SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 3: k nearest neighbours (Part 2)

DR. Rui Zhu, Dr. Feng Zhou

2020-01-21

In R exercises 3, you will know[1]:

- How to calculate the classification accuracy/error rate on the test set
- How to use LOOCV on the training set to choose k
- How to use LOOCV to obtain an average performance measure
- How to use kNN for regression task
- More real data examples

Don't forget to change your working directory!

1 Calculate classification accuracy

Get the training and test sets and use `knn` to obtain predictions of the test set.

```
library(class)
# get indexes for training data
n=25 # the number of training data in each class
NN=dim(iris3)[1]# the total number of observations in each class
set.seed(983)
index_s=sample(1:NN,n)
index_c=sample(1:NN,n)
index_v=sample(1:NN,n)
# get training and test set
train_rand = rbind(iris3[index_s,,1], iris3[index_c,,2], iris3[index_v,,3])
test_rand = rbind(iris3[-index_s,,1], iris3[-index_c,,2], iris3[-index_v,,3])
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# classification using knn, with k=5
kk=5 # number of nearest neighbours
set.seed(275)
knn_pred=knn(train=train_rand,test=test_rand,cl=train_label, k=kk, prob=FALSE)
```

How do we calculate the classification accuracy or error rate of applying the `knn` model on the test set?

Method 1

```
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
table(knn_pred,cl)
```

[1]Please note: This Part 2 may require some codes/outputs/packages/datasets from Part 1 in the previous lab.

```
##          cl
## knn_pred  c  s  v
##        c 25  0  3
##        s  0 25  0
##        v  0  0 22
```

```
(22+25+25)/75
```

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

```
sum(diag(table(knn_pred,cl)))/nrow(test)
```

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

We usually prefer the last line than the second last line. This is because if we change the training and test sets, the table may be changed. In this case, we can still use the last line to calculate the accuracy while we have to change the numbers in the second last line to get the correct answer.

Method 2

```
test_label_true=factor(c(rep("s",25), rep("c",25), rep("v",25)))
sum(knn_pred==test_label_true)/length(test_label_true)
```

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

```
mean(knn_pred==test_label_true)
```

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

Make sure you understand all the codes.

You can choose the one you like or think about other ways to calculate the classification accuracy. Try to calculate the error rate by yourself.

Play with the above codes, to see how change of training/test data, change of variables or change of k will affect the classification accuracy.

Exercise: Calculate the classification accuracies of the predictions in the Standardise data section in the previous exercise.

2 LOOCV to choose the value of k using the training data

Suppose we want to determine which k is the best for classification from five values $1, 3, 5, 7, 9$. Then we need to do LOOCV five times on the training data: calculate the mean accuracy of LOOCV using each k value and then choose the k value with the largest accuracy. We then use this k value in the kNN model to classify the test samples.

To effectively do this, we can use the `for` loop. Here is an example to calculate $1 + 2 + \ldots + 10$:

```
Sum=0
for(ii in 1:10){
  Sum=Sum+ii
}
Sum
```

```
## [1] 55
```

We first initialise `Sum` variable as 0, to store the sum value in each loop. `ii` is the index for the loops and is from 1 to 10. The values of `ii` are indicated in `( )` using `in`. The steps of each loop is written in `{ }`. `ii` starts from 1, so the first loop calculates $0 + 1$ and give this value to `Sum`. In the next loop, `ii` becomes 2,

and `Sum` becomes $1+2$ (think about why!). `ii` ends in 10, so the last loop is $45+10$. After `ii` reaches 10, the loop ends and `Sum` is 55.

