Classificeren

Het k-NN-algoritme

Het is het handigste om de *nearest neigbour* methode (verder aan te duiden als de k-NN-methode) te illustreren aan de hand van een voorbeeld.

> *Een instituut voor microfinanciering in Afrika stelt leningen ter beschikking aan startende (groepen van) ondernemers.*
>
> *Tot nu toe worden aanvragen voor leningen vooral beoordeeld en behandeld door veldwerkers (loan officers) die dorpen langsgaan. Ongeveer 40% van de aanvragen wordt toegewezen.*
>
> *De veldwerkers hebben in de loop van de tijd veel ervaring opgedaan, en weten vooraf goed in te schatten of cliënten in staat zijn de rente te betalen en de lening af te lossen (ruim 90% van de leningen wordt op tijd afgelost).*
>
> *Met de introductie van mobiele technologie komen echter steeds meer aanvragen direct op het hoofdkantoor terecht waar ze worden beoordeeld door administratieve medewerkers die geen direct contact hebben met de aanvragers.*
>
> *Het hoofdkantoor beschikt over een schat aan gegevens over toegewezen en afgewezen leningen, en kenmerken van de aanvragers. De vraag is nu, of het mogelijk is om aan de hand van een aantal simpele kenmerken (o.a. leeftijd; geslacht; ervaring) te weten komen of een lening moet worden toegewezen of afgewezen.*

Afstand: een maatstaf voor ‘lijken op’

Het uitgangspunt is dat een nieuwe aanvrager wordt vergeleken met een set van “soortgelijke” aanvragers uit het verleden. Als aan de meerderheid van die set een lening werd toegewezen dan wijzen we ook aan de nieuwe aanvrager de lening toe (en we wijzen de lening af in het andere geval). De set van soortgelijke aanvragers wordt bepaald aan de hand van een aantal kenmerken waarvan we vermoeden dat ze relevant zijn.

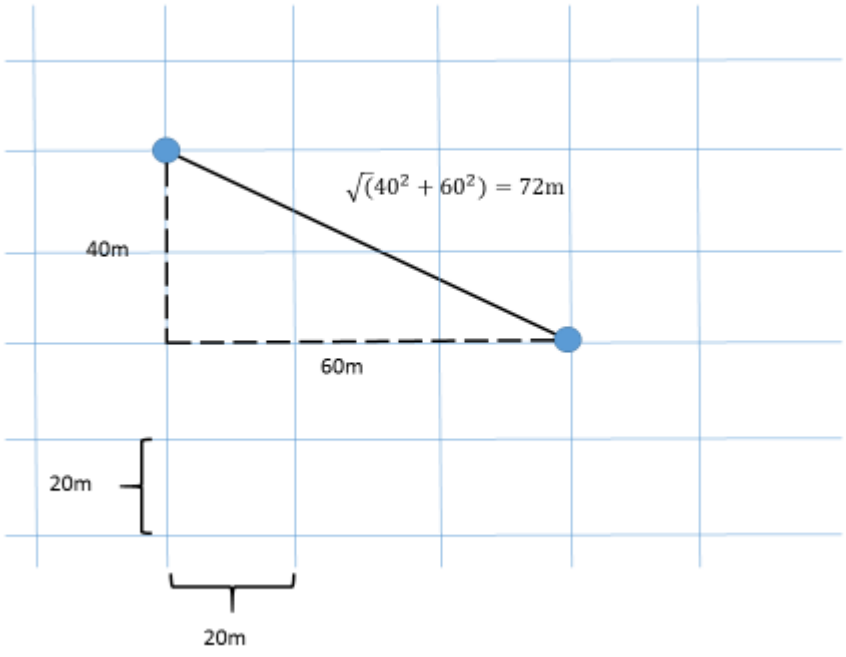
Uit de gegevens op het aanvraagformulier zijn 10 kenmerken geselecteerd:

- Geslacht (een *dummy-variabele*: 0=vrouw; 1=man);
- Leeftijd (in jaren);
- Ervaring (aantal jaren werkzaam, zelfstandig of in loondienst);
- Eigen oordeel over ondernemingsvaardigheden (een 10-puntsschaaal; 1=zeer slecht; 10=uitmuntend);
- Zelfstandig ondernemer geweest in het verleden (0=nee; 1=ja);
- Opleidingsniveau (1=geen; 2=primair onderwijs; 3=voortgezet onderwijs; 4=universitair);
- Opleidingsniveau ouders (zelfde codering);
- In bezit van een spaarrekening (0/1);
- Woonachtig in stad of platteland (0=platteland; 1=stad);
- Burgerlijke staat (0=niet getrouwd; 1=getrouwd).

Uit gesprekken met veldwerkers is gebleken dat deze kenmerken relevant zijn, maar op een vaak vrij complexe manier. Oudere aanvragers brengen meer ervaring met zich mee, wat positief is. Tegelijkertijd missen ze vaak de energie om succesvol iets nieuws te starten zeker als ze niet eerder een bedrijfje hebben gehad. Jongere ondernemers hebben geen ervaring, maar hebben betere vaardigheden (zoals kennis van IT). Opleidingsniveau is vooral een goede indicatie als het hoger is dan dat van de ouders (het duidt op doorzettingsvermogen en *commitment*). Kortom, de beslissing om wel of niet een lening toe te wijzen is complex, en bovendien deels intuïtief.

Als een nieuwe aanvrager zich meldt, dan gaan we zijn of haar gegevens vergelijken met de aanvragers uit het verleden. We zoeken in het bestand naar aanvragers met ongeveer gelijke kenmerken. Misschien vinden we aanvragers met exact dezelfde kenmerken: we zeggen dan dat de afstand tussen de nieuwe aanvrager en de bestaande klant nul is. De afstand tussen twee personen wordt bepaald aan de hand van de verschillen op de tien kenmerken. Je kunt dat op verschillende manieren doen. Je kunt de *absolute* verschillen meten, en die vervolgens sommeren over de 10 kenmerken. Maar meestal gebruiken we de zogenaamde *Euclidische* afstand. We werken daarbij in drie stappen. In de eerste stap bepalen we de gekwadrateerde verschillen tussen twee objecten (of zoals in ons voorbeeld, personen). In de tweede stap tellen we alle (in ons voorbeeld, tien) gekwadrateerde verschillen bij elkaar op. In stap 3, ten slotte, trekken we de wortel uit het resultaat van stap 2.

Opmerking: dit is eigenlijk een uitbreiding van de stelling van Pythagoras die u waarschijnlijk nog wel kent van de middelbare school! De hemelsbrede afstand tussen twee punten kan daarbij worden berekend als de wortel uit de som van de gekwadrateerde verschillen in horizontale en in verticale richting. De uitbreiding zit erin dat we nu meer dan twee dimensies hebben.



Figuur 1. De (Euclidische) afstand. Bron OGN.

Normaliseren of standaardiseren van de data

Bij het berekenen van afstanden moeten we goed rekening houden met de gebruikte eenheden. Voor leeftijd gebruiken we jaren, en het bereik van de metingen (de hoogst waargenomen leeftijd minus de laagst waargenomen leeftijd) in ons klantenbestand kan wel 40 (jaren) zijn. Voor geslacht gebruiken we een (0/1) dummyvariabele die als maximaal verschil 1 geeft. Zouden we ook het

bedrag op de spaarrekening als kenmerk meenemen, dan zouden de waarde (in eenheden lokale valuta) in de duizenden of zelfs miljoen kunnen lopen.

Het probleem van verschillende en onvergelijkbare eenheden kan op twee manieren worden opgelost: normaliseren, of standaardiseren.

Normaliseren

Normaliseren is het herberekenen van de gegevens tot waardes die liggen tussen 0 (het minimum) en 1 (het maximum) met de formule:

$$x_n = (x - x_{min})/(x_{max} - x_{\min})$$

Als voorbeeld: als de leeftijden liggen tussen 20 en 50, dan heeft iemand met een leeftijd van 35 jaar een genormaliseerde leeftijd van (35-20)/(50-20) = 0.50 (halverwege de laagst en de hoogst waargenomen leeftijd).

Als we alle variabelen normaliseren dan zijn ze vergelijkbaar.

Zelftest

U zit in een klas van 20 studenten. Uw score voor het examen Machine Learning is een 7. De cijfers van alle studenten lopen uiteen van 4 (minimum) tot 9 (maximum). Wat is us genormaliseerde score voor het examen?

Standaardiseren

Een alternatief voor normaliseren is standaardiseren. Hier bepalen we de op de normale verdeling gebaseerde Z-score. De Z-score is negatief (positief) voor waardes die kleiner (groter) zijn dan het gemiddelde. Standaardiseren heeft de voorkeur boven normaliseren als de criteria onbegrensd zijn en er uitschieters naar boven en/of naar beneden voorkomen.

Stel dat alle personen in het bestand tussen 0 en 100 geldeenheden verdienen, op een persoon na die 10,000 verdient. De bulk van de genormaliseerde waarden zou zich dan tussen 0 en 0.01 bevinden, en weinig gewicht hebben bij het vinden van *nearest neigbours*.

Het kiezen van het aantal *neighbours* (k)

De uitkomst van het algoritme is gevoelig voor de keuze van *k* (het aantal *nearest neigbours*). Kiezen we ze *k* heel groot, dan wordt het steeds waarschijnlijker dat de meerderheid in de dataset gaat domineren. In ons geval: aangezien de meerderheid van de aanvragen wordt afgewezen, zal er vaker een meerderheid van afwijzingen zijn als k toeneemt. In het extreme geval wijzen we alles af omdat dat nu eenmaal de meerderheid is. We zijn dan helemaal niets opgeschoten! En als de bank iedereen afwijst en dus geen leningen meer verstrekt zal het ook niet aan zijn doelstellingen (winst, en/of het steunen van veelbelovende ondernemers) kunnen voldoen.

Kiezen we daarentegen een zeer kleine k (in het uiterste geval, k=1) dan lopen we het risico dat kleine groepen die misschien niet correct gelabeld zijn een ongewenst grote invloed uitoefenen op de voorspellingen. Bijvoorbeeld, een persoon met extreme scores op, zeg, leeftijd en ondernemersvaardigheden, die een lening heeft gekregen maar die eigenlijk niet had moeten krijgen, heeft veel invloed op de voorspellingen van nieuwe aanvragers met dezelfde kenmerken.

We moeten dus op zoek naar de gulden middenweg.

Als vuistregel voor k gebruiken we vaak de wortel uit het aantal waarnemingen. Bij bijvoorbeeld 500 waarnemingen zetten we k op $\sqrt{500}$ = 22. In het geval van een binaire classificatie (een indeling in twee groepen) prefereren we echter een oneven getal zodat er altijd een meerderheid is; we kunnen k op 21 of 23 zetten. Doorgaans is dit slechts een ijkpunt, want om het model zo goed mogelijk te maken zullen we ook experimenteren met de waarde van k!

Toepassing van k-NN in R

Laten we het k-NN-algoritme stapsgewijs toepassen op een voorbeeld. De stappen zijn als volgt.

1. Inlezen en beschrijven van de data.
2. Gereed maken van de data voor toepassing van het k-NN algoritme:
   a. Normaliseren (of standaardiseren) van de data.
   b. Splitsen van de data in een training-set en een test-set.
3. Het trainen van het model op de data.
4. Het evalueren van het model.
5. Het verbeteren van het model.

Stap 1: inlezen en beschrijven van de data

We gaan weer werken in een **R**-script. Zoals we geleerd hebben in het eerste hoofdstuk beginnen we ons **R**-script met een aantal algemene zaken:

- Het instellen van de *working directory*, met het **`setwd()`** commando.
- Het laden van eventuele zelfgeschreven functies.
- Commentaar-regels met datum, omschrijving van de data en het doel van het script, om het script leesbaar te maken.

Onze data zijn afkomstig uit een **STATA** bestand (**ML3_1_KNN.dta**) dat we kunnen inlezen met **`read.dta()`**. Voor het gebruik van deze functie moeten we eerst het package **foreign** laden, met **`library(foreign)`**. We lezen het bestand in als een dataframe met de naam **knndata**. We kunnen de kenmerken van dat bestand bekijken met het **`str()`** commando.

```
## LOI Machine Learning
## Hoofdstuk 3
# kNN methode
setwd("C:\\LOI\\LOI DATA\\CHAPTER 3"); getwd()
library(foreign)
knndata<-read.dta("ML3_1_KNN.dta",convert.factors=FALSE)
str(knndata)
```

```
> str(knndata)
'data.frame':   500 obs. of 13 variables:
 $ id     : num  1 2 3 4 5 6 7 8 9 10 ...
 $ male   : num  1 0 0 0 0 1 0 1 0 0 ...
 $ age    : num  49 29 40 33 27 40 46 46 25 32 ...
 $ exp    : num  2 1 20 12 1 8 21 2 7 1 ...
 $ skills : num  3 1 10 10 10 10 1 10 10 10 ...
 $ busexp : num  0 0 1 0 0 0 1 0 0 0 ...
 $ educ   : num  1 1 1 1 2 3 3 2 2 1 ...
 $ educpar: num  1 1 1 1 4 2 4 2 4 1 ...
 $ saving : num  0 0 0 0 0 0 1 0 1 0 ...
 $ city   : num  0 1 0 0 1 1 0 0 0 0 ...
 $ marital: num  1 0 1 1 1 0 1 1 0 1 ...
 $ savacc : num  6.28 14.37 11.51 5.16 17.86 ...
 $ accept : num  0 0 1 0 0 0 1 1 0 1 ...
```

Uit het overzicht zien we dat het dataframe 500 observaties telt, met 13 variabelen. De waarden van de eerste records in het dataframe zijn weergegeven. Voordat we aan de slag gaan moeten we nog wel het nodige voorwerk verrichten.

- Sommige variabelen zijn niet nodig voor de analyse (**id**; **savacc**). De variabele **id** is een volgnummer van de records in het bestand en bevat geen nuttige informatie. De variabele **savacc** is een geschat bedrag aan spaargeld van de aanvrager. Een spaarrekening kan alleen worden toegekend als het minimale spaartegoed 20 geldeenheden is. Omdat het spaartegoed slechts een

momentopname is, gebruiken de *loan officers* het al of niet hebben van een spaarrekening als indicatie, en niet de hoogte van het spaartegoed.

- De variabele die we in de k-NN-methode het *label* noemen, moet een factor variabele zijn. Het *label* is hier **accept** maar dat is een numerieke variabele.
- Verder willen we controleren of alle variabelen waardes hebben in het juiste bereik (geen leeftijden boven 80 of onder 16; alle *dummies* zijn inderdaad 0 of 1; de ondernemersvaardigheden zijn op een 10-puntsschaal met waarden van 1 tot 10; enzovoorts).

Selectie en beschrijving van variabelen

Allereerst maken we een selectie van de variabelen die we nodig hebben in de analyse. Er zijn diverse manieren om dit te doen. Bij dit soort analyses is het een goed gebruik om de *label* variabele die aangeeft van welke groep het object (of de persoon) deel uitmaakt in de eerste kolom te plaatsen. De *label* variabele (**accept**) staat nu in de laatste van de 13 kolommen. De variabelen **id** en **savacc** hebben we niet nodig.

Eerst maken we een nieuwe factor variabele aan, **acceptFac**, op basis van de numerieke variabele **accept**. We gebruiken de **`factor()`** functie. In die functie kunnen we aangeven wat de waarden (*levels*) zijn in de oorspronkelijke variabele, en hoe we die waarden willen labelen. We kunnen de nieuwe variabele beschrijven met de in het vorige hoofdstuk geïntroduceerde zelfgeschreven **`CFR()`** functie. Van de 500 aanvragen is inderdaad ongeveer 60% afgewezen.

```
> knndata$acceptFac <- factor(knndata$accept, levels=c(0,1),
labels=c("Rejected","Accepted"))
> CFR(knndata$acceptFac, 2)
Frequencies (absolute/relative/cumulative)
LOI_Machine Learning/2016

           f cum.f rel.f cum.rel.f
Rejected 304   304  0.61      0.61
Accepted 196   500  0.39      1.00
```

Nu we de nieuwe variabele **acceptFac** hebben kunnen we een nieuw bestand maken met alleen de benodigde variabelen, en **acceptFac** in kolom 1.

Met de **`head()`** functie geven we de eerste (6) observaties weer, en **`summary()`** geeft een beschrijving van alle variabelen. Bestudeer de output altijd aandachtig: eventuele codeerfouten of uitschieters komen dan snel aan het licht! Dit bestand lijkt geen codeerfouten of vreemde waardes te bevatten. De codering voor opleidingsniveau is zoals verwacht van 1 tot 4; de leeftijden lopen uiteen van 21 tot 50; enzovoorts.

```
> knndata2 <- knndata[,c("acceptFac","male","age","exp","skills","busexp","educ","educpar","saving",
+                        "city","marital")] # id is niet nodig; accept ook niet
> head(knndata2)
  acceptFac male age exp skills busexp educ educpar saving city marital
1  Rejected    1  49   2      3      0    1       1      0    0       1
2  Rejected    0  29   1      1      0    1       1      0    1       0
3  Accepted    0  40  20     10      1    1       1      0    0       1
4  Rejected    0  33  12     10      0    1       1      0    0       1
5  Rejected    0  27   1     10      0    2       4      0    1       1
6  Rejected    1  40   8     10      0    3       2      0    1       0
> summary(knndata2)
    acceptFac        male             age              exp             skills          busexp
 Rejected:304   Min.   :0.000   Min.   :21.00   Min.   : 0.000   Min.   : 1.000   Min.   :0.000
 Accepted:196   1st Qu.:0.000   1st Qu.:29.00   1st Qu.: 4.000   1st Qu.: 1.000   1st Qu.:0.000
                Median :0.000   Median :37.00   Median : 8.000   Median : 3.000   Median :0.000
                Mean   :0.412   Mean   :35.98   Mean   : 9.214   Mean   : 5.224   Mean   :0.232
                3rd Qu.:1.000   3rd Qu.:43.00   3rd Qu.:13.000   3rd Qu.:10.000   3rd Qu.:0.000
                Max.   :1.000   Max.   :50.00   Max.   :30.000   Max.   :10.000   Max.   :1.000
      educ          educpar         saving           city          marital
 Min.   :1.000   Min.   :1.00   Min.   :0.000   Min.   :0.000   Min.   :0.000
 1st Qu.:1.000   1st Qu.:1.00   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000
 Median :2.000   Median :2.00   Median :0.000   Median :0.000   Median :1.000
 Mean   :2.016   Mean   :2.16   Mean   :0.244   Mean   :0.498   Mean   :0.614
 3rd Qu.:3.000   3rd Qu.:3.00   3rd Qu.:0.000   3rd Qu.:1.000   3rd Qu.:1.000
 Max.   :4.000   Max.   :4.00   Max.   :1.000   Max.   :1.000   Max.   :1.000
```

Stap 2: prepareren van de data

Normaliseren van de data

Zoals gezegd is de k-NN-methode – net als andere methodes voor *clustering* van objecten – gebaseerd op afstanden tussen de objecten. We hebben echter te maken met variabelen die in verschillende eenheden zijn uitgedrukt. Er zijn dummyvariabelen die als waarde 0 of 1 kunnen aannemen; leeftijd in jaren (met als maximum 50); vaardigheden op een 10-puntsschaal; en onderwijsniveau op een 4-puntsschaal. Om alle variabelen op hetzelfde niveau te brengen kunnen we ze normaliseren of standaardiseren. Vaak maken we gebruik van normalisering gebaseerd op de waarde die een object heeft op een schaal die loopt vanaf 0 (het minimum van de variabele) tot 1 (het maximum van de variabele). De genormaliseerde waarde ligt dan altijd tussen 0 en 1 wat een prettige eigenschap is. Merk op dat de dummyvariabelen in deze methode ongewijzigd blijven!

