10
n_mod <-nrow(grid)
N_train <- nrow(dat)
out <- vector("list", R)

#function for defining classification accuracy
class_acc <-function(y, yhat) {
  tab <-table(y, yhat)
  return(sum(diag(tab))/sum(tab) )
}

for( r in 1:R ) {
  acc <-matrix(NA, K, n_mod)# accuracy of the classifiers in the K folds
  folds <-rep(1:K,ceiling(N_train/K) )
folds <-sample(folds)# random permute
folds <-folds[1:N_train]# ensure we got N_train data points
for( k in 1:K ) {
  train_fold <-which(folds!=k)
  validation <-setdiff(1:N_train, train_fold)# fit SVM on the training data and assess on the validatio
  for( j in 1:n_mod ) {
    fit <-ksvm(X[train_fold,], Y[train_fold],type ="C-svc",kernel ="rbfdot",C =grid$C[j],
               kpar =list(sigma   =grid$sigma[j]))
    pred <-predict(fit,newdata =X[validation,])
    acc[k,j] <-class_acc(pred, Y[validation])
    }
}
out[[r]] <- acc
}

avg_fold_acc <-t(sapply(out, colMeans) )
avg_acc <-colMeans(avg_fold_acc)# estimated mean accuracy
grid_acc <-cbind(grid, avg_acc)
```

The table below shows the average validation accuracy across all 5 folds and 10 replications for each value of Cost and σ.

```
##     C sigma   avg_acc
## 1   1 0.010 0.9528562
## 2   2 0.010 0.9617926
## 3   5 0.010 0.9695183
## 4  10 0.010 0.9737167
## 5  20 0.010 0.9743951
## 6   1 0.015 0.9581366
## 7   2 0.015 0.9659956
## 8   5 0.015 0.9715462
## 9  10 0.015 0.9729059
## 10 20 0.015 0.9722265
## 11  1 0.020 0.9604403
## 12  2 0.020 0.9677579
## 13  5 0.020 0.9715527
## 14 10 0.020 0.9718211
## 15 20 0.020 0.9719535
## 16  1 0.025 0.9617981
## 17  2 0.025 0.9676209
## 18  5 0.025 0.9712787
## 19 10 0.025 0.9714148
## 20 20 0.025 0.9699237
## 21  1 0.030 0.9632846
## 22  2 0.030 0.9685696
## 23  5 0.030 0.9714148
## 24 10 0.030 0.9708733
## 25 20 0.030 0.9697858
```

With the best combination of the variables from the set we considered being

```
##    C sigma   avg_acc
## 5 20  0.01 0.9743951
```

Based on this the optimal hyperparameters are C =20, $\sigma = 0.01$.However, the C obtained is the maximum value of the search grid and σ is the minimum value of it's vector. It could be the case that the true optimal C is greater than C = 20, and the optimal σ is less than $\sigma = 0.01$. I will run a similar search again but with the following values.Using the same folds generated in the previous test.

Running the same test for C = {20,25,30,35,40} σ = {0.005,0.0075,0.010}

```
##     C  sigma   avg_acc
## 1  20 0.0025 0.9661427
## 2  25 0.0025 0.9701967
## 3  30 0.0025 0.9702059
## 4  35 0.0025 0.9715573
## 5  40 0.0025 0.9702059
## 6  20 0.0050 0.9742600
## 7  25 0.0050 0.9756113
## 8  30 0.0050 0.9742600
## 9  35 0.0050 0.9756113
## 10 40 0.0050 0.9756113
## 11 20 0.0075 0.9756113
## 12 25 0.0075 0.9756113
## 13 30 0.0075 0.9756113
## 14 35 0.0075 0.9756113
```

```
## 15 40 0.0075 0.9742600
## 16 20 0.0100 0.9729086
## 17 25 0.0100 0.9729086
## 18 30 0.0100 0.9701967
## 19 35 0.0100 0.9701967
## 20 40 0.0100 0.9715481
```

```
best <-which.max(grid_acc$avg_acc)
grid_acc[best,]
```

```
##    C sigma   avg_acc
## 7 25 0.005 0.9756113
```

So based on the second search C = 25 , $\sigma = 0.005$ is optimal. There does appear to be significant differences between accuracy for the σ values but the difference across the values of C are quite small and the optimal value of C differs from test to test. The value of σ does seem significant and $\sigma = 0.005$ is consistently the best.

