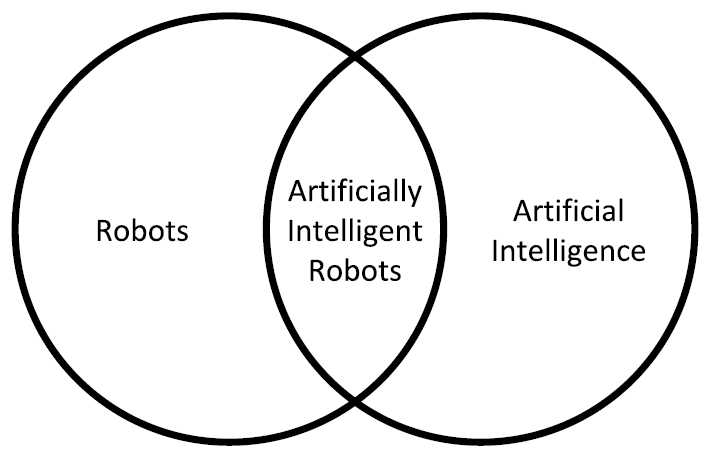
Goals AI & Robotics

# Week 1

## \* can describe in own words the relation between robots and AI, including the intersection and draw a venn diagram depicting this relation.

We hebben robots of machines die zelfstandig een of meerdere taken kunnen uitvoeren. AI is het kunnen vervolledigen van menselijke taken maar dan door robots. Voorbeelden hiervan zijn visuele begrijping, spraakherkenning, beslissingen maken en vertalingen. De relatie is dat deze 2 zonder elkaar of met elkaar kunnen bestaan. Samen is het best of both worlds.



## \* can explain the term robotics, robots and AI in own words.

Robotics is een tak van technologie dat zich bezig houdt met het ontwerpen, bouwen, uitvoeren en toepassen van het gebruik van robots.

Een robot is een programmeerbare machine die een serie van complexe acties automatisch kan uitvoeren.

AI is het kunnen vervolledigen van menselijke taken maar dan door robots. Voorbeelden hiervan zijn visuele begrijping, spraakherkenning, beslissingen maken en vertalingen.

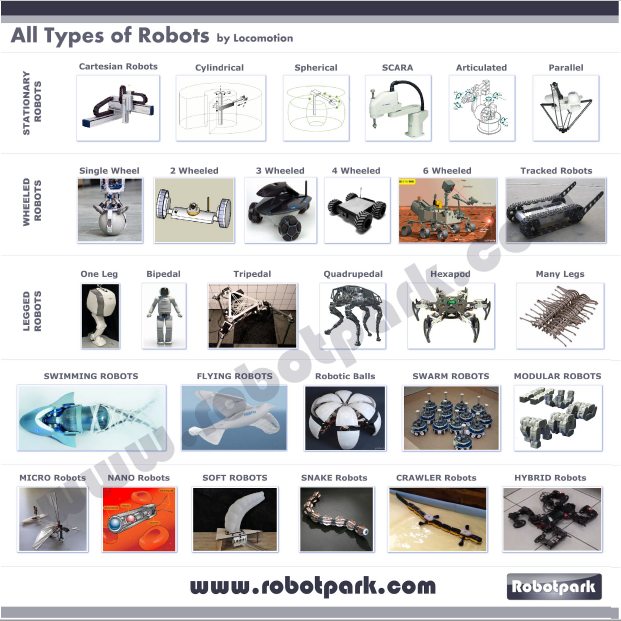
## \* can describe the history of robots in and provide examples of historically important robots.

4de eeuw VC. Vogel gemaakt die vloog op stoom  
1495 Leonardo da Vinci – eerste menselijke LIJKENDE robot  
1810 eerste menselijke robot  
1970 opkomst Japanse robots  
2013 BD – Wildcat  
2015 BD – Spot  
2016 BD – Atlas  
2017 BD – Handle & Atlas & Spotmini  
2018 BD – Spotmini

## \* knows the origin of the term “robot”.

Robot werd gebruikt voor fictionele automatisering. Het woord robot is uitgevonden van het Czechische ‘robota’ wat ‘dienen’ betekent.

## \* can categorize robots according to its type.



## \* can describe the difference between weak and strong AI and use examples to further clarify the difference.

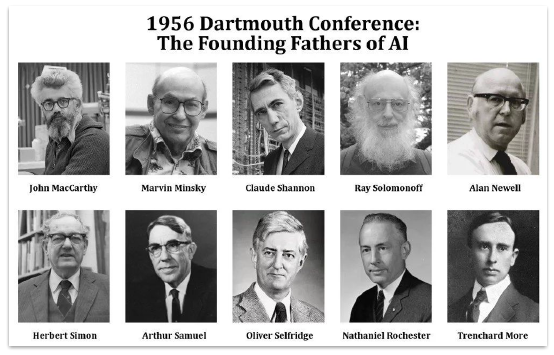
|  |  |
| --- | --- |
| Weak | Strong |
| Eén taak | Alle taken |
| Geen bewustzijn | Wel bewustzijn |
| Automatisatie | Gelijk of beter dan  Menselijke intelligentie |
| Optimalisatie |
| Analyse |
| Gebasseerd op data en statistieken | Gebasseerd op redenering |
| Alle huidige systemen | **Toekomst & science fiction?** |

## \* can describe the history of AI and realize the historical importance of Turing, Zuse, Ada, The Founding Fathers of AI.

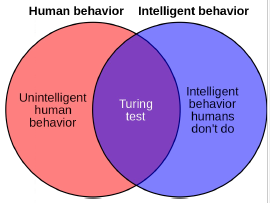
Alan Turing:  
Een computer is intelligent als de gebruiker niet kan zeggen of het een mens of robot is.

Zuse:  
Het gevaar van computers die zoals mensen worden is niet zo groot als het gevaar van mensen die zoals computers worden.

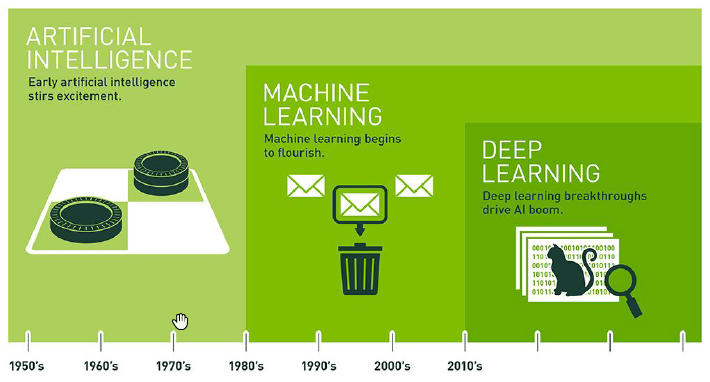
Ada:  
Ada werd de eerste ‘programmeur’ gevonden hoewel er in die tijd nog geen computers bestonden. Zij heeft als eerste een zeer getailleerde reeks van bewerkingen opgeschreven die een computer moest doen om een probleem op te lossen.

The Founding Fathers of AI:  


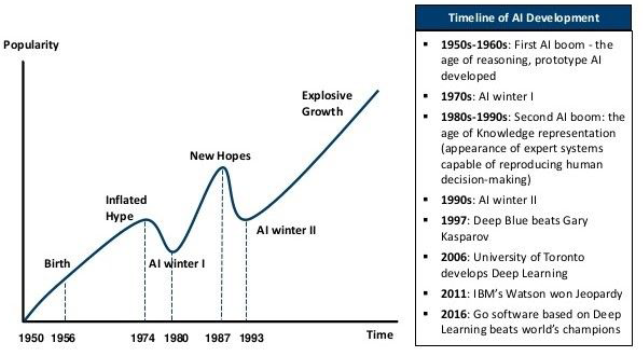
## \* can explain the Turing test in his or her own words.



## \* can describe the subsets of AI: ML and DL in own words using a venn diagram and a timeline.



## \* can draw a timeline of the long history of AI, pointing out the numerous AI winters.



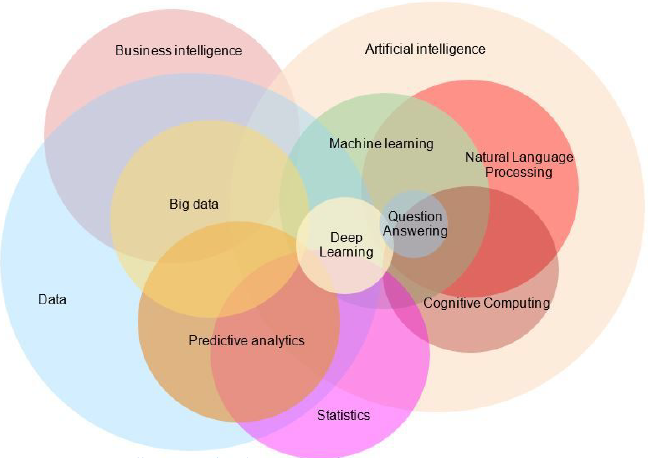
## \* can describe the relation between Moore’s Law and AI.

Moore’s law zegt dat wanneer performance verdubbelt, de kost halveert. AI streeft ernaar meer data te kunnen verwerken terwijl dit minder resource-intensief is.

## \* can describe the relation between the amount of available data and AI.

Hoe meer data er beschikbaar is, hoe meer situaties een AI voor zich krijgt en hoe beter hij getraind is.

## \* can describe the limitations of the standard AI venn diagram and explain AI using a provided diagram depicting different fields.



## \* can explain the different learning types (supervised, unsupervised and reinforcement) using own words and examples.

Supervised:  
Data met labels, directe feedback en voorspellen van uitkomst of toekomst.  
Een voorbeeld hiervan is objectherkenning.