We kunnen voor elke variabele afzonderlijk de genormaliseerde waarde bepalen. Bijvoorbeeld voor leeftijd kunnen we de volgende regels schrijven:

```
> # voorbeeld normaliseren (zonder een aparte functie); voor leeftijd
> knndata2$age_n <- (knndata2$age - min(knndata2$age)) / (max(knndata2$age) - min(knndata2$a
ge))
> head(knndata2[,c("age","age_n")])
  age     age_n
1  49 0.9655172
2  29 0.2758621
3  40 0.6551724
4  33 0.4137931
5  27 0.2068966
6  40 0.6551724
```

Ter controle: iemand van 29 jaar heeft een genormaliseerde leeftijd van (29-21)/(50-21)=0.2759. Je zult begrijpen dat deze methode een hoop typewerk vergt, en erg foutgevoelig is. Het is beter een functie te schrijven die we **`normaliseer()`** zullen noemen.

```
# normaliseren van data
normaliseer <- function(x){
  return((x-min(x))/(max(x)-min(x)))
}
```

De functie **`normaliseer()`** wordt gedefinieerd als een object. Eenmaal gerund verschijnt de functie **`normaliseer()`** als object in het scherm rechtsboven. Tussen de accolades staat wat de

functie doet: de functie wordt toegepast op een argument (x), en past de eenvoudige formule toe met behulp van standaardfuncties in **R**, namelijk **`max(x)`** en **`min(x)`**.

We kunnen nu de functie toepassen op de 10 variabelen in ons bestand (niet op de label variabele: dat is een factor en kan niet worden genormaliseerd!). Dat kan in een stap door de functie toe te passen op een lijst (*list*) van gegevens: een dataframe is in feite een lijst van aan elkaar geplakte vectoren. Voor het toepassen van bewerkingen op een lijst gebruiken we de **`lapply()`** functie.

We schrijven de genormaliseerde data in een nieuw data frame **knndata2_n**. Merk op dat we de eerste kolom (de label variabele: accepteren of afwijzen) niet meenemen!

```
> # maak een "genormaliseerd" bestand; niet voor de "label" variabele acceptFac!!
> knndata2_n <- as.data.frame(lapply(knndata2[2:11], normaliseer))
> summary(knndata2_n)
      male             age               exp             skills          busexp           educ
 Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
 1st Qu.:0.000   1st Qu.:0.2759   1st Qu.:0.1333   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000
 Median :0.000   Median :0.5517   Median :0.2667   Median :0.2222   Median :0.000   Median :0.3333
 Mean   :0.412   Mean   :0.5164   Mean   :0.3071   Mean   :0.4693   Mean   :0.232   Mean   :0.3387
 3rd Qu.:1.000   3rd Qu.:0.7586   3rd Qu.:0.4333   3rd Qu.:1.0000   3rd Qu.:0.000   3rd Qu.:0.6667
 Max.   :1.000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
    educpar          saving            city           marital
 Min.   :0.0000   Min.   :0.000   Min.   :0.000   Min.   :0.000
 1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000
 Median :0.3333   Median :0.000   Median :0.000   Median :1.000
 Mean   :0.3867   Mean   :0.244   Mean   :0.498   Mean   :0.614
 3rd Qu.:0.6667   3rd Qu.:0.000   3rd Qu.:1.000   3rd Qu.:1.000
 Max.   :1.0000   Max.   :1.000   Max.   :1.000   Max.   :1.000
```

De output van **`summary()`** bevestigt dat de **`normaliseer()`** functie doet wat we verwachten: de minima en maxima van alle variabelen zijn nu allemaal 0 en 1. De variabelen zijn nu uitgedrukt in vergelijkbare eenheden!

Normaliseren een belangrijke stap is. De software zal zonder enige waarschuwing ook werken met data die niet zijn genormaliseerd. Maar de variabelen met het grootste bereik (hier leeftijd) leggen dan veel meer gewicht in de schaal, en dat is op voorhand niet wat we willen.

Splitsen van de data in een trainingset en een testset

De volgende stap is het splitsen van de data in een training- en een testset.

Met de trainingset willen we een model ontwikkelen dat de groepering (accepteren of afwijzen) voorspelt aan de hand van de 10 criteria waarvan we vermoeden dat ze belangrijk zijn. Of het model goed werkt moeten we bepalen aan de hand van een testset met data die niet zijn gebruikt bij de ontwikkeling van het model.

Een belangrijke vraag is het bepalen van de omvang van de twee sets. Aan de ene kant willen we natuurlijk zoveel mogelijk informatie gebruiken om een zo goed mogelijk model te bepalen. Maar tegelijkertijd willen we ook een testset hebben die groot genoeg is om statistisch onderbouwde uitspraken te doen over de kwaliteit van het model. Veelgebruikte vuistregels zijn om 20% of 25% van de dataset te gebruiken voor het testen van het model. Maar bij zeer grote gegevensbestanden kan dat percentage naar beneden worden bijgesteld.

Een ander belangrijk punt is de manier waarop de gegevens worden gesplitst in een training- en een testset. Als de data in willekeurige volgorde in het bestand staan dan kunnen we simpelweg de eerste 80% van de records gebruiken voor de training set en de laatste 20% voor de testset. Maar als de data op een of andere manier zijn gesorteerd (bijvoorbeeld op leeftijd, of op de label variabele) dan is het zeer onverstandig om deze methode te volgen. In het algemeen is het beter een random steekproef te trekken uit het databestand.

Het trekken van een steekproef gaat als volgt. We weten dat het databestand zo'n 500 records telt (maar we kunnen **R** het werk laten doen met de **`nrow()`** functie). We willen voor het gemak een testset van exact 100 records hebben. Allerlei functies die te maken hebben met steekproeven en toevalsgetallen, zijn eigenlijk niet helemaal toevallig: uitgaande van een bepaald startpunt (aangegeven met een getal dat we *seed* noemen) ligt de uitkomst van de trekkingen vast. Dat is vaak

wel handig, omdat met dat gegeven uitkomsten kunnen worden gereproduceerd. Het startpunt dat wij hebben gekozen met **`set.seed()`** is 12321. Het runnen van **`set.seed(12321)`** levert dezelfde resultaten op als hieronder beschreven. Het niet runnen of het gebruik van een ander startpunt geeft andere resultaten! Als de steekproef maar groot genoeg is zullen de conclusies echter dezelfde zijn.

De steekproef die wordt bewaard in de vector **train_steekproef**, is een reeks van 400 unieke getallen. De omvang van ons bestand blijkt immers 500 te zijn (**R** berekent dat met de **`nrow()`** functie), en we hebben zelf bepaald dat de test set een omvang moet hebben van 100 records.

```
> set.seed(12321)                    # voor reproduceerbaarheid van de resultaten!
>
> x <- nrow(knndata2_n)              # bepaal aantal observeringen (500)
> train_steekproef <- sample(x,x-100) # trek een steekproef van 400 uit 500
> str(train_steekproef)              # laat de structuur van de vector zien
 int [1:400] 422 214 481 31 300 210 30 114 449 482 ...
```

We kunnen de vector met de 400 geselecteerde records direct gebruiken voor het aanmaken van de training set **knn_train**. Voor de testset selecteren we alle records die NIET in de trainingset voorkomen, met de selectie "**-train_steekproef**". Uit de controle blijkt dat we inderdaad twee bestanden hebben gecreëerd van 400 en 100 records.

```
> knn_train <- knndata2_n[train_steekproef ,]
> knn_test  <- knndata2_n[-train_steekproef,]
> str(knn_train)
'data.frame':   400 obs. of 10 variables:
 $ male    : num  0 0 1 0 1 0 0 1 0 1 ...
 $ age     : num  0.103 0.276 0.207 0.276 0.862 ...
…
 $ marital: num  0 0 0 0 1 0 1 0 1 0 ...
> str(knn_test)
'data.frame':   100 obs. of 10 variables:
 $ male    : num  0 1 0 0 0 0 1 1 0 1 ...
 $ age     : num  0.655 0.655 0.379 0.31 0.483 ...
…
 $ marital: num  1 0 1 0 1 0 1 0 1 1 ...
```

Merk op dat dit trekkingen zijn uit het bestand met genormaliseerde data, en in dat bestand hebben we de label variabele **acceptFac** achterwege gelaten. We moeten dus hetzelfde doen voor de label variabele (kolom 1 in het bestand met niet-genormaliseerde data).

```
> knn_train_label <- knndata2[train_steekproef, 1] # de eerste kolom
uit het niet-genormaliseerde bestand!
> knn_test_label  <- knndata2[-train_steekproef, 1]
> str(knn_train_label); length(knn_train_label)
 Factor w/ 2 levels "Rejected","Accepted": 1 1 1 1 2 1 2 1 1 1 ...
[1] 400
> str(knn_test_label) ; length(knn_test_label)
 Factor w/ 2 levels "Rejected","Accepted": 2 1 2 1 2 2 1 1 2 1 ...
[1] 100
```

Stap 3: het trainen van het model

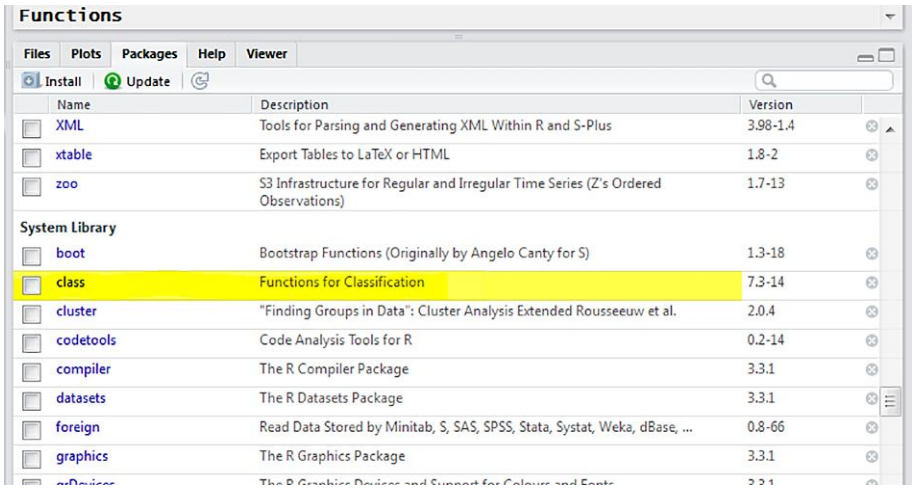
Voor het trainen van het model met de k-NN-methode maken we gebruik van het *package* **class**. Eerst installeren we het *package* met **`install.packages("class")`**, en vervolgens laden we de functies van het pakket met **`library(class)`**.

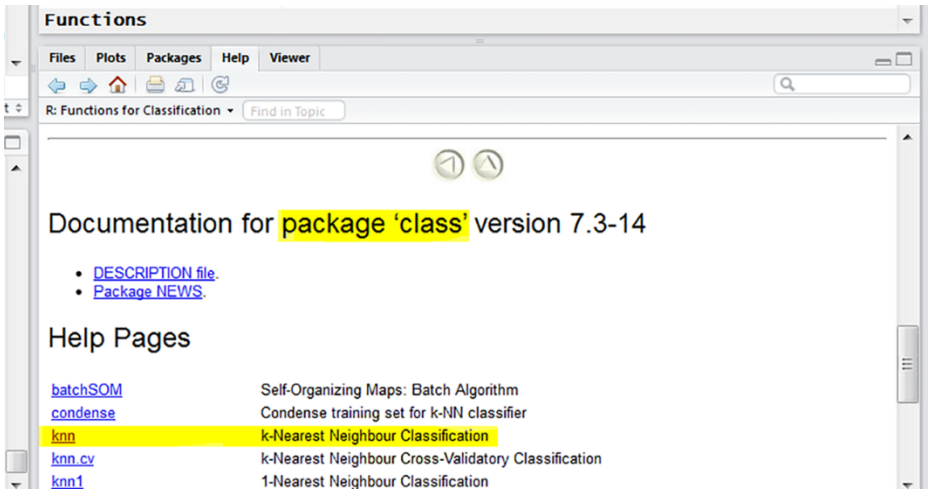
```
install.packages("class") # eenmalig, op uw computer!
library(class)
```

Noot: in RStudio vindt u rechtsonder het scherm "Functions". Onder de tab "Packages" staan alle beschikbare packages. Sommige daarvan hebt u zelf geïnstalleerd, en andere zijn al in het systeem beschikbaar.

De door uzelf geïnstalleerde packages staan in de "user library", en de binnen het systeem beschikbare packages staan in de "system library". Afhankelijk van de versie van R die u in gebruik hebt vallen sommige packages in de tweede categorie. In de hier gebruikte versie behoren de pakketten ***foreign*** *en* ***class*** *tot de system library, en dat betekent dat ze niet hoeven te worden geïnstalleerd met* ***`install.packages()`****! Maar installeren doet geen kwaad. Zoek zelf naar* ***class*** *en klik op het pakket om meer over het package en de beschikbare functies te weten te komen!*

Ook packages in de system library moeten worden aangeroepen met het ***`library()`*** *commando!*





Figuur 2. Meer informatie over de functies in het ***class*** *package.* *Bron OGN.*

Voor het trainen van het model gebruiken we de functie **`knn()`**. Tussen haakjes voeren we de naam in van de trainingset, de testset, en de groeperingen die horen bij de training set. Deze drie bestanden hebben we in het bovenstaande aangemaakt.

```
model <- knn(train=trainingset, test=testset, cl=trainlabel, k1)
```

We moeten aangeven hoeveel *nearest neigbours* (**k1**) dienen te worden gebruikt bij het bepalen van de groepering. Als vuistregel is een goede eerste keuze de vierkantswortel uit het aantal records. Bovendien willen we vermijden dat het algoritme geen beslissing kan nemen omdat evenveel *nearest neigbours* in de twee groepen vallen, door een oneven getal te kiezen. In ons voorbeeld is de wortel uit 500 (afgerond) 22. Het kan in een aantal gevallen voorkomen dat van de 22 *nearest*

neigbours er 11 zijn toegewezen en 11 zijn afgewezen waardoor een toewijzing willekeurig moet gebeuren. Deze situatie kunnen we eenvoudig vermijden door **k1** op 21 of 23 te zetten. Maar laten we **R** vragen om het werk voor ons te doen!

```
k1 <- round(sqrt(nrow(knndata2_n))) # de wortel uit het aantal
observeringen, als vuistregel voor k
k1 <- ifelse(k1%%2==0, k1-1, k1)
cat("De keuze voor k is:", k1,"[ = de wortel uit",x,"]\n")
```

```
> cat("De keuze voor k is:", k1,"[ = oneven getal, ongeveer de wortel
uit",x,"]\n")
De keuze voor k is: 21 [ = oneven getal, ongeveer de wortel uit 500 ]
```

In de eerste regel hierboven bepalen we **k1** als de wortel uit het aantal records in ons databestand. In de tweede regel checken of **k1** een geheel getal is, met de *modulo* operator (%%). Als **k1** een even getal is en dus een veelvoud is van 2, dan is de modulo (de rest, bij deling van **k1** door 2) gelijk aan 0. Is **k1** oneven, dan is de modulo gelijk aan 1. De **`ifelse()`** luidt in woorden: als **k1** een even getal is, trek er dan 1 van af. Als **k1** oneven is laat het dan zoals het is. In de derde regel printen we de gekozen waarde voor **k1** met toelichtende tekst.

Vervolgens vullen we de **`knn()`** -functie in met de namen van de te gebruiken data frames, en **k1**. Het runnen van de eerste regel geeft een voorspellend model. Het object **knn_pred** is een vector met een lengte van 100, met voorspellingen voor de test set. U kunt de inhoud van **knn_pred** bekijken met de diverse functies die we hebben besproken: **`str()`** en **`summary()`**. Probeer dit voor uzelf!

U kunt de 100 voorspellingen afdrukken in de console, zoals we hieronder hebben gedaan.

```
knn_pred <- knn(train=knn_train, test=knn_test, cl=knn_train_label, k1)
```

```
> knn_pred
  [1] Accepteren Afwijzen   Afwijzen   Afwijzen   Accepteren Afwijzen   Afwijzen
  [8] Afwijzen   Accepteren Afwijzen   Afwijzen   Afwijzen   Accepteren Accepteren
 [15] Accepteren Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren
 [22] Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren Afwijzen
 [29] Afwijzen   Accepteren Afwijzen   Afwijzen   Afwijzen   Accepteren Afwijzen
 [36] Accepteren Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren
 [43] Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen
 [50] Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren
 [57] Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren Afwijzen   Afwijzen
 [64] Accepteren Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen   Afwijzen
 [71] Afwijzen   Afwijzen   Accepteren Accepteren Afwijzen   Afwijzen   Afwijzen
 [78] Afwijzen   Accepteren Accepteren Afwijzen   Afwijzen   Afwijzen   Afwijzen
 [85] Afwijzen   Afwijzen   Afwijzen   Afwijzen   Accepteren Accepteren Afwijzen
 [92] Afwijzen   Afwijzen   Afwijzen   Accepteren Accepteren Accepteren Accepteren
 [99] Afwijzen   Accepteren
Levels: Afwijzen Accepteren
```

Stap 4: evalueren van het model

Voor het evalueren van het model zijn we geïnteresseerd in de werkelijke waarde en de voorspelde waarde in de test set. Aangezien we een tweedeling hebben (accepteren of afwijzen), zijn er vier mogelijkheden:

1. Een correct voorspelde acceptatie.
2. Een correct voorspelde afwijzing.
3. Een afwijzing die incorrect voorspeld is als acceptatie.
4. Een acceptatie die incorrect voorspeld is als afwijzing.

		Voorspelling	
		Verwerpen	Accepteren
Werkelijke uitkomst	Verwerpen	(2)	(3)
	Accepteren	(4)	(1)

Tabel 1. De classificatie-tabel.

De groengearceerde cellen in de tabel geven de correcte voorspellingen weer, en roodgearceerde cellen de incorrecte voorspellingen.

Een eerste test voor de waarde van het model is de optelsom van de eerste twee categorieën (de groen gearceerde cellen): hoeveel procent van de records in de test set kan correct worden geclassificeerd met het model?

Maar – tenzij het model 100% correct voorspelt – moeten we ook kijken naar de laatste twee categorieën afzonderlijk. Het is zeer goed denkbaar dat de ene fout categorie ernstiger is dan de andere. In ons voorbeeld: het kan zijn dat het kostbaarder is om ten onrechte een lening toe te wijzen (de bank is dan al het geld kwijt) dan ten onrechte een lening af te wijzen (de bank loopt dan alleen de winst op een lening mis). Wat erger is, is natuurlijk aan de uiteindelijke beslisser, en niet zozeer aan de analist.

R kan met bijvoorbeeld **`table()`** simpele tabellen maken. Voor mooiere tabellen met meer informatie kunnen we gebruikmaken van de functies in get **gmodels** package. Aannemend dat het package al is geïnstalleerd kunnen we het volgende doen.

```
library(gmodels)
CrossTable(knn_test_label,knn_pred,chisq=TRUE, expected=TRUE,
format="SPSS")
```

```
> CrossTable(knn_test_label,knn_pred,chisq=TRUE, expected=TRUE, format="SPSS")

   Cell Contents
|-------------------------|
|                   Count |
|         Expected Values |
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  100

               | knn_pred
knn_test_label | Rejected  | Accepted  | Row Total |
---------------|-----------|-----------|-----------|
      Rejected |       63  |        3  |       66  |
               |   48.840  |   17.160  |           |
               |    4.105  |   11.684  |           |
               |   95.455% |    4.545% |   66.000% |
               |   85.135% |   11.538% |           |
               |   63.000% |    3.000% |           |
---------------|-----------|-----------|-----------|
      Accepted |       11  |       23  |       34  |
               |   25.160  |    8.840  |           |
               |    7.969  |   22.682  |           |
               |   32.353% |   67.647% |   34.000% |
               |   14.865% |   88.462% |           |
               |   11.000% |   23.000% |           |
---------------|-----------|-----------|-----------|
  Column Total |       74  |       26  |      100  |
               |   74.000% |   26.000% |           |
---------------|-----------|-----------|-----------|
```

Bron: OGN.

```
> CrossTable(knn_test_label,knn_pred,chisq=TRUE, expected=TRUE, format="SPSS")

   Cell Contents
|-------------------------|
|                   Count |
|         Expected Values |
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  100

               | knn_pred
knn_test_label | Rejected  | Accepted  | Row Total |
---------------|-----------|-----------|-----------|
      Rejected |       63  |        3  |       66  |
               |   48.840  |   17.160  |           |
               |    4.105  |   11.684  |           |
               |   95.455% |    4.545% |   66.000% |
               |   85.135% |   11.538% |           |
               |   63.000% |    3.000% |           |
---------------|-----------|-----------|-----------|
      Accepted |       11  |       23  |       34  |
               |   25.160  |    8.840  |           |
               |    7.969  |   22.682  |           |
               |   32.353% |   67.647% |   34.000% |
               |   14.865% |   88.462% |           |
               |   11.000% |   23.000% |           |
---------------|-----------|-----------|-----------|
  Column Total |       74  |       26  |      100  |
               |   74.000% |   26.000% |           |
---------------|-----------|-----------|-----------|
```

Bron: OGN.

In de tabel is de werkelijke groepering uitgesplitst naar de voorspelde score.

Van de 66 records die in de praktijk zijn afgewezen zijn er 63 correct voorspeld met het kNN-model; 3 afgewezen aanvragen zijn ten onrechte voorspeld als geaccepteerd.

Van de 34 geaccepteerde aanvragen zijn er 23 correct voorspeld, zodat het totale aantal correcte voorspellingen 63+23=86 bedraagt. Bij de geaccepteerde aanvragen is het aantal incorrecte voorspellingen (11/34=32%) vrij hoog, maar dit kan als minder ernstig worden ervaren dan het aantal ten onrechte geaccepteerde aanvragen.

Stap 5: verbeteren van het model

Binnen de kaders van de data die we ter beschikking hebben zijn er enkele mogelijkheden om te komen tot een verbetering van het model.

Buiten die kaders zouden we kunnen denken aan het verzamelen van belangrijke ontbrekende informatie, of een indeling in meer dan twee groepen (naast afwijzing en acceptatie ook een groep twijfelgevallen). We kunnen ook kijken of modellen met juist minder variabelen ook goed werken (in dat geval hoeven we voortaan niet meer naar overtollige informatie te vragen bij aanvragers).

Hier gaan we twee dingen veranderen: we gaan standaardisatie toepassen in plaats van normalisatie. En we veranderen het aantal *nearest neighbours*.

Standaardisering als alternatief

Standaardisering is een goede optie als sommige van de variabelen scheve verdelingen hebben en/of uitschieters naar boven of naar beneden. In onze dataset zijn er geen variabelen die zich daarvoor lenen en de verwachte verandering in het resultaat is daarom gering. We laten het toch zien omwille van de illustratie.

Voor normalisering gebruikten we de zelfgeschreven **`normaliseer()`** functie in combinatie met **`lapply()`**. Voor standaardisering is de **`scale()`** functie in **R** beschikbaar. De functie kan worden toegepast op een dataframe, en werkt op elke variabele afzonderlijk (dat wil zeggen, we hebben de **`lapply()`** functie niet nodig).

Het label variabele kunnen we ongewijzigd laten, en ook voor de steekproeven maken we gebruik van de bestanden die we al hebben aangemaakt. Het is onverstandig om andere steekproeven te gebruiken omdat veranderingen in de uitkomsten dan het gevolg kunnen zijn van willekeurige veranderingen in de steekproef en niet van veranderingen in de methode!

```
> knndata3_z <- as.data.frame(scale(knndata2[2:11]))
> summary(knndata3_z)
      male              age               exp              skills            busexp             educ        
 Min.   :-0.8362   Min.   :-1.8184   Min.   :-1.3228   Min.   :-0.9954   Min.   :-0.5491   Min.   :-1.01309  
 1st Qu.:-0.8362   1st Qu.:-0.8470   1st Qu.:-0.7486   1st Qu.:-0.9954   1st Qu.:-0.5491   1st Qu.:-1.01309  
 Median :-0.8362   Median : 0.1243   Median :-0.1743   Median :-0.5241   Median :-0.5491   Median :-0.01595  
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000  
 3rd Qu.: 1.1935   3rd Qu.: 0.8529   3rd Qu.: 0.5435   3rd Qu.: 1.1255   3rd Qu.:-0.5491   3rd Qu.: 0.98118  
 Max.   : 1.1935   Max.   : 1.7028   Max.   : 2.9842   Max.   : 1.1255   Max.   : 1.8176   Max.   : 1.97832  
    educpar            saving             city            marital      
 Min.   :-1.0823   Min.   :-0.5675   Min.   :-0.995   Min.   :-1.2600  
 1st Qu.:-1.0823   1st Qu.:-0.5675   1st Qu.:-0.995   1st Qu.:-1.2600  
 Median :-0.1493   Median :-0.5675   Median :-0.995   Median : 0.7921  
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.000   Mean   : 0.0000  
 3rd Qu.: 0.7837   3rd Qu.:-0.5675   3rd Qu.: 1.003   3rd Qu.: 0.7921  
 Max.   : 1.7168   Max.   : 1.7585   Max.   : 1.003   Max.   : 0.7921  
```

Bron: OGN.

Merk op dat de gemiddeldes van alle variabelen inderdaad 0 zijn. Gestandaardiseerde variabelen hebben een gemiddelde van 0 en een standaarddeviatie van 1. Met de **`sd()`** functie kun je verifiëren dat de standaarddeviaties van alle variabelen inderdaad 1 zijn. Bijvoorbeeld:

```
> sd(knndata3_z$age)
[1] 1
> sd(knndata3_z$male)
[1] 1
```

```
# de steekproef vectoren bestaan nog!
knn_train3 <- knndata3_z[train_steekproef ,]
knn_test3  <- knndata3_z[-train_steekproef,]
knn_pred3 <- knn(train=knn_train3, test=knn_test3,
cl=knn_train_label, k1)
CrossTable(knn_test_label,knn_pred3,chisq=TRUE, expected=TRUE,
format="SPSS")
```

```
   Cell Contents
|-------------------------|
|                   Count |
|         Expected Values |
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  100

                | knn_pred3
knn_test_label | Rejected  | Accepted  | Row Total |
---------------|-----------|-----------|-----------|
      Rejected |        63 |         3 |        66 |
               |    48.840 |   17.160  |           |
               |     4.105 |   11.684  |           |
               |   95.455% |    4.545% |   66.000% |
               |   85.135% |   11.538% |           |
               |   63.000% |    3.000% |           |
---------------|-----------|-----------|-----------|
      Accepted |        11 |        23 |        34 |
               |    25.160 |    8.840  |           |
               |     7.969 |   22.682  |           |
               |   32.353% |   67.647% |   34.000% |
               |   14.865% |   88.462% |           |
               |   11.000% |   23.000% |           |
---------------|-----------|-----------|-----------|
  Column Total |        74 |        26 |       100 |
               |   74.000% |   26.000% |           |
---------------|-----------|-----------|-----------|
```

Bron: OGN.

Zoals verwacht maakt het niet uit of we normaliseren of standaardiseren.

Aangepaste waarden voor k

De beste waarde voor *k* is een kwestie van *trial and error*. Bij een hoge waarde voor *k* domineren groepen die in enig opzicht in de meerderheid zijn, en past het model zich minder flexibel aan bij kleinere groepen. Bij een lage waarde voor *k* kunnen allerlei uitzonderingsgevallen (of foute coderingen) een grote invloed hebben op de voorspellingen. Uitgaande van de vuistregel die leidde tot een keuze k=21, kunnen we kijken of voorspellingen systematisch veranderen bij veranderingen in k. We kunnen bijvoorbeeld kiezen voor waardes voor k van 3, 5, 9, 15, 25 en 29 (steeds oneven getallen).

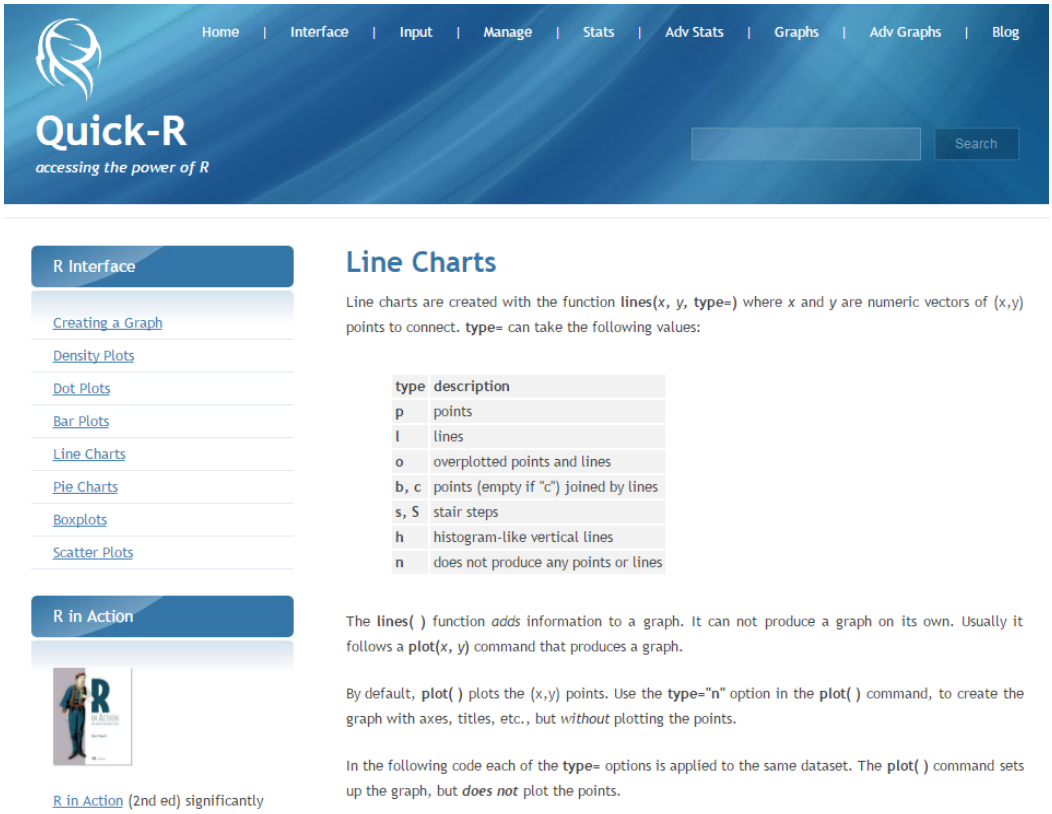
Alle datasets staan gereed, en we kunnen hetzelfde commando een aantal keer herhalen met steeds nieuwe waardes voor *k*. Eleganter is om een *loop* constructie te gebruiken, en gemakkelijk leesbare output te schrijven. Als belangrijkste uitkomst nemen we het aantal correcte voorspellingen. Het aantal correct voorspellingen staat op de diagonaal van de tabel: in de eerste rij en eerste kolom, en in de tweede rij en tweede kolom. We kunnen de getallen uit de diagonaal ophalen met de indices [1,1] en [2,2] van de tabel. Het object **corn** is het aantal correcte classificeringen, en **corp** het percentage. Met de `cat()` functie schrijven we de uitkomsten in leesbare vorm.

```
kset <- c(3, 5, 9, 15, 21, 25, 29)
ssize <- 100

for (val in kset) {
  knn_pred <- knn(train=knn_train,
test=knn_test,cl=knn_train_label,val)
  tb_true_neg <- table(knn_test_label,knn_pred)[1,1]
  tb_true_pos <- table(knn_test_label,knn_pred)[2,2]
  corn = tb_true_neg + tb_true_pos
  corp = 100*round((corn / ssize), 4)
  cat("Correct voorspeld (k=", val,"):",corn, "(",corp,"%)\n")
```

```
Correct voorspeld (k= 3 ): 88 ( 88 %)
Correct voorspeld (k= 5 ): 89 ( 89 %)
Correct voorspeld (k= 9 ): 91 ( 91 %)
Correct voorspeld (k= 15 ): 88 ( 88 %)
Correct voorspeld (k= 21 ): 86 ( 86 %)
Correct voorspeld (k= 25 ): 85 ( 85 %)
Correct voorspeld (k= 29 ): 86 ( 86 %)
```

