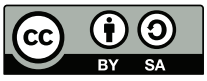
Pieter Delobelle
Anton Danneels

Internetapplicaties

KU Leuven

I HAVE CLEARED THE AUGEAN STABLES OF ASTRONOMY OF CYCLES AND SPIRALS, AND LEFT BEHIND ME A SINGLE CARTLOAD OF DUNG.

—JOHANNES KEPLER



Inhoudsopgave

Introductie

Deze cursus hoort bij het vak *internetapplicaties* van Tony Wouters, gegeven op de Technologiecampus Gent van de Katholieke Universiteit van Leuven.

De broncode van de gebruikte scripts is ook beschikbaar op `https://github.com/iPieter/internetapplicaties`.

Web data mining

Waarom data mining

Met de opkomst van het internet, of meer bepaald het *World Wide Web*, is er een explosie aan nieuwe data beschikbaar gekomen. Het web is een onderdeel geworden van het dagelijkse leven waarbij miljarden mensen met elkaar verbonden zijn via biljoenen documenten. Deze documenten zijn aan elkaar gelinkt via hyperlinks. Hierdoor is een nieuw soort maatschappij ontstaan: de virtuele maatschappij.

Ondanks deze grote hoeveelheid nieuwe info is er een fenomeen ontstaan waarbij er een grote hoeveelheid data beschikbaar is die eigenlijk niet gebruikt wordt: de *data gap*. De opslag van gegevens wordt steeds goedkoper, maar het analyseren van de data blijkt niet eenvoudig.

Er zijn drie grote elementen die de analyse moeilijk maken. Een eerste component is dat de data, beschikbaar op het internet, vaak geen vaste structuur heeft. Men moet dus eerst de data herwerken zodat men een vast patroon heeft dat gebruikt wordt in een bepaald algoritme. Een tweede probleem is dat de data vaak veel ruis bevat, data die men niet nodig heeft. Tot slot is de data steeds veranderend: het is dynamisch.

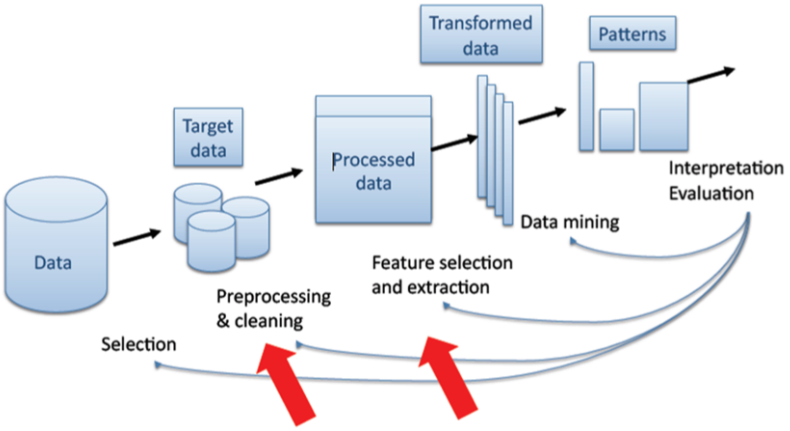
Om deze grote hoeveelheid data toch te verwerken gebruikt men *web data mining*. Dit is een automatische manier om informatie en kennis te extraheren uit het web. Deze techniek heeft 3 toepassingen:

- **Web content mining**: Het analyseren van data op basis van de inhoud van webpagina's.
- **Web usage mining**: Het analyseren van gedragspatronen op basis van de gebruikers logbestanden.
- **Web structure mining**: Het analyseren van het data op basis van hyperlinks.

Data mining

Voor we verder ingaan op het concept van web mining, bekijken we eerst *data mining*. Data mining is het proces om patronen of kennis te ontdekken in databronnen. Het staat daarom ook bekend als

KDD: *Knowledge discovery in databases*. Dit proces is multidisciplinair en beslaat domeinen zoals *machine learning*, statistiek, databanken, visualisatie . . . Het proces is samengevat in de volgende figuur:



Figuur 1: Het data mining proces.

Zoals we zien bestaat dit proces uit meerdere stappen.

1. Eerst wordt de data verzameld. Dit kan uit een database komen of kan bestaan uit verzameling van allemaal bronnen

2. In de tweede stap wordt een selectie gemaakt uit de data. Stel bijvoorbeeld dat we tweets analyseren. Indien we dit opvragen, dan krijgen we extra metadata zoals de locatie en of de tweet een antwoord is op iemand anders. We kunnen er dan voor kiezen om enkel tweets te selecteren die geen antwoord zijn.

3. Vervolgens is er de *data cleaning*-stap. Hierbij wordt ontbrekende info aangevuld of wordt er data weggelaten indien blijkt dat deze niet bruikbaar is.

4. Nadat er enkel schone data overblijft moet deze nog worden omgezet naar data die gebruikt kan worden voor mining. Hierbij gebruikt men technieken zoals aggregatie van data, normalisatie.

5. De data is uiteindelijk klaar voor *data mining*. Hierbij worden technieken zoals clustering en *association* toegepast zodat er patronen kunnen vormen in de data.

6. Vervolgens wordt de data mining geëvalueerd en eventueel gevisualiseerd.

7. Uiteindelijk wordt er gekeken of de data effectief nuttig is en tot meer kennis leidt.

Merk op dat dit proces niet lineair is. Stel dat in stap 6 blijkt dat de mining slechte resultaten oplevert, dan kan men terugkeren naar de vorige stappen en kijken of er bijvoorbeeld geen nieuwe selectie

gemaakt moet worden. Dit proces gaat door tot men resultaten bereikt waardoor men effectief kennis bijkrijgt.

Data mining bestaat dus uit een aantal technieken die in deze cursus besproken zullen worden. Hier onderscheid men onder andere:

- Supervised learning (~classificatie): Hierbij beschikt men over een reeds geclassificeerde database waarmee het algoritme kan leren en een database om het algoritme te testen.
- Unsupervised learning(~clustering): Een groep van algoritmen waarbij het algoritme zelf bijleert over de dataset.
- Association rule mining: Hierbij maakt men gebruik van technieken zoals: stel dat gebruiker 1 A en B koopt, en gebruiker 2 koopt A. Wat is dan de kans dat gebruiker 2 ook B zal kopen?
- Sequential pattern mining: Een voorbeeld hiervan zijn DNA sequences waarbij blijkt dat sommige combinaties vaak voorkomen.

Data mining vs Web mining

Data mining werkt dus in op gestructureerde data. Dit is dus een probleem voor de data verzamelt op het web omdat dit vaak ongestructureerd is. Hoe meer structuur er is in de data, hoe rijker en complexer de query's kunnen zijn. Wegens deze verschillen maakt men dus een onderscheid tussen *web mining* en *data mining*. Enkele voorbeelden van *web mining* zijn:

- Tekst zoals contact info halen uit webpagina's
- Video's uit webpagina's halen
- Tabellen op bijvoorbeeld Wikipedia gebruiken om een analyse uit te voeren.

Besluit

Algemeen kan men dus stellen dat web content mining een brede waaier is waarbij data mining ook gebruikt kan worden. De technieken die men hierbij onderscheid zijn:

- Unstructured data mining(=information retrieval)
- Structured data mining op tabellen en databanken
- Semi-structured: er is wel iets van structuur, maar deze moet nog gedefinieerd worden
- Multimedia mining

De structuur van de data bepaalt het type van algoritmen. Gestructureerde data is vaak eenvoudiger te analyseren en leidt tot een grotere verzameling kennis.

Association rules & sequential patterns

Association rules

Deze *machine learning*-techniek is ontstaan dankzij supermarkten, welke grote datasets hadden met aankopen van verschillende klanten. Voor deze winkels is het interessant om patronen te vinden in deze aankopen, waar ze dan kunnen op inspelen met commerciële aanbiedingen. Het doel is om *association rules* te vinden tussen verschillende aankopen. Een voorbeeld is dat **als** een klant wijn koopt, **dan** zal de klant ook kaas zal kopen. Dit noteert men formeel door:

$$wijn \Rightarrow kaas \tag{1}$$

Ieder product dat de klant aankoopt wordt een item i genoemd, welke in de ongeordende set van alle aangeboden items I zit:

$$I = \{i_1, i_2, i_3, .., i_m\} \tag{2}$$

Daarnaast worden alle transacties (afrekenen aan de kassa) bijgehouden, waarbij elke transactie t bestaat uit een subset van de verzameling van items I.

$$T = \{t_1, t_2, t_3, .., t_n \mid t \subseteq I\} \tag{3}$$

Een *association rule* is dus een regel tussen twee subsets X en Y van de itemset I, welke—uiteraard—verschillende items bevatten. Want niemand is geïnteresseerd in het feit dat als klanten brood kopen, dan ook brood kopen.

$$X \Rightarrow Y \text{ met } X, Y \subseteq I \text{ en } X \cap Y = \emptyset \tag{4}$$

Support en confidence

Om nuttige regels te kunnen onderscheiden van alle mogelijke regels, dienen deze regels een minimale drempelwaarde te halen voor bepaalde parameters. Twee veelgebruikte parameters zijn *support* en *confidence*.

Support Deze parameter geeft aan hoe vaak een itemset A voorkomt in de dataset van transacties T.

$$\text{supp}(X \cup Y) = \frac{|X \cup Y|}{|T|} \tag{5}$$

Om een *association rule* te kunnen opstellen, dient een frequente itemset gevonden te worden. Dit is een itemset $X \cup Y$ welke een bepaalde drempelwaarde overschrijdt.

Confidence Deze parameter is een indicatie van hoe vaak een regel $X \Rightarrow Y$ waar blijkt te zijn. Dus hoe vaak alle items $X \cup Y$ voorkomen ten opzichte van hoe vaak X in totaal voorkomt.

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} = \frac{|X \cup Y|}{|X|} \tag{6}$$

Voorbeeld Bij wijze van oefening beschouwen we de volgende set met zeven transacties, waarop we de regel $\{eieren, kleding\} \Rightarrow \{melk\}$ toetsen.