Unsupervised:

Data zonder labels, geen feedback en het zoeken van een verborgen structuur in de data.  
Een voorbeeld hiervan is het classificeren van vogels op basis van tekeningen.

Reinforced:  
Process van keuzes maken, gebasseerd op systeem met beloningen en het leren van een serie van acties.  
Een voorbeeld hiervan is het leren spelen van Mario.

## \* can describe 10 ethical aspects of AI in own words.

- Werkloosheid  
- Verdeeldheid (geen gelijke rechten of verdeling van verdiensten)  
- Menselijkheid  
- Artificiële domheid (behoeden op fouten)  
- Racistisch  
- Veiligheid  
- Kwaadaardige geniën  
- Vereenvoudiging of simpliciteit  
- Rechten van robots  
- Verantwoordelijkheid

## \* can explain the link between the technology/humanity balance and the failure or success for all of us.

Als we de balans juist krijgen, zullen robots een grote hulp zijn in alles wat we doen. We moeten streven dat robot en mens naast elkaar samen kunnen werken.

## \* can explain why (current) AI/technology has no ethics.

Momenteel is de ontwikkelaar verantwoordelijk voor de keuzes van de robot. De robot kan nog niet zelfstandig denken of over ethiek denken.

## \* can describe the term ethics in own words.

Ethics is het verschil weten tussen wat maatschappelijk aanvaard wordt om te doen en welke kracht je hebt om iets te doen.

## \* can explain the link between AI and GDPR.

Er is heel veel data nodig om aan AI te doen. Deze data is niet altijd publiek beschikbaar. Vooral als er men data van mensen gewerkt wordt, is GDPR belangrijk.

## \* can think critically about the ethical aspects of killerbots.

Ethisch te gevaarlijk en niet verantwoord.

## \* can describe in simple terms the Belgian killerbots law.

Het Belgische leger mag geen gebruik maken van volledig autonome wapens (killerrobots). Streeft naar een internationaal verbod. Met autonoom bedoelt de wet, zonder menselijke tussenkomst.

## \* can explain the implications of hacking on devices/robots.

Als een robot gehackt wordt, kan er mee gedaan worden wat de hacker wil. Dit geldt ook op illegale dingen of zelfs moorden.

## \* can describe the (possible) impact of robots/AI on the workforce, scientific community, ...

Robots en AI-systemen kunnen gemakkelijk zonder extra kosten tot 50% van het menselijke werk overnemen. Dit zorgt dat er banen verdwijnen zoals kassierster maar er komen ook weer banen bij zoals onderhoudstechnici.

## \* can explain the implications of face news & the (possible) link with AI.

AI is zo goed geworden dan het beelden en video’s van bekende personen kan genereren die iets controversieel zeggen. Dit is foutief nieuws en wordt moeilijker en moeilijker om van het echte nieuws te onderscheiden.

## \* can provide some repercussions already happened in the world.

Zelfrijdende auto’s aangevallen door mensen, kassiersters die zelfscankassa’s saboteren, anti-robot protesten, …

## \* can describe the role of open source in AI and the ethical “limits”.

Een voorbeeld hiervan is het project dat zelf teksten kon genereren. Dit project was zo goed dat het te gevaarlijk geacht werd om open source te maken. Er was een ethische limiet dat mensen dit zouden misbruiken voor slechte dingen.

## \* can explain the 9 ethical issues of AI according to the WEF.

- Werkloosheid  
- Verdeeldheid (geen gelijke rechten of verdeling van verdiensten)  
- Menselijkheid  
- Artificiële domheid (behoeden op fouten)  
- Racistisch  
- Veiligheid  
- Kwaadaardige geniën  
- Vereenvoudiging of simpliciteit  
- Rechten van robots

## \* can explain the 3 levels of moral agency.

1. Operational morality:  
Moraal ligt volledig bij ontwikkelaars en gebruikers.

2. Functional morality:  
De mogelijkheid waarbij een AMA (Artificial Moral Agent) keuzes maakt i.p.v. ontwikkelaars.

3. Full moral agency:  
Volwaardige aanvoelende robots.

## \* can describe the difference between roboethics and machine ethics.

Roboethics:  
Het morele gedrag van mensen hoe zij AI-dingen ontwerpen, bouwen, gebruiken en behandelen.

Machine ethics:  
Hoe implementeren we morele keuzes bij computers en robots?

## \* EXTRA goal van Joachim – wat zijn Asimov’s 3 wetten

* Een robot kan geen mens verwonden of, door niet-handelen, een mens schade toebrengen
* Een robot moet de regels volgen die hij heeft gekregen van de mens als ze niet tegen de eerste regel in gaan.
* Een robot moet zijn eigen kunnen beschermen zolang het niet tegen de boven gaande wetten gaat.

## \* EXTRA goal van Joachim – wat zijn Murphy & Wood’s 3 wetten

* Een mens mag een robot niet gebruiken zonder het mens-robot werksysteem dat voldoet aan de hoogste wettelijke en professionele normen van veiligheid en ethiek.
* Een robot moet reageren op mensen als geschikt voor hun rollen.
* Een robot moet voldoende zijn eigen bestaan beschermen bestaan zolang deze bescherming niet conflict met de Eerste en Tweede Wetten.

## \* EXTRA goal van Joachim – wat zijn de EPSRC principes van robots?

* Robots zijn multi-use tools en mogen niet gemaakt worden om mensen pijn te doen of te vermoorden behalve als het te maken heeft met de nationale veiligheid.
* Mensen, geen robots, zijn verantwoordelijke agenten. Robots moeten zo ontworpen en geëxploiteerd worden als voor zover uitvoerbaar om te voldoen aan bestaande wetgeving, fundamentele rechten en vrijheden, inclusief privacy.
* Robots zijn producten en moeten worden ontworpen worden met verantwoordelijkheid op veiligheid.
* Robots zijn vervaardigde artefacten. Ze moeten niet op een misleidende manier worden ontworpen om misbruik maken van kwetsbare gebruikers; in plaats daarvan moet de aard van hun werking transparant zijn.
* De persoon met wettelijke verantwoordelijkheid voor een robot moet worden toegeschreven.

## \* can write fluent Python code.

N.v.t.

## \* can set up a Python environment using pip, ... and conda.

N.v.t.

## \* can set up a Jupyter environment.

N.v.t.

## \* can log data to the console and a file.

N.v.t.

## \* can describe the pros and cons to logging to the console and a file.

+ De logging komt niet in je algemene outputstream.  
+ De logging is bij een zware crash of reboot nog altijd beschikbaar.  
+ Er kan gewerkt worden met levels van ernstigheid.  
+ Propere code.

## \* can track changes in a file using tail -f.

N.v.t.

## \* can create unit tests for Python.

N.v.t.

# Week 2

## \* can explain in own words what a tree is and its use cases

Een boom is een hiërarchisch ADT (Abstract Data Type) en is recursief gedefinieerd.

## \* can determine when a tree is a good data structure to use for a given problem

Een boom is handig voor biologische taxonomiën, familiebomen, organisatorische structuren, …

## \* can explain in own words what a binary tree is and its use cases

Een binaire boom heeft maximum 2 kinderen (linker en rechter).

## \* can implement an arbitrary tree

N.v.t.

## \* can implement pre-order, post-order and in-order tree traversal for arbitrary trees

Pre-order is bol links tekenen op de node.

In-order is bol onderaan tekenen op de node.

Post-order is bol rechts tekenen op de node.

Bij alle 3 de gevallen gaat het pad van links naar rechts terwijl hij alle nodes afgaat.

## \* can explain in own words the state space of a problem

De state space is een soort status van het huidige probleem. Als men verschillende acties onderneemt, zullen er verschillende statussen zijn.

## \* can represent the state space of a problem

Oefening.

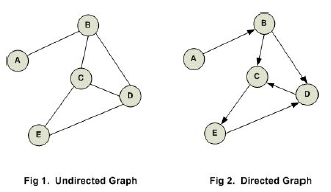
## \* can apply the state space procedure for a given problem

1. Neem een manier om de statussen in het probleem voor te stellen (duidelijk)  
2. Formuleer alle mogelijke acties (Productieregels)  
3. Representeer de initiële statussen  
4. Formuleer wanneer een status het doel van ons probleem bereikt.  
5. Activeer de productieregels op de initiële status en zijn kinderen tot een doelstatus bereikt is.

## \* can explain in own words what a graph is and its use cases

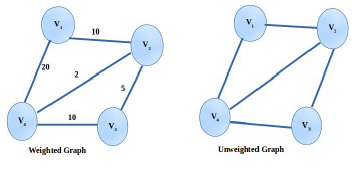
Een graaf is een andere soort voorstelling dan een boom maar laat een oneindig aantal linken naar een andere node toe.

## \* can explain in own words the difference between undirected and directed in the context of graphs

Bij undirected moet er geen richting die gevolgd worden. Bij directed wel. 

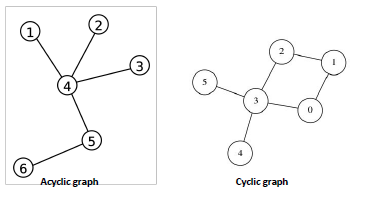
## \* can explain in own words the difference between unweighted, weighted in context of graphs.

Unweighted betekent dat elke route evenveel kost. Bij weighted kosten sommige routes meer dan anderen.



## \* can explain in own words the difference between acyclic and cyclic in context of graphs.

Cyclische grafen hebben connecties waar cirkels in zitten of waar je bij oneindig door kan loopen. Bij Acyclische kan dit niet.



## \* can determine when a graph is a good data structure to use for a given problem

Een graaf wordt gebruikt voor sociale relaties, mappen, neurale netwerken, etc voor te stellen.

## \* can implement and differentiate the different graph representations

N.v.t.

## \* Extra goal van Joachim: matrix <-> list

Een adjacency matrix wordt gebruikt voor grafen met onderling veel connecties (dense). Een adjacency list wordt gebruikt voor grafen met onderling weinig connecties (sparse).

## \* can explain in own words what Big O notation means

Big O notatie in matematisch geschreven door O(f) waarbij O staat voor de grootte van de omvang en f waartegen we de complexiteit aan het vergelijken zijn.

## \* can explain why only the term with the highest order of magnitude matters

De andere operaties die niet tot de hoogste macht horen, zijn verwaarloosbaar klein. De tijd gaat per macht, exponentieel omhoog.

## \* can derive the term with highest order of magnitude from a given polynomial

Oefening.

## \* can describe in own words the link between complexity and efficiency

Hoe minder complex een algoritme is, hoe sneller en dus hoe efficiënter het is.

## \* can explain the two most important parameters to measure efficiency

- Aantal stappen  
- Input

## \* can describe in own words the three different atomic step/instruction types of an algorithm

Atomic assignment instruction  
bvb. N = 10

Atomic Read/Write instruction  
bvb.

Atomic test instruction  
bvb. for i in range(1, n + 1):

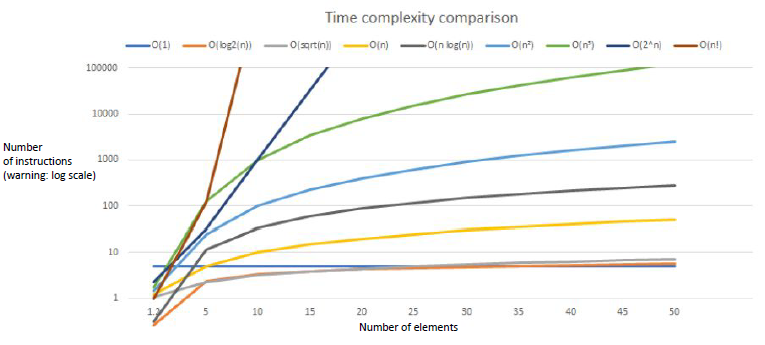
## \* can evaluate the time complexity of a given algorithm

Oefening.

## \* can explain the difference between worst-, average- and best-case time complexity

Slechtste, gemiddelde en slechtste geval.

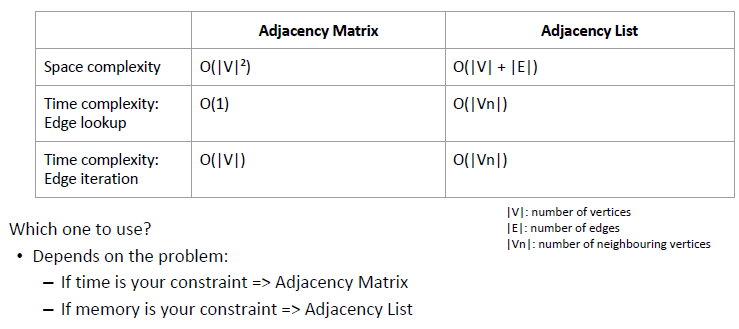
## \* can understand the implications of complexity growth w.r.t. input for different mathematical functions (constant, linear, logarithmic, quadratic, cubic, exponential and factorial)



## \* can evaluate the space complexity of data structures and algorithms

Beschrijft het aantal geheugen er nodig is om het algoritme op te lossen.

## \* can describe the complexity (space and time) for graph representations



## \* can describe and transform a graph with a start and end node into a partial paths tree representation

Oefening.

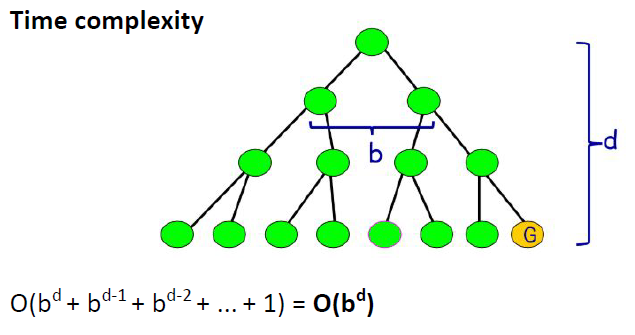
## \* can explain and implement depth-first search

- Start vanaf de root  
- Selecteer een kind (conventie links eerst)  
- Ga zo ver mogelijk door tot het laatste kind.  
- Ga een niveau hoger tot er terug kinderen zijn.  
- Ga naar stap 2

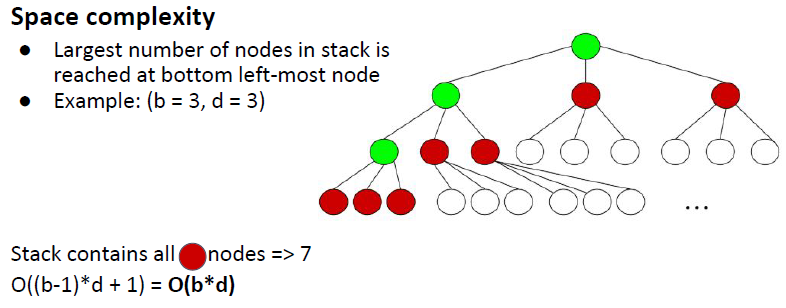
## \* can analyze the completeness of depth-first search

Gaat altijd een pad bij een bepaald aantal nodes hoewel dit niet altijd het kortste zal zijn.

## \* can evaluate the time complexity of depth-first search



## \* can evaluate the space complexity of depth-first search



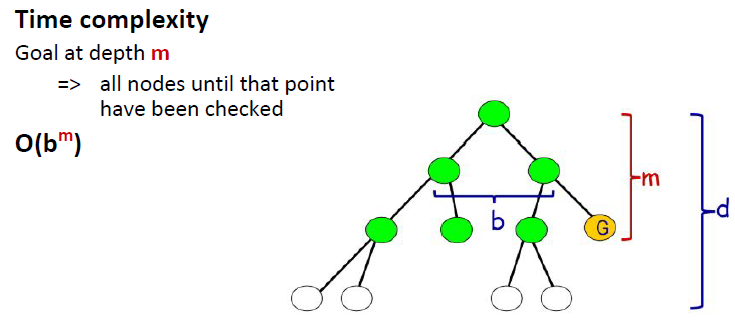
## \* can explain and implement breadth-first search

- Start aan de root  
- Ga naar het volgende niveau  
- Itereer over alle nodes op dit niveau  
- Ga naar stap 2

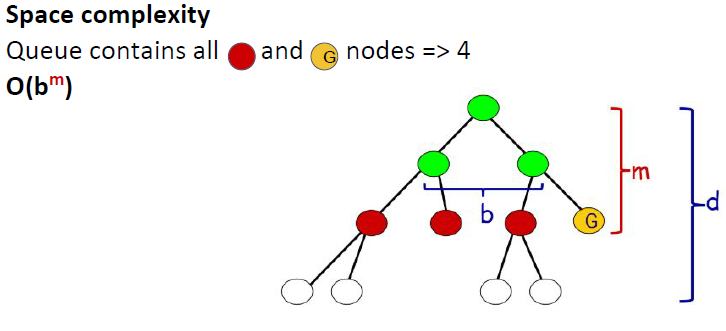
## \* can analyze the completeness of breadth-first search

Gaat altijd een pad vinden, ook bij oneindig aantal nodes. Gaat altijd het kortste pad vinden.

## \* can evaluate the time complexity of breadth-first search



## \* can evaluate the space complexity of breadth-first search



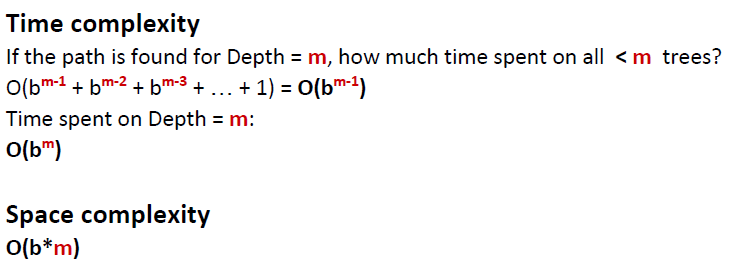
## \* can explain and implement iterative deepening search

Beperking van depth-first tot een vaste diepte. Als er geen pad gevonden werd, wordt de diepte verhoogd. Combinatie van het beste van depth-first en breadth-first.

## \* can analyze the completeness of iterative deepening search

Gaat altijd het kortste pad vinden.

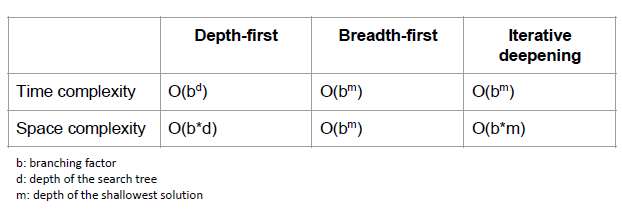
## \* can evaluate the time complexity of iterative deepening search



## \* can evaluate the space complexity of iterative deepening search



## \* can compare the different blind search techniques



## \* can explain uniform cost search in own words

UCS is “het beste eerst” zoekalgoritme. Bij elke stap wordt de node met de laagste kost geselecteerd tot het doel wordt bereikt.

## \* can implement uniform cost search

N.v.t.

## \* can describe the link between UCS and breadth-first search

Breath-first gaat ook eerst het hele niveau scannen en dan naar het volgende niveau. Daar wordt dan bij UCS gekeken welk kind de laagste kost heeft en gaat daar naartoe.

## \* can explain in own words what accumulated cost means

AC betekent de kost van het huidig al gedane pad + de afstand tot de volgende node.

## \* can explain how UCS selects the next node

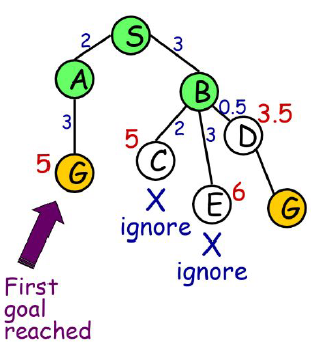
UCS kijkt welk kind de laagste kost heeft en gaat daar naartoe.

## \* can explain using an example why UCS does not always find the most optimal path

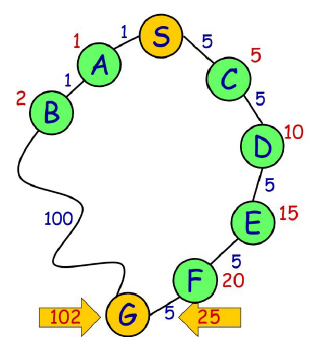


## \* can explain in own words and using an example the branch-and-bound principle

Branch and bound houdt rekening met de oplossing en zijn kost en blijft doorzoeken naar een betere oplossing.



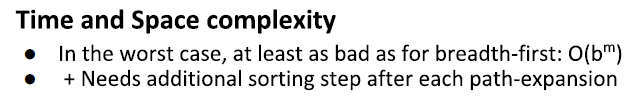
## \* can explain using an example why UCS with branch-and-bound finds the most optimal path



## \* can implement optimal UCS (using branch-and-bound)

N.v.t.

## \* can describe the time and space complexity of optimal UCS (using branch-and-bound)



## \* can explain the link and difference with Dijkstra’s algorithm

Dijkstra’s is een algemenere vorm van UCS. Dijkstra vindt het kortste pad van de startnode tot elke andere node. Historisch belangrijk algoritme.

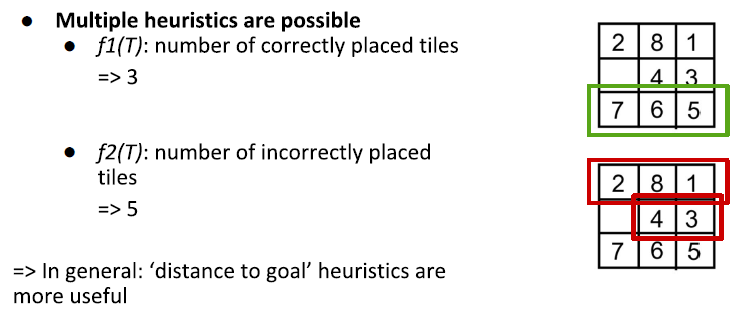
## \* can explain in own words what a heuristic is

Een heuristic is een gok voor hoeveel de volgende stap zou kosten.

## \* can use a given heuristic for a state space

Oefening.

## \* can explain the heuristics used in the 8-puzzle problem



## \* can describe in own words what path deletion is and how it works

Path deletion verwijdert paden vanaf nodes die al eerder gevonden zijn door met een korter pad. M.a.w. als een node opnieuw ontdekt is terwijl de eerste keer toen dit ontdekt werd, er een kleinere kost is dan nu, wordt het nu ontdekte pad verwijdert.

## \* can implement path deletion

N.v.t.

## \* can explain A\* search in own words

A\* gebruikt B&B principe, Heuristics, Redundant PD en UCS.

## \* can implement A\* search

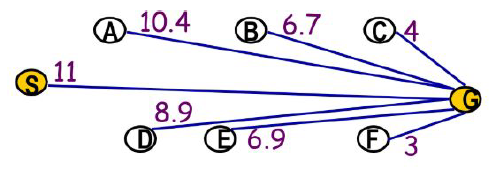
N.v.t.

## \* can describe the heuristic function used in A\*

De heuristic in A\* is de kost van het path + de heuristic van dat pad. De heuristic wordt met berekent door de straight-line distance.

## \* can explain the straight line distance using an example

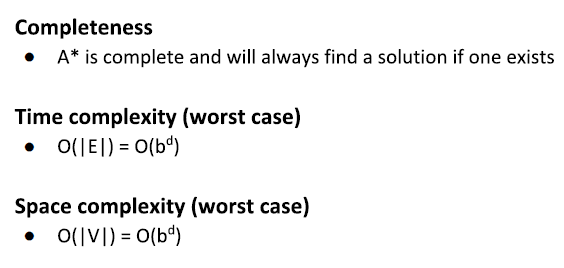
De straight-line distance berekent de afstand tot die node in vogelvlucht.



## \* can explain the importance of underestimation in the context of heuristics using an example

Bij een overschatting bestaat er de kans dat het optimale pad niet gevonden wordt.

## \* can describe completeness, time and space complexity of A\*



# Week 3

## \* can describe & explain in own words the position game AI as a subfield

Een van de oudste subvelden van AI. Het is een abstracte en pure vorm van competitie die intelligentie nodig lijkt te hebben. Spellen zijn een speciale vorm van zoekproblemen en hebben enkele nieuwe requirements.

## \* can describe in own words the link between tree search and game AI

Een game AI gaat via een tree zoeken naar een oplossing waarbij hij wint. Hij gaat zijn huidige status bekijken en daarna zijn kinderen afgaan en met een algoritme kijken welke optie de beste lijkt.

## \* can explain in own words the differences between simple and more complex board games in context of game AI using real world games

Simpele spellen zoals bordspellen hebben relatief weinig staten en acties. Er is ook weinig wereldkennis vereist. Een voorbeeld hiervan is schaken.

Complexere spellen hebben oneindig veel staten en acties. Te groot om de state space van weer te geven. Een voorbeeld hiervan is Go.

## \* can explain in own words the term “contingency” of a problem

De AI weet niet wat de volgende stap van de tegenspeler gaat zijn.

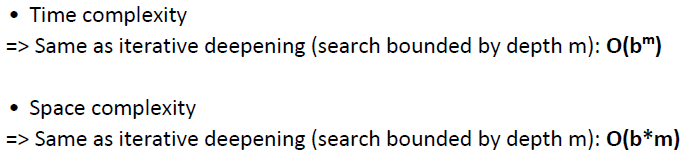
## \* can describe in own words how a game AI using tree search on an abstract level

Het game AI algoritme zoekt in de boom naar boven tot een bepaalde diepte. Daarna wordt een evaluatiefunctie gebruikt. Deze evaluatie wordt verder naar boven in de boom gebruikt.

## \* can explain in own words Minimax in context of game AI

Stel dat we MAX als AI speler nemen en MIN als zijn tegenstander. We voeren minimax uit door een diepte te kiezen, de boom op te bouwen tot die diepte en dan de evaluatiefunctie uit te voeren voor de kinderen. MAX gaat altijd het maximum nemen en MIN gaat altijd het minimum nemen. Dit is omdat MAX MIN’s score wilt maximaliseren en omdat MIN MAX’s score wilt minimaliseren.

## \* can explain in own words the time and space complexity of Minimax



## \* can implement Minimax for a given problem

N.v.t.

## \* can explain in own words an improvement of Minimax

Een probleem van Minimax is dat het geen verschil maakt tussen een vroeg of laat verlies. Een oplossing hiervoor is om de diepte mee in de evaluatiescore te steken.

## \* can implement an improvement of Minimax

N.v.t.

## \* can explain in own words Alpha-Beta pruning in context of game AI

Dit is een optimalisatie voor Minimax. In plaats van tot op diepte de hele boom te bouwen en de functie uit te voeren, gaat een deel van de generatie genomen worden om de functie op uit te voeren. Dit zal leiden tot dat sommige delen redundant zullen zijn en deze niet in de boom mee moeten opgenomen worden.

## \* can explain in own words the best case and worst case gain of Alpha-Beta pruning in context of game AI

Best case:  
Op elke laag is de beste node, de meest linkse.

Worst case:  
Geen verbetering.

## \* can describe in own words the term Heuristic continuation in context of game AI and what problem it solves in context of Alpha-Beta pruning using a real world example

Bij schaken zijn bijvoorbeeld strategisch gezien, bepaalde stukken waardevoller dan andere stukken. Hierdoor moet er voor sommige paden voorbij de voor gedefinieerde diepte gezocht worden.

## \* can implement Alpha-Beta pruning for a given problem

N.v.t.

## \* can implement Heuristic continuation for a given problem

N.v.t.

# Week 4

## \* can explain why an IDE needs to be run from the command line

Als dit niet gedaan wordt, wordt .bashrc niet uitgevoerd en wordt ROS niet gesourced en zal de IDE of de terminal in de IDE geen ROS verstaan.

## \* can describe ROS in own words

ROS is een open source robotica-framework die de mogelijkheid biedt om via 1 programma met heel veel verschillende robots te communiceren.

## \* can describe flexible robotics in own words

Flexible robotics betekent dat het ROS het makkelijk maakt om met wiskundig moeilijk te beschrijven en te controleren robots te werken.

## \* can explain in own words the benefits of using ROS

+ Verschillende drivers reeds ingebouwd  
+ Samenhangend framework voor zowel perceptie, navigatie en planning  
+ Makkelijk log en error handling inbouwen

## \* can explain "structured communications layer" in context of ROS

ROS voorziet buiten management en planning ook een gestructureerde communicatielaag of m.a.w. een ingebouwde tussen verschillende nodes of componenten.

## \* can explain the salient characteristics of and its consequence to ROS

- Alleenstaande robot  
- Workstation-soort berekenkracht aan boord  
- Geen real-time requirements  
- Uitstekende netwerkconnectiviteit  
- Applicaties reeds in onderzoek (academia)  
- Maximum flexibiliteit

## \* can describe in own words "distributions" in context of ROS

Distributies zijn verschillende versies van ROS.

## \* can describe in own words different usages of ROS

- Robotica research  
- Wedstrijden  
- Robot prototypes  
- Robot specifieke ontwikkeling  
- Industrie

## \* can explain the features of ROS

- Gedistribueerd framework (verschillende resources worden als 1 geheel gezien)  
- Interactie tussen nodes  
- Robot platform onafhankelijk  
- Sensor onafhankelijk  
- Makkelijk testen  
- Programmeertaal onafhankelijk  
- Grote hoeveelheid reeds gedeelde code

## \* can explain in own words distributed systems in the context of ROS

- Gedistribueerd framework (verschillende resources worden als 1 geheel gezien)

## \* can explain the differences between a framework and a library

Een framework is in controle. Die roept jouw code aan en jouw code roept een library aan.

- controle life-cycle  
- Reguleert alle communicatie  
- Biedt standaardisatie  
- Uitbreidbaar  
- Kan meerdere libraries bevatten

## \* can explain interaction of nodes in context of ROS

Elke node kan met een willekeurige node praten. Elke node heeft zijn unieke functie.

## \* can describe the downsides of ROS

- Trage vooruitgang naar nieuwe versies  
- Beperkte ondersteuning voor Python 3

## \* can describe the difference between ROS(1) and ROS2 using the salient characteristics of ROS(1) and the new use cases.

|  |  |
| --- | --- |
| ROS 1 | ROS 2 |
| Alleenstaande robot | Team van meerdere robots |
| Workstation-soort berekenkracht aan boord | Kleine embedded platformen |
| Geen real-time requirements | Real-time systemen |
| Uitstekende netwerkconnectiviteit | Niet-ideale netwerken |
| Applicaties reeds in onderzoek (academia) | Productieomgevingen |
| Maximum flexibiliteit | Voorgeschreven patronen voor het opbouwen en structureren van systemen |

## \* can describe in own words roscore, master, parameter server and rosout in own words

Roscore:  
Roscore is een service die moet draaien om de communicatie tussen nodes toe te laten. Elke node registreerd zichzelf bij de roscore.

Master:  
Een ROS master zorgt voor de benaming en registratie van de services van de nodes. Het controleert ook alle publishers, subscribers en services.

Parameter server:  
Dit is een gedeelde bibliotheek toegankelijk via API’s. Nodes kunnen dit gebruiken om runtime parameters op te slagen of op te halen.

Rosout:  
Rosout is een console logging mechanisme in ROS.

## \* can explain in own words nodes, topics, pub/sub, service/client packages in context of ROS

Node:  
Een package of onderdeel dat een bepaalde taak doet.

Topic:  
Een topic bevindt zich tussen 2 nodes en zorgt voor de communicatie tussen beiden bij een Pub/Sub.

Pub/Sub:  
Versturen van continue stream aan data.

Service/Client:  
Eenmalig versturen van iets, one to many relatie. Client moet opvragen, anders gebeurd er niks.

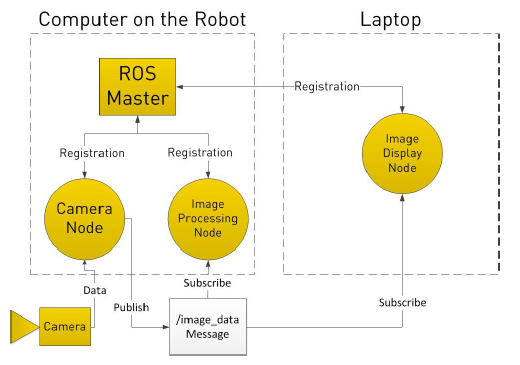
## \* can start, list, inspect and ping existing nodes

N.v.t.

## \* can describe the difference in communication characteristics of pub/sub and service/client in context of ROS

Service/client is trager dan Pub/Sub en is eenmalig.

## \* can explain in own words the ROS architecture on one and multiple computers using schemes.



## \* can explain a complete ROS system from a given scheme

Oefening.

## \* can visualize a ROS system as a graph

Oefening.

## \* can inspect ROS topics

N.v.t.

## \* can inspect ROS messages

N.v.t.

## \* can plot ROS messages

N.v.t.

## \* can navigate a ROS system

N.v.t.

## \* can describe a catkin workspace in own words

Een catkin workspace is een map waar alle catkin packages zich in bevinden.

## \* can explain the package structure of a catkin Workspace

Een workspace heeft meerdere packages. Hierin zit standaard een CMakeLists.txt-bestand in dat zorgt voor bepaalde instellingen voor die package. Daarnaast is er een package.xml die zorgt voor de installatie van de juiste dependencies. C++-bestanden zitten in src en Python-bestanden in scripts.

## \* can create a package with dependencies inside a catkin workspace

N.v.t.

## \* can inspect the dependencies of a ROS Package

N.v.t.

## \* can create and run their own ROS publisher using Python

N.v.t.

## \* can create and run their own ROS subscriber using Python

N.v.t.

## \* can create and run their own ROS service using Python

N.v.t.

## \* can create and run their own ROS client using Python

N.v.t.

## \* can use OpenCV to analyse image (streams) with Python in projects.

N.v.t.

# Week 5

## \* can log information using the correct level with the standard logging module

N.v.t.

## \* can log information using the correct level with the ROS logging system

N.v.t.

## \* can create a Qt graphical user interfaces for ROS nodes

N.v.t.

## \* can connect ROS to Webots

N.v.t.

## \* can create multithreaded GUIs using QT and Python

N.v.t.

# Week 6

## \* can implement the given PIT assignment

N.v.t.

# Week 7

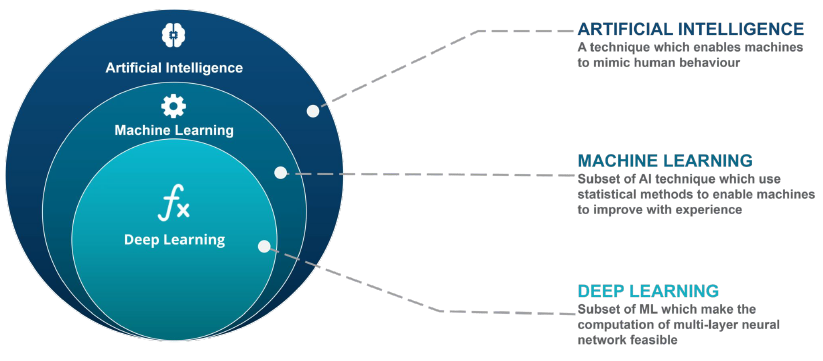
## \* has a working anaconda environment

N.v.t.

## \* can explain Machine Learning in their own words

ML is een branch van AI waar de computer regels genereert gebasseerd op rauwe data.

## \* can situate Machine Learning in the broader context of AI



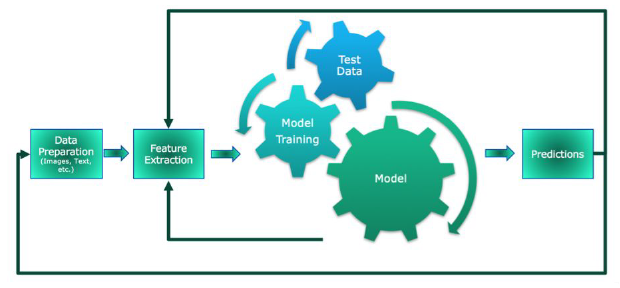
## \* can explain the difference between traditional programming and Machine Learning

Traditioneel programmeren verwacht data en een programma als input en geeft na computatie een output terug. Bij ML worden er data en resultaten samengenomen als input en door computatie ontstaat er een model.

## \* can explain the importance of practical Machine Learning

Praktische ML is gebaseerd op studies waarbij we meer inzichten krijgen over de verschillende soorten oplossingen binnen ML.

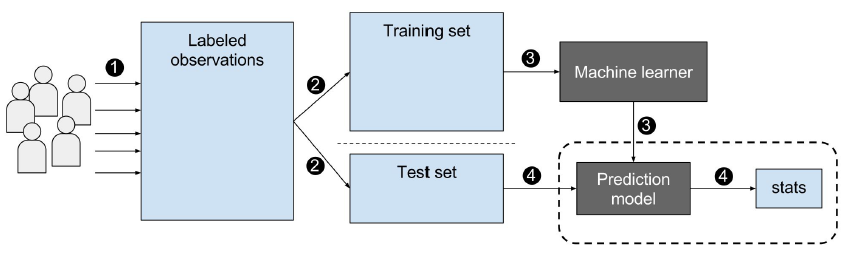
## \* can describe a standard ML pipeline



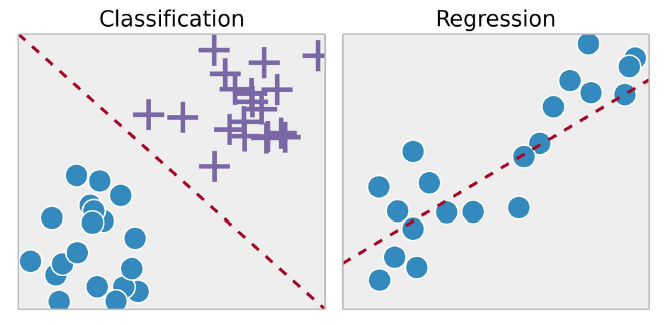
## \* can explain Supervised Learning in their own words

Supervised learning is met gelabelde data, die direct feedback geeft. Er zitten betekenisvolle patronen in de data. Meest gebruikt en gebruikensklaar voor de wereld.

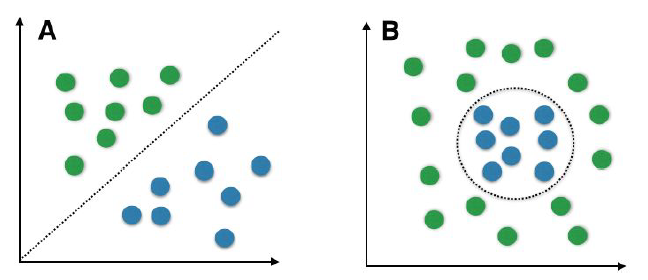
## \* can describe the general flow of a supervised learning pipeline



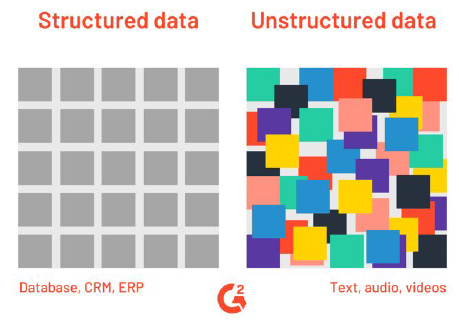
## \* can explain the difference between classification and regression



## \* can explain the difference between linear and nonlinear spread of classification examples



## \* can explain the difference between structured and unstructured data



## \* can explain in their own words what a decision tree is

Een decision tree gaat een doelvariabele y voorspellen door regels te leren die te maken hebben met de features van de inputdata x. Elke branch is een keuze of regel. Elke leaf is een uitkomst.

## \* can explain the difference between classification and regression in context of a decision tree

Een decision tree opgesplitst d.m.v. ja en nee-keuzes is een classificatie terwijl een opsplitsing door waardes een regression is.

## \* can explain the usages of decision trees

Een decision tree wordt vooral bij gelabelde data gebruikt waar snel een goed resultaat verwacht wordt.

## \* can explain how to decide on a split point for a classification tree based on the gini index

Decision tree’s zullen altijd de Gini impurity willen minimaliseren.

Een gini index wordt berekend door 1 – (voor elk deeltje: - deel²)

## \* can explain how to decide on a split point for a classification tree based on entropy and information gain

Decision tree’s zullen altijd de information gain willen maximaliseren. Het attribuut met de hoogste information gain zal eerst gesplitst worden.

## \* can explain how to decide on a split point for a regression tree

Een regression tree gaat kijken naar de mean squared error en gaat de variantie minimaliseren.

## \* can construct a classification decision tree for a given problem based on the gini index

Oefening.

## \* can construct a classification decision tree for a given problem based on entropy and information gain

N.v.t.

## \* can implement a decision tree for a given problem in scikit-learn

N.v.t.

## \* can explain the advantages and disadvantages of decision trees

|  |  |
| --- | --- |
| Voordelen | Nadelen |
| Simpel om te begrijpen | Grote kans op overfitting |
| Makkelijk te visualiseren | Kwetsbaar voor kleine variaties in de data |
| Kan numerische en categorische variabelen aan | Moeilijk om een optimale decision tree te maken |
| Mogelijkheid om multiklass classificatie te doen |  |
| Snel |  |

## \* can explain in their own words the difficulty of describing the “goodness” of a ML model

Elk model is uniek en de goedheid meten is relatief.

## \* can explain in their own words the metric used for classification

De metric voor classificatie is d.m.v. een confusion matrix. (TP, FP, FN, TN) met true, false, positive en negative.

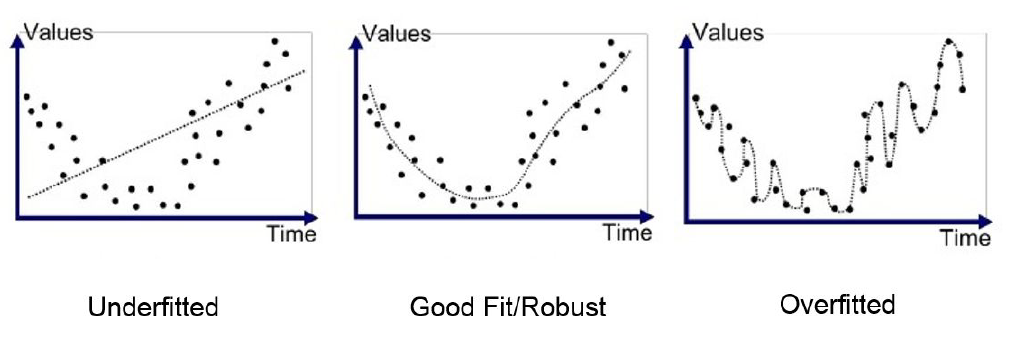
## \* can explain in their own words the metrics used for regression

De metric voor regressie is door het meten van de afstand van de voorspelde waarde tot de echte waarde. Hiervan komt de MSE of de Mean Squared Error verder.

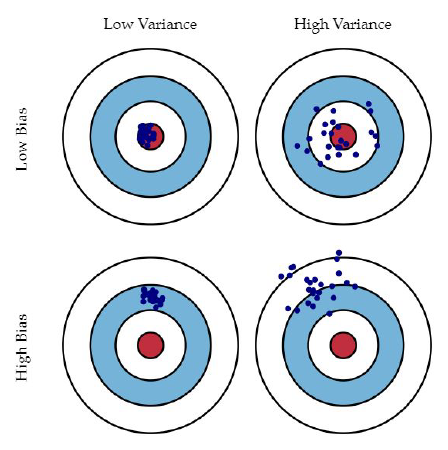
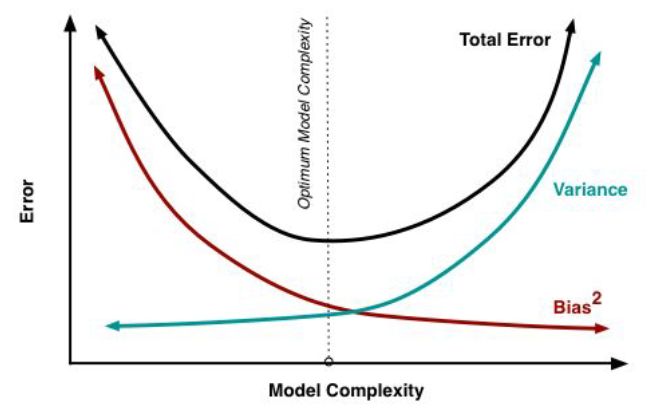
## \* can describe how to go about measuring the “goodness” of an ML model

Het beste dat we kunnen doen is verschillende modellen die op dezelfde data getraind zijn, vergelijken. Dit kan met RMSE en R².

## \* can explain overfitting and underfitting in their own words



## \* can explain the tradeoff between bias and variance

## \* can explain the importance of dividing your data sets in training, validation and test sets

Dit gaat vooral overfitting tegen.

## \* can explain the purpose of the training set

Is nodig om het model te trainen.

## \* can explain the purpose of the validation set

Wordt gebruikt om de hyperparameters te finetunen.

## \* can explain the purpose of the test set

Wordt gebruikt om het model te evalueren.

## \* can explain bagging in the context of random forests in their own words

Door gebruik te maken van bagging, gaan we veel verschillende modellen uitkomen. We van al deze modellen dan het gemiddelde. Dit vermindert bias en variance.

## \* can explain bootstrap sampling

Een aantal voorbeelden worden geselecteerd en in willekeurige subsamples geplaatst.

## \* can explain a random forest in their own words

Een random forest gaat eerst subsamples maken door bootstrap sampling. Daarna worden een hele reeks modellen gebouwd. Daarna wordt er door bootstrap aggregating alle voorspellingen samengenomen en een gemiddelde van genomen.

## \* can explain the importance of random sampling in the context of random forests

Als dit willekeurig gebeurd, betekent dit dat de data zo niet-samenhangend als mogelijk is.

## \* can describe the habit of overfitting in context of decision trees

Er bestaat een kans op overfitting binnen een enkele boom maar dit wordt opgelost omdat er het gemiddelde van veel verschillende bomen wordt genomen.

## \* can describe why the different decision trees in a random forest need to be as uncorrelated as possible

Elke boom of elke predictie gaat op zijn eigen manier hetzelfde proberen te voorspellen. Als deze data niet-samenhangende data is, dan verhogen onze kansen op een correcte voorspelling.

## \* is able to sum up and explain 5 advantages and 2 disadvantages of random forests

|  |  |
| --- | --- |
| Voordelen | Nadelen |
| Algemeen bruikbaar voor gestructureerde data | Tijdrovend voor grote datasets |
| Flexibel en hoge accuraatheid | Niet makkelijk te interpreteren zoals een enkele decision tree |
| Makkelijk te visualiseren |  |
| Numerische en categorische variabelen aankunnen |  |
| Multiklass classificatie aankunnen |  |

# Week 8

N.v.t.

# Week 9

## \* can explain the link between data & optimization

Je kan alleen dingen optimaliseren die je begrijpt.

## \* can describe the reasons for understanding your data

Het is belangrijk te weten hoe het opgeslagen is, in welk formaat en met welke types.

## \* can explain how to prepare and clean a dataset for training a Machine Learning model (categorical variables, dates, missing values)

Categorical variables worden meestal als strings opgeslagen. Het beste is om deze om te zetten naar categorieën. Intern worden deze dan als nummers behandeld. Het is dan ook optioneel om deze te ordenen. Datums bevatten veel neveninformatie over de dag van de week, dag van de maand, schrikkeljaar, etc. Missende waarden krijgen meestal de mediaan van de andere waarden.

## \* can explain one-hot encoding

One-hot encoding is het omzetten van categorieën (zoals kleuren) die als strings opgeslagen zijn naar verschillende kolommen (voor elke kleur eentje) en dan true of false wanneer toepasselijk.

## \* can explain the difference between parameters and hyperparameters

Parameters worden geleerd tijdens de training. Hyperparameters worden door de programmeur bepaald voor de training.

## \* can list and describe 3 ways to tune the hyperparameters of a Machine Learning model

Trial & error, grid search (elke combinatie proberen) en random search.

## \* can list and describe 3 hyperparameters of a Random Forest

N\_estimators:  
aantal bomen dat gebruikt wordt

Min\_samples\_leaf:  
minimale aantal dingen in een node. Verkleint de kans op overfitting.

Max\_features:  
Het aantal features waarbij bij elke node het splitspunt gekozen wordt. Verhoogt randomness.

## \* can explain how to acquire the out-of-bag score for a Random Forest and why it’s useful

Is een goede check voor overfitting, is vooral nodig bij kleine datasets.

## \* can explain why feature importance is important in the context of Machine Learning

Verwijderen van minder belangrijke features zodat de belangrijke naar voren komen, zorgt voor meer correlatie.

## \* can explain the concept of Occam's Razor in the context of Machine Learning

Simpelere oplossing hebben meer kans om correct te zijn dan complexe. Oplossing kiezen met de minste assumpties.

## \* can describe ways to speed up the training process of a Random Forest

Subsampling: kies een willekeurige subset van samples voor elke boom.

# Week 10

## \* can explain boosting in their own words

Boosting is een techniek om bias en variance naar beneden te halen.

## \* can describe the stopping criteria for boosting algorithms

Enkele stopcriteria’s zijn om te stoppen na een aantal bomen, wanneer de foutenratio omhoog gaat of wanneer er geen verbetering meer zichtbaar is op de validatieset.

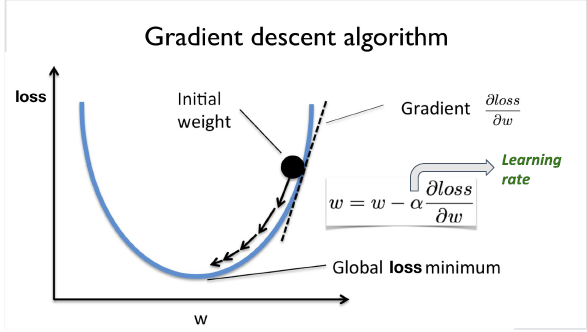
## \* can explain AdaBoost in their own words

Adaptive boosting start met een model te fitten. Bij de errors worden dan de gewichten verhoogt. Bij correcte classificaties worden de gewichten verlaagd. Er wordt opnieuw gefit. Dit wordt herhaald tot een stopcriterium bereikt wordt. Dit wordt voor verschillende modellen gedaan waar uiteindelijk het gemiddelde van genomen wordt. Deze techniek is wel kwetsbaar voor ruisdata.

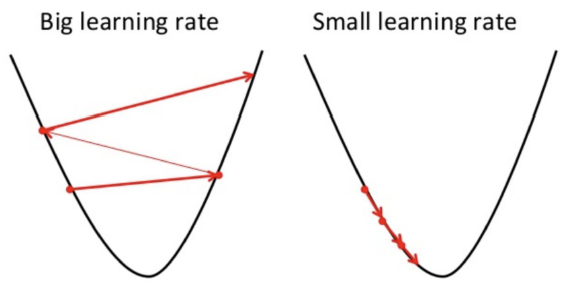
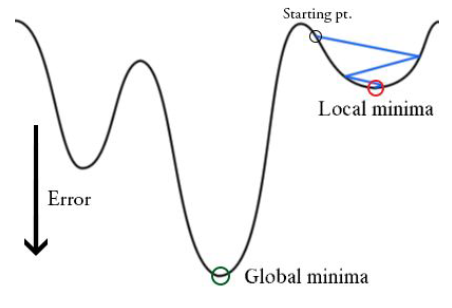
## \* can explain what a loss function is and why we use them

Het doel van de optimalisatie is om fouten te minimaliseren of “loss” te minimaliseren. Een loss-functie beschrijft hoeveel loss er is bij een gegeven soort accuraatheid.

## \* can explain the idea behind gradient descent in their own words



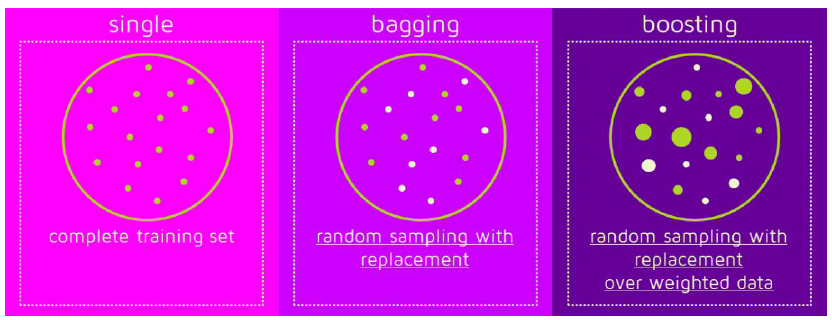
## \* can explain the importance of the learning rate in the context of gradient descent

## \* can explain gradient boosting in their own words

Elke ronde leer je fouten. Dit komt overeen met gradient descent op een loss-functie.

## \* can explain the differences between bagging and boosting



## \* can explain the advantages of XGBoost over other bagging and boosting algorithms

+ Robuust  
+ Niet kwetsbaar voor ruisdata  
+ Underfitted? -> Niet genoeg rondes gedaan  
+ Overfitted? -> Te veel rondes gedaan

# Week 11

## \* can explain how a perceptron works in their own words

Een perceptron neemt meerdere inputs en vormt dit om naar 1 output.

## \* can explain the importance of the activation function in the context of neural networks

Een activatiefunctie beslist of een neuron geactiveerd moet worden of niet. Dit maakt ook back-propagation mogelijk.

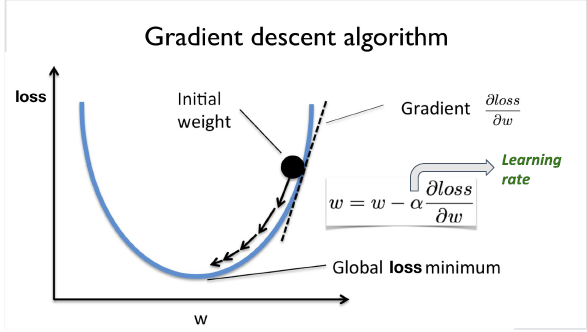
## \* can explain the limitations of perceptrons

Perceptrons kunnen alleen klassen aan die lineair gesplitst kunnen worden.

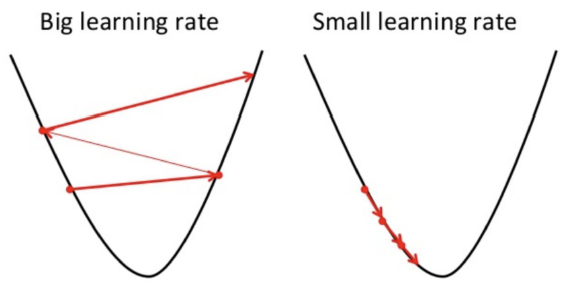
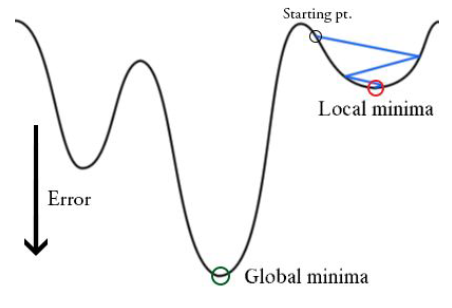
## \* can explain how Neural Networks can handle nonlinear separation

Non-lineaire splitsing wordt gedaan door meerdere lagen toe te voegen in een netwerk.

## \* can describe the concept of gradient descent in the context of neural networks in their own words



## \* can explain the importance of the learning rate in the context of gradient descent

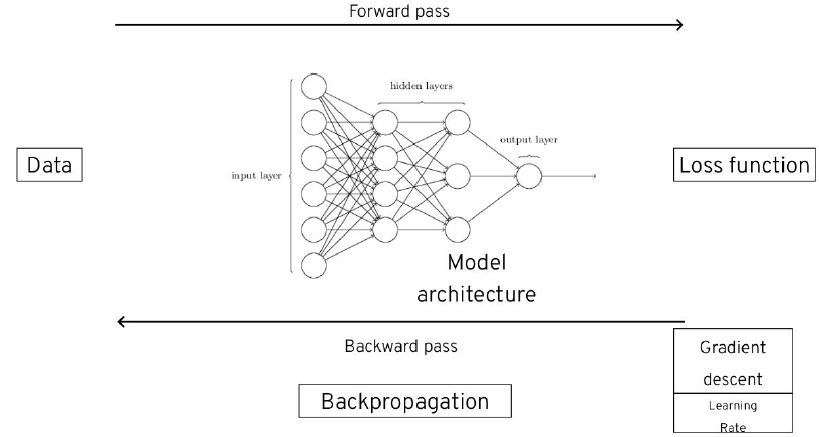
## \* can explain how backpropagation works in their own words

Na een gegeven output door het NN, gaat backpropagatie kijken welke gewichten er moeten aangepast worden. Dit doet hij in omgekeerde volgorde, dus van de laatste laag richting de inputlaag.

## \* can explain how a Neural Networks is trained in their own words

Een neural netwerk traint zichzelf door inputdata te verwerken. De data komt dus bij de eerste laag binnen en daarna wordt gekeken welke weg of welke node het beste lijkt voor tot de gewenste output te komen. Dit voor elke laag herhaalt. Daarna krijgen we een output in de laatste laag. Dan controleren we of dit ge gewenste output is en gaan we aan backpropagatie doen.

## \* can draw a schematic of a neural network and mark the different parts of the network (input, hidden, output layers)

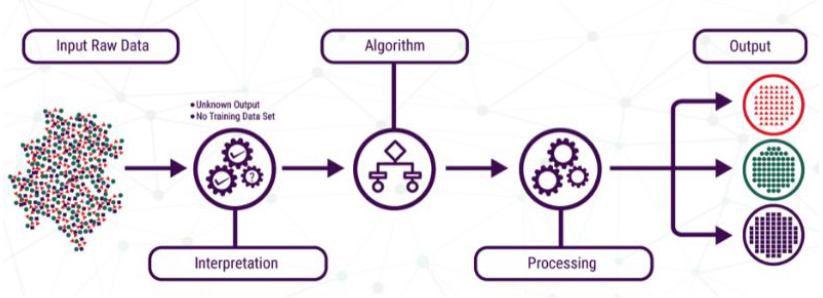


# Week 12

## \* can explain Unsupervised Learning in their own words

Bij unsupervised learning heeft de data geen labels en is er ook geen feedback. Er wordt naar betekenisvolle patronen gezocht in data zonder labels. Het gebruik ervan is aan het stijgen.

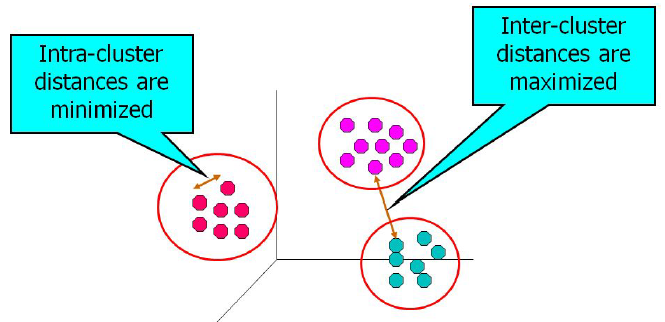
## \* can describe the general flow of an unsupervised learning pipeline



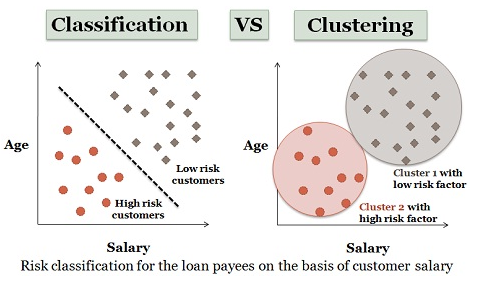
## \* can list at least 3 examples of unsupervised learning

- Aanbevelingen  
- Compressie-algoritmen  
- Marktsegmentatie

## \* can explain the purpose of clustering



## \* can explain the difference between clustering and classification



## \* can explain how to prepare data for clustering

- Categorieën omzetten naar numerische data  
- Missende waardes invullen  
- Data schalen (zelfde eenheden gebruiken)

## \* can explain how the k-means algorithm works in their own words

K-means gaat op zoek naar een local optimum. K-means weet niet hoeveel clusters er zijn.

## \* can describe the advantages and disadvantages of k-means

|  |  |
| --- | --- |
| Voordelen | Nadelen |
| Snel & efficiënt | Hoe kies je k? |
| Simpel en makkelijk te begrijpen | Hoe kies je de initiële centroids? |
|  | Gevoelig voor ruisdata |
|  | Slecht met clusters van verschillende grootte |
|  | Slecht met clusters van verschillende densiteit |
|  | Slecht met clusters die niet rond zijn |

## \* can explain how k-means++ optimizes the centroid choice problem of k-means

K-means++ kiest niet random maar kiest de volgende centroid rekening houdend met een afstandsfunctie.

## \* can explain how the Elbow method in the context of k-means works

De Elbow-methode is een manier om k te kiezen. Dit wordt gedaan door de afstandsscore voor verschillende k’s te proberen en te kijken waar de elleboog zit of waar de grafiek een knik maakt (laagste punt of kleinste afstand).

## \* can implement k-means and DBSCAN for a given clustering problem

N.v.t.

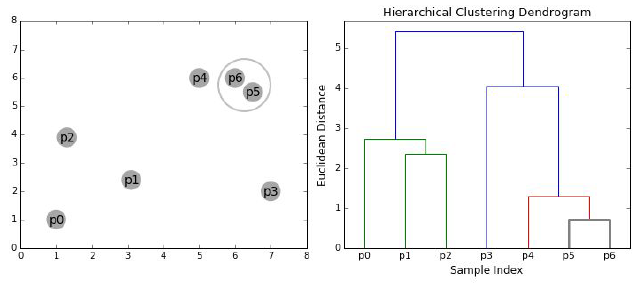
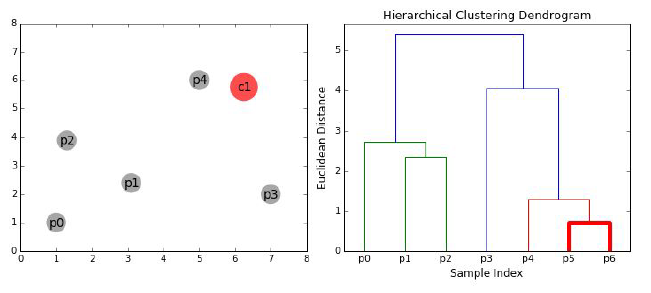
## \* can explain how the DBSCAN clustering algorithm works in their own words

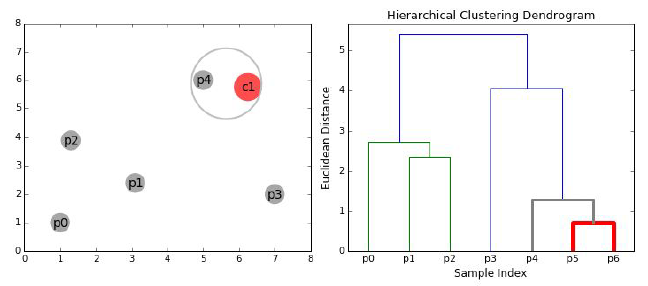
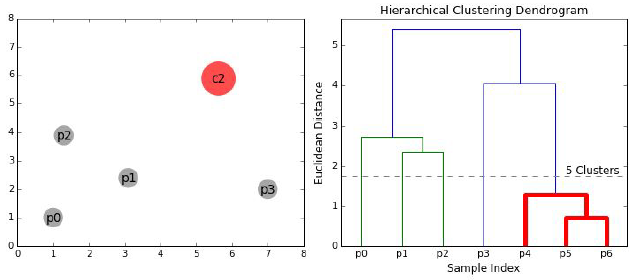
Een cluster is een hoog geconcentreerde regio waarbij de vorm niet uitmaakt en omringt is door een laag geconcentreerde regio. Heeft geen initiële schatting nodig over hoeveel clusters er gaan zijn. Robuuster tegen ruisdata.

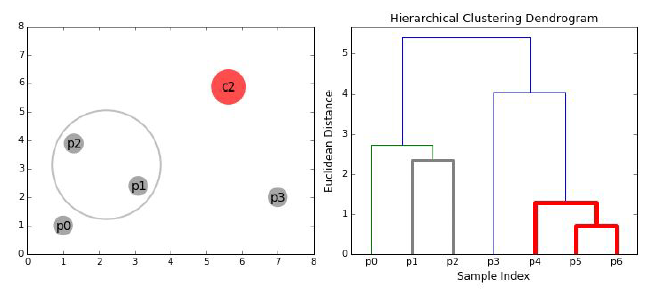
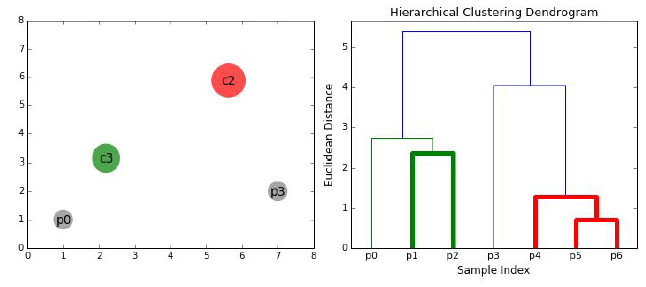
## \* can describe the advantages and disadvantages of DBSCAN

|  |  |
| --- | --- |
| Voordelen | Nadelen |
| Geen schatting van aantal clusters nodig | Parameter epsilon (ɛ) moet gekozen worden |
| Robuust tegen ruisdata | Minpunten moet gekozen worden |
|  | Slecht met clusters van verschillende densiteit |

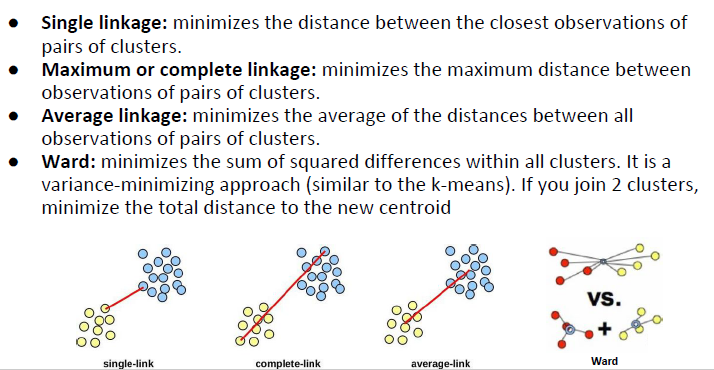
## \* can explain how hierarchical clustering works on the basis of a visual representation of a dataset

## \* can describe the different linkage forms for hierarchical clustering



## \* can describe the advantages and disadvantages of hierarchical clustering

|  |  |
| --- | --- |
| Voordelen | Nadelen |
| Makkelijk te visualiseren (dendrogram) | Moeilijk om aantal clusters aan te duiden |
| Biedt hierarchische relaties tussen clusters | Gevoelig voor ruisdata |
| Vooraf aantal clusters niet weten | Vroege stages -> wss verkeerd gegroepeerd |
| Niet gevoelig voor vormen en dichtheden | Traag |