Het beste resultaat vinden we bij *k*=9. We kunnen de resultaten grafisch weergeven in een lijndiagram, met op de verticale as (y-as) het percentage correct voorspellingen, en op de horizontale as (de x-as) de verschillende waarden voor het aantal *neighours* (k). Voor het maken van grafieken zijn er standaard functies in **R**, zoals **`plot()`**. Voor voorbeelden kun je het beste zoeken in **Quick-R** (voor lijndiagrammen, http://www.statmethods.net/graphs/line.html).



Figuur 3. Screenshot van Quick-R: line charts. *Bron: Quick-R/ http://www.statmethods.net.*

We zoeken naar een voorbeeld van wat we willen hebben, kopiëren het in ons script, en passen het aan onze gegevens aan. In de **`plot()`** functie staan *x* en *y* voor de waarden op de x- en y-as. Verder kunnen we titels aangeven voor de grafiek, en voor de x- en de y-as.

```
plot(x, y, type="n", main="grafiek titel", xlab="x-as titel", ylab="y-as titel" )
```

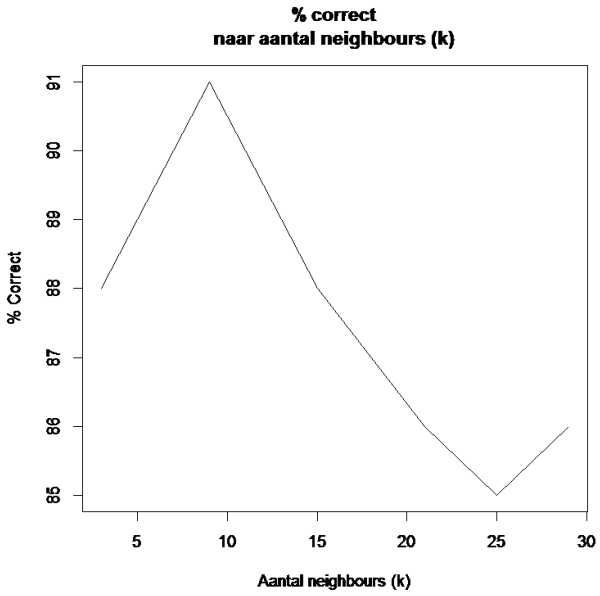
De waarden voor *x* staan al in de vector **kset**. De waarden voor *y* nemen we over uit de output:

```
correct <- c(88,89,91,88,86,85,86)
```

Aangezien we een lijndiagram willen kiezen we voor de optie **`type="l"`**. Het commando wordt dan:

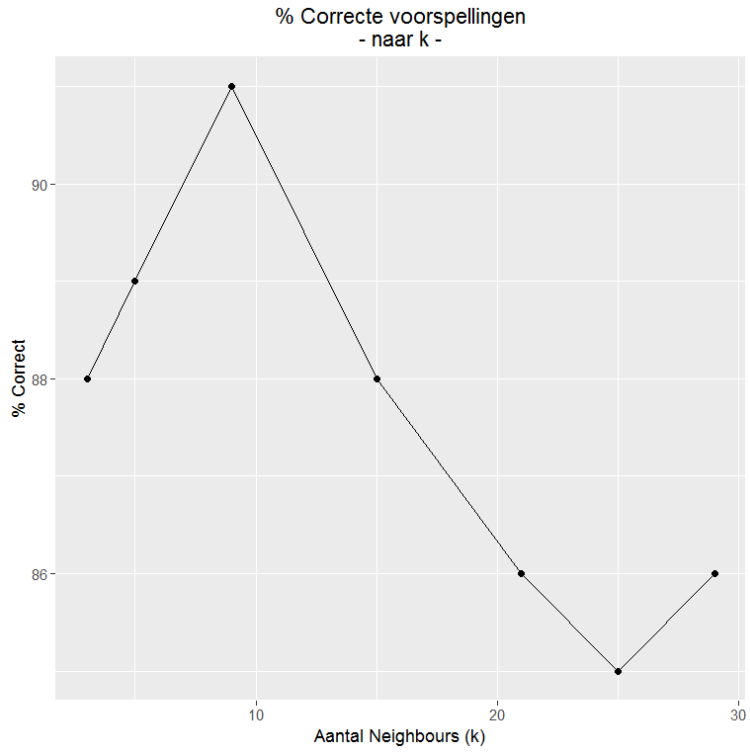
```
plot(kset, correct, type="l", main="% correct\nnaar aantal neighbours (k)", xlab="Aantal neighbours (k)", ylab="% Correct")
```

De "\n" in de grafiektitel zorgt ervoor dat wat volgt op een volgende regel wordt afgedrukt.



*Figuur 3.4(a). Bepaling van de optimale waarde voor k (met **plot()**).* *Bron: OGN.*

Sommigen geven de voorkeur aan mooiere grafieken van publicatiekwaliteit. Daarvoor kan gebruik worden gemaakt van de vele functies (met de vele opties) in het *package* **ggplot2**. Het gaat te ver om alle mogelijkheden van **ggplot2** te bespreken, maar voor een overzicht en voorbeelden is **Cookbook-R** een goed startpunt omdat vrijwel alle voorbeelden gebaseerd zijn op het gebruik van dit *package* (zie http://www.cookbook-r.com/Graphs/). Een voorbeeld van hetzelfde lijndiagram met **ggplot2** is hieronder weergegeven; de commando's zijn onderdeel van het script.



*Figuur 3.4(b). Bepaling van de optimale waarde voor k (met **ggplot()**).* *Bron: OGN.*

Als we tevreden zijn met het model en het willen gaan toepassen op nieuwe aanvragers, dan kan een laatste stap zijn om maximaal gebruik te maken van alle informatie (alle 500 records) bij de training van het model, om het vervolgens toe te passen op nieuwe aanvragers.

K-means clustering

In de k-NN-methode gingen we uit van een bekende groepering van objecten, met als uitdaging een link te leggen met kenmerken die kunnen worden gebruikt als voorspeller van het deel uitmaken van een groep. We doen dat door te kijken naar de groep waarvan soortgelijke objecten (*neigbours*) in meerderheid deel uitmaken. We kunnen het model trainen omdat we al gegevens hebben over de groepering (in ons voorbeeld: de verworpen en geaccepteerde aanvragen).

In sommige gevallen is het juist de uitdaging om groepen te vinden van objecten die op elkaar lijken in termen van een aantal geselecteerde criteria. Het verschil ten opzichte van de k-NN methode is dat we niet van tevoren weten welke groepen er zijn, en zelfs niet hoeveel groepen er zijn. Een tweede verschil is dat we nadrukkelijk wel zijn geïnteresseerd in hoe de groepen eruit zijn. Bij de k-NN methode lag het accent op de correcte classificering maar bekommerden we ons niet om hoe twee groepen (geaccepteerde en afgewezen leningen) verschilden in termen van leeftijd, geslacht, ervaring, vaardigheden, enzovoorts.

Het algoritme

De aanpak in K-means clustering heeft veel gemeen met de k-NN-methode, maar er zijn ook verschillen.

De letter *k* heeft in de twee methodes (k-NN en K-means) een verschillende betekenis: in de k-NN methode staat de *k* voor het aantal *nearest neigbours* waarmee het te classificeren object wordt vergeleken, en in K-means staat de *k* voor het aantal clusters (groepen) dat we willen samenstellen.

Net als in die methode maken we gebruik van het begrip afstand. Bij K-means worden objecten toegewezen aan een cluster op basis van de Euclidische afstand tussen het object en het midden van het cluster, ook wel aangeduid als het cluster *centroid*. De uitdaging is dat we op voorhand niet weten hoeveel clusters er zijn, en dat we dus ook niet weten hoe de clusters eruit zien. Daarom werken we in twee stappen. Allereerst bepalen we het optimale aantal clusters, en vervolgens bepalen we startwaardes voor ieder cluster.

Bepalen van het aantal clusters

Het aantal clusters kan op drie manieren worden bepaald.