$t_1 = \{wijn, eieren, melk\}$
$t_2 = \{wijn, kaas\}$
$t_3 = \{kaas, laarzen\}$
$t_4 = \{wijn, eieren, kaas\}$
$t_5 = \{wijn, eieren, kleding, kaas, melk\}$
$t_6 = \{eieren, kleding, melk\}$
$t_7 = \{eieren, kleding, melk\}$

Tabel 1: Set van transacties T

De set $\{eieren, kleding, melk\}$ komt 3 maal voor in de totale set van 7 transacties, dus de *support* is:

$$\text{supp}(\{eieren, kleding, melk\}) = \frac{3}{7} \tag{7}$$

in tabel 2 is ook te zien dat de set $\{eieren, kleding\}$ driemaal voorkomt. Daarnaast is de set $\{melk\}$ ook altijd aanwezig is indien de voorgaande set een subset van een transactie t is, de gecombineerde set komt dus ook drie keer voor. In dit geval is de *confidence* dus:

$$\text{conf}(\{eieren, kleding\} \Rightarrow \{melk\}) = \frac{3}{3} \tag{8}$$

Mining algoritmes

Om in een dataset *association rules* te vinden, bestaan er veel algoritmes. Deze zouden, gegeven een minimum *support* en minimum *confidence*, uit dezelfde dataset eenzelfde set regels moeten extraheren. Op vlak van efficiëntie en geheugengebruik kunnen er uiteraard wel verschillen zijn.

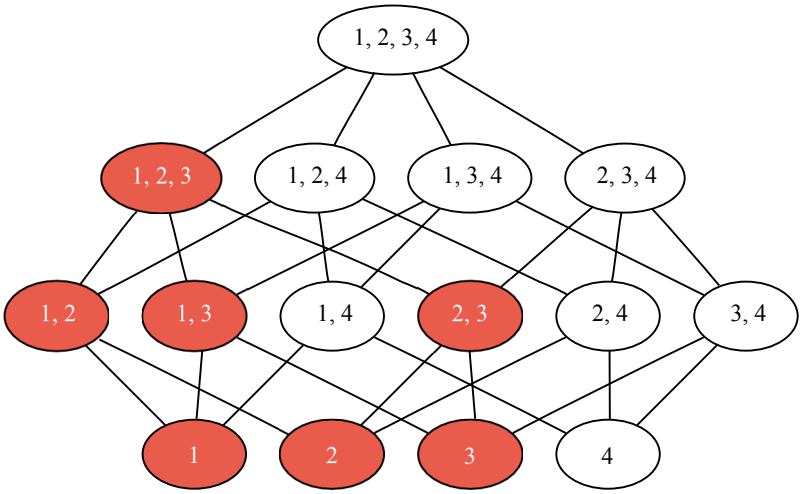
Een naïeve strategie om alle regels te vinden in een bepaalde dataset met m items, is om alle regels te genereren en te toetsen aan de vooropgestelde minima. Uiteraard is dit —zeker met grote itemsets $m >> 1$—niet altijd realiseerbaar. Het aantal regels (inclusief deze die de vooropgestelde drempelwaarden niet halen) is namelijk:

$$|R| = 3^m - 2^{m+1} + 1 \tag{9}$$

Het Apriori algoritme is in 1994 bedacht, met als doel het vinden van alle *association rules* in een transactionele dataset op basis van frequente itemsets. Men onderscheidt twee stappen:

1. Vind alle frequente itemsets met een minimale *support* tot en met een k-itemset, waarbij k het aantal items voorstelt.
2. Gebruik deze frequente itemsets om *association rules* te genereren, welke de boven de minimale *confidence* scoren.

Om de eerste stap te realiseren, is het algoritme gebasseerd op de *downward closure*-eigenschap: iedere subset van van een frequente itemset is ook een frequente itemset.



Figuur 2: Itemset $\{1, 2, 3\}$ is een frequente itemset; als gevolg van de *downward closure*-eigenschap zijn de subsets ook frequente itemsets.

Algoritme 1: Pseudocode van Apriori, waarbij de frequente itemsets ontdekt worden.

Require: T: Transactionset
  function APRIORI(T)
    $C_1 \leftarrow initialPass(T)$ ▷ Generating frequent 1-itemsets
    $F_1 \leftarrow \{f \mid f \in C_1 \wedge minsup \leq \frac{|f|}{|T|}\}$
    $k \leftarrow 2$
    while $F_{k-1} \neq \emptyset$ **do**
      $C_k \leftarrow candidateGen(F_{k-1})$
      for $t \in T$ **do** ▷ Iterate over all transactions
        for $c \in C_k$ **do**
          if $c \subseteq t$ **then**
            $|c| \leftarrow |c| + 1$ ▷ Increase count of candidate c
          end if
        end for
      end for
      $F_k \leftarrow \{c \mid minsup \leq \frac{|c|}{|T|}\}$
      $k \leftarrow k + 1$
    end while
    return $F \leftarrow \cup_k F_k$
  end function

Apriori past deze eigenschap toe door eerst de 1-item frequente itemsets te zoeken; door permutatie onstaan dan kandidaat-2-itemsets. Deze worden getest of ze voldoen aan de minimale *support*. Indien dit het geval is, worden ze toegevoegd aan de frequente 2-itemset. Dit proces wordt iteratief herhaald tot alle frequente k-itemsets gevonden zijn.

Het genereren van de kandidaten wordt beschreven in algoritme 2; de functie genereert een superset van frequente k-itemsets $C_k \supseteq F_k$ op basis van de frequente itemsets F_{k-1}. Dit gebeurt in in twee stappen:

1. **Join**: Alle mogelijke kandidaat-k-itemsets C_k worden gegenereerd op basis van F_{k-1}.
2. **Prune**: Kandidaten welke—door de *downward closure property*—niet frequent kunnen zijn worden verwijderend uit C_k.

Om de *join*-stap te voltooien, worden kandidaat-k-itemsets beschreven volgens:

$$C_k \leftarrow \{a \cup \{b\} \mid a \in F_{k-1} \wedge b \notin a\} \tag{10}$$

De itemset a is dus een element van de set van frequente-k-1-itemset, wat maakt dat a zelf een frequente-k-1-itemset is. Bij deze itemset a wordt een item b toegevoegd, wat niet voorkomt in de itemset a. Hierdoor is een k-itemset ontstaan.

$$\{s \mid s \subset c \wedge |s| = k-1\} \tag{11}$$

De *prune*-stap verwijdert alle subsets $s \subset c$ uit de kandidaten $c \in C_k$ die niet voorkwomen in de frequente itemset F_{k-1}. De subset s moet voor deze reden $k-1$ items bevatten.

Algoritme 2: Pseudocode om kandidaten te genereren uit frequente k-1-itemsets.

```
Require: F_{k-1}: Frequent k-1-itemset
  function CANDIDATEGEN(F_{k-1})
    C_k ← ∅                                   ▷ Starting with an empty set
    for C_k ← {a ∪ {b} | a ⊂ F_{k-1} ∧ b ∉ a} do          ▷ Join
      for {s | s ⊂ c ∧ |s| = k − 1} do
        if s ∉ F_{k-1} then
          C_k ← C_k − c                                  ▷ Prune
        end if
      end for
    end for
    return C_k
  end function
```

Algoritme 1 is dus in staat alle frequente itemsets met een minimale *support minsup* te ontdekken, hieruit kunnen dan *association rules* gedestilleerd worden. Dit gebeurt door voor iedere frequente itemset $F_k \in F$ iteratief iedere strikte subset $E \subset F_k$ te toetsen aan de minimale *confidence minconf*.

Algoritme 3: Pseudocode om *assso-ciation rules* uit frequente itemsets te genereren.

```
Require: F: Set of frequent itemsets
  function RULESGEN(F)
    for F_k ∈ F do
      for {E ⊂ F_k | E ≠ ∅ ∧ E ≠ F_k} do
        if conf(E ⇒ F_k − E) ≥ minconf then
          Assocation rule E ⇒ F_k − E found
        end if
      end for
    end for
  end function
```

Zeldzame itemprobleem

In bepaalde datasets komen bepaalde items significant meer voor dan andere items. In het voorbeeld van de supermarkt zien we bijvoorbeeld terug in de wekelijkse aankoop van brood en de—hopelijk—eenmalige aankoop van een broodrooster. Het probleem is dus dat de minimaal benodigde *support* voor ieder item hetzelfde is, onafhankelijk van de frequentie van dat item. De keuze van deze drempelwaarde heeft dus een twee gevolgen:

- **Minimale *support* is hoog**: *Association rules* met *rare items* zullen niet gevonden worden.
- **Minimale *support* is laag**: Frequente items zullen met elkaar gecombineerd worden, wat ook minder betekenisvolle regels tot gevolg heeft.

Door individuele items een *minimum item support* MIS toe te kennen, wat we noteren als $\text{MIS}(i)$, kan dit probleem opgelost worden. De *support* dient dan groter te zijn dan de minimale MIS van itemset $X \cup Y = \{i_1, i_2, ..., i_r\}$ met $i_j \in I$.

$$\text{supp}(X \cup Y) \geq \min\left(\text{MIS}(i_1), \text{MIS}(i_2), ..., \text{MIS}(i_r)\right) \tag{12}$$

Om te verhinderen dat frequente items samen met *rare items* voorkomen, mag het verschil tussen de grootste en kleinste MIS niet groter zijn dan een bepaalde drempelwaarde. Dit wordt de *support difference constraint* ϕ genoemd.

$$\phi \geq \max_{i \in X \cup Y} \left(\text{supp}(i) - \min_{i \in X \cup Y} \left(\text{supp}(i) \right) \right) \tag{13}$$

Wat vereenvoudigd kan worden tot:

$$\phi \geq \max_{i \in X \cup Y} \left(\text{supp}(i) \right) - \min_{i \in X \cup Y} \left(\text{supp}(i) \right) \tag{14}$$

Door de introductie van *minimum item supports* is de *downward closure*-eigenschap niet meer geldig[1]. Een alternatief hiervoor is *sorted closure*; waarbij items gesorteerd worden volgens hun MIS. Dit valt echter buiten de scope van deze cursus.

