Introduction to supervised learning

Olga Dethlefsen

Table of contents

Preface

Aims

- to introduce supervised learning for classification and regression

Learning outcomes

- to be able to explain supervised learning
- to be able to split data into training, validation and test sets
- to be able to explain basic performance metrics for classification and regression
- to be able to use knn() function to select the optimal value of k and build KNN classifier

Do you see a mistake or a typo? We would be grateful if you let us know via edu.ml-biostats@nbis.se

1 Introduction to supervised learning

1.1 What is supervised learning?

- When we talked earlier about PCA and clustering, we were interested in finding patterns in the data. We treated data set as a whole, using all the samples, and we did not use samples labels in any way to find the components with the highest variables (PCA) or the number of clusters (k-means).
- In supervised learning, we are using samples **labels** to build (train) our models. When then use these trained models for interpretation and **prediction**.

1.2 Supervised classification

- Classification methods are algorithms used to categorize (classify) objects based on their measurements.
- They belong under **supervised learning** as we usually start off with **labeled** data, i.e. observations with measurements for which we know the label (class) of.
- If we have a pair $\{\mathbf{x_i}, g_i\}$ for each observation i, with $g_i \in \{1, \ldots, G\}$ being the class label, where G is the number of different classes and $\mathbf{x_i}$ a set of exploratory variables, that can be continuous, categorical or a mix of both, then we want to find a **classification rule** $f(.)$ (model) such that

$$f(\mathbf{x_i}) = g_i$$

1.3 KNN example

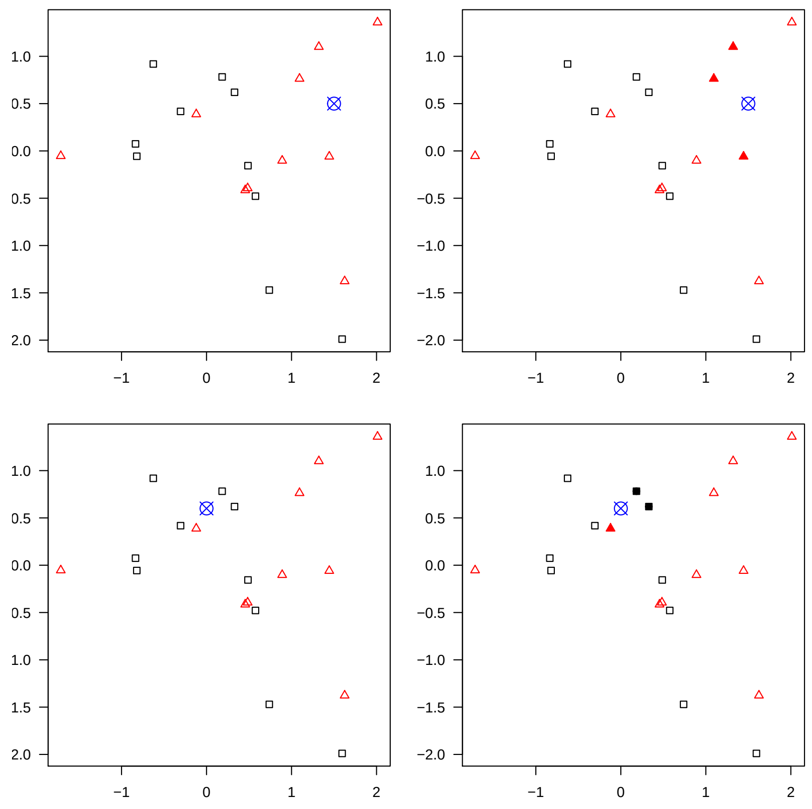


Figure 1.1: An example of k-nearest neighbours algorithm with k=3; in the top new observation (blue) is closest to three red triangales and thus classified as a red triangle; in the bottom, a new observation (blue) is closest to 2 black dots and 1 red triangle thus classified as a black dot (majority vote)

1.4 Data splitting

1.4.1 train, validation & test sets

- Part of the issue of fitting complex models to data is that the model can be continually tweaked to adapt as well as possible.
- As a results the trained model may not be generalizable to future data due to the added complexity that only works for given unique data set, leading to so called overfitting.
- To deal with overconfident estimation of future performance we randomly split data into training data, validation data and test data.

