PRACTICAL 10

Aim:

The aim of this practical is to investigate the effect of overfitting in trees and *k*-nearest neighbours classifiers. We will also briefly investigate the use of a fitting graph to find a suitable value for *k* in *k*-nearest neighbours.

Before you start

Load the `tree, caret` and `ggplot2` packages.

Instructions:

This document contains discussions of some basic concepts, and instructions which you should execute in RStudio. **Text in bold** indicate actions that are to be taken within RStudio. `Text highlighted in grey` shows the code you should enter in RStudio.

Preliminaries

1. We will once again use the `churn.csv` dataset, which was also used in Practical 5 and Tutorial 9.
Read about the background of this dataset in the Practical 5 instruction sheet, which also refers to the relevant pages in the textbook.

2. **Import this dataset into RStudio and store it in an object named `churn`.**

```
churn <- read.csv("churn.csv", header = TRUE,
stringsAsFactors = TRUE)
```

Model fitting

3. We will first fit a basic tree, without changing any of the default parameters.

3.1 **Fit a basic decision tree, using the `tree()` function from the `tree` package, and plot the tree. Store the model in an object named `tree.basic`.**

```
tree.basic <- tree(LEAVE~., data = churn)
plot(tree.basic)
text(tree.basic, pretty = 0)
```

3.2 **View the summary output for the `tree.basic` model object, and take note of the misclassification error rate.**

```
summary(tree.basic)
```

4. We now explicitly specify some of the parameters for the `tree()` function, to force the construction of a more complex tree. You do not have to be concerned about what any of the arguments mean; you should simply take note of the resultant tree.

```
tree.over <- tree(LEAVE~., data = churn, control =
tree.control(nobs = nrow(churn), mindev = 0.00015, mincut
= 1))
plot(tree.over)
text(tree.over, pretty = 0)
summary(tree.over)
```

5. For the trees fit in Step 3 and 4 above, the full dataset was used. We should instead use a training dataset to fit the model and then evaluate performance on a test set.
Randomly select 80% of the data for a training dataset and designate the remaining 20% of the data as test set. Set a seed equal to 9 before starting, to ensure reproducibility of your results.

```
set.seed(9)
churn_index <- sample(1:nrow(churn), size = nrow(churn) *
0.8, replace = FALSE)
churn_train <- churn[churn_index,]
churn_test <- churn[-churn_index,]
```

6. **Fit a complex tree on the training data. Use the exact code below, and consider the summary output of the tree to see what the misclassification error rate on the training data is.**

```
tree.train <- tree(LEAVE~., data = churn_train, control =
tree.control(nobs = nrow(churn_train), mindev = 0.0002,
mincut = 1))
summary(tree.train)
```

7. **Now evaluate the performance of the tree fit in Step 6 on the test data.**

```
churn_pred <- predict(tree.train, churn_test[,-11], type =
"class")
table(predicted = churn_pred, actual = churn_test$LEAVE)
```

You should note that the misclassification error rate is higher on the test data than on the training data.

8. **We now fit a simpler tree. Use the exact code below, and consider the summary output of the tree to see what the misclassification error rate on the training data is.**

```
final.tree <- tree(LEAVE~., data = churn_train, control =
tree.control(nobs  =  nrow(churn_train),  mindev  =  0.001,
minsize = 100))
summary(final.tree)
```

9. Now evaluate the performance of the tree fit in Step 8 on the test data.

```
final_pred <- predict(final.tree, churn_test[,-11], type =
"class")
table(predicted = final_pred, actual = churn_test$LEAVE)
```

You should now note that the misclassification error rate for the training and test data is more similar.

Model complexity and kNN

In prior discussions about k-nearest neighbours, we touched on the risk of overfitting when using a value of $k = 1$. Choosing higher values of k might lead to models that are more robust to noise, but might lead to a drop in performance.

10. Train a 1NN model on the training dataset, and obtain the training as well as test accuracy.

```
onenn <- knn3(LEAVE~., data = churn_train, k = 1)
trainpred <- predict(onenn, churn_train[,-11], type =
"class")
confusionMatrix(trainpred, churn_train$LEAVE)
testpred <- predict(onenn, churn_test[,-11], type =
"class")
confusionMatrix(testpred, churn_test$LEAVE)
```

11. **Train a 300NN model on the training dataset, and obtain the training as well as test accuracy.**

```
manynn <- knn3(LEAVE~., data = churn_train, k = 300)
trainpred <- predict(manynn, churn_train[,-11], type =
"class")
confusionMatrix(trainpred, churn_train$LEAVE)
testpred <- predict(manynn, churn_test[,-11], type =
"class")
confusionMatrix(testpred, churn_test$LEAVE)
```

12. Recall from the theoretical lecture that a *fitting graph* can be used to visualise the difference between accuracy on the training data vs. the test data.

Construct a fitting graph to help you find a suitable value of *k* to use.

The construction of fitting graphs in R is beyond the scope of this course; you should only be able to interpret fitting graphs. However, you should be able to follow the code below fairly easily, as it just wraps what was done in Steps 10 and 11 above in a "for" loop.

Note that this code will take a while to run, as 300 different kNN models are fit and then used for prediction!

Two new vectors, `trainacc` and `testacc`, are created. After each model is fit, the training and test accuracy are saved in the corresponding position of each vector. For instance, the training accuracy for the 3NN model is saved in the 3rd position of the `trainacc` vector, and the test accuracy for the 3NN model is saved in the 3rd position of the `testacc` vector.

After all 300 models have been fit, a plot is created, showing both the training and test accuracy for all values of *k*. It is clear to see that initially the training accuracy drops steeply as *k* increases, but then stabilises, while the test accuracy increases fairly steeply as *k* increases initially but then also stabilises.

```
trainacc = rep(0, 300)
testacc = rep(0, 300)
for (i in 1:300) {
     fit <- knn3(LEAVE~., data = churn_train, k = i)
     trainpred <- predict(fit, churn_train[,-11], type =
     "class")
     conf <- confusionMatrix(trainpred, churn_train$LEAVE)
     trainacc[i] <- (conf$table[1,1] + conf$table[2,2]) /
     nrow(churn_train)
     testpred  <-  predict(fit,  churn_test[,-11],  type  =
     "class")
     conft <- confusionMatrix(testpred, churn_test$LEAVE)
     testacc[i] <- (conft$table[1,1] + conft$table[2,2]) /
     nrow(churn_test)
}
k <- c(1:300)
plotdata <- data.frame(cbind(k, trainacc, testacc))
ggplot() + geom_line(data = plotdata, mapping = aes(x = k,
y = trainacc), color = "blue") + geom_line(data = plotdata,
mapping  =  aes(x  =  k,  y  =  testacc),  color  =  "red")  +
labs(title = "Fitting graph for the customer churn (kNN)
model", subtitle = “Training accuracy in blue and Test
accuracy  in  red”,  x  =  "k  (nr.  of  neighbours)  ",  y  =
"Accuracy")
```