`kNN.cv` in the `class` package provides the leave-one-out cross-validation (LOOCV) evaluation. Type the following code to see the input options and the outputs.

```
?knn.cv
```

Below are the codes for our task:

```
# get indexes for training data
n=25 # the number of training data in each class
NN=dim(iris3)[1]# the total number of observations in each class
set.seed(983)
index_s=sample(1:NN,n)
index_c=sample(1:NN,n)
index_v=sample(1:NN,n)
# get training and test set
train_rand = rbind(iris3[index_s,,1], iris3[index_c,,2], iris3[index_v,,3])
test_rand = rbind(iris3[-index_s,,1], iris3[-index_c,,2], iris3[-index_v,,3])
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# evaluate k=1,3,5,7,9 on the training data using LOOCV
kk=c(1,3,5,7,9)
# initialise acc to store the mean accuracy of LOOCV for
# five $k$
acc=vector("numeric",length=length(kk))
set.seed(649)
for(ii in 1:length(kk)){
  knn_pred=knn.cv(train=train_rand, cl=train_label, k = kk[ii])
  acc[ii]=mean(knn_pred==train_label)
}
# get the first k that has the largest accuracy
# here we can also randomly choose any k that has the largest
# accuracy if we have several largest values
acc
```

```
## [1] 0.9466667 0.9733333 0.9733333 0.9733333 0.9466667
```

```
kk_LOOCV=kk[which.max(acc)]
# use the k tuned by the training set to classify the test set
set.seed(275)
knn_pred_test=knn(train=train_rand,test=test_rand,cl=train_label, k=kk_LOOCV, prob=FALSE)
acc_test=mean(knn_pred_test==test_label_true)
acc_test
```

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

Play with the codes to make sure you understand them!

3 LOOCV to evaluate the performance of kNN with a specific k value

Use LOOCV to evaluate the classification performance of a kNN model with $k = 3$ on the iris data:

```
# LOOCV on the whole dataset
train=rbind(iris3[,,1], iris3[,,2], iris3[,,3])
cl=factor(c(rep("s",50), rep("c",50), rep("v",50)))
# evaluate kNN model with k=3
kk=3
set.seed(382)
knn.pred=knn.cv(train, cl, k = kk)
acc=mean(knn.pred==cl)
acc
```

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

Exercise: Can we get an average classification accuracy of using kNN on the iris data, with k tuned by LOOCV? Hint: combine sections 2 and 3.

Any easier ways to do this? The `caret` package.

4 kNN for regression

We use the Advertising data to predict the sales of a product.

Load data from .csv file and create training and test set.

```
Advertising=read.csv("Advertising.csv",header=TRUE)
set.seed(1)
n=100
ind.train=sample(nrow(Advertising),n)
train.feature=Advertising[ind.train,2:4]
train.y=Advertising[ind.train,5]
test.feature=Advertising[-ind.train,2:4]
test.y=Advertising[-ind.train,5]
```

We can use **knn.reg** function from **FNN** package.

```
##### knn regression
library(FNN)
```

```
##
## Attaching package: 'FNN'
```

```
## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
## k=3
knn3=knn.reg(train.feature, test.feature, train.y, k = 3)
knn3$pred
```

```
##   [1] 19.566667 10.133333 17.700000  7.166667 13.300000  4.733333 13.666667
##   [8]  8.133333 13.833333 11.400000 13.466667 18.500000 14.566667  9.200000
##  [15] 13.100000 20.300000  9.866667 24.066667 10.500000 20.433333 17.800000
##  [22] 13.666667 10.133333 14.266667  9.866667 13.666667 10.233333 18.000000
##  [29] 20.933333  6.900000 16.766667 14.566667  9.700000 19.933333 19.100000
##  [36]  9.066667  7.166667  6.266667 14.133333  3.500000 11.266667 11.400000
##  [43] 18.433333 16.166667 12.300000  6.266667 18.400000 14.566667 15.933333
##  [50] 25.633333 17.566667 14.266667 15.933333  6.266667 20.433333 15.166667
##  [57] 17.900000  6.266667  6.900000 19.566667  9.800000 20.333333  8.133333
```

```
##  [64] 10.900000  9.300000 10.666667 18.500000 21.933333 11.566667 12.300000
##  [71] 14.333333 10.133333 13.100000 16.266667 18.200000 13.533333 11.700000
##  [78]  6.900000 12.900000 14.933333 16.266667 14.966667 11.733333 17.566667
##  [85]  4.966667  8.133333 16.300000 14.366667 12.000000 20.300000 12.600000
##  [92] 12.100000 16.666667 16.166667  6.266667  9.700000 15.966667  9.333333
##  [99] 15.166667 14.333333
```

```
sum((knn3$pred-test.y)^2)/100 ## mean square error
```

```
## [1] 2.835189
```

We can also use the **caret** package.

```
##### knn regression in caret
library(caret)
```

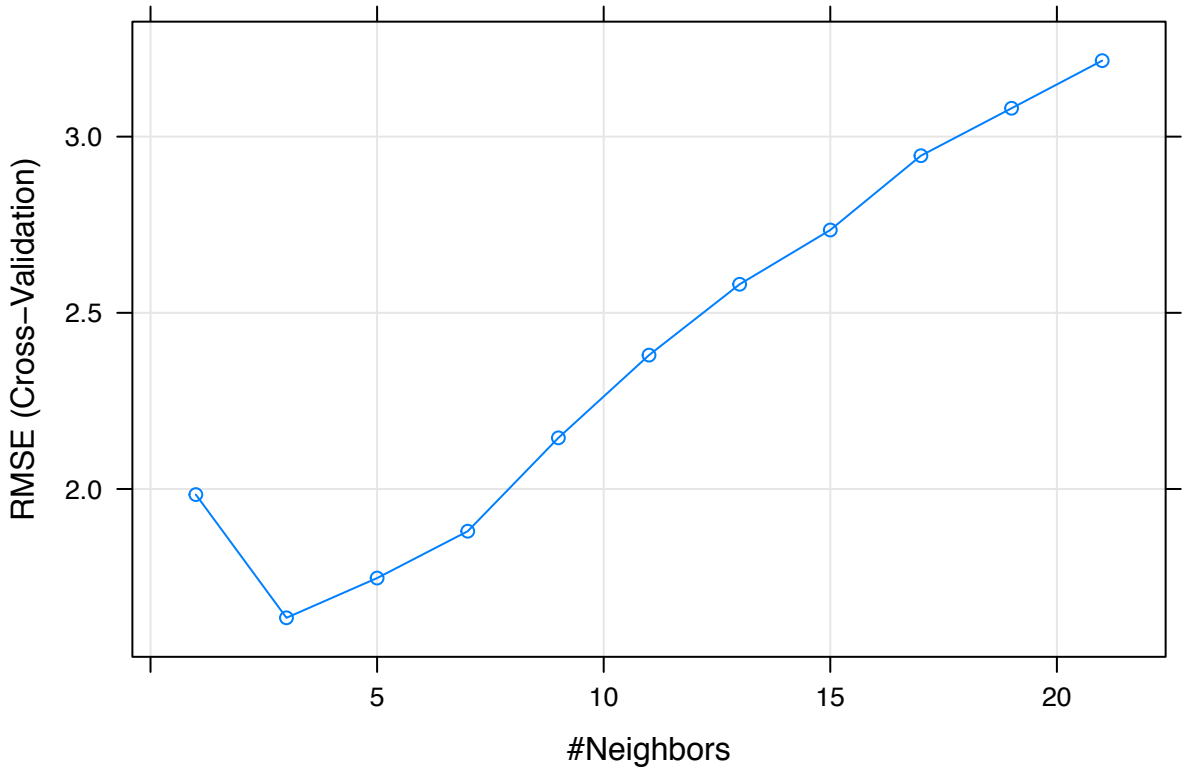
```
## Loading required package: lattice
```

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

```
set.seed(1)
knnreg.caret = train(
   train.feature,train.y,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5),
  tuneGrid = expand.grid(k = seq(1, 21, by = 2))
)
knnreg.caret$modelType
```

```
## [1] "Regression"
```

```
plot(knnreg.caret)
```



```
knnreg.caret$finalModel
```

```
## 3-nearest neighbor regression model
```

```
sum((predict(knnreg.caret,test.feature)-test.y)^2)/100 ## mean square error
```

```
## [1] 2.835189
```

References: Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R.* 2013.