```
n <-nrow(grid_acc)
summary(grid_acc[,3])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9661  0.9702  0.9736  0.9729  0.9756  0.9756
```

The values are all extremely close together, there is only a difference of 0.0123 between the maximum and minimum values obtained in the search. So whichever value of C we choose from the second search will probably have little impact on the overall outcome. So we will leave C = 20 so as not to over complicate the model, $\sigma = 0.005$ since it is marginally better across all folds.

cross validation

The boosting and random forest classifiers don't require the same level of tuning that the SVM. The only input parameter of interest is the number of trees for the forest or replications for boosting, we will leave these as their defaults. We will use a 5-fold cross validation procedure using the training and validation set to compare the models. The code below outputs a matrix of the classification accuracy for each model and fold. The model with the highest validation accuracy will be used as the best model.

```
K<- 5
N <- nrow(dat)

folds <- rep( 1:K, ceiling(N/K) )
folds <- sample(folds)              # random permute
folds <- folds[1:N]                 # ensure we got N data points


out<- matrix(nrow = K , ncol = 3)
for ( k in 1:K ) {
  train <- which(folds != k)
  test <- setdiff(1:N, train)
  X <- scale(dat[train,-1])
  Y <- dat[train,1]
```

```
  # fit Random forest
  forest <- randomForest(class ~ . , data = dat , subset = train, importance = TRUE  )

  #Support Vector Machine
  SVM <- ksvm(X, Y  , type ="C-svc",kernel ="rbfdot",C = 30,
              kpar = list(0.005) )
  boost <- boosting(class ~ . , data = dat, subset = train , coeflearn = "Breiman")

  # classify the test data observations for random forest
  Pred_test_forest <- predict(forest, type = "class", newdata = dat[test,])
  tabTest_forest <- table(dat$class[test], Pred_test_forest)
  out[k,1] <- sum(diag(tabTest_forest))/sum(tabTest_forest)

  #classify the test data for support vector machine
  predTest_svm <- predict(SVM, newdata = scale(dat[test,-1]))
  tabTest_svm <- table(dat$class[test], predTest_svm)
  out[k,2] <- sum(diag(tabTest_svm))/sum(tabTest_svm)

  #classify test data for boost
  predTest_boost <- predict(boost, newdata = dat[test,-1])
  tabTest_boost <- table(dat$class[test], predTest_boost$class)
  out[k,3] <- sum(diag(tabTest_boost))/sum(tabTest_boost)

