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
```

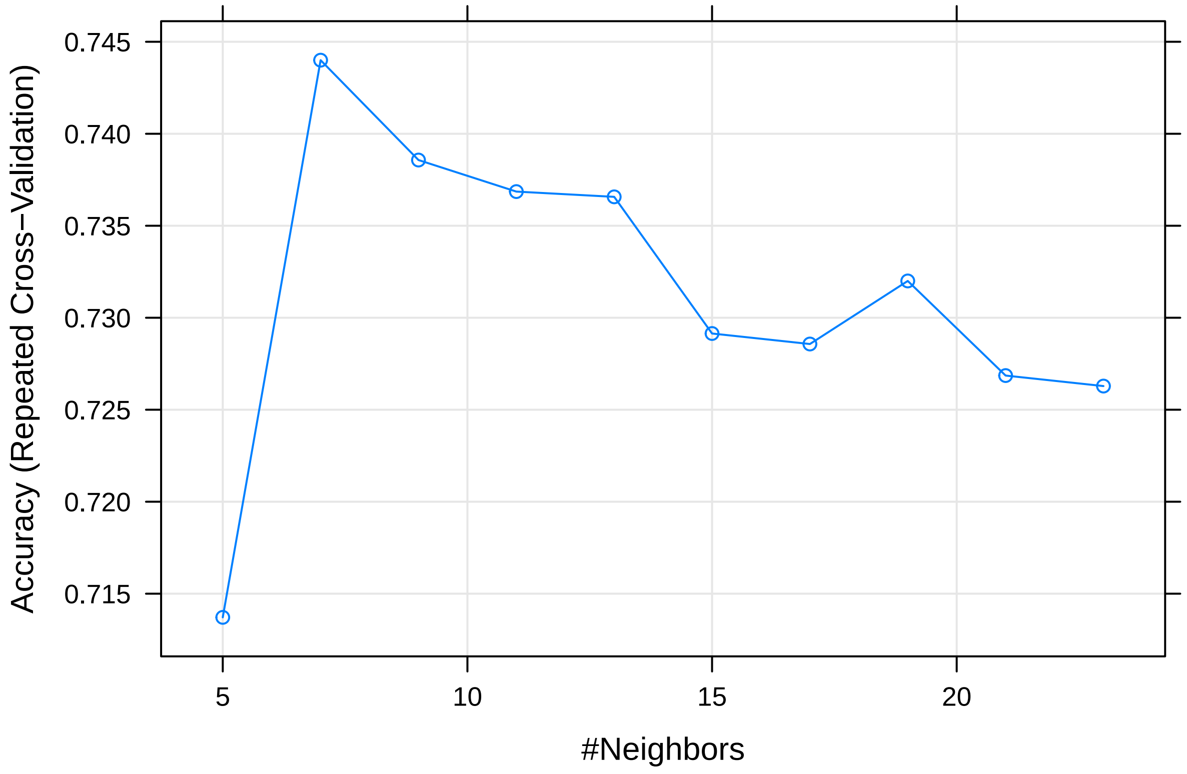
Train the kNN classifier and tune the value of `k` by 10-fold CV.

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

```r
  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.
```

```r
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
```

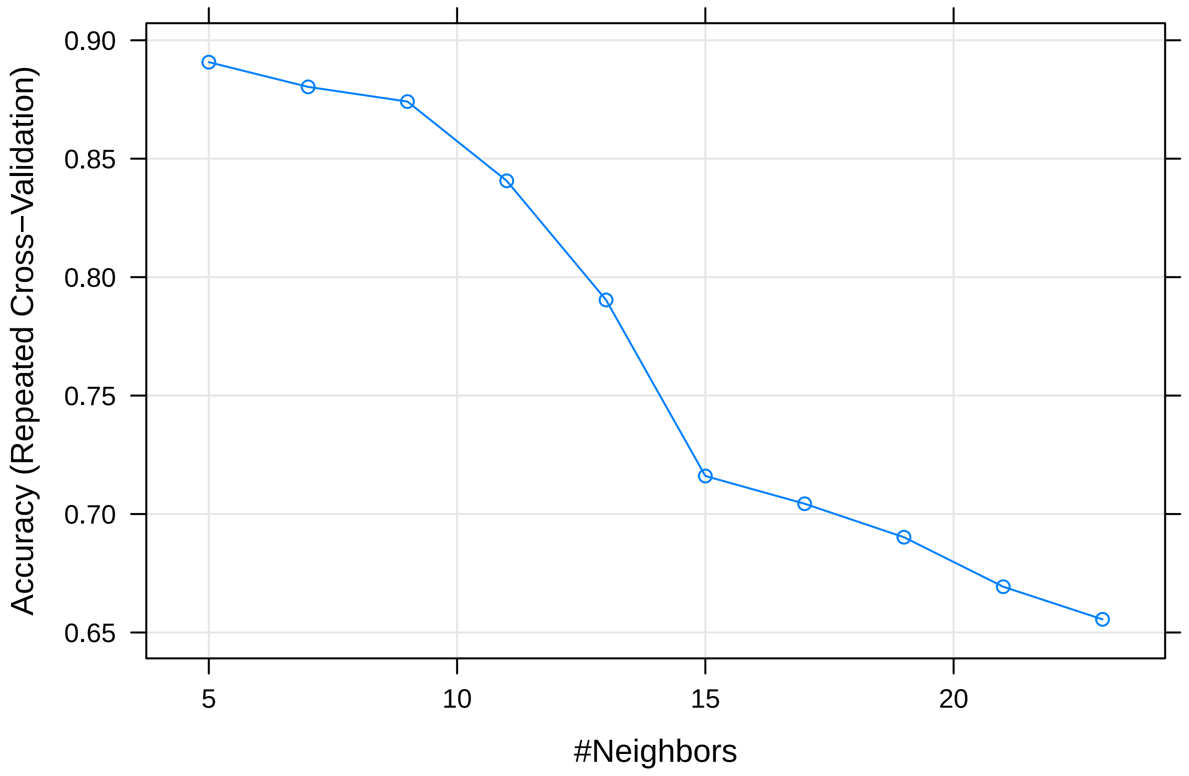
```
## oilType
##  A  B  C  D  E  F  G
## 37 26  3  7 11 10  2
```

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
```

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

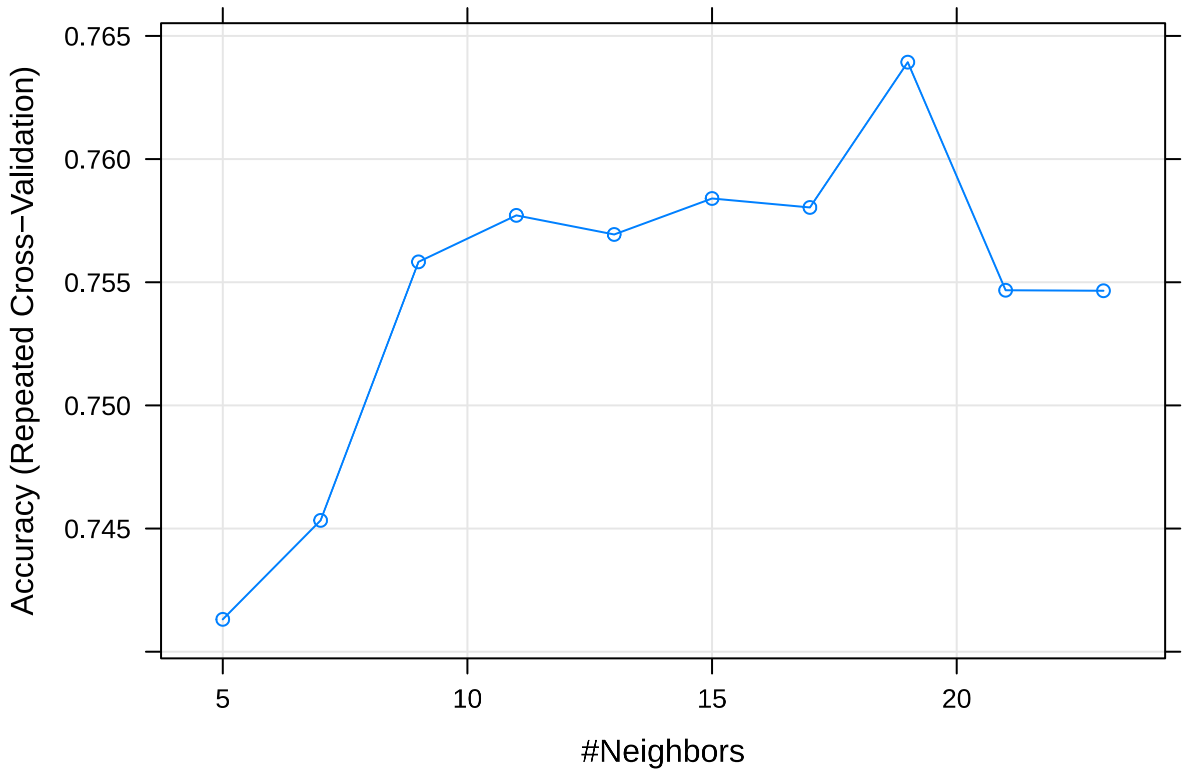
```
##
## neg pos
## 500 268
```

Divide the dataset to training and test sets.

```r
# 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
```