More real data examples

German Credit Data

These data have two classes for the credit worthiness: good or bad. There are predictors related to attributes, such as: checking account status, duration, credit history, purpose of the loan, amount of the loan, savings accounts or bonds, employment duration, Installment rate in percentage of disposable income, personal information, other debtors/guarantors, residence duration, property, age, other installment plans, housing, number of existing credits, job information, Number of people being liable to provide maintenance for, telephone, and foreign worker status.

This dataset is also included in the `caret` package.

```
library(caret)
```

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

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

```
#load german credit data from caret package
data(GermanCredit)
# classify two status: good or bad
## Show the first 10 columns
str(GermanCredit[, 1:10])
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                  : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration       : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                     : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits   : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone               : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                   : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

```
## Delete two variables where all values are the same for both classes
GermanCredit[,c("Purpose.Vacation","Personal.Female.Single")] <- list(NULL)
```

Now we divide the dataset to training and test sets.

```
# create training and test sets
set.seed(12)
trainIndex = createDataPartition(GermanCredit$Class, p = 0.7, list = FALSE, times = 1)
train.feature=GermanCredit[trainIndex,-10] # training features
train.label=GermanCredit$Class[trainIndex] # training labels
test.feature=GermanCredit[-trainIndex,-10] # test features
test.label=GermanCredit$Class[-trainIndex] # test labels
```

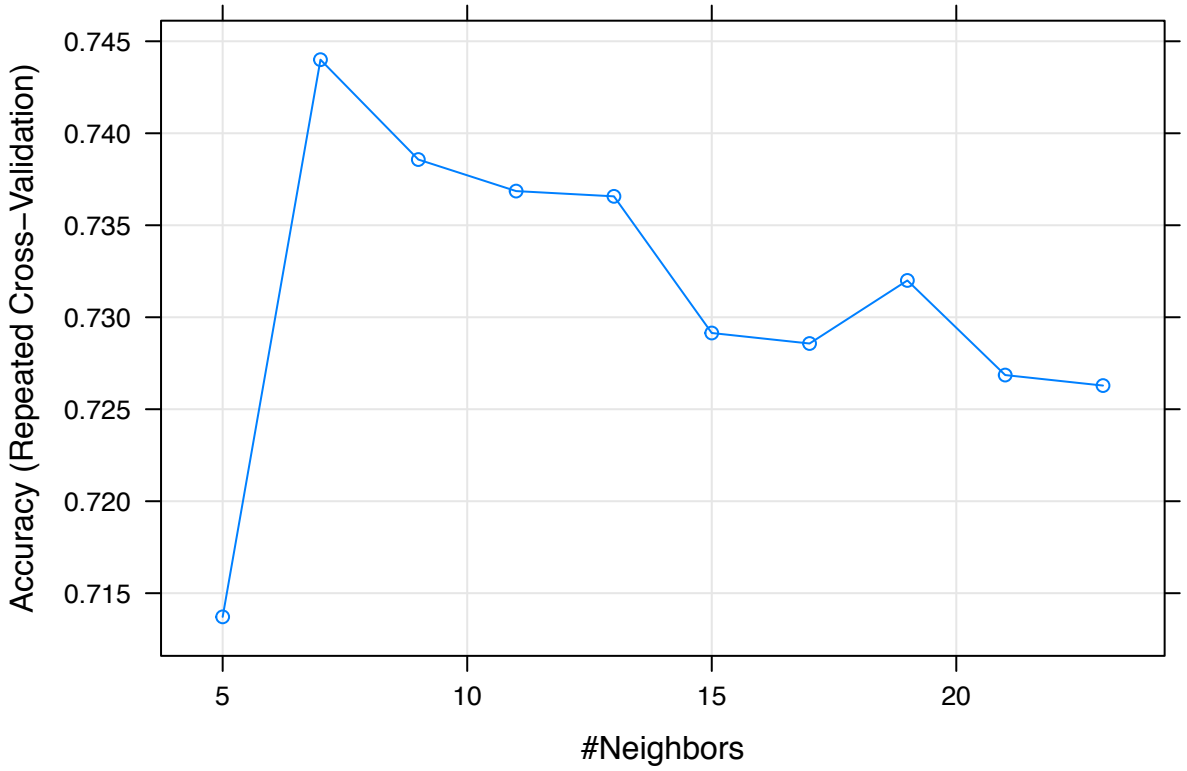
Train the kNN classifier and tune the value of `k` by 10-fold CV.

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