De eerste manier is een vuistregel die het aantal clusters stelt op de vierkantswortel uit de helft van het aantal objecten. Als we 200 objecten willen clusteren, dan zou het aantal clusters $\sqrt{200/2} = 10$ zijn. We hebben dan relatief veel clusters (10) ten opzichte van de gemiddelde omvang van de clusters (20 objecten per cluster). Vermoedelijk stamt de vuistregel uit de tijd dat clusteranalyse vooral werd toegepast op vrij kleine bestanden. In de tijd van *big data* met bestanden van duizenden objecten (transacties; personen; berichten) zou het aantal clusters enorm toenemen. Met 2 miljoen objecten is het aantal groepen 1000, en is er nog nauwelijks sprake van compacte, overzichtelijke informatie.

Een betere methode – maar niet altijd mogelijk – is om het aantal clusters te bepalen aan de hand van bekende indelingen die bijvoorbeeld worden gebruikt in andere onderzoeken. Als het gaat om de intensiteit van het gebruik van een product of dienst zouden we een driedeling kunnen gebruiken om op zoek te gaan naar *light, medium* en *heavy users*. Voor bepaalde producten zijn er segmentatiemodellen die de kopers indelen naar motieven en gedragskenmerken. In toerisme

bijvoorbeeld maakt het onderzoeksbureau Motivaction gebruik van een indeling in 5 categorieën (*traditionals; mainstreamers; upperclass quality seekers; postmoderns* en *achievers*).

Als we geen houvast hebben aan bestaande indelingen kunnen we het aantal clusters statistisch onderbouwen. Het doel van clusteranalyse is om groepen te vormen van objecten die (i) zoveel mogelijk op elkaar lijken, en (ii) zoveel mogelijk verschillen van objecten in de andere clusters. We zijn, anders gezegd, geïnteresseerd in de homogeniteit van een cluster. De homogeniteit kan worden bepaald aan de hand van de variatie van de kenmerken van de objecten binnen het cluster: hoe geringer die variatie is, des te homogener het cluster is. De variatie zal afnemen als de clusters kleiner worden; in het extreme geval is één enkel object zijn eigen cluster en is de variatie gereduceerd tot 0. Helaas zijn we dan niets opgeschoten. We accepteren een zekere variatie binnen het cluster, zolang de clusters maar voldoende herkenbaar zijn, en afwijken van andere clusters.

Een statistische maatstaf voor de variatie is de zogeheten *within group sum of squares* (**wss**). We kunnen een plot maken van de **wss** naar het aantal clusters. Vaak zien we dan een knik (de "elleboog") in de curve: de **wss** daalt eerst snel maar vlakt dan af. Zodra de curve afvlakt is het optimale aantal clusters bereikt. Het maken van meer (en dus kleinere) clusters leidt dan nauwelijks tot homogenere clusters, en vaak tot afsplitsing van kleine clusters die nauwelijks interessant zijn.

In **Quick-R** (http://www.statmethods.net/advstats/cluster.html) staat uitgelegd hoe we een "elleboogcurve" kunnen maken. Laten we het toepassen op een voorbeeld.

Een reisbureau biedt een veelheid van reizen aan, waaronder reizen die gebaseerd zijn op de principes van "sustainable tourism". Om erachter te komen wat toeristen belangrijk vinden wordt gebruikgemaakt van gegevens uit een korte enquête waaruit scores worden afgeleid op het belang van zes kenmerken (bereidheid meer te betalen voor "sustainable tourism"; het belang van "sustainable tourism"; de prijs van de reis; de kwaliteit van de accommodatie; de kwaliteit van de dienstverlening; en het belang van natuur en omgeving).

De gegevens van 200 respondenten worden gelezen uit een in STATA gemaakt bestand. Naast de zes scores hebben we ook informatie over geslacht, leeftijd (jong/middelbaar/oud); inkomen (laag/midden/hoog), en nationaliteit (een dummy voor Europese Unie; en een dummy voor Verenigde Staten van Amerika).

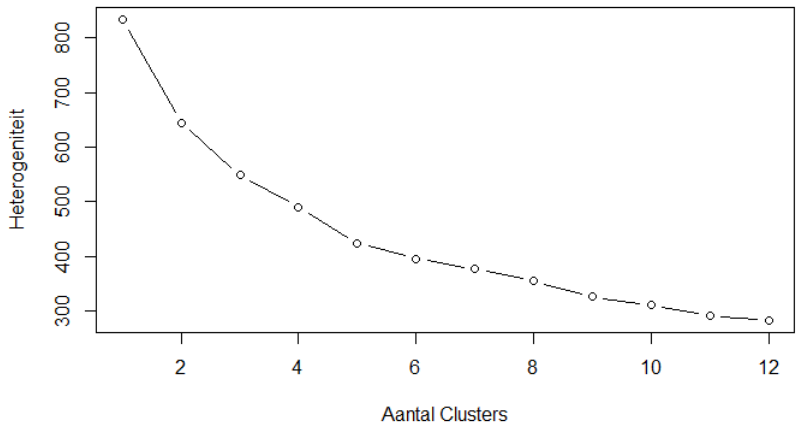
De onderstaande regels lezen de data, en geven de structuur met het **`str()`** commando. Ga voor jezelf na met het **`summary()`** commando dat alle waarden in het verwachte bereik liggen. Ook zijn er geen ontbrekende waarden.

```
> library(foreign)
> toerist <- read.dta("H2_LOI_RT.dta",convert.factors=FALSE)
> str(toerist)
'data.frame':   200 obs. of  11 variables:
 $ male   : num  1 0 1 1 0 0 0 0 0 0 ...
 $ age    : int  2 2 2 2 3 1 3 1 3 3 ...
 $ income: num  2 1 2 1 2 2 1 2 1 2 ...
 $ eu     : num  1 0 1 1 0 1 0 0 1 0 ...
 $ us     : num  0 1 0 0 1 0 0 0 0 1 ...
 $ wtp    : num  3 2 2 2 2 3 2 3 5 2 ...
 $ price : num  4 3.5 3 2 3.5 3 1 4.5 3 3 ...
 $ accom : num  2.5 5 4.5 5 4.5 4 5 4 3.5 4 ...
 $ serv   : num  3 4 4 2 3 4 5 4 3 4 ...
 $ rt     : num  4.6 4.6 4.6 3.6 4.4 ...
 $ geo    : num  5 5 5 5 5 5 5 5 5 4 ...
```

Ons doel is de toeristen te clusteren op wat zij belangrijk vinden, zoals aangegeven in de laatste 6 variabelen in het dataframe (kolommen 6 t/m 11). We maken een dataframe met alleen deze 6 kenmerken. Vervolgens maken we een plot van de **wss** naar het aantal clusters.

```
k <- 12 # bepaal aantal clusters
wss <- (nrow(toeristBelang)-1)*sum(apply(toeristBelang,2,var))
```

```
for (i in 2:k) wss[i] <- sum(kmeans(toeristBelang, centers=i)$withinss)
plot(1:k, wss, type="b", xlab="Aantal Clusters", ylab="Heterogeniteit")
```



Figuur 5. Bepaling van het optimale aantal clusters. *Bron: OGN.*

Uit de plot blijkt dat de heterogeniteit van de clusters afneemt als we meer clusters toevoegen. De afname vlakt af: de daling is het sterkst als we van 1 naar 2 clusters gaan. De daling is iets minder sterk van 2 naar 3; enzovoorts. Er is in dit geval niet een heel duidelijke knik in de curve. Wel is de heterogeniteit bij 5 clusters gehalveerd, en is de winst bij uitbreiding van 5 naar 12 clusters gering (van ongeveer 400 naar 300); 5 clusters lijkt een goede keuze.

Clusteranalyse is een exploratieve methode: we kennen op voorhand de clusters niet. We noemen dit in Machine Learning ook wel *unsupervised learning*. De methode genereert nieuwe data (clusters) die door de analist moeten worden benoemd. De waarde van de analyse schuilt in de interpretatie. De uitkomsten kunnen niet worden toegepast op een test set! Vanwege het exploratieve karakter is het verstandig om te experimenteren met andere keuzes voor het aantal clusters.

Experimenteren is ook van belang omdat K-means verschillende algoritmes heeft voor het bepalen van clusters. Een iteratieve methode is om willekeurig *k* objecten te selecteren als initieel cluster, en vervolgens andere objecten aan het cluster toe te wijzen, waarna het midden (*centroid*) van het cluster opnieuw wordt bepaald. De procedure eindigt zodra geen objecten meer van cluster veranderen. De uitkomst van K-means is gevoelig voor de willekeurige selectie van objecten voor de initiële oplossing, en het is daarom aan te raden de procedure te herhalen met andere startpunten, en te kijken hoe robuust de oplossing is.

Standaardiseren of normaliseren

Net als in de k-NN methode moeten de kenmerken waarop wordt geclusterd in vergelijkbare eenheden zijn gemeten. In dat geval is dat geen probleem omdat alle 6 kenmerken zijn uitgedrukt op een 5-puntsschaal; normaliseren of standaardiseren is niet nodig.

Stap 1: prepareren van de data

Stap 1 hebben we eigenlijk al doorlopen. We hebben de data ingelezen. Het gaat om 200 toeristen en hun gegevens over leeftijd, geslacht, inkomen, en nationaliteit, en hun mening over het belang van een zestal kenmerken van toeristische bestemmingen en verblijven. Omdat we de toeristen alleen willen clusteren op de zes kenmerken, hebben we een bestand gemaakt met alleen die variabelen.

We hebben al geconstateerd dat normaliseren of standaardiseren van de data niet nodig is, in dit geval. Overigens kan het geen kwaad om het wel te doen, en veel onderzoekers standaardiseren de data altijd, met als reden dat de clustermiddens dan gemakkelijker te interpreteren zijn.

Stap 2: het trainen van het model

Aangezien we bij veel machine-learningmethoden uitgaan van een training- en een testset, suggereert deze stap – het trainen van het model – dat er ook een evaluatie volgt met een testset. Maar dat is niet het geval. De K-means methode is *unsupervised learning*, waarbij een cluster resulteert als nieuw gegeven. Een eventuele test set heeft dat gegeven niet, en we kunnen dus ook niet evalueren of het cluster goed wordt voorspeld!