- Common split strategies include 50%/25%/25% and 33%/33%/33% splits for training/validation/test respectively
- **Training data**: this is data used to fit (train) the classification model, i.e. derive the classification rule
- **Validation data**: this is data used to select which parameters or types of model perform best, i.e. to validate the performance of model parameters
- **Test data**: this data is used to give an estimate of future prediction performance for the model and parameters chosen

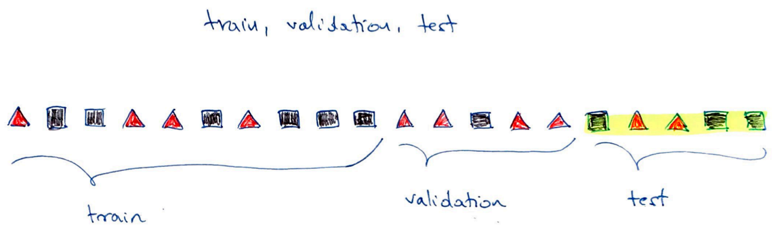


Figure 1.2: Example of splitting data into train (50%), validation (25%) and test (25%) set

1.4.2 cross validation

- It could happen that despite random splitting in train/validation/test dataset one of the subsets does not represent data. e.g. gets all the difficult observation to classify.
- Or that we do not have enough data in each subset after performing the split.
- In **K-fold cross-validation** we split data into K roughly equal-sized parts.
- We start by setting the validation data to be the first set of data and the training data to be all other sets.
- We estimate the validation error rate / correct classification rate for the split.
- We then repeat the process $K - 1$ times, each time with a different part of the data set to be the validation data and the remainder being the training data.
- We finish with K different error of correct classification rates.

- In this way, every data point has its class membership predicted once.
- The final reported error rate is usually the average of K error rates.

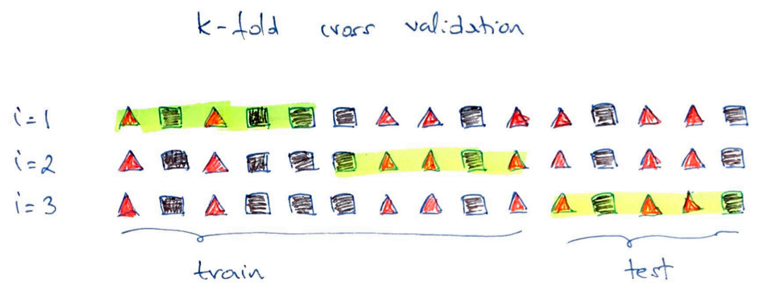


Figure 1.3: Example of k-fold cross validaiton split (k = 3)

- Leave-one-out cross-validation is a special case of cross-validation where the number of folds equals the number of instances in the data set.

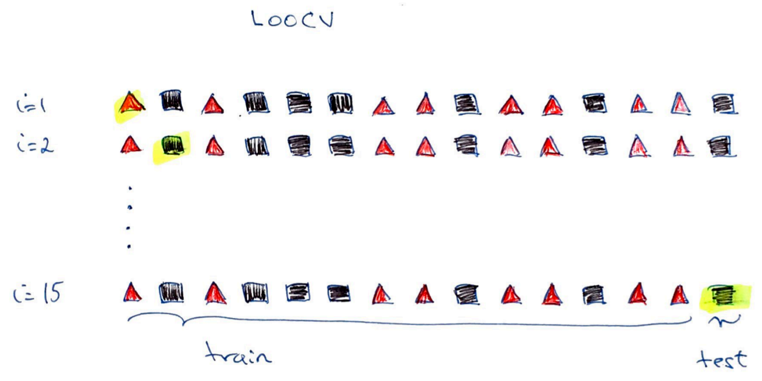


Figure 1.4: Example of LOOCV, leave-out-out cross validation

1.5 Evaluating Classification Model Performance

- To train the model we need some way of evaluating how well it works so we know how to tune the model parameters, e.g. change the value of k in KNN.

- There are few measures being used that involve looking at the truth (labels) and comparing it to what was predicted by the model.
- Common measures include: correct (overall) classification rate, missclassification rate, class specific rates, cross classification tables, sensitivity and specificity and ROC curves.