```
  repeats = 5)
#### training process
set.seed(5)
knnFit=train(train.feature,train.label, method = "knn",
             trControl = fitControl,
             metric = "Accuracy",
             preProcess = c("center","scale"),
             tuneLength=10)
knnFit
```

```
## k-Nearest Neighbors
##
## 700 samples
##  59 predictor
##   2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 630, 630, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7137143  0.2393261
##    7  0.7440000  0.3036671
##    9  0.7385714  0.2722017
##   11  0.7368571  0.2558584
##   13  0.7365714  0.2392079
##   15  0.7291429  0.2097804
##   17  0.7285714  0.1969141
##   19  0.7320000  0.2027894
##   21  0.7268571  0.1811377
##   23  0.7262857  0.1696587
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
plot(knnFit)
```



The test set is then classified as follows.

```
#### test process
pred=predict(knnFit,test.feature)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7066667
```

Fatty Acid Composition Data

A set of data where seven fatty acid compositions were used to classify commercial oils as either pumpkin (labeled A), sunflower (B), peanut (C), olive (D), soybean (E), rapeseed (F) and corn (G). Our aim is to classify an unknown commercial oil to one of the seven categories based on its seven fatty acid compositions.

This dataset is in the `caret` package. We can load the data by using the `data( )` command.

```
library(caret)
#load oil data from caret package
data(oil)
```

Now we can check some basic properties of this dataset.

```
dim(fattyAcids) # check the dimensions of features
```

```
## [1] 96  7
```

```
table(oilType) # table of oil types
```

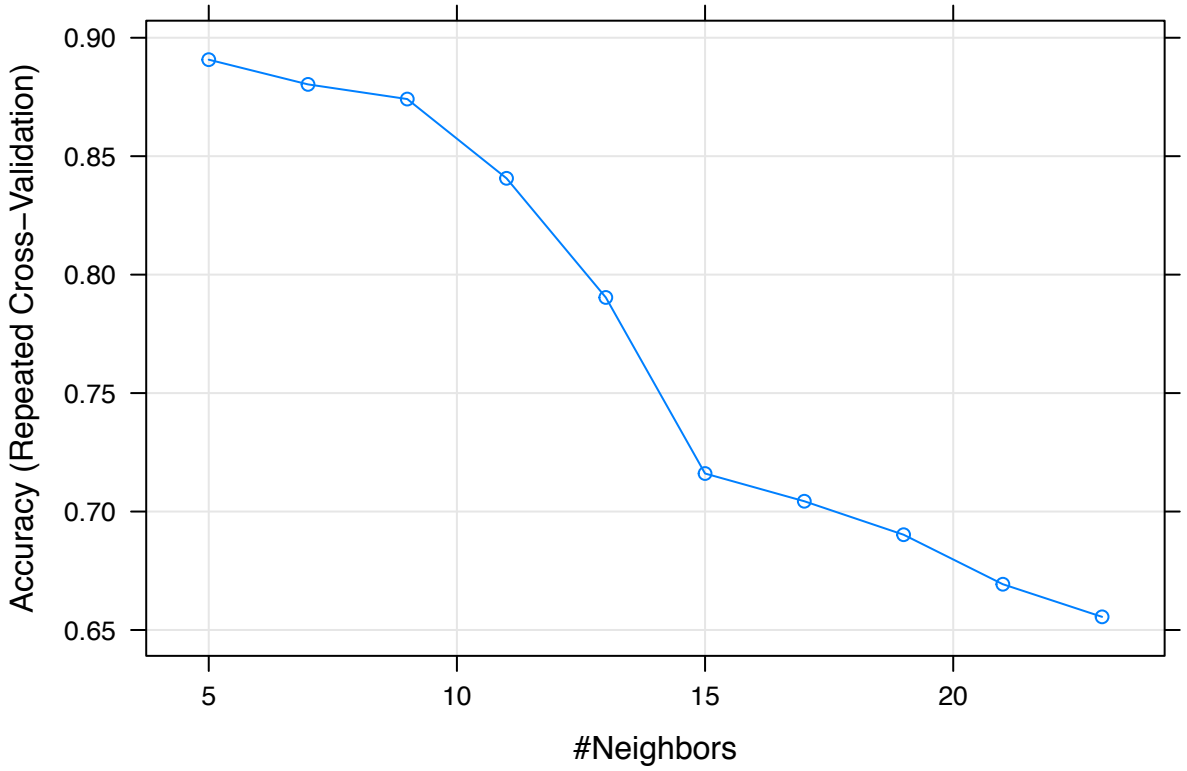
```
## oilType
##  A  B  C  D  E  F  G
## 37 26  3  7 11 10  2
```

First, we divide the dataset to a training and a test set.