Wel zullen we het model evalueren aan de hand van andere kenmerken in de dataset. Zijn de clusters ook verschillend in de zin van achtergrondkenmerken (zoals leeftijd en nationaliteit) die het mogelijk maken om, in marketingtermen gesproken, de segmenten (clusters) met een gedifferentieerde marketingstrategie te benaderen?

Het trainen van het model doen we met functies uit het package **stats**; dit is (net als bijvoorbeeld het package **foreign** een package uit de *system library* die niet apart behoeft te worden geïnstalleerd, maar wel moet worden aangeroepen met het **`library()`** commando.

De functie uit het **stats** package is **`kmeans()`**; tussen haakjes geven we de dataset en het aantal clusters. De uitkomsten schrijven we in een object dat alle informatie bevat die we nodig hebben - zoals we zullen zien. Een van de kenmerken is de omvang van de clusters (het aantal toeristen in ieder cluster). De omvang van de clusters is een vector (**size**) in het cluster object. We zien dat de omvang van de clusters uiteenloopt van 23 (cluster 3) tot 76 (cluster 5). Voor de marketingmanager is het natuurlijk belangrijk te weten dat er een homogeen cluster is, en dat dat cluster (marktsegment) voldoende groot is om een aparte strategie op los te laten!

Om dezelfde uitkomst te genereren gebruiken we weer het **`set.seed()`** commando. Het niet gebruiken van dit commando of het veranderen van het startnummer, geeft een iets andere uitkomst: de uitkomst is gevoelig voor de startset! Test daarom de robuustheid van de analyse door herhaling met andere startwaarden. Ter geruststelling, het komt maar zelden voor dat herhaling wezenlijk andere resultaten oplevert, en zeker niet als de variabelen geen uitschieters laten zien. Met een 5-puntsschaal is het sowieso niet mogelijk om uitschieters te hebben, aangezien de schaal begrensd is. Het is om dezelfde redenen van groot belang de data te controleren op fouten: een codeerfout (een record met een waarde van 55, op een schaal van 1-5) kan gevolgen hebben voor de uitkomst.

```
> library(stats)
> set.seed(35753)
> toeristCluster <- kmeans(toeristBelang, 5)
> toeristCluster$size
[1] 31 31 23 39 76
```

Minstens even interessant is het om te kijken naar wat de vijf clusters precies voorstellen. De meest inzichtelijke manier is het maken van een overzicht van de clustermiddens (de *centroids*). Deze informatie staat in het zelfde object (**toeristCluster**), in de matrix **centers**. We kunnen deze informatie direct in de console printen:

```
> toeristCluster$centers
       wtp    price    accom     serv       rt      geo
1 4.354839 3.919355 4.467742 4.145161 4.283871 4.677419
2 2.774194 1.870968 4.596774 4.322581 3.956452 4.838710
3 4.521739 3.086957 3.782609 2.869565 4.000000 4.826087
4 2.205128 3.141026 3.615385 2.641026 4.082051 4.743590
5 2.263158 3.861842 4.381579 4.157895 4.238158 4.802632
```

Iedere rij stelt een cluster voor. Het eerste cluster (met, zoals we zagen, 31 toeristen), heeft hoge gemiddelde scores op alle variabelen. Het tweede cluster scoort laag op de **wtp** (bereidheid om extra

te betalen), en op het belang van de prijs van de reis; de scores op kwaliteit van accommodatie en dienstverlening (**accom** en **serv**) zijn daarentegen hoger dan in de andere clusters. Voor alle clusters zien we dat de scores op **geo** hoog zijn (4.67 of hoger, op een 5-puntsschaal). Dit aspect is derhalve nauwelijks onderscheidend en kan wellicht in een volgende run worden weggelaten.

Stap 3: het evalueren van het model

Aangezien het model exploreert, en zoekt naar homogene groepen toeristen, kunnen we niet aan de hand van een test set controleren of het model goed voorspelt. Er valt immers niets te voorspellen! De evaluatie gebeurt aan de hand van de antwoorden op drie vragen:

- Zijn de clusters groot genoeg om rekening mee te houden (*substance*)?
- Zijn er duidelijke verschillen tussen de clusters, op grond waarvan we de clusters kunnen benoemen (*interpretation*)?
- Geven de clusters informatie die aanleiding geeft tot specifieke acties (*actionable*)?

Velen zullen zeggen dat uiteindelijk alleen de laatste vraag belangrijk is. Als we niets met de informatie kunnen doen dan is de analyse op zijn best leuk om te weten, maar niet nuttig.

Voor de interpretatie schetsen we een profiel van de clusters. Nu we weten dat er clusters zijn met andere motieven en gedragskenmerken, kunnen we nagaan wie er vooral deel uitmaken van de clusters. Zijn het oude of jonge toeristen? Rijke of arme? Europeanen of Amerikanen?

Voor het schetsen van de profielen kunnen we een vector uit het object **toeristCluster** lichten die voor alle toeristen het cluster bevat waarin de toerist is ingedeeld. Deze vector plakken we aan het oorspronkelijke bestand waarin we ook informatie hebben over leeftijd, geslacht, inkomen en nationaliteit. In de tweede regel vragen we **R** voor de eerste 6 toeristen in het bestand de informatie af te drukken over cluster, leeftijd, inkomen, en nationaliteit.

```
toerist$cluster <- toeristCluster$cluster
toerist[1:10, c("cluster","age","income","eu","us")]
```

```
   cluster age income eu us
1        4   2      2  1  0
2        5   2      1  0  1
3        5   2      2  1  0
4        4   2      1  1  0
5        4   3      2  0  1
6        5   1      2  1  0
```

Voor een totaalbeeld kunnen we gebruik maken van de **`aggregate()`** functie die statistische informatie geeft (bijvoorbeeld de gemiddelde leeftijd, of de gemiddelde nationaliteit) voor ieder cluster. Natuurlijk klinkt gemiddelde nationaliteit vreemd maar we zullen laten zien wat we ermee bedoelen.

```
> aggregate(data=toerist, age     ~ cluster, mean)
  cluster      age
1       1 1.967742
2       2 2.096774
3       3 2.217391
4       4 1.871795
5       5 2.000000
> aggregate(data=toerist, eu      ~ cluster, mean)
  cluster        eu
1       1 0.6451613
2       2 0.5161290
3       3 0.6086957
4       4 0.6923077
5       5 0.6315789
> aggregate(data=toerist, us      ~ cluster, mean)
  cluster        us
1       1 0.2258065
2       2 0.2580645
3       3 0.2173913
4       4 0.1794872
5       5 0.2236842
```

Bron: OGN.

We hebben geen informatie over de precieze leeftijd van de toeristen. We weten alleen of ze jonger dan 35 zijn (code 1), tussen 35 en 55 (code 2), of ouder dan 55 (code 3). Een indicatie voor de gemiddelde leeftijd is dus een getal tussen 1 en 3. De verschillen zijn niet heel groot, behalve dat cluster 4 een aanzienlijk lagere score heeft dan cluster 3. We zouden in de profielschetsen van clusters 3 en 4, relatief oud en relatief jong kunnen opnemen.

Wat betreft nationaliteit valt op dat cluster 2 relatief weinig Europeanen telt. Aangezien de variabele **eu** een dummyvariabele is, is slecht 52% van de toeristen in cluster 2 Europeaan, vergeleken met meer dan 60% in alle andere clusters, en zelfs 69% in cluster 4! In lijn hiermee zijn relatief veel toeristen in cluster 2 Amerikaan en relatief weinig in cluster 4.

We kunnen alle informatie samenvatten in een overzicht zoals in figuur 7. In de groen en rood gearceerde boxen in staat weergegeven op welke kenmerken de clusters een relatief hoge (+; groen gearceerd) of lage (-; rood) scores hebben. Cluster 2 bijvoorbeeld let sterk op de kwaliteit van accommodatie en dienstverlening, en vindt prijs minder belangrijk. Ook zijn de in dit cluster ingedeelde toeristen minder bereid om meer te betalen voor toeristische accommodaties die op maatschappelijke verantwoorde wijze worden geleid (**wtp** staat voor *willingness to pay*). Dit segment bestaat vooral uit Amerikaanse toeristen met een relatief hoog inkomen (de grijs gearceerde box, onderaan). We zouden op dit segment als label “de rijke toerist” kunnen plakken.

Figuur 6. Interpretatie van de clusters. *Bron: OGN.*

Samenvatting

In dit hoofdstuk hebben we twee aan elkaar verwante methoden besproken die kunnen worden gebruikt om objecten te groeperen op basis van hun gelijkenis.

In de k-NN-methode gaan we uit van een vooraf bepaalde indeling in groepen. Wij weten bijvoorbeeld van een groep aanvragers van leningen of de lening wel of niet wordt toegewezen. Vervolgens kijken we of we aan de hand van een aantal kenmerken kunnen voorspellen tot welke groep het object behoort. We doen dat door te kijken naar de groep waartoe de meerderheid van k-buren (objecten die veel gelijkenis tonen met het te classificeren object) behoort. We hebben geleerd hoe we de voorspelkracht kunnen evalueren, en op welke manieren we het model kunnen verbeteren.

Bij K-means clustering is de aanpak wezenlijk anders. We weten nu niet van tevoren hoeveel clusters er zijn, laat staan tot welke cluster objecten behoren. De geformeerde clusters voegen informatie toe.

Beide methodes zijn relatief eenvoudig maar zeer krachtig, en populair. De K-means clustering is niet de meeste geavanceerde manier om objecten te clusteren maar zeker bij omvangrijke gegevensbestanden erg efficiënt. De k-NN-methode wordt beschouwd als *lazy learning* omdat er door het toewijzen van objecten aan een meerderheid eigenlijk niets wordt geleerd over het proces. De K-means methode is *unsupervised learning* in de zin dat het getrainde model niet kan worden getest, omdat we nu eenmaal niet weten tot welke clusters objecten behoren. Dat maakt de methode bij uitstek geschikt voor exploratief (verkennend) onderzoek. Een harde maatstaf voor evaluatie zoals bij de k-NN-methode wel aanwezig is, ontbreekt voor K-means clustering.