[1] Bing Liu, Wynne Hsu en Yiming Ma. "Mining association rules with multiple minimum supports". In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 1999, p. 337–341

Class association rules

Bij *association mining* kunnen items voorkomen als een conditie $i \in X$ of als een consequentie $i \in Y$. Indien men echter enkel geïnteresseerd is in de regels die een bepaalde consequentie opleveren, dan willen we alle *class association rules* (CAR) met een klasse y van de set van klasse-labels Y vinden. Formeel wordt dit:

$$X \Rightarrow y \text{ met } X \subseteq I \text{ en } y \in Y \tag{15}$$

Aangezien de klasse-labels al gekend zijn, is het vinden van frequente k-itemsets voldoende om de regels te genereren. Dit betekent dus dat Apriori aangepast kan worden voor CAR.

Daarnaast is het ook nuttig om op te merken dat de gekende definities voor de *support* en *confidence* hetzelfde blijven.

Doc	Dataset	Klasse
1	$\{student, leerkracht, school\}$	*onderwijs*
2	$\{student, school\}$	*onderwijs*
3	$\{leerkracht, school, stad, wedstrijd\}$	*onderwijs*
4	$\{basketball, lopen\}$	*sport*
5	$\{basketball, speler, toeschouwer\}$	*sport*
6	$\{basketball, coach, wedstrijd, team\}$	*sport*
7	$\{basketball, team, stad, wedstrijd\}$	*sport*

Tabel 2: Set van transacties T.

Voorbeeld Stel $minsup = 20\%$ en $minconf = 60\%$. We beschouwen de dataset in tabel 2, welke bestaat uit enkele frequente woorden van documenten die geclassificeerd zijn als *sport* of *onderwijs*.

Regel	supp	conf
$\{student, school\} \Rightarrow onderwijs$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$
$\{basketball\} \Rightarrow sport$	$\frac{4}{7} = 57\%$	$\frac{4}{4} = 100\%$
$\{wedstrijd\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{3} = 67\%$
$\{team\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$
$\{basketball, wedstrijd\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$
$\{basketball, team\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$
$\{team, wedstrijd\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$
$\{basketball, wedstrijd, team\} \Rightarrow sport$	$\frac{2}{7} = 29\%$	$\frac{2}{2} = 100\%$

Tabel 3: Ontdekte regels uit de transacties opgesomd in tabel 2.

Sequentiële patronen

Een limitatie van voorgaande *association rules* is dat er geen rekening gehouden wordt met de volgorde van transacties. Het kan echter wel nuttig zijn om deze volgorde in beschouwing te nemen, wat leidt tot sequenties. Men definiëert een sequentie s als een **geordende** lijst van itemsets a_i. Een k-sequentie is een sequentie van k itemsets; wat gelijkaardig is aan k-itemsets.

$$s = \langle a_1, a_2, a_3, ..., a_r \mid a \subseteq I \rangle \tag{16}$$

sub- en supersequenties Indien we twee sequenties $s_1 = \langle a_1, ..., a_r \rangle$ en $s_2 = \langle b_1, ..., b_v \rangle$ beschouwen, dan kunnen we stellen dat sequentie s_1 een *subsequentie* is van sequentie s_2 als iedere itemset ook een deelverzameling is.

$$s_1 \subseteq s_2 \iff \{a_i \subseteq b_i \mid \{i \in \langle \mathbb{Z}^+ \rangle \mid i \leq r\}\} \tag{17}$$

Mining algoritmes

Om sequenties te *minen*, kan het *Generalized Sequential Pattern* of kortweg GSP-algoritme gebruikt worden. Dit algoritme lijkt zeer sterk op Apriori—wat bij algoritme 1 is besproken. Wat wel verschilt is de generatie van kandidaten. De twee stappen (*join* en *prune*) van Apriori komen ook wel weer terug:

1. **Join**: Alle mogelijke kandidaat-k-sequenties worden gegenereerd door F_{k-1} te *joinen* met F_{k-1}. De volledige sequentie s_1 *joint* de volledige sequentie s_2 als de subsequentie van s_1 zonder het eerste item gelijk is aan de subsequentie van s_2 zonder het laatste item.

   Afhankelijk of het laatste element van s_2 een aparte set vormt, zijn er twee gevallen:

   (a) Het item vormt een aparte set op het einde van s_1 als het ook een aparte set vormde in s_2.

   (b) Het item wordt aan het laatste element van s_1 toegevoegd indien het geen aparte set vormde in s_2.

   Het *joinen* van F_1 met F_1 is ook bijzonder, aangezien er hier nog geen itemsets zijn ($k = 1$). Dus wordt het item van s_2 zowel als itemset als apart element toegevoegd.

2. **Prune**: Kandidaten waarbij een van de k-1-subsequenties niet frequent is, worden verwijderd.

Voorbeeld In tabel 4 wordt de generatie van kandidaten uitgewerkt. Indien we we het algoritme overlopen en stellen dat $s_1 = \langle\{1,2\}\{4\}\rangle$, dan is de subsequentie zonder het eerste item $\langle\{2\}\{4\}\rangle$. Dit komt overeen met de laatste twee frequente-3-sequenties. Deze worden dan gecombineerd, rekening houdende met de bovengestelde voorwaarde.

Frequente-3-sequenties	**Kandidaat-4-sequenties**	
	Na *join*	**Na *prune***
$\langle\{1,2\}\{4\}\rangle$	$\langle\{1,2\}\{4,5\}\rangle$	$\langle\{1,2\}\{4,5\}\rangle$
$\langle\{1,2\}\{5\}\rangle$	$\langle\{1,2\}\{4\}\{6\}\rangle$	
$\langle\{1\}\{4,5\}\rangle$		
$\langle\{1,4\}\{6\}\rangle$		
$\langle\{2\}\{4,5\}\rangle$		
$\langle\{2\}\{4\}\{6\}\rangle$		

Tabel 4: Kandidaten uit frequente-3-sequenties.

De R programmeertaal

Wat is R

R is een scriptingtaal speciaal ontwikkeld voor het gebruik in statistische software. De taal is open source en dus vrij beschikbaar. Er is ook een gratis IDE beschikbaar. De grote kracht van deze taal is de grote hoeveelheid beschikbare packages waarbij gebruikers reeds algoritmen geïmplementeerd hebben. Zo kan de gebruiker van de taal dus focussen op zijn specifiek probleem, zonder noodzakelijk bezig te zijn met details.

R is beschikbaar op `https://www.r-project.com` en RStudio is verkrijgbaar op `https://www.rstudio.com`

Basisconcepten

Variabelen

Omdat R een dynamische taal is, is het niet nodig om het type te vermelden. Men kan variabelen op 2 manieren toewijzen:

```
x = 123
y <- 321
z = "Dit is een test string"
```

Merk op dat indien statements op 1 lijn staan er geen puntkomma's nodig zijn om deze af te sluiten. R heeft heel wat functies al ingebouwd. Voorbeelden hiervan zijn `sqrt(x)`,`cos(x)`,`log(x)` en veel meer. Verder volgt R ook dezelfde commando's als op Unix systemen. Indien een gebruiker dus een variable wil verwijderen uit de workspace kan met `rm(x)` gebruiken. Gelijkaardig kan men `ls()` gebruiken. Indien men RStudio gebruikt als IDE dan staan de variabelen automatisch rechts vermeld.

Bij *data mining* en statistiek maakt men vaak gebruik van vectoren en matrices om data voor te stellen. Het is daarom ook logisch dat R heel wat methoden bevat om matrices en vectoren te manipuleren. Hieronder zijn enkel voorbeelden gegeven.

Voorbeeldcode: vectoren en matrices

```
#Comments beginnen met een #-teken.
#Het alloceren van een vector
positie <- c(0,5,0)
#Het is ook mogelijk om automatisch een vector op te vullen
sequence <- seq(from=0, to=10, by=1) #sequence = {0,1,2,3,4,5,6,7,8,9,10}
#Of om een bepaalde herhaling uit te voeren
rep = rep(2, times=5) #rep = {2, 2}, 2 kan ook een vector zijn
#Optellen van 2 vectoren
delta_pos <- {1,0,0}
positie = positie + delta_pos
#constanten en een vector:
positie = positie * 0.5
#Vectoren in R beginnen, in tegenstelling tot veel andere talen, bij 1:
first_element = sequence[1]
last_element = sequence[11]
#Alles behalve 1 element:
all_without_first = sequence[-1]
#Subset:
first_five = sequence[1:5]
#Matrices
mat = matrix(1:9, nrow=3, byrow=TRUE) #Vult de matrix met waarden per rij( dus rij 1: 1,2,3 )
mat = matrix(1:9, nrow=3, byrow=FALSE) #Vult de matrix met waarden per kolom( dus rij 1: 1,4,7 )
#Matrices indexering:
top_left = mat[1,1] #top_left = 1
row_two = mat[2,] #row_to = {2,5,8}
```

Controle structuren

R heeft dezelfde controlestructuren zoals men aantreft in andere programmeertalen. Deze zijn:

- **if**: `if(condition) expr1 else expr2` Hierbij kan men braces gebruiken indien de expressies uit meerdere statements bestaan.
- **for**: `for( var in seq ) expr`
- **while**: `while( cond ) expr`
- **switch**: `switch( expr, case1, case2, .. )`

Methoden

Methoden in R worden als volgt gedefinieerd en opgeroepen:

```
mean <- function( input )
{
        return( sum( input ) / length( input ) )
}
mean( c( 5, 10 ) ) #print 7.5 uit
```

Dataframes

Een andere vaak gebruikte datastructuur is een *dataframe*. Deze structuur wordt het best vergeleken met een matrix. Het bestaat uit een collectie lijsten met dezelfde lengte. Een voordeel aan dataframes vergeleken met matrices is dat ze gebruikt kunnen worden zoals een database. De gebruiker kan rijen toevoegen of een query uitvoeren op een bepaalde kolom. Men kan een dataframe bekijken via `View(dataframe)`. Vectoren toevoegen doet met via `attach()` en `detach()`. Bepaalde data opvragen kan als volgt: `data[ gender=="male" & age > 18,]`. Let hierbij op de komma om alle kolommen te selecteren.

Het gebruik van packages

Een overzicht van alle packages is beschikbaar op: `https://cran.r-project.org/web/packages/available_packages_by_date.html`

De grote sterkte van R zijn de grote hoeveelheid aanwezige *packages*, bibliotheken met een bepaald doel. Het installeren van deze *packages* is zeer eenvoudig: `install.packages(wordcloud)`. en het gebruik nog eenvoudiger: `library(wordcloud)`. Een *package* moet maar eenmaal geïnstalleerd worden en kan daarna met bovenstaand commando geladen worden.

Association rules in R

Als concreet voorbeeld nemen we de *association rules* aangeleerd in vorig hoofdstuk. Hiervoor is het package `arules` beschikbaar. Stel dat we volgend bestand beschikbaar hebben:

De inhoud van het testbestand: transactions.txt

```
A,B,C
B,C
A,B,D
A,B,C,D
A,B
B,C
A
B
```

Het inlezen is zeer eenvoudig:

```
transactions <- read.transactions("transactions", format="basket", sep=",")
```

Hier geven we de bestandsnaam mee, samen met het formaat: een *basket*. Dit maakt het duidelijk voor het package dat de data een winkelmand voorstelt. Tot slot vermelden we de seperator waarmee de data gescheiden is.

Indien we de transactions willen bekijken dan kan dit via een aantal methoden: `inspect(transactions)` print de data uit in de console, `image(transactions)` geeft een afbeelding van een grid weer waarbij een cel zwart is ingekleurd indien een item aanwezig is en `itemFrequencyPlot(transactions, support=0.1)` visualiseert hoe vaak elk item voorkomt.

In het vorig hoofdstuk hebben het apriori algoritme besproken. Ook dit algoritme is aanwezig in deze *package*. We kunnen automatisch de regels laten genereren via:

```
rules <- apriori(transactions, paramater=list(supp=0.5,conf=0.5))
inspect( rules )
```

Dan krijgen we de volgende output:

```
> inspect(rules)
    lhs    rhs support confidence lift
[1] {}  => {C} 0.500   0.5000000  1.0000000
[2] {}  => {A} 0.625   0.6250000  1.0000000
[3] {}  => {B} 0.875   0.8750000  1.0000000
[4] {C} => {B} 0.500   1.0000000  1.1428571
[5] {B} => {C} 0.500   0.5714286  1.1428571
[6] {A} => {B} 0.500   0.8000000  0.9142857
[7] {B} => {A} 0.500   0.5714286  0.9142857
```

We kunnen ook sequentiële patronen laten genereren via het `arulesSequences` package.

Unsupervised learning & clustering

Introductie

Algemeen gezien bouwen *machine learning*-technieken een model op basis van een (voorbeeld)-dataset; waarbij het doel is om een structuur die in de dataset zit naar boven te brengen. Het kan zijn dat deze structuur al aangeduid wordt doordat de data gelabeld is, in dit geval spreken we van *supervised learning*. Indien de data nog niet gelabeld is, dient een *unsupervised learning*-techniek gebruikt te worden.

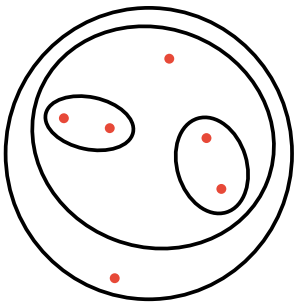
Clustering

Clustering is een typische *unsupervised learning*-techniek, waarbij *similarity groups* of clusters worden gezocht. Dit zijn data-elementen welke (per cluster) onderling sterk op elkaar gelijken. De data is dus nog niet geclassificeerd; er zijn geen labels voorzien.

Om elementen met elkaar te kunnen vergelijken is een *distance*-functie nodig. Door te clusteren dienen dan twee parameters geoptimaliseerd te worden:

- **Interclusterafstand**: De afstand tussen elementen van verschillende clusters dient maximaal te zijn.
- **Intraclusterafstand**: De afstand tussen elementen van eenzelfde cluster dient minimaal te zijn.

Het spreekt voor zich dat deze parameters een maatstaaf zijn voor de kwaliteit van het resultaat. De *distance*-functie heeft een invloed op dit resultaat, alsook het algoritme zelf. Ook speelt het mee hoe de clusters gevormd zijn. Wat algoritmes betreft, beschouwt



Figuur 3: Een voorbeeld van 6 elementen die met een hiërarchisch clusteringsalgoritme gegroepeerd zijn.

men meestal twee soorten algoritmes. De eerste groep zijn de *partitional clustering*-algoritmes, waar K-means onder valt. Hierbij wordt een groep elementen onderverdeeld in niet-overlappende subsets. *Hierarchical clustering* is de tweede soort, waarbij een boomstructuur van geneste clusters wordt opgebouwd.

K-means

Zoals eerder vermeld, is K-means een *partitional clustering*-algoritme. De data wordt dus verdeeld in k niet-overlappende clusters. Deze parameter k wordt door de gebruiker opgegeven.

De data D is een set van r punten in een n-dimensionale vectorruimte.

$$D = \{x_1, \ldots, x_r \mid x_i \in \mathbb{R}^n\} \tag{18}$$

Elke cluster heeft een *centroid*, een punt in het midden van de cluster. Bij de start van het algoritme worden k punten random aangeduid als *centroid*. Ieder punt wordt daarna een aan cluster toegewezen, afhankelijk van de afstand tot de *centroid*. Hierna worden de *centroids* herberekend, op basis van de huidige clusters. Dit wordt herhaald tot er geen convergentie meer waargenomen wordt.

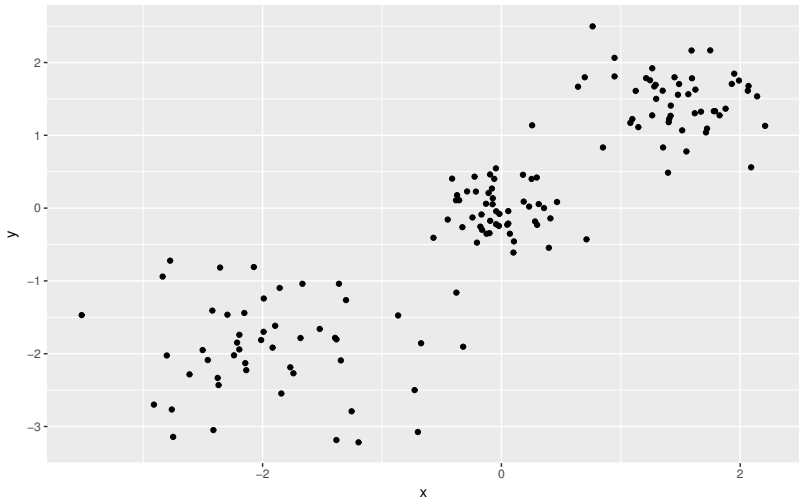
Algoritme 4: Pseudocode voor het K-means-algoritme.

```
Require: k: Number of clusters
Require: D: Data
  function K-MEANS(k,D)
    C ← randomSelection(k, D)              ▷ Random centroids C
    repeat
      for x ∈ D do
        for m ∈ C do
          compute distance from x to m
        end for
        assign x to closest centroid m
      end for
      C ← recomputeCentroids
    until stopping criterion is met
    return C
  end function
```

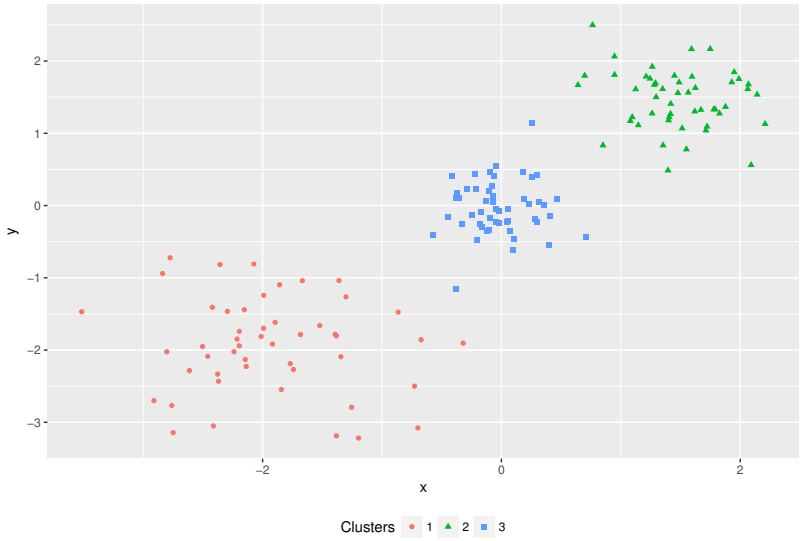
Het *stopping criterion* in algoritme 4 kan op meerdere manieren geïmplementeerd worden. Enkele mogelijkheden zijn:

1. Geen of weinig toekenningen van datapunten aan andere clusters.
2. Geen of minimale verandering van de *centroids*.
3. Minimale afname in de *sum of squared error*. Hierbij wordt per cluster C_i de afstand d tussen ieder punt $x \in C_i$ en de *centroid* m_i berekend.

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_i} d\,(x, m_i)^2 \tag{19}$$



Figuur 4: Een voorbeeld van een gegenereerde tweedimensionale dataset, welke opgedeeld is in drie clusters van gelijke grootte.



Figuur 5: Op de dataset van figuur 4 is het *K-means*-algoritme toegepast. De code voor het genereren van deze dataset en het clusteren staat op Github.

Distance functions In een Euclidische vectorruimte $\mathbb{R}^n$ kan de afstandsfunctie d op een volgende manier beschreven worden:

$$d(x,m) = ||x-m|| = \sqrt{(x_1-m_1)^2 + \cdots + (x_n-m_n)^2} \tag{20}$$

Voor de *centroids* is het dan mogelijk om het gemiddelde te gebruiken, wat per cluster C_i tot de volgende formule leidt:

$$m_i^t = \frac{1}{|C_i^{t-1}|} \cdot \sum_{x \in C_i^{t-1}} x \tag{21}$$

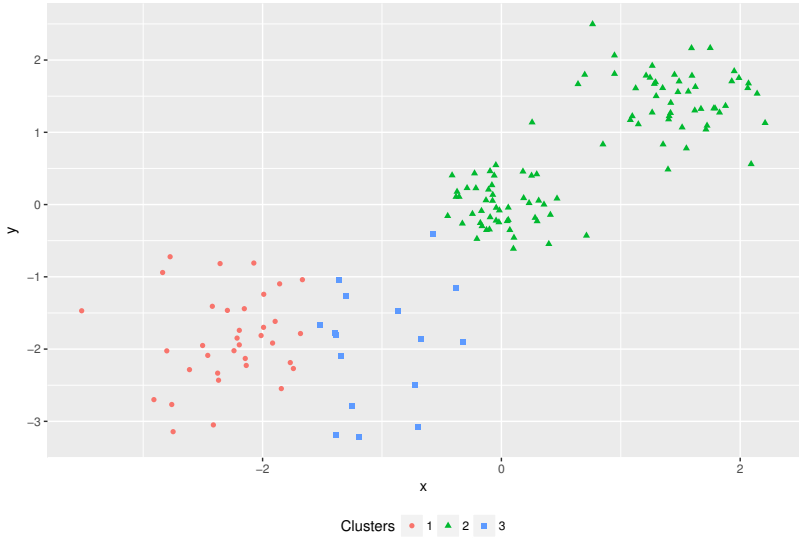
Voor- en nadelen Het K-means-algoritme is vrij eenvoudig om te begrijpen en te implementeren. Daarnaast is het ook nog behoorlijk efficiënt, met een tijdscomplexiteit $O(k \cdot n \cdot i \cdot r)$ met i iteraties [2]. Er wordt ook veel werk gestoken in een paralleliseerbare versie [3]. Dit alles maakt dat K-means het populairste clusterings-algoritme is.

[2] Xin Jin en Jiawei Han. "K-means clustering". In: *Encyclopedia of Machine Learning*. Springer, 2011, p. 563–564

[3] Weizhong Zhao, Huifang Ma en Qing He. "Parallel k-means clustering based on mapreduce". In: *IEEE International Conference on Cloud Computing*. Springer. 2009, p. 674–679

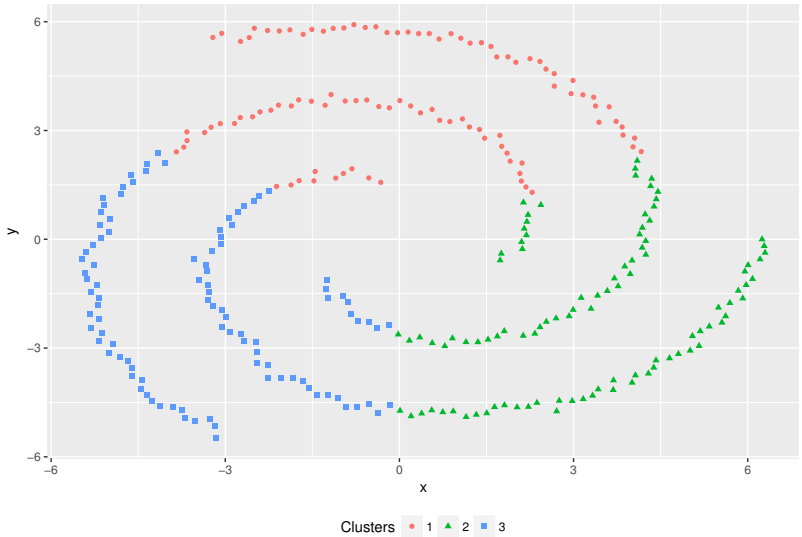
Er zijn echter ook nadelen aan het algoritme; waarvan de volgende het meest noemenswaardig zijn.

- Het gemiddelde van een cluster dient gedefiniëerd te zijn.
- Het algoritme is gevoelig voor initiële *centroids*; dit is geïlustreerd in figuur 6. Door meerdere *passes* te doen, kan dit probleem verholpen worden. Daarnaast zijn er ook methoden om goede *seeds* te kiezen voor de initiële *centroids*.



Figuur 6: Dezelfde dataset van 4, maar met ongelukkig gekozen *centroids*. Merk op dat er nog steeds 3 clusters zijn, maar dat dit niet het globale optimum heeft gelokaliseerd.

- De gebruiker dient zelf het aantal clusters k op te geven.
- Het algoritme is gevoelig voor *outliers*. Dit kan opgelost worden door deze punten na enkele iteraties te verwijderen of door te clusteren op een random geselecteerde subset van de data. Dit verkleint de kans dat een *outlier* geselecteerd wordt.
- K-means is enkel bruikbaar voor hyper-ellipsoïden.



Figuur 7: Een spiraalvormige dataset; nog steeds met 3 clusters. K-means is niet in staat de clusters correct te classificeren. De code hiervoor staat op Github.

Supervised learning

Introductie

In tegenstelling tot bij *unsupervised learning* beschikken we bij *supervised learning* over een dataset waarbij het gewenste resultaat al bijgevoegd is. Deze dataset wordt ook nog eens opgedeeld in twee delen: Een eerste deel waarmee het algoritme bijleert, en een tweede om de nauwkeurigheid van het algoritme te testen. Merk op dat indien de tweede dataset ook gebruikt zou worden om het algoritme te trainen, dit geen goed beeld zou geven van de werking.

Voorbeeld Een eerste voorbeeld waarop dit mogelijk is, is een creditcardmaatschappij. Stel dat een nieuwe klant een aanvraag indient om een creditcard te krijgen. Dan moet het bedrijf verschillende parameters analyseren zoals de leeftijd, of de persoon getrouwd is, het loon, welke leningen er al aanwezig zijn, ... Het resultaat van de aanvraag is dan binair: wordt de aanvraag aanvaard of niet?

Voorbeeld Een tweede voorbeeld is in een ziekenhuis. Hierbij moet er opnieuw aan de hand van een aantal parameters beslist worden of een patiënt opgenomen moet worden op intensive care. Aangezien dit een duur proces is, wensen ziekenhuizen dit enkel te doen voor patiënten met een hoog risico.

Algemeen heeft met dus een probleem met een aantal (discrete) uitkomsten. Hiervoor ontwikkelen we een oplossing waarbij een functie getraind wordt met een bestaande dataset. Dit wordt ook vaak *classification* of *inductive learning* genoemd.

Voorbeeld en terminologie

De data wordt gedefinieerd als een set van k attributen: $A_1, A_2, .., A_k$. Elke rij heeft ook een label met een reeds gedefinieerde klasse. Hiervan komt de naam *supervised*, we beschikken over leermateriaal. Het doel is om deze gegevens te gebruiken om een model zo te trainen dat een toekomstige rij, waarbij alle attributen aanwezig zijn, een correct label toe te kennen. Na de training kan de data getest worden op een tweede dataset. Deze dataset bestaat uit een

aantal rijen waarvan de klasse reeds bekend is. Met kan de *accuracy* van het model dan als volgt berekenen:

$$\text{Correctheid} = \frac{\text{juiste gevallen}}{\text{totaal aantal testcases}} \tag{22}$$

Algemeen volgen we dus dit patroon:

$$\text{Training data} \rightarrow \text{Learning algoritm} \rightarrow \text{Model} \rightarrow \text{Test data} \rightarrow \text{Accuracy} \tag{23}$$

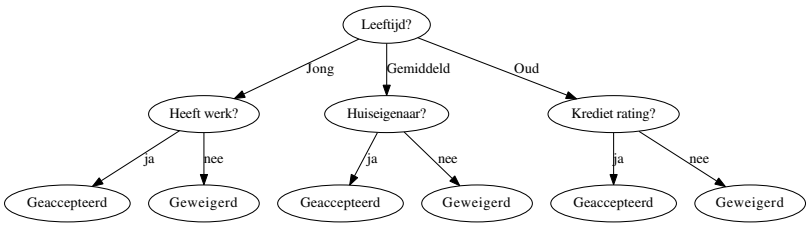
Het is belangrijk om zich het doel te blijven herinneren tijdens dit proces. Stel in het voorbeeld van het creditcardmaatschappij dat we gewoon steeds *geaccepteerd* antwoorden. Dan zullen we dus een *accuracy* van 50% hebben. Het doel van het supervised learning algoritme is dus om (veel) beter te doen dan de reeds gebruikte methode.

Het aanleren wordt dus als volgt gedefinieerd. Stel dat we een dataset D hebben met een taak T en een manier om de kwaliteit te meten M. Een computersysteem leert dan van D om taak T uit te voeren, indien na het leren het systeem beter presteert als het taak T uitvoert. Dit wordt gemeten via M.

Er is dus een cruciaal concept waar we steeds rekening mee moeten houden indien we gebruik maken van *supervised learning*: De test- en trainingsdata moet voldoende representatief zijn voor de data die men in het echt zal krijgen.

Decision trees

Een eerste, simpele maar toch efficiënte vorm van supervised learning zijn *decision trees*. Deze beslissingsbomen werken volgens een simpel principe. Bovenaan de boom staat een eerste attribuut. Vanuit deze knoop vertrekken verbindingen naar andere knopen. Elke verbinding stelt een waarde van het attribuut voor.



Figuur 8: Een voorbeeld van een beslissingsboom voor een creditcard-maatschappij.

Deze boom is afhankelijk van de gegeven data en is niet uniek, noch optimaal. Er bestaat een kleinere boom die dezelfde info geeft in minder nodes. We moeten dus proberen een zo klein mogelijk boom te construeren. Dit is echter geen triviale opgave en alle huidige algoritmes geven een benadering.

De boom kan worden omgezet naar een serie regels. Elk pad van de beginknoop tot aan een blad is een regel. Voor figuur 8 geeft dit dus volgende regels:

- Leeftijd = jong, heeft werk? = ja ⇒ Class = Geaccepteerd
- Leeftijd = jong, heeft werk? = nee ⇒ Class = Geweigerd
- ...

Elk van deze regels heeft dan ook een *support* en *confidence* afhankelijk van de gebruikte dataset.

Een simpel algoritme

Een mogelijk algoritme is een *divide-and-conquer* algoritme.

Algoritme 5: Voorbeeld van een beslissingsboom constructie algoritme

```
Require: D: Training data, A: Attributen, T: Een knoop van de
    boom, C: De verzameling van klassen
 1: function MAAKBESLISSINGSBOOM(D,A,T)
 2:     if D bevat enkel data van een bepaalde klasse c_j ∈ C then
 3:         maak T een blad met klasse c_j;
 4:     else if A = ∅ then
 5:         Maak van T een blad met het label van klasse c_j,
 6:         waarbij c_j de meest voorkomende klasse in D is.;
 7:     else
 8:         // D bevat een mix van klassen.We selecteren een
 9:         // attribuut om D verder te verdelen in subsets
10:         // zodat elke subset minder onzuiverheden bevat.
11:         p_0 = puriteitEval(D)
12:         for A_i ∈ {A_1, A_2, ..., A_k} do
13:             p_i = puriteitsEval(A_i, D);
14:         end for
15:         Selecteer A_g ∈ {A_1, A_2, ..., A_k} met de minste puurheid,
16:         berekend met p_0 − p_i;
17:         if p_0 − p_g < drempelwaarde then
18:             Maak van T een blad met label c_j;
19:         else
20:             Maak van T een beslissingsknoop voor A_g;
21:             Verdeel D dan in m sets D_1, D_2, ..., D_m gebaseerd op
22:             A_g, met de mogelijke waarden: {v_1, v_2, ..v_m}
23:             for D_j in {D_1, D_2, ..D_m} do
24:                 if D_j ≠ ∅ then
25:                     Maak een tak T_j voor v_j als kind van T.
26:                     decisionTree( D_j, A − {A_g}, T_j )
27:                 end if
28:             end for
29:         end if
30:     end if
31: end function
```

Bij dit algoritme nemen we aan dat alle data categorisch is. De boom wordt op een *top-down* recursieve manier opgesteld. Dit wil zeggen dat we beginnen bij de bovenste knoop van de boom en

zo verder afdalen tot de bladen gevormd zijn. Elementen uit de dataset worden verder onderverdeeld gebaseerd op hun attributen.

Het bovenste algoritme heeft enkele voorwaarden op te stoppen:

- Indien er geen elementen meer over zijn
- Er zijn geen attributen meer die onderverdeeld moeten worden
- Alle elementen bij een knoop behoren tot dezelfde klasse

De sleutel tot succes bij dit algoritme is de keuze van het attribuut om een tak te maken. Het doel is om de data zo puur mogelijk te maken: Een subset van data wordt als puur aanzien indien alle data tot dezelfde klasse behoort. Voor we verder ingaan op het maken van deze keuze moeten we eerst begrijpen wat puurheid nu eigenlijk is. We verklaren dit aan de hand van *informatietheorie*.

Informatietheorie

Informatietheorie is een onderdeel van de wiskunde dat zich bezig houdt de opslag en verwerking van informatie. Het levert ons een basis om de inhoud van data te beschrijven. Een manier om dit beter te begrijpen is om dit voor te stellen als een antwoord op een vraag: Zal mijn worp kop of munt zijn? Indien men op voorhand al info heeft, men weet bijvoorbeeld dat de munt vervalst is, dan is het antwoord op de vraag minder relevant. Indien men dit nog niet weet, dan zou extra kennis dus welkom zijn. Het antwoord heeft dus een grotere waarde.

Informatietheorie gebruikt ditzelfde concept, maar gebruikt geen geld om de waarde van informatie uit te drukken, maar *bits*. Een bit is net genoeg info om het antwoord te geven op een ja/neen vraag waarover men geen idee heeft, zoals het opwerpen van een munt.

Een belangrijke manier om hoeveelheden informatie uit te drukken is entropie. Entropie is een maat voor informatiedichtheid in een reeks gebeurtenissen. Deze entropie wordt gedefinieerd als:

$$E(p) = -\sum_{i=1}^{n} p_i \log_2 p_i [bit] \tag{24}$$

Waarbij p_i een kans is op de uitkomst van u_i bij experiment A. Of in het geval van beslissingsbomen: De kans op klasse c_j in data set D. We kunnen deze entropie gebruiken als een maat voor de puurheid van een dataset D. Een voorbeeld: Stel dat een dataset bestaat uit een lijst van positieve en negatieve tweets over een bepaald onderwerp.

1. Een dataset D heeft 50% positieve tweets en 50% negatieve. De entropie is dan: $E(D) = -0,5 \cdot \log_2 0,5 - 0,5 \cdot \log_2 0,5 = 1 bits$
2. Een dataset D heeft 20% positieve tweets en 80% negatieve. De entropie is dan: $E(D) = -0,2 \cdot \log_2 0,2 - 0.8 \cdot \log_2 0,8 = 0,722 bits$
3. Een dataset D heeft 100% positieve tweets en 0% negatieve. De entropie is dan: $E(D) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0 bits$

Hieruit volgt dat naarmate de data puurder wordt, de entropie steeds kleiner wordt. We kunnen deze eigenschap gebruiken voor het opstellen van beslissingsbomen.

De toename van informatie en afname van entropie is dan het verschil tussen de entropie voor een tak wordt toegevoegd en de entropie erna. Deze toename wordt in de literatuur vaak *information gain* genoemd.

Stel nu dat we een attribuut A_i met v mogelijke waarden hebben en deze attribuut de wortel van de huidige boom maken, dan zal dit de dataset D onderverdelen in v subsets $D_1, D_2, ...$ De entropie wordt dan:

$$E_{A_i}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \cdot E(D_j) \tag{25}$$

Een concreet voorbeeld bij het opstellen van een beslissingsboom voor een creditcardmaatschappij met volgende data:

ID	Leeftijd	Heeft werk	Huiseig.	Credit rating	Klasse
1	jong	neen	neen	redelijk	neen
2	jong	neen	neen	goed	neen
3	jong	ja	neen	goed	ja
4	jong	ja	ja	redelijk	ja
5	jong	neen	neen	redelijk	neen
6	gem.	neen	neen	redelijk	neen
7	gem.	neen	neen	goed	neen
8	gem.	ja	ja	goed	ja
9	gem.	neen	ja	excellent	ja
10	gem.	neen	ja	excellent	ja
11	oud	neen	ja	excellent	ja
12	oud	neen	ja	goed	ja
13	oud	ja	neen	goed	ja
14	oud	ja	neen	excellent	ja
15	oud	neen	neen	redelijk	neen

Tabel 5: Een verzameling met applicaties D.

De entropie van deze totale dataset wordt berekend a.d.h.v. de klasse:

$$E(D) = -\frac{6}{15} \cdot \log_2 \frac{6}{15} - \frac{9}{15} \cdot \log_2 \frac{9}{15} = 0.971 \tag{26}$$

Vervolgens berekenen we per attribuut de entropie en de winst in informatie indien dit attribuut de wortel van de beslissingsboom zou zijn.

$$E_{leeftijd}(D) = \frac{5}{15} \cdot E(D_1) + \frac{5}{15} \cdot E(D_2) + \frac{5}{15} \cdot E(D_3) \tag{27}$$

Met:

$$E(D_1) = -\frac{3}{5} \cdot \log_2 \frac{3}{5} - \frac{2}{5} \cdot \log_2 \frac{2}{5} = 0.971 \tag{28}$$

$$E(D_2) = -\frac{3}{5} \cdot \log_2 \frac{3}{5} - \frac{2}{5} \cdot \log_2 \frac{2}{5} = 0.971 \tag{29}$$

$$E(D_3) = -\frac{4}{5} \cdot \log_2 \frac{4}{5} - \frac{1}{5} \cdot \log_2 \frac{1}{5} = 0.722 \tag{30}$$

Levert dit:

$$E_{leeftijd}(D) = \frac{5}{15} \cdot 0.971 + \frac{5}{15} \cdot 0.971 + \frac{5}{15} \cdot 0.722 = 0.888 \tag{31}$$

Met als informatiewinst:

$$winst(D, leeftijd) = 0.971 - 0.888 = 0.083 \quad (32)$$

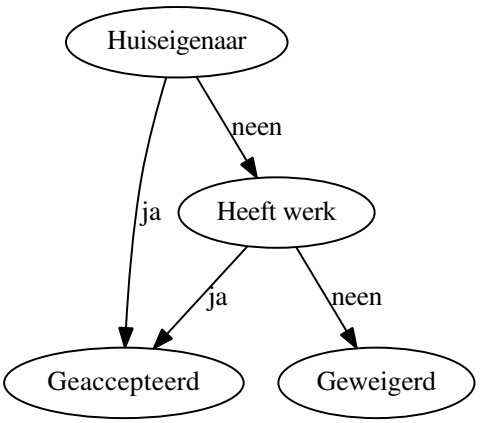
Zo kunnen we voor elk attribuut de winst berekenen. Dit levert voor de attributen *Heeft werk*, *Huiseigenaar*, *Krediet rating*:

$$winst(D, Heeft\ werk) = 0.9721 - 0.647 = 0.324 \quad (33)$$

$$winst(D, Huiseigenaar) = 0.9721 - 0.551 = 0.420 \quad (34)$$

$$winst(D, Kredietrating) = 0.9721 - 0.608 = 0.363 \quad (35)$$

We kiezen het attribuut met de grootste winst aan informatie, in dit geval *Huiseigenaar* als wortel van onze boom. Deze ziet er dan zo uit:



Figuur 9: De beslissingsboom opgesteld door het algoritme

Tot nu toe zijn we ervan uitgegaan dat attributen discreet zijn. Het algoritme werkt echter ook met continue attributen. Dit kunnen we doen door met intervallen te werken. Het moeilijke hierbij is de keuze van de waarde die dient als onderverdeling van deze intervallen. Hiervoor gebruiken we opnieuw de informatiewinst. We sorteren eerst de waarden in onze dataset in toenemende volgorde: $\{v_1, v_2, ...v_k\}$. Een van de mogelijke waarden voor de drempel is dan deze tussen v_i en v_{i+1}. We proberen alle drempels en kiezen degene die de maximale winst oplevert. Het probleem bij deze aanpak is dat men zo kan blijven onderverdelen aangezien de waarden continu zijn.

