De mogelijkheden voor data-extractie in ongestructureerde masterdata op basis van clustering: een vergelijkende studie.

Kobe Dehandschutter.

Scriptie voorgedragen tot het bekomen van de graad van
Professionele bachelor in de toegepaste informatica

Promotor: S. Lievens
Co-promotor: M. Verstraeten
Academiejaar: 2022–2023
Eerste examenperiode

Departement IT en Digitale Innovatie .

Woord vooraf

Aan de hand van deze bachelorproef zoek ik naar goede manieren om ongestructureerde masterdata te clusteren. Ik was persoonlijk nog weinig in contact gekomen met machine learning, maar toch heb ik dit altijd een interessant onderwerp gevonden.
Toen ik dit voorstel tegenkwam op Chamilo, heb ik mij er direct voor opgegeven. Ik ben direct begonnen allerlei informatie over clustering en machine learning op te zoeken, aangezien mijn weinige voorkennis waarschijnlijk niet genoeg was.
Als eerste wil ik graag mijn promotor, Stijn Lievens, bedanken voor de ondersteuning tijdens het volledige proces en de feedback die ik geregeld mocht ontvangen. Mijn co-promotor, Mario Verstraeten, wil ik graag bedanken dat ik de kans kreeg om dit onderzoek uit te voeren en ook voor de steun, tips en hulp bij het opstellen van de algoritmes. Ik wil ook graag mijn ouders bedanken voor de steun en de motivatie die ze mij gaven. Tot slot wil ik ook nog mijn stagepartner, Jona De Neve, met wie ik tijdens de lange autoritten van en naar onze stageplaats in Mechelen veel ideeën besproken en kennis gedeeld heb.
Ik wens u veel leerplezier toe en ik hoop dat ook u gebruik kan maken van deze bachelorproef naar de toekomst toe!

Samenvatting

Het beheren van ongestructureerde masterdata is een probleem waar veel bedrijven mee kampen. Het is heel lastig om structuur te brengen in zulke tabellen. In dit onderzoek worden verschillende manieren vergeleken om deze data te clusteren zodat gelijkaardige data gegroepeerