Exercise: the `caret` package}

In this R exercise, you will know:

- How to use the `caret` package

Don't forget to change your working directory!

1 The caret package

The `caret` package provides easy ways to use/train a wide range of machine learning models, tune parameters, calculate model performance measures and draw nice plots. You can read the following nice page of `caret` for more information: https://topepo.github.io/caret/index.html. The manual of this package: https://cran.r-project.org/web/packages/caret/caret.pdf. Try to explore different functions of `caret` by yourselves.

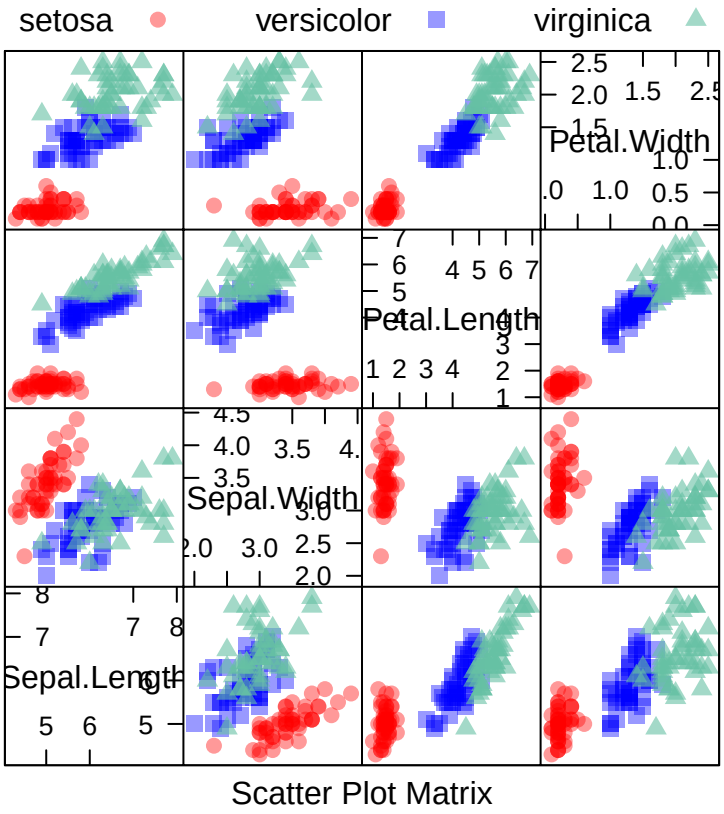
```
install.packages("caret")
library(caret)
```

2 Nice plots by caret

This example is from https://topepo.github.io/caret/index.html. The `featurePlot` function in `caret` can be used to get nice pairs plot, to visualise the relationship between variables.

```
install.packages("AppliedPredictiveModeling")
library(AppliedPredictiveModeling)
```

```
transparentTheme(trans = .4)
featurePlot(x = iris[, 1:4],
            y = iris$Species,
            plot = "pairs",
            ## Add a key at the top
            auto.key = list(columns = 3))
```



Scatter Plot Matrix

3 Use kNN in caret

Get training/test sets by the `createDataPartition` function.

```
set.seed(215)
train.indx=createDataPartition(iris$Species,p=0.5,list=FALSE,times = 1)
train=iris[train.indx,-5]; train.label=iris[train.indx,5]
test=iris[-train.indx,-5]; test.label=iris[-train.indx,5]
```

Set up things to control the training process.

```
#### set up train control
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated ten times
                           repeats = 5)
```

Train the kNN model by tuning k with repeated 10-fold cross-validation:

```
#install.packages("e1071")
#library(e1071)
#### training process
set.seed(596)
knnFit1=train(train,train.label,
              method="knn",
              trControl=fitControl,
              metric="Accuracy",
```

```
                tuneLength=10)
```

Use help or read the manual to understand the control commands.