```
# create training and test sets
set.seed(32)
trainIndex = createDataPartition(oilType, p = 0.7, list = FALSE, times = 1)
train.feature=fattyAcids[trainIndex,] # training features
train.label=oilType[trainIndex] # training labels
test.feature=fattyAcids[-trainIndex,] # test features
test.label=oilType[-trainIndex] # test labels
```

Then, we can train a kNN classifier based on the training set. The value of `k` is tuned by 10-fold CV.

```
## k-Nearest Neighbors
##
## 70 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## Pre-processing: centered (7), scaled (7)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 62, 63, 62, 63, 62, 65, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8907381  0.8558033
##    7  0.8803016  0.8387934
##    9  0.8741111  0.8300201
##   11  0.8406587  0.7842398
##   13  0.7903730  0.7094608
##   15  0.7160476  0.5899361
##   17  0.7043492  0.5664201
##   19  0.6902222  0.5452280
##   21  0.6693175  0.5151346
##   23  0.6555476  0.4942422
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```



Now we can classify the test data by the trained kNN classifier.

```
#### test process
pred=predict(knnFit,test.feature)
acc=mean(pred==test.label)
acc
```

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

Pima Indians Diabetes Data

In this dataset, we aim to predict the onset of diabetes in female Pima Indians from medical record data. This dataset can be loaded from the `mlbench` package.

```
library(mlbench)
#load Pima Indians Diabetes data from mlbench package
data(PimaIndiansDiabetes)
dim(PimaIndiansDiabetes)
```

```
## [1] 768   9
```

```
levels(PimaIndiansDiabetes$diabetes)
```

```
## [1] "neg" "pos"
```

```
head(PimaIndiansDiabetes)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35       0 33.6    0.627  50      pos
## 2        1      85       66      29       0 26.6    0.351  31      neg
## 3        8     183       64       0       0 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
```

```
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74       0       0 25.6    0.201  30      neg
```

```
table(PimaIndiansDiabetes$diabetes)
```