Correct (miss)classification rate

- The simplest way to evaluate in which we count for all the n predictions how many times we got the classification right.

$$Correct\ Classifcation\ Rate = \frac{\sum_{i=1}^{n} 1[f(x_i) = g_i]}{n}$$

where 1[] is an indicator function equal to 1 if the statement in the bracket is true and 0 otherwise

Missclassification Rate = 1 - Correct Classification Rate

1.6 Putting it together: k-nearest neighbours

Algorithm

- Decide on the value of k
- Calculate the distance between the query-instance (new observation) and all the training samples
- Sort the distances and determine the nearest neighbors based on the k-th minimum distance
- Gather the categories of the nearest neighbors
- Use simple majority of the categories of the nearest neighbors as the prediction value of the new observation

Euclidean distance is a classic distance used with KNN; other distance measures are also used incl. weighted Euclidean distance, Mahalanobis distance, Manhatan distance, maximum distance etc.

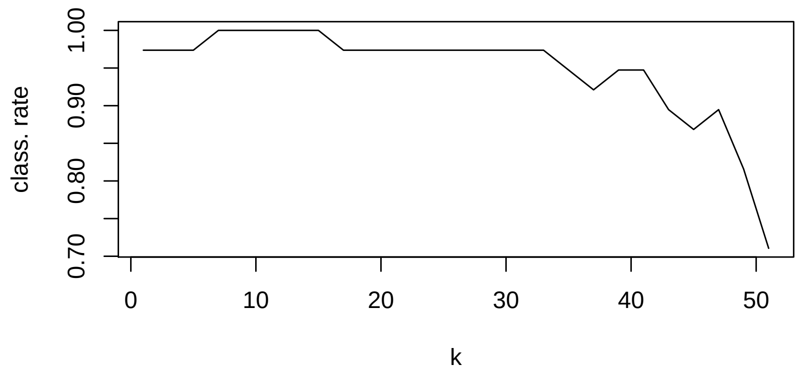
choosing k

- for problems with 2 classes, choose an odd number of k to avoid ties
- use validation data to fit the model for a series of k values
- pick the value of k which results in the best model (as assessed by the method of choice, e.g. overall classification rate)

Let's see how it works in practice on a classical iris dataset containing measurements on petals and sepals as well as species information (setosa, versicolor, virginica)

```
library(class) # library with knn() function
library(splitTools) # load library for data splitting
## Warning: package 'splitTools' was built under R version 4.0.5

# preview iris dataset
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
tail(iris)
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica

# summary statistics
summary(iris)
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##

# split data into train 50%, validation 25% and test dataset 25%
#
randseed <- 102
set.seed(randseed)
inds <- partition(iris$Species, p = c(train = 0.5, valid = 0.25, test = 0.25), seed = randse
str(inds)
## List of 3
##  $ train: int [1:74] 1 4 5 6 8 11 13 15 17 19 ...
##  $ valid: int [1:38] 3 10 12 18 21 27 28 30 32 33 ...
##  $ test : int [1:38] 2 7 9 14 16 20 23 29 31 35 ...

data.train <- iris[inds$train, ]
data.valid <- iris[inds$valid,]
data.test <- iris[inds$test, ]

dim(data.train)
## [1] 74  5
dim(data.valid)
## [1] 38  5
dim(data.test)
## [1] 38  5

summary(data.train$Species)
##     setosa versicolor  virginica
##         24         25         25
summary(data.valid$Species)
##     setosa versicolor  virginica
##         13         12         13
summary(data.test$Species)
```

```
##     setosa versicolor  virginica
##         13         13         12

# run knn with different values of k from 1, 3, 5 to 51
k.values <- seq(1, 51, 2)
class.rate <- rep(0, length(k.values)) # allocate empty vector to collect correct classifica
for (k in seq_along(k.values))
{

  pred.class <- knn(train = data.train[, -5],
                    test = data.valid[, -5],
                    cl = data.train[, 5], k.values[k])

  class.rate[k] <- sum((pred.class==data.valid[,5]))/length(pred.class)
}

# for which value of k we reach the highest classification rate
k.best <- k.values[which.max(class.rate)]
print(k.best)
## [1] 7

# plot classification rate as a function of k
plot(k.values, class.rate, type="l", xlab="k", ylab="class. rate")
```



```
# how would our model perform on the future data using the optimal k?
pred.class <- knn(train = data.train[, -5], data.test[, -5], data.train[,5], k=k.best)
class.rate <- sum((pred.class==data.test[,5]))/length(pred.class)
```

```
print(class.rate)
## [1] 0.9473684
```