Een laatste probleem bij het opstellen van beslissingsbomen is *overfitting*. Hierbij is de boom teveel aangepast aan de trainingdata en niet genoeg aan het algemene geval. De symptomen hiervan zijn vaak duidelijk omdat de boom diep is en enorm veel takken bevat. Dit is vaak te wijten aan ruis of andere afwijkingen. Er zijn twee manieren om dit te vermijden:

1. Bij *pre-pruning* wordt de constructie van de boom vroegtijdig gestart. De moeilijkheid hierbij is om te weten wanneer men moet stoppen.

2. Bij *post-pruning* worden takken verwijderd van een reeds gemaakte boom gebaseerd op een statisch model dat de fouten per tak berekent.

K-nearest neighbour

Een tweede algoritme voor *supervised learning* dat we zullen bespreken is het *K-nearest neighbour* algoritme. In tegenstelling tot bijvoorbeeld beslissingsbomen bouwt KNN geen model van de training data.

Algoritme 6 beschrijft hoe dit gebruikt wordt in R en het resultaat van dit script is figuur 11.

Algoritme 6: Het kNN algoritme

Require: D: Training data, d: het element dat geclassificeerd moet worden, k: Het aantal buren.

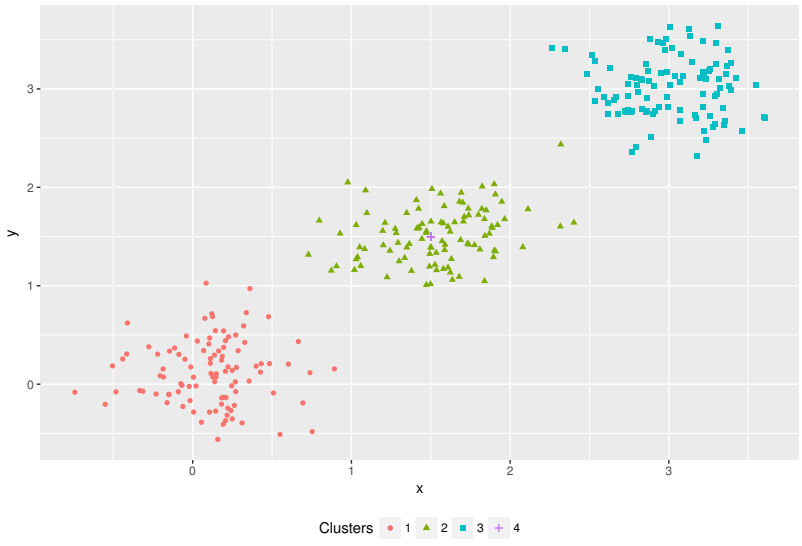
```
1: function kNN(D,d,k)
2:     bereken_afstand(d, D);
3:     P = kies_k_dichtste(D,k);
4:     d->klasse = meest_frequente(P);
5: end function
```

Hierbij geven we d dus een klasse door eerst alle afstanden in D tot aan d te berekenen. Hieruit kiezen we dan k punten die we in de verzameling P plaatsten. De klasse van d stellen we dan gelijk aan de meest voorkomende klasse in P. We gaan er dus van uit dat de kans op klasse c_j, indien we n kennen, gelijk is aan $\frac{n}{k}$.

$$p(c_j|d) = \frac{n}{k} \tag{36}$$

Cruciaal hierbij is de keuze van de afstandsberekening. Het meest voor de hand liggende is natuurlijk de euclidische afstand, maar alternatieven—zoals de *Manhattan distance*—zijn ook mogelijk. De keuze is vaak afhankelijk van de applicatie.

k is meestal empirisch gekozen door handmatig te kiezen of via een controle set.



Figuur 10: De data voor het knn algoritme wordt toegepast.

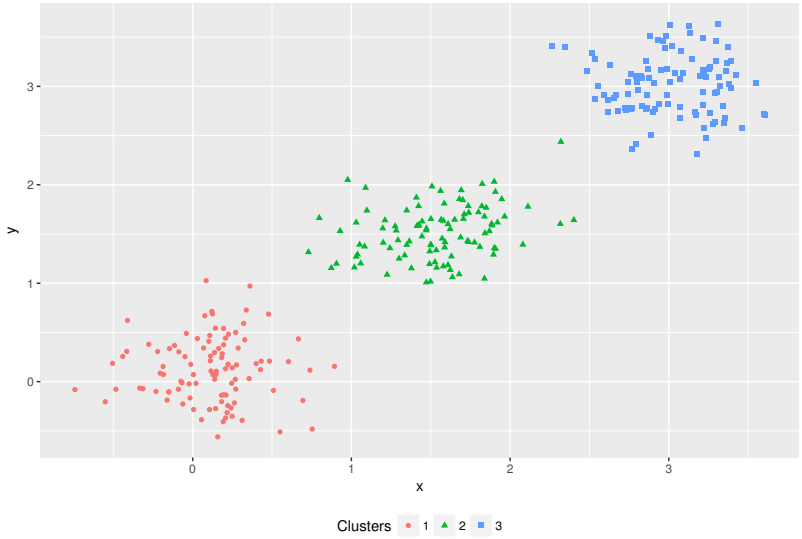
Voorbeeld We maken gebruik van de *package* FNN. Stel dat we reeds geclassificeerde data hebben, welke in figuur 10 gevisualiseerd is. In de figuur zijn er 3 groepen met elk een klasse en 1 testpunt. De code om het punt te classificeren ziet er als volgt uit:

```
library("ggplot2")
library("FNN")

gen_matrix <- function(n, sd, mean, class)
{
  cluster <- matrix( rnorm(n*2, sd = sd, mean = mean),ncol = 2 )
  cluster <- cbind( cluster, rep(class,times=n) )
  colnames( cluster ) <-c("x","y","class")
  return( cluster )
}

#generate 3 clusters

df <- data.frame( gen_matrix( 100, 0.3, 0.1, 1 ) )
df <- rbind( df, gen_matrix( 100, 0.3, 1.5, 2 ) )
df <- rbind( df, gen_matrix( 100, 0.3, 3, 3 ) )

test_point <- c(1.5, 1.5, 4)

ggplot(df,aes(x,y))+
  geom_point(aes(colour=factor(class),shape=factor(class))) +
  geom_point( aes(x=test_point[1], y=test_point[2],
                  colour=factor(test_point[3]),
                  shape=factor(test_point[3]))) +
  scale_colour_discrete(guide="legend", name="Clusters") +
  scale_shape(name="Clusters") +
  theme(legend.position="bottom")

result <- knn( df, test_point , df$class, k = 15  )
test_point[3] = as.numeric( sub("Level:","",result) )

ggplot(df,aes(x,y))+
  geom_point(aes(colour=factor(class),shape=factor(class))) +
  scale_colour_discrete(guide="legend", name="Clusters") +
  scale_shape(name="Clusters") +
  theme(legend.position="bottom")
```

Bemerkingen bij KNN

Het KNN algoritme heeft enkele voordelen:

- De eenvoudigheid. KNN is een zeer eenvoudig algoritme om te begrijpen en implementeren. Desondanks heeft KNN al heel wat goede resultaten opgeleverd.



Figuur 11: De data nadat het KNN algoritme wordt toegepast: het punt behoord tot de juiste klasse.

- KNN kan overweg met complexe grenzen tussen data.

Er zijn echter ook enkele nadelen aan verbonden.

- In tegenstelling tot bijvoorbeeld beslissingsbomen levert KNN geen begrijpbaar model op.
- Omdat er geen model wordt gemaakt en de classificatie meteen gebeurt, is KNN traag tijdens deze classificatie.

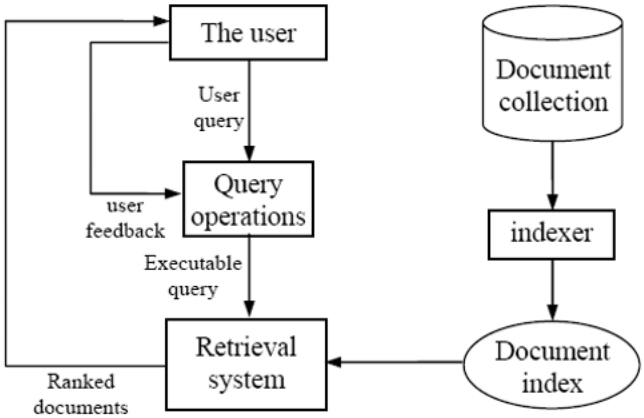
Besluit

Supervised learning is een reeks van technieken die reeds geclassificeerde data gebruikt om toekomstige classificaties te voorspellen. Twee van deze technieken hebben we besproken in deze cursus. Bij de eerste, beslissingsbomen, wordt er een boomstructuur opgesteld a.d.h.v. de trainingsdata waarna de testdata eenvoudig geclassificeerd kan worden. Bij de tweede techniek, KNN, krijgen we geen verstaanbaar model maar deze techniek werkt wel voor complexe data.

