# DECISION TREES – MACHINE LEARNING

## Hoe werkt de techniek?

### Intro

Goedemorgen iedereen en welkom bij onze presentatie. We gaan het vandaag hebben over Decision Trees.

Decision Trees zijn een vorm van Supervised Learning, en om preciezer te zijn; onder de vorm van classificatie. Maar wat is dat nu juist?

Classificatie is een techniek waar een model getraind wordt, dat op basis van de input bepaalt in welke categorieën je input thuishoort.

In onze oefeningen hebben we gebruik gemaakt van een binaire classifier. De output die we probeerden te voorspellen had maar twee mogelijke waarden, namelijk transported = true of false.

Maar hoe werkt een Decision Tree nu?

Met dit simpel voorbeeldje kunnen we de werking uitleggen. Maar eerst enkele definities die belangrijk zijn om te snappen.

De root node is altijd de eerste node in een Decision Tree. In deze node begint de boom, en alle data zit hierin.

Tussen de de root node en leaf nodes zijn er internal nodes. Dit is waar de boom verder groeit of wordt ‘gesplit’. Het gekozen algoritme van de Decision Tree zal berekenen welke vraag te stellen en op welke plek, om zo de best mogelijke ‘split’ te maken.

Je kan een internal node interpreteren met een vraag. In de internal node wordt de vraag gesteld, en de lijn naar de volgende node is het antwoord op die vraag.

We hebben besloten om de achterliggende werking hiervan niet uit te leggen om het simpel en begrijpelijk te houden voor iedereen. Het is enorm veel wiskunde en met de formules uit te leggen zijn we niet zo heel veel. Het doel van de splits is om een zo zuiver mogelijke distributie van data te hebben op elke node.

Op het einde van de decision tree zijn er dan leaf nodes. Dit zijn nodes die niet verder worden gesplitst. Dit is de node waar de voorspelling in voorkomt.

Deze decision tree vertelt ons of we willen gaan joggen of niet. We beginnen bovenaan, en vragen ons af welk weer is het? We splitsen op in zonnig, regen en bewolkt.

Als het bewolkt is, krijgen we yes dus gaan we joggen. Als het zonnig is, gaan we kijken hoe vochtig het is. Als er een normale vochtigheid is gaan we joggen, en als die hoog is dan niet.

Dan gaan we naar als het regent, is er een sterke wind? Zo ja gaan we niet joggen, als er geen sterke wind is gaan we wel joggen.

## Over welke hyperparameters beschikt de techniek? Wat stellen die voor? Wat zijn vaak gebruikte waarden?

De Decision Tree Classifier die wij hebben gebruikt is die van scikit-learn. Deze beschikt 12 hyperparameters.

* Criterion
* Splitter
* max\_depth
* min\_samples\_split
* min\_samples\_leaf
* min\_weight\_fraction\_leaf
* max\_features
* random\_state
* max\_leaf\_nodes
* min\_impurity\_decrease
* class\_weigth
* ccp\_alpha

### Critection

Is functie om de kwaliteit van een splitsing te meten. De ondersteunde criteria zijn **gini**, **entropy** en **log\_loss**. De standaardwaarde is **gini**. Gini maakt gebruik van het **CART** algoritme. Dat staat voor **Classification And Regression Tree**. Als je gebruik maakt **entropy** zal de decision tree het **ID3** algoritme gebruiken. Dat staat voor **Iterative Dichotomiser 3**.

### Splitter

De strategie die gebruikt wordt om de splitsing op elke node te kiezen. Ondersteunde strategieën zijn **best**, om de beste split te nemen en **random** om de beste random split te nemen. De standaardwaarde is **best**.

### Max\_depth

De maximale diepte van de boom. Ondersteund **integers** als waarde of **none**. De standaardwaarde is **none**. Als er none wordt gebruikt, worden de nodes uitgebreid tot alle **leafs pure** zijn of tot als alle leafs **minder samples** bevatten dan **min\_samples\_split**.

### Min\_samples\_split

Het minimumaantal samples dat nodig is om een node te splitten. Dit kan een **integer** of een **float** zijn. Als het een integer is wordt het gewoon als dat getal aanschouwd, als het een float is wordt er eerst een formule toegepast: “ceil(min\_samples\_split \* n\_samples)”, dan wordt de uitkomst hiervan de minimale aantal samples voor te splitten. De standaardwaarde is **2**.

## Min\_samples\_leaf

Het minimumaantal samples dat vereist is om bij een leaf node te zijn. Dit kan een **integer** of een **float** zijn. Als het een integer is wordt het gewoon als dat getal aanschouwd, als het een float is wordt er eerst een formule toegepast: “ceil(min\_samples\_leaf \* n\_samples)”, dan wordt de uitkomst hiervan de minimale aantal samples dat vereist is bij een leaf node. De standaardwaarde is **1**.

### Min\_weight\_fraction\_leaf

De minimale gewogen fractie van de som van de gewichten (van alle samples) die vereist is op een leaf node. Samples hebben een gelijk gewicht wanneer sample\_weight niet wordt opgegeven. Dit kan enkel een **float** zijn. De standaardwaarde is **0.0**.

### Max\_features

Het aantal features waarmee rekening moet worden gehouden bij het zoeken naar de beste split. Dit kan een **integer**, **float** of **none** zijn.

Er kan ook gebruik gemaakt worden van ‘auto’, ‘sqrt’ of ‘log2’ De standaardwaarde is **none**.

Als het een integer is wordt het gewoon als dat getal aanschouwd, als het een float is wordt er eerst een formule toegepast: “max(1, int(max\_features \* n\_features\_in\_))”,   
bij auto & sqrt wordt: “max\_features=sqrt(n\_features)” toegepast en   
bij log2: “max\_features=log2(n\_features)”.

### Random\_state

Regelt de **willekeurigheid** van de **estimator**. Kan een **integer** zijn, een **instance van RandomState** of **none**. De standaardwaarde is **none**.

### Max\_leaf\_nodes

Het **maximaal aantal leaf nodes** die de decision tree mag hebben. Het kan alleen een **integer** zijn of none. De standaardwaarde is none.

Min\_impurity\_decreaseEen node wordt gesplitst als deze split een daling van de onzuiverheid veroorzaakt die groter is dan of gelijk is aan deze waarde. Hier vul je een **float** in. De standaardwaarde is **0.0**.

### Class\_weight

Het **gewicht** **geassocieerd** met een **klasse**. Je kan een **dictionary** invoeren, **lijst van dictionary’s** of ‘**balanced’**. De standard waarde van class\_weight is **none**. Als het none is, heeft elke klasse een gewicht van **1**.

### Ccp\_alpha

De complexiteitsparameter gebruikt voor Minimal Cost-Complexity Pruning. De subtree met de grootste cost complexity die kleiner is dan ccp\_alpha wordt gekozen. Je voert een positieve float in. Standaardwaarde is 0.0, dat wil zeggen dat er standaard niet wordt gesnoeid in de boom.

### Volledig voorbeeld

Moesten we een decision tree maken waar elke hyperparameter manueel is ingesteld zou het er zo uitzien.

## In welke situaties werkt de techniek goed en in welke werkt het minder goed?

### Goede werking

De techniek zal ook alleen goed werken als de hyperparameters goed getuned zijn. Als dit niet het geval is kan er overfitting optreden. Dat is wanneer het model zo specifiek getraind is op de train data, dat de test data eigenlijk niet toepasselijk is en als gevolg de test geen goede resultaten zal opleveren. Dit was een probleem bij ons, meer hierover in het volgende hoofdstuk.

De techniek gaat over het algemeen goed werken als het doel enkel en alleen is om aan EDA (exploratory data analysis) te doen. Als men meer accurate antwoorden wil zou je beter gebruik maken van Random Forest. Dat is een techniek waar er gebruik gemaakt wordt van meerdere Decision Trees waar dan een gemiddelde Decision Tree wordt van gemaakt.

Enkele voordelen van Decision Trees zijn:

* Decision Trees zijn vrij eenvoudig om te begrijpen omdat ze gevisualiseerd kunnen worden.
* Ze kunnen gebruikt worden voor zowel classificatie als regressie problemen.
* Normalisatie van de data is niet verreist.
* Ze zijn snel en efficiënt en vragen niet veel computing power.

### Slechte werking

De techniek gaat niet goed werken als er sprake is van overfitting of underfitting.

Underfitting is als er te weinig data is om een goed model te trainen. Om dit op te lossen heb je simpelweg eigenlijk gewoon meer data nodig.

Zoals al vermeld, overfitting is als een model te specifiek getraind is en hierdoor geen betrouwbare resultaten zal opleveren met test data.

Enkele nadelen van Decision Trees zijn:

* Geen garantie dat de Decision Tree een model oplevert dat 100% juiste voorspelling geeft.
* Onstabiel want de kleinste aanpassing in de trainingsdata kan heel het model veranderen.
* Niet volledig geïmplementeerd in scikit-learn. Normaal gezien kan een Decision Tree werken met categorieke data. Het probleem is dat de Decision Tree Classifier van scikit-learn dat niet kan. Daarom moesten we eerst alles encoden of omvormen naar numerieke data in onze oefening.

## Hoe kan je overfitting tegengaan met deze techniek.

### Hyperparameter tuning

### Pruning

Standaard mag het decision tree model tot zijn volledige diepte groeien. Prunen is een techniek om de delen van de decision tree te verwijderen om te voorkomen dat deze tot zijn volle diepte groeit. Door de hyperparameters van het decision tree model af te stemmen, kunnen we de bomen prunen en voorkomen dat ze te sterk passen.

Er zijn twee vormen van prunen Pre-pruning en Post-pruning.

#### Pre-pruning

De pre-pruningtechniek zorgt voor het vroegtijdig stoppen van de groei van de decision tree. Bij deze techniek worden de hyperparameters van het decision tree vóór de training afgestemd. De hyperparameters van de decision tree, waaronder max\_depth, min\_samples\_leaf, min\_samples\_split, kunnen worden ingesteld om de groei van de tree vroegtijdig te stoppen en te voorkomen dat het model te sterk past.

#### Post-pruning

De post-pruningtechniek laat het decision tree model groeien tot zijn volledige diepte en verwijdert achteraf de tree branches om te voorkomen dat het model te sterk past. Cost complexity pruning (ccp) is één soort post-pruning techniek. Bij cost complexity pruning kan de ccp\_alpha worden ingesteld om het best passende model te krijgen.

Het Scikit-learn-pakket wordt geleverd met de implementatie om de ccp\_alpha-waarden van de decision tree te berekenen met de functie cost\_complexity\_pruning\_path(). Naarmate de ccp\_apha-waarden toenemen, worden meer knooppunten van de tree gesnoeid.

### Random Forest

Random Forest is een ensemble techniek voor classificatie en regressie door bootstrapping van meerdere decision trees. Random Forest gebruikt bootstrap sampling en aggregatietechnieken om overfitting te voorkomen.

Bootstrapping helpt bij het maken van meerdere subsets van de training data. Hierdoor kunnen we meerdere decision trees maken op basis van de gebootstrapte samples. Het zal dus op willekeurige wijze herhaaldelijke gegevenspunten selecteren van de trainingsdata en hierdoor meerdere subsets maken.

Random Forest kan worden geïmplementeerd met behulp van de Scikit-Learn library. We kunnen de hyperparameters van het Random Forest-algoritme verder afstemmen om de prestaties van het model te verbeteren. De parameter n\_estimator kan worden afgestemd om de overfitting van het model te verminderen.

## Hoe gebruik je de techniek? Train hiervoor minstens 2 modellen per teamlid met verschillende en bespreek de impact van de gewijzigde hyperparameters. Toon en leg uit tijdens je presentatie de code van 1 model.

Om de Decision Tree Classifier van sci-kit learn te gerbuiken moeten we eerst zorgen dat alle data numerieke data is. Zo vorm je de booleans kolommen om naar 0 of 1 integer kolommen en kolommen met een string voeren we encoding op uit. Dit doen we zowel voor de train als test data.

Verder kregen wij foutmeldingen omdat er cellen waren met niks in (null). Dit geeft fouten bij de scikit-learn decision tree classifier. Daarom hebben we besloten om alle null cellen op te vullen met de meest voorkomende waarde van die kolom. Dit is normaal hoe een decision tree zich zou moeten gedragen, maar we moesten dit zelf doen bij de decision tree classifier van scikit-learn.

Eens dat alle data numeriek is en er geen lege cellen meer zijn kunnen we aan de slag met het model te trainen.

We gaan dit uitleggen aan de hand van de decision tree met de hyperparameters die de hoogste score op Kaggle haalde.

Eerst moeten we een instantie van de DecisionTreeClassifier aanmaken met de gewenste hyperparameters. In ons geval maken we dus een decision tree met criterion = entropy, max\_depth = 8 en min\_samples\_split = 4. Al de rest van de hyperrameters die we niet impliciet toewijzen nemen hun standaardwaarde.

Daarna gaan we het model trainen. Hiervoor hebben we een X en een y nodig. De X is de train data (zonder de kolom dat hij moet voorspellen). En de y is de kolom van de te voorspellen data.

Als het model getraind is kunnen we er nu een voorspelling op maken met de test data. Vergeet niet dat de test data dezelfde aanpassingen heeft moeten gedaan dan de train data. Dat was bij ons het opvullen lege cellen, strings encoden en de te voorspellen kolom droppen.

De voorspelling is nu gemaakt, nu rest ons enkel nog om hem in een correct formaat om te vormen zodat we hem op Kaggle konden uploaden. We pakken hiervoor de test data tabel, resetten de index en nemen alleen de kolom PassengerId. We voegen een kolom toe en plaatsen hierin de voorspelling. We vormen deze om naar een boolean, exporteren hem als csv en dan zijn we klaar.

Maar hoe ziet ons model er nu uit?

Omdat onze boom 8 niveaus heeft is het een beetje onduidelijk dus laten we even inzoomen. Dit is hoe onze boom begint. Het achterliggende algoritme heeft beslist dat de beste splitsing om te beginnen op de kolom CryoSleep is. Als CryoSleep dus 0 is, gaat hij verder in de linkse tak, als hij 1 is in de rechtse. En zo wordt eigenlijk heel de boom overlopen per te voorspellen rij van de data.

Samples is het gehele aantal samples, zowel links en rechts en bij values aan de linker kant zie je hoeveel samples er zitten in de linker vertakking en rechts bij de rechtse vertakking.

## Bespreek de behaalde resultaten op basis van verscheidene metrieken zoals accuraatheid, f1-score en training/predict tijd. Is er overfitting/underfitting?

Omdat Kaggle ons niet de juiste voorspelling aanbied hebben we eerst onze train data zelf opgesplitst zodat we een deel hebben waar we 100% zeker zijn wat de correcte voorspelling zou moeten zijn.

We hebben de traindata opgesplitst in 75% train gedeelte en 25% test gedeelte. Vervolgens hebben we een model getraind met dezelfde hyperparameters als het model waar we de hoogste score op Kaggle kregen. Dan hebben we voorspellingen gemaakt met het model en een confusion matrix ervan opgesteld.

Met dit model scoren we vrij goed met een accuraatheid van 0.934 en een f1-score van ook 0.934.

Het model heeft 1009 keer transported = true correct voorspeld, en 1023 keer transported = false correct voorspeld.

Het model heeft 69 keer transported = true fout voorspeld en 73 keer transported = false fout voorspeld.

Het model maken, trainen en een voorspelling maken ging altijd super snel. Het nam gemiddeld van 0.1 seconden tot anderhalve second.

Ook was er geen sprake van underfitting omdat de train data genoeg was.

Hetzelfde geldt voor overfitting, we hebben het model niet getraind met de standaardwaarden maar met dezelfde hyperparameters als die van het model waar we de hoogste score op Kaggle mee verkegen.

## Bereken de voorspelling op basis van de testdata. Sla deze resultaten op in een csv en dien het in op Kaggle. Op welke plaats eindig je?

We hebben van elk model een voorspelling gemaakt en op Kaggle ingediend. We hebben het hoogst gescoord op plaats 1618 met een score van 0.78279.

Dit was met model met de volgende hyperparameters:

* Criterion: entropy
* Max\_depth: 8
* Min\_samples\_split: 4

Dit lijkt misschien relatief niet goed omdat we op 1618 zijn geëindigd op 2273, maar je moet weten dat de persoon die eerste staat een score heeft van 0.87584, dus technisch gezien is onze voorspelling maar voor 0.09 verschillend met de persoon op plaats 1. Verder is het niet verplicht om Decision Trees te gebruiken voor deze competitie dus kan het zijn dat de persoon in kwestie een andere of betere machine learning techniek heeft gebruikt.

Dit hebben wij ook eens gedaan. We hebben eerder al vermeld dat dat Random Forests een hele groep van decision trees zijn, waarvan een gemiddelde boom wordt van gemaakt en dat deze doorgaans accurater kan voorspellen dan één decision tree.

Om dit te bewijzen hebben we ook eens een random forest gemaakt en ingediend op Kaggle. Als we dit deden, zijn we op plaats 703 geëindigd met een score van 0.79985.

Dit was met het model met de volgende hyperparameters:

* N\_estimators: 100
* Criterion: gini
* Max\_depth: 6

Dit bewijst dan nog eens dat random forests accurater zijn dan decision trees.

En met dat gezegd te hebben sluiten we onze presentatie af.

Bedankt voor jullie aandacht en zijn er nog vragen?