1.7 Going back to regression

- The idea of using data splits to train the model holds for fitting (training) regression models.
- Earlier, we used the entire data set to fit the model and we used the fitted model for prediction given a new observation.
- If we were to use regression in supervised learning context, we would use data splits to train and assess the regression model.
- For instance, given a number of variables of interest, we could try to find the best regression model using train data to fit the model and assess on the validation data; while keeping the test to assess the performance on the final model. Or we could use cross validation (We have seen before how to fit the model and assess the model fit, e.g. with R^2.)
- Other popular regression performance metrics include **RMSE**, root mean square error

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

- and **MAE**, mean absolute error,

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

KNN demo with k-fold cross-validation

```
# load libraries and data
library(class)
library(splitTools)
## Warning: package 'splitTools' was built under R version 4.0.5
data(iris)
```

```
# make copy of data
data.iris <- iris

# set random seed
randseed <- set.seed(124)

# set aside test data
splits <- partition(data.iris$Species,
                    p = c(rest = 0.80, train = 0.20))
data.test <- data.iris[splits$train, ]

# define remaining data as data.rest and split these samples into k-folds
data.rest <- data.iris[splits$rest, ]
kfolds.train <- create_folds(data.rest$Species,
                             k = 5,
                             seed = randseed)

kfolds.valid <- create_folds(data.rest$Species,
                             k = 5,
                             invert = TRUE, # gives back indices of the validation samples
                             seed = randseed) # OBS! use the same seed as above in kfolds.tr
# check folds
str(kfolds.train)
```

```
List of 5
 $ Fold1: int [1:96] 1 2 3 4 5 7 8 9 10 11 ...
 $ Fold2: int [1:96] 1 2 3 5 6 7 9 10 11 12 ...
 $ Fold3: int [1:96] 1 2 3 4 5 6 8 9 10 12 ...
 $ Fold4: int [1:96] 2 4 5 6 7 8 9 10 11 14 ...
 $ Fold5: int [1:96] 1 3 4 6 7 8 11 12 13 14 ...
```

```
str(kfolds.valid)
```

```
List of 5
 $ Fold1: int [1:24] 4 11 14 15 22 23 28 40 47 61 ...
 $ Fold2: int [1:24] 1 8 18 20 24 29 33 35 41 44 ...
 $ Fold3: int [1:24] 3 5 6 7 12 25 26 38 43 46 ...
 $ Fold4: int [1:24] 10 13 21 30 31 34 37 39 42 48 ...
 $ Fold5: int [1:24] 2 9 16 17 19 27 32 36 45 49 ...
```

```
# verify that the k-folds were done correctly
# (let's just check fold fold1 but should check all the folds in reall life)
# check that fold samples are mutually exclusive
intersect(kfolds.train$Fold1, kfolds.valid$Fold1) # there should be no overlap
```

```
 [1]   4  11  14  15  28  40  63  69  74  83  99 103 110 112 115
```

```
# check that all the samples were used (we should get 120 unique samples in our case, sames
length(c(kfolds.train$Fold1, kfolds.valid$Fold1))
```

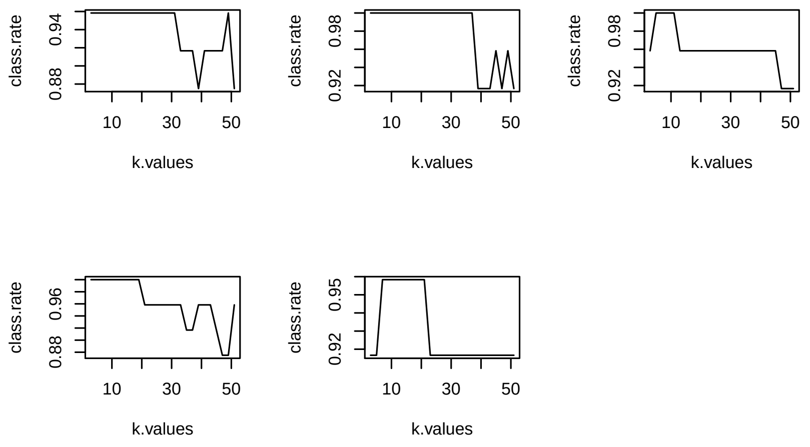
```
[1] 120
```

```
dim(data.rest)
```

```
[1] 120   5
```

```
# fit the model
k.values <- seq(3, 51, by = 2) # parameter space
k.best.folds <- c() # vector to collect the best value of k across the three folds

par(mfrow=c(2,3))
for (j in 1:5){

  # for each fold define train and validation sets
  data.train <- data.rest[kfolds.train[[j]], ]
  data.valid <- data.rest[kfolds.valid[[j]],]

  # repeat fitting KNN as in the previous example
  class.rate <- c()
  for (i in seq_along(k.values)){

    set.seed(randseed)
    pred.class <- knn(train = data.train[, -5],
                      test = data.valid[, -5],
                      cl = data.train[, 5],
                      k = k.values[i])

    class.rate.truefalse <- pred.class == data.valid[, 5]
    class.rate[i] <- sum(class.rate.truefalse)/length(pred.class)

  }

  plot(k.values, class.rate, type = "l")
  k.best.folds[j] <- k.values[which.max(class.rate)]

}

k.best.folds
```

```
[1] 3 3 5 3 7
```



```r
# now we have five models and we can find corresponding accuracy on the test data for each m
# as well as average of these
final.accuracy <- c()
for (j in 1:5){

  data.train <- data.rest[kfolds.train[[j]], ]
  class.pred <- knn(train = data.train[, -5],
                    test = data.test[, -5],
                    cl = data.train[, 5],
                    k = k.best.folds[j])

  final.accuracy[j] <- sum(class.pred ==
                     data.test$Species)/length(class.pred)

}
print(mean(final.accuracy))
## [1] 0.9666667
```