  print(k)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
colnames(out) <- c("Random forest", "SVM", "Boosting")
out
```

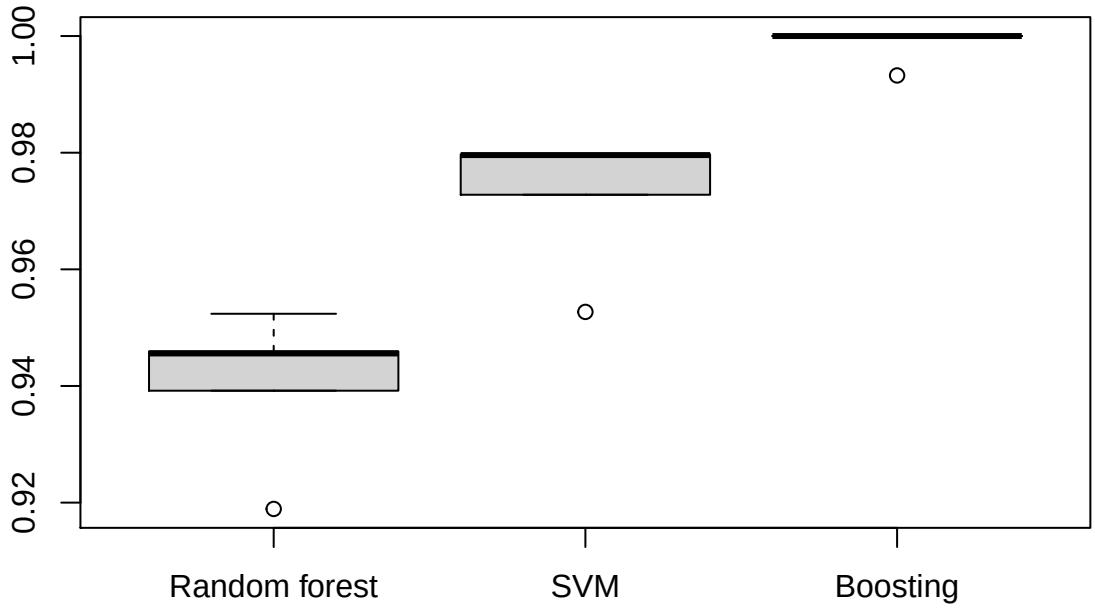
```
##      Random forest       SVM  Boosting
## [1,]     0.9455782 0.9727891 1.0000000
## [2,]     0.9391892 0.9797297 1.0000000
## [3,]     0.9189189 0.9527027 1.0000000
## [4,]     0.9459459 0.9797297 0.9932432
## [5,]     0.9523810 0.9795918 1.0000000
```

```
colMeans(out)
```

```
## Random forest           SVM      Boosting
##     0.9404026     0.9729086     0.9986486
```

The boosting classifier performs best across all 5 folds of the cross validation and has a higher mean accuracy shown in the table above. Based on the predictive performance in the validation we will use boosting as the model to take forward to the test data. Below is a boxplot of the validation accuracys from the 5 folds.

```
boxplot(out)
```



Question 2

To evaluate the test performance I will train the Boosting classifier on the whole training and validation set then test it's predictive performance on the unseen test data

```
boost <-  boosting(class ~ . , data = dat , coeflearn = "Breiman" )
predTest_boost <- predict(boost, newdata = dat_test)
tabTest_boost <- table(dat_test$class, predTest_boost$class, dnn = c("Observed", "Predicted"))
Test_Accuracy <-sum(diag(tabTest_boost))/sum(tabTest_boost)
tabTest_boost
```

```
##                Predicted
## Observed        nonstrawberry strawberry
##   nonstrawberry           134          4
##   strawberry                3        104
```

```
Test_Accuracy
```

```
## [1] 0.9714286
```

```
predict(boost, newdata = dat_test)$error
```

```
## [1] 0.02857143
```

The output above shows a table comparing the true class values against the predicted value, the test accuracy and the test error (1-test accuracy). With an accuracy of 97.1% the model correctly classified 239 observations. The model had 3 false negatives and 4 false positives. Below is the sensitivity followed by specificity of the model on the test data.

```
sensitivity <-  tabTest_boost[2,2]/sum(tabTest_boost[2 , 1:2])
specificity_ <-  tabTest_boost[1,1]/sum(tabTest_boost[1, 1:2])
sensitivity
```

```
## [1] 0.9719626
```

```
specificity_
```

```
## [1] 0.9710145
```

Question 3

For the application in the real world we would like the sensitivity to be higher. The performance of the model in this light depends on how much worse we consider a false positive to be. The current model correctly classifies 134/138 of the non strawberry samples or 97.8%. The sensitivity and the specificity are both very close in value, this is because the boosting method does not distinguish between false positives and false negatives, only classified and misclassified. Maybe changing the way in which the weights are adjusted to give more importance to false positives could be implemented for boosting at the expense of more false negatives.