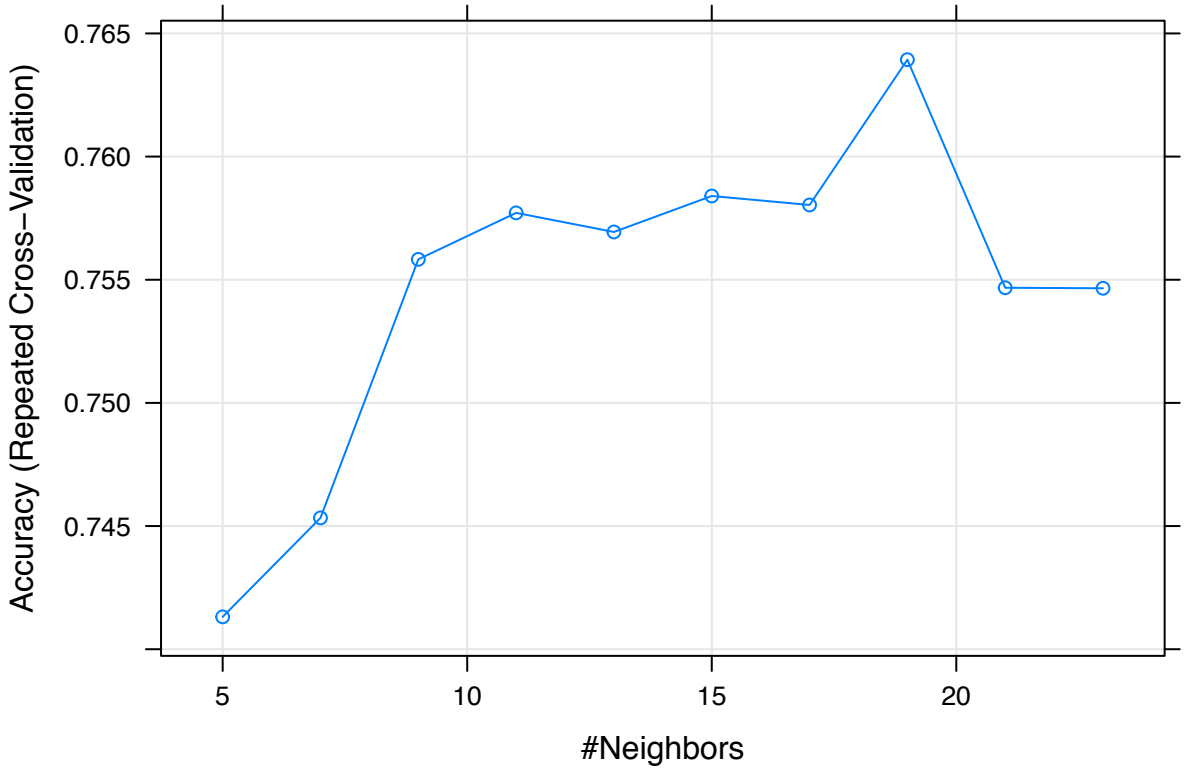
```
##
## neg pos
## 500 268
```

Divide the dataset to training and test sets.

```
# create training and test sets
set.seed(76)
trainIndex = createDataPartition(PimaIndiansDiabetes$diabetes, p = 0.7, list = FALSE, times = 1)
train.feature=PimaIndiansDiabetes[trainIndex,-9] # training features
train.label=PimaIndiansDiabetes$diabetes[trainIndex] # training labels
test.feature=PimaIndiansDiabetes[-trainIndex,-9] # test features
test.label=PimaIndiansDiabetes$diabetes[-trainIndex] # test labels
```

Train a kNN classifier and tune the value of `k` by 10-fold CV.

```
## k-Nearest Neighbors
##
## 538 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 484, 484, 485, 484, 484, 484, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7413138  0.4134968
##    7  0.7453319  0.4167808
##    9  0.7558281  0.4383072
##   11  0.7577149  0.4403751
##   13  0.7569392  0.4327286
##   15  0.7583997  0.4329293
##   17  0.7580363  0.4253536
##   19  0.7639343  0.4376814
##   21  0.7546751  0.4135613
##   23  0.7546541  0.4117932
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 19.
```



Predict the labels of the test set.

```
#### test process
pred=predict(knnFit,test.feature)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7043478
```