Exercises

Exercise 1.1 (Breast Cancer classifier)**.** Given BreastCancer data shown below build the best KNN classification model that you can to predict the cancer type (benign or malignant).

Note: you may need to do some data cleaning.

- First column contains patients IDs that should not be used
- Data set contains some missing values. You can keep only the complete cases e.g. using na.omit() function.

```
# install "mlbench" package
library(mlbench)

# Look at the Breast Cancer data
# more about data is here: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(
data(BreastCancer)
dim(BreastCancer)
## [1] 699  11
levels(BreastCancer$Class)
## [1] "benign"    "malignant"
head(BreastCancer)
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
## 4 1016277            6         8          8             1            3
## 5 1017023            4         1          1             3            2
## 6 1017122            8        10         10             8            7
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
## 1           1           3               1       1    benign
## 2          10          3              2       1    benign
## 3           2          3              1       1    benign
## 4           4          3              7       1    benign
## 5           1          3              1       1    benign
## 6          10          9              7       1 malignant
```

Exercise 1.2 (Training regression model)**.** Let's look again at the "fat" data set from the "faraway" package. This time we would like to use regression to predict the Brozek scores given new observations as accurately as possible.

Split data into train (60%), validation (20%) and test (20%). Assess three models on the validation set using RMSE. Which model seems to be the best in terms of predicting the Brozek scores. What would be the expected performance on the new (test) data?

1. Model 1: use "age" only as predictor
2. Model 2: use "age", "weight" and "heigth" as predictors
3. Model 3: use "age", and "wrist" as predictors