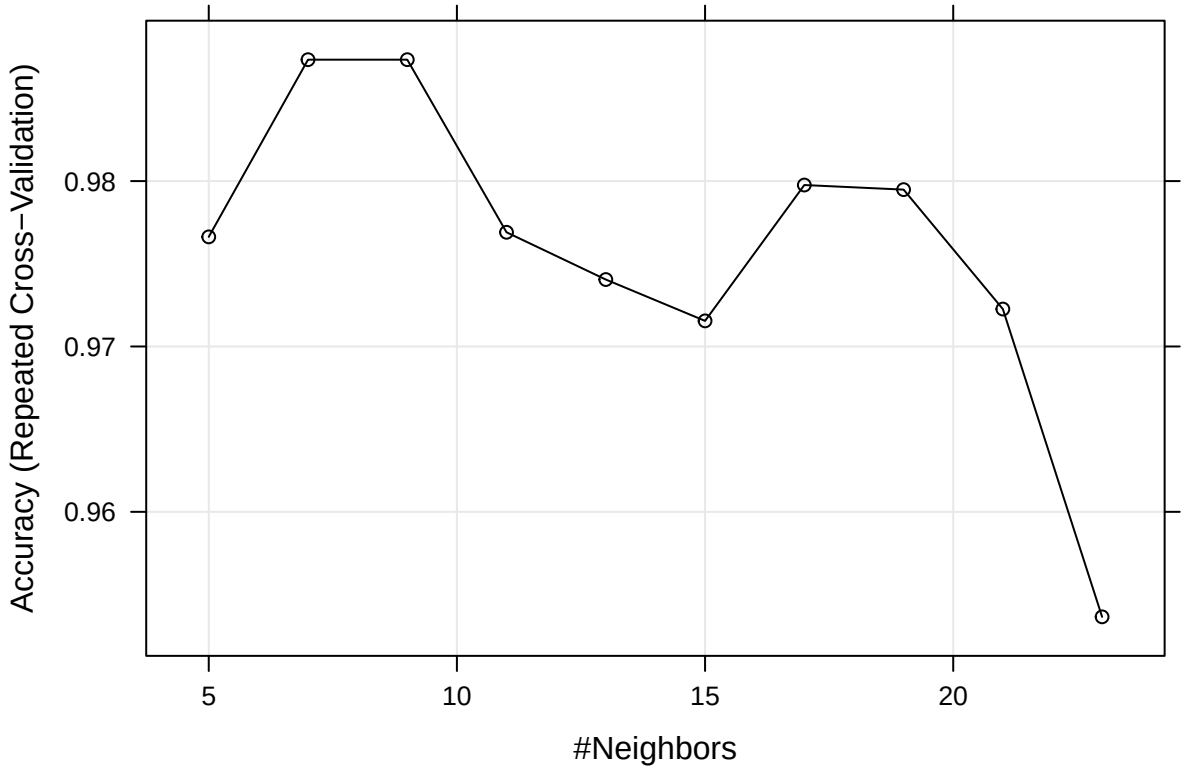
Type `knnFit1` to see the output:

```
knnFit1
```

```
## k-Nearest Neighbors
##
## 75 samples
##  4 predictor
##  3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 67, 68, 66, 69, 67, 67, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.9766270  0.9641089
##    7  0.9873413  0.9806723
##    9  0.9873413  0.9806723
##   11  0.9769048  0.9646780
##   13  0.9740476  0.9601619
##   15  0.9715476  0.9565339
##   17  0.9797619  0.9690134
##   19  0.9794841  0.9686258
##   21  0.9722619  0.9576734
##   23  0.9536508  0.9296624
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

We can also have a plot to show the parameter tuning process in the training data:

```
plot(knnFit1)
```



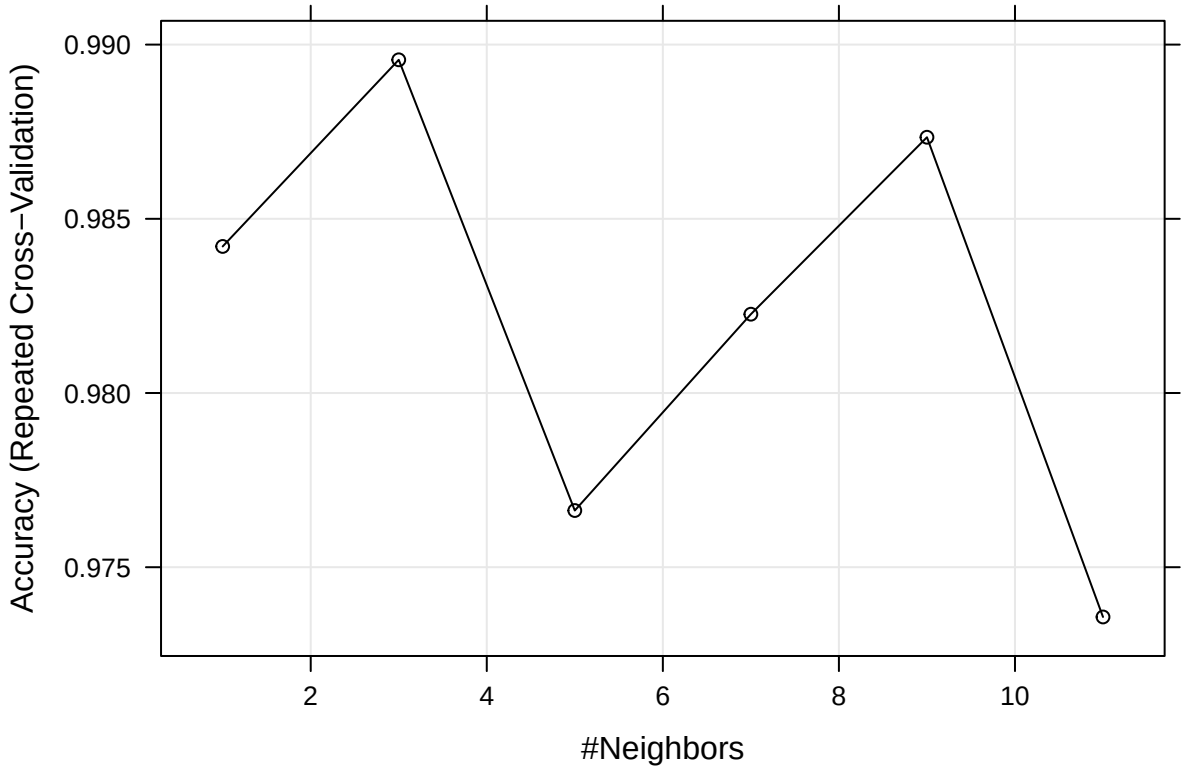
It seems that the default setting of k starts from 5. We can also create a grid containing our own selected parameters.

```
#### specify a tuning grid####################
kNNGrid=expand.grid(k=c(1,3,5,7,9,11))
#### training process
set.seed(596)
knnFit2=train(train,train.label, method = "knn", trControl = fitControl,
              metric = "Accuracy", tuneGrid=kNNGrid)
knnFit2
```

```
## k-Nearest Neighbors
##
## 75 samples
##  4 predictor
##  3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 67, 68, 66, 69, 67, 67, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9842063  0.9755871
##    3  0.9895635  0.9840057
##    5  0.9766270  0.9641089
##    7  0.9822619  0.9728229
##    9  0.9873413  0.9806723
##   11  0.9735714  0.9596780
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

```
plot(knnFit2)
```



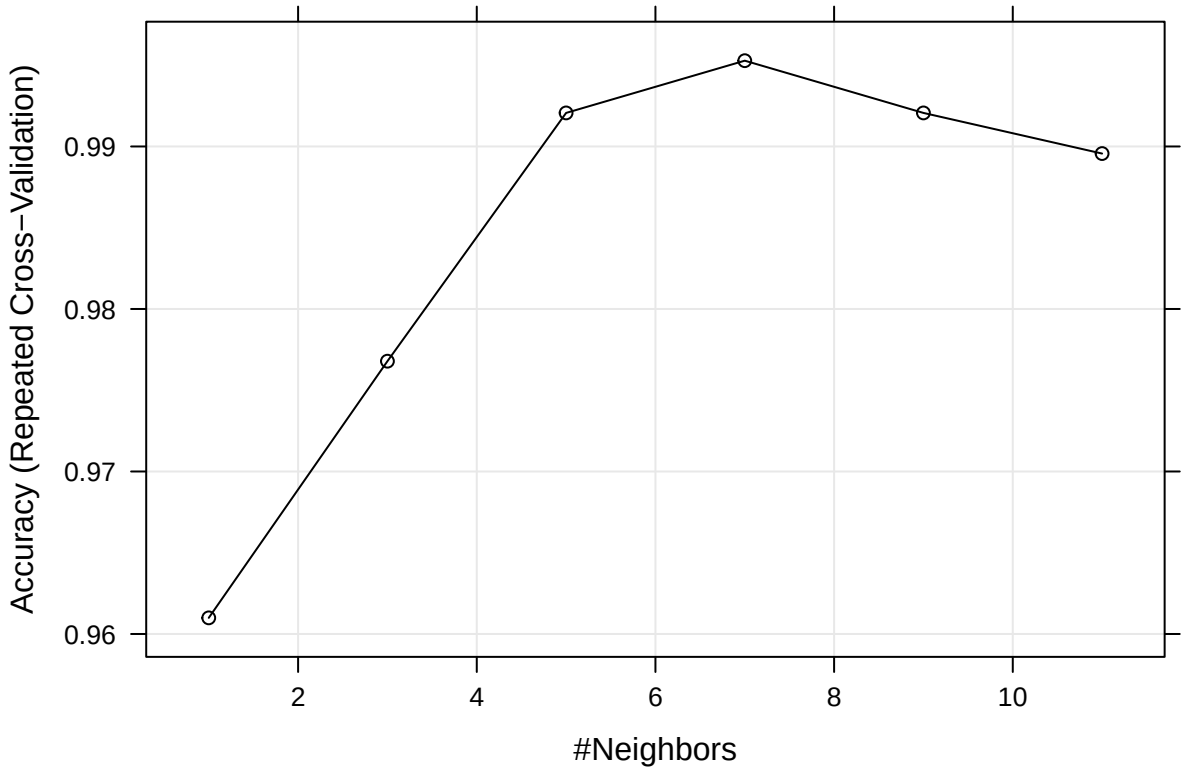
We can also do preprocessing on the data.

```
set.seed(596)
knnFit3=train(train,train.label, method = "knn", trControl = fitControl,
              metric = "Accuracy", preProcess = c("center","scale"),
              tuneGrid=kNNGrid)
knnFit3
```

```
## k-Nearest Neighbors
##
## 75 samples
##  4 predictor
##  3 classes: 'setosa', 'versicolor', 'virginica'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 67, 68, 66, 69, 67, 67, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9609921  0.9404439
##    3  0.9767857  0.9647156
##    5  0.9920635  0.9879081
##    7  0.9952778  0.9927642
##    9  0.9920635  0.9879081
##   11  0.9895635  0.9840057
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
plot(knnFit3)
```



We can use the usual way to do prediction for the test set:

```
#### test process
pred1=predict(knnFit1,test)
acc1=mean(pred1==test.label)
acc1
```

```
## [1] 0.96
```

```
pred2=predict(knnFit2,test)
acc2=mean(pred2==test.label)
acc2
```

```
## [1] 0.9466667
```

```
pred3=predict(knnFit3,test)
acc3=mean(pred3==test.label)
acc3
```

```
## [1] 0.9466667
```

There are a lot of things to explore in this package. We'll talk about some during lectures, but it'll be better if you could discover new things by yourselves!