Information retrieval and web search

Information retrieval of kortweg IR is een reeks van technieken die gebruikt worden om tekst te bewerken voordat deze gebruikt worden in data mining algoritmes. IR helpt gebruikers met behulp van *queries* zodat deze informatie kunnen bijwinnen. Algemeen definieert men IR dus als de studie die de verzameling, verwerking, opslag en verspreiding van informatie mogelijk maakt.

Architecturaal kan men IR als volgt voorstellen:



Figuur 12: De architectuur van IR.

Hierbij kan de gebruiker dus aan de hand van query's informatie opvragen. Gebaseerd op het document dat hij hierbij terugkrijgt kan de query dan eventueel aangepast worden. Er zijn verschillende soorten query's mogelijk:

- **Keyword queries**: Zoeken met kernwoorden zoals bijvoorbeeld op Google.
- **Boolean queries**: AND, OR, NOT gebruiken om query's samen te stellen.
- **Phrase queries**: Zoeken via zinnen.
- **Proximity queries**: Zoeken naar woorden die normaal gezien vaak samen voorkomen.
- **Full document queries**: Gebruikt bij bijvoorbeeld plagiaatcontrole.
- **Natural language questions**: Gebruikt bij *voice assistants* zoals Siri, Cortana, Alexa ...

Een IR model bepaalt hoe een document en een query zijn opgebouwd. De meest voorkomende zijn booleaanse modellen, *vector space* modellen en *statistical language* modellen.

Booleaans model

Bij een booleaans model wordt elk document en elke query beschouwd als een groep van woorden. De volgorde waarin deze voorkomen is niet belangrijk. Algemeen stellen we dat voor een groep documenten D we $V = \{t_1, t_2, ..., t_{|V|}\}$ definiëren als de woordenschat van deze documenten. Aan elk woord t_i uit document $d_j \in D$ wordt een gewicht $w_{ij} = 1$ toegekend. Voor termen die niet voorkomen in document d_j is het gewicht gelijk aan nul. Het document is dus gelijk aan:

$$d_j = \left(w_{1j}, w_{2j}, .., w_{|V|j}\right) \tag{37}$$

Met behulp van booleaanse operatoren kunnen query's op opgebouwd worden; waarbij het gewicht w_{ij} in beschouwing wordt genomen. Ieder document waarvoor het resultaat *true* is, wordt opgehaald en teruggegeven. De resultaten voor deze query's zijn vaak slecht omdat er geen rekening wordt gehouden met de frequentie van woorden. De gewichten zijn immers 0 of 1. Een voorbeeld van zo een query: (*voetbal* AND (NOT *anderlecht*)).

Vector space model

Bij een *vector space model* of VSM wordt een document opnieuw voorgesteld als een groep van woorden. Elk document is een vector met daarin per woord een gewicht. In tegenstelling tot bij het booleaans model is het gewicht niet gewoon 0 of 1. Om dit gewicht te berekenen maken we gebruik van een *Term Frequency* (TF) model. De definitie hiervan is eenvoudig; het gewicht van een term t_i in een document d_j is het aantal keren dat t_i voorkomt in d_j voorgesteld in f_{ij}. We kunnen deze waarden ook normaliseren. We spreken dan van TF-IDF model. IDF staat voor *Inverse Document Frequency*. De gewichten bij TF-IDF worden dan als volgt berekend:

$$tf_{ij} = \frac{f_{ij}}{max_{f_j \in d_j} f_j} \tag{38}$$

Met N het aantal documenten en df_i het aantal documenten dat t_i bevat:

$$idf_i = \log \frac{N}{df_i} \tag{39}$$

Wordt het gewicht:

$$w_{ij} = tf_{ij} \cdot idf_i \tag{40}$$

We houden dus niet alleen rekening met de frequentie van een woord in 1 document, maar kijken over alle documenten heen.

Het opstellen van een query is bij VSM is minder eenvoudig dan bij een booleaans model en maakt gebruik van een inwendig

product (dot product) tussen 2 vectoren. We stellen immers zowel de query als het document voor als een vector. Hoe beter de vectoren overeenkomen, hoe groter de cosinus tussen deze vectoren. Algemeen berekenen we de overeenkomst als

$$\cos(d_j, q) = \frac{d_j \cdot q}{||d_j||||q||} = \frac{\sum_{i=1}^{|V|} w_{ij} w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2 \sum_{i=1}^{|V|} w_{iq}^2}} \tag{41}$$

Voorbeeld Stel dat we volgende woordenschat hebben: *hardware,software,users* en volgende set van documenten met als query: 'hardware and software':

Set	**Inhoud**
A_1	$\{1,0,0\}$
A_2	$\{0,1,0\}$
A_3	$\{0,0,1\}$
A_4	$\{1,1,0\}$
A_5	$\{1,0,1\}$
A_6	$\{0,1,1\}$
A_7	$\{1,1,1\}$
A_8	$\{1,0,1\}$
A_9	$\{0,1,1\}$

Tabel 6: Een verzameling met applicaties D.

Bij het booleaanse model krijgen we dan A_4 en A_7 terug. We kijken immers gewoon of de term voorkomt en gebruiken een AND-relatie.

Bij VSM berekenen we telkens de cosinus tussen de twee vectoren. De query—voorgesteld als vector—ziet er als volgt uit:

$$q(1,1,0) \tag{42}$$

We krijgen de volgende waarden per document:

Set	**Inhoud**	**cos**
A_1	$\{1,0,0\}$	0,71
A_2	$\{0,1,0\}$	0,71
A_3	$\{0,0,1\}$	0
A_4	$\{1,1,0\}$	1
A_5	$\{1,0,1\}$	0,5
A_6	$\{0,1,1\}$	0,5
A_7	$\{1,1,1\}$	0,82
A_8	$\{1,0,1\}$	0,5
A_9	$\{0,1,1\}$	0,5

Tabel 7: Een verzameling met applicaties D.

Hieruit krijgen we dan de volgende documenten terug: $A_1, A_2, A_4, A_5, A_6, A_7, A_8, A_9$.

Text processing

Indien we het huidig model zouden gebruiken, dan zou dit geen goede resultaten geven. Documenten bevatten vaak woorden die geen waarde toevoegen: stopwoorden. Het verwijderen van deze

stopwoorden verhoogt de efficiëntie en accuraatheid van het zoeksysteem. Men kan lijsten met stopwoorden online vinden. Voor sommige toepassingen wordt bovenop deze lijst nog een extra reeks woorden toegevoegd die men ook als stopwoorden beschouwd.

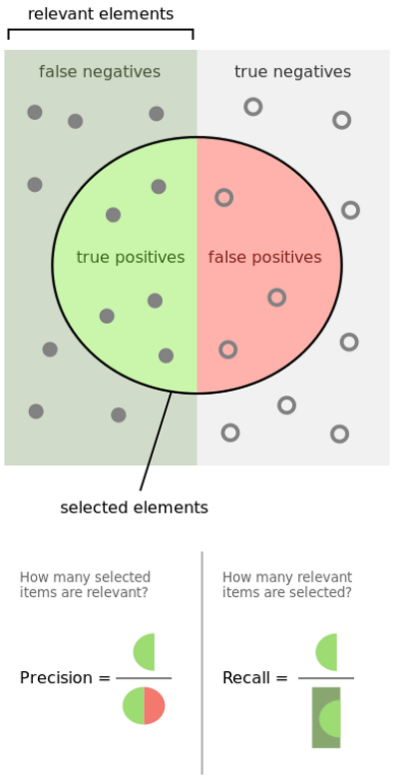
Naast stopwoorden moet men ook nog *stemming* toepassen, waarmee herleid wordt naar de stam van een woord. Woorden zoals loop, loper, loopt, gelopen . . . zijn kleine veranderingen op het woord loop. *Stemming* maakt het mogelijk om dit allemaal terug te laten wijzen op hetzelfde basiswoord. Ook hier is het doel om de efficiëntie te verhogen en de grootte van de index te verkleinen. Per taal is er een reeks technieken beschikbaar om aan *stemming* te doen. Zo kan men in het Engels bij woorden die op -ing eindigen, dit laatste wegdoen.

Tot slot kan men opnieuw TF-IDF toepassen om de termen bij te houden in de index.

Zoals gezien zijn er meerdere methoden om op queries te reageren. We hebben dus een manier nodig om deze te evalueren. Hierbij stellen we ons 2 vragen:

- Is elke opgehaald document relevant?
- Zijn alle relevante documenten opgehaald?

De eerste vraag heeft betrekking tot de nauwkeurigheid van de zoekopdracht, terwijl de tweede betrekking heeft op de volledigheid. Dit wordt duidelijk gemaakt in de volgende figuur:



Figuur 13: Nauwkeurigheid en volledigheid van een zoekopdracht.

We kunnen deze algoritmes met elkaar vergelijken met de volgende formule; waarmee we dan grafieken kunnen opstellen om de verschillende algoritmes visueel met elkaar te vergelijken.

$$\overline{p}\,(r_i) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} p_j(r_i) \tag{43}$$

Inverted index

Men kan een zoekopdracht via het internet beschouwen als een groot IR systeem. Hierbij moet een *webcrawler* alle webpagina's die deze tegenkomt opslaan in wat men een geïnverteerde indexing database noemt. Voor ieder item kan een lijst met documenten worden bijgehouden waar het item in voorkomt. Dit is een geïnverteerde index, wat als voordeel heeft dat het vinden van documenten in vaste tijdsduur kan gebeuren. Daarnaast kunnen meerdere query-elementen ook eenvoudig verwerkt worden.

Als we een query q beschouwen, dan omvat zoeken de volgende stappen:

1. **Vocabulary search**: Zoek alle woorden in q in de geïnverteerde index.
2. **Resultaten mergen**: *Merge* de resulaten die alle of enkele woorden uit q bevatten.
3. **Rank score computation**: Order de documenten.