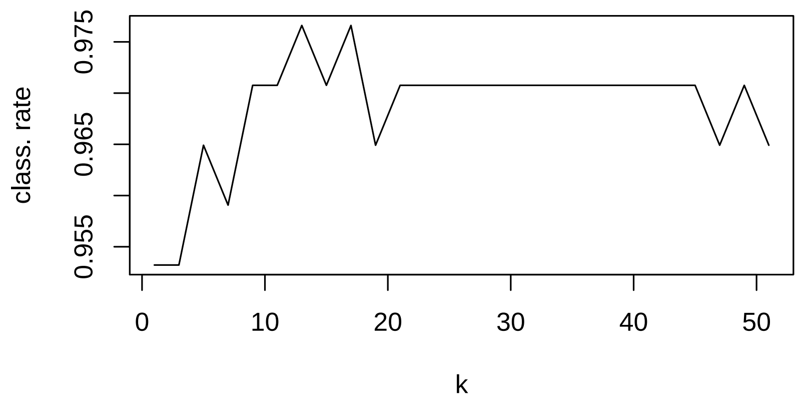
Answers to selected exercises

Solution. Exercise 1.1

```
library(splitTools)
library(class)

# set random seed to be able to repeat analysis
randseed <- 123
set.seed(randseed)

# clean data by removing first column of IDs
data.cancer <- BreastCancer[, -1]

# keep only complete cases
data.cancer <- na.omit(data.cancer)
```

```
# split data into train, validation and test
inds <- partition(data.cancer$Class,
                  p = c(train = 0.5, valid = 0.25, test = 0.25),
                  seed = randseed)

# preview lists with splits
str(inds)
## List of 3
##  $ train: int [1:341] 2 3 4 8 13 17 18 20 21 24 ...
##  $ valid: int [1:171] 7 9 10 11 22 23 25 30 35 49 ...
##  $ test : int [1:171] 1 5 6 12 14 15 16 19 26 27 ...

# make train, validation and test sets
data.train <- data.cancer[inds$train, ]
data.valid <- data.cancer[inds$valid,]
data.test <- data.cancer[inds$test, ]

# check their dimensions
dim(data.train)
## [1] 341  10
dim(data.valid)
## [1] 171  10
dim(data.test)
## [1] 171  10

# and print our group summaries
summary(data.train$Class)
##    benign malignant
##       222       119
summary(data.valid$Class)
##    benign malignant
##       111        60
summary(data.test$Class)
##    benign malignant
##       111        60

# find optimal value of k
k.values <- seq(1, 51, 2)
class.rate <- rep(0, length(k.values))
```

```r
for (k in seq_along(k.values))
{

  pred.class <- knn(train = data.train[, -10],
                    test = data.valid[, -10],
                    cl = data.train[,10], k=k.values[k])

  class.rate[k] <- sum((pred.class==data.valid[,10]))/length(pred.class)
}

# for which value of k we reach the highest classification rate
k.best <- k.values[which.max(class.rate)]
print(k.best)
## [1] 13

# plot classification rate as a function of k
plot(k.values, class.rate, type="l", xlab="k", ylab="class. rate")
```



```r
# how would our model perform on the future data using the optimal k?
pred.class <- knn(train = data.train[, -10], data.test[, -10], data.train[,10], k=k.best)
class.rate <- sum((pred.class==data.test[,10]))/length(pred.class)
print(class.rate)
## [1] 0.9707602
```

Other solutions could include:

- trying and comparing models with different number of

variables
- trying to implement k-fold cross validation or LOOVC to assess model performance when selecting the optimal value of k

Solution. Exercise 1.2

```
library(tidyverse)
library(splitTools)

# access data
# data(fat, package = "faraway")
# data.fat <- fat

data.fat <- read_csv("data/brozek.csv")

# split into train, validation and test: stratify by Brozek score
inds <- partition(data.fat$brozek,
                  p = c(train = 0.6, valid = 0.2, test = 0.2),
                  seed = randseed)
str(inds)
## List of 3
##  $ train: int [1:145] 1 2 4 5 6 7 11 12 13 15 ...
##  $ valid: int [1:54] 3 8 9 10 14 18 22 27 29 34 ...
##  $ test : int [1:53] 16 37 38 40 42 43 48 49 54 62 ...

data.train <- data.fat[inds$train, ]
data.valid <- data.fat[inds$valid,]
data.test <- data.fat[inds$test, ]

# Model 1
m1 <- lm(brozek ~ age, data = data.train) # fit model on train
m1.pred <- predict(m1, newdata = data.valid[,-1]) # predict brozek score using validation se
m1.rmse <- sqrt((1/nrow(data.valid))*sum((data.valid$brozek - m1.pred)^2)) # calculate RMSE

# Model 2
m2 <- lm(brozek ~ age + weight + height, data = data.train) # fit model on train
m2.pred <- predict(m2, newdata = data.valid[,-1]) # predict brozek score using validation se
m2.rmse <- sqrt((1/nrow(data.valid))*sum((data.valid$brozek - m2.pred)^2)) # calculate RMSE
```

```
# Model 3
m3 <- lm(brozek ~ age + wrist, data = data.train) # fit model on train
m3.pred <- predict(m3, newdata = data.valid[,-1]) # predict brozek score using validation se
m3.rmse <- sqrt((1/nrow(data.valid))*sum((data.valid$brozek - m3.pred)^2)) # calculate RMSE

# Compare models
rmse <- data.frame(model = c("Model 1", "Model 2", "Model 3"), rmse = c(m1.rmse, m2.rmse, m3
rmse
##     model     rmse
## 1 Model 1 6.970207
## 2 Model 2 6.281986
## 3 Model 3 6.543864

# Out of the three models, Model 2, has the smallest RMSE and is thus selected as best

# Expected performance on the test data
m.pred <- predict(m2, newdata = data.test[,-1])
sqrt((1/nrow(data.test))*sum((data.test$brozek - m.pred)^2))
## [1] 8.129801
```

Note: It is possible to use glm() function to fit linear regression. The advantage of this is that one can use easily use cross validation with cv.glm(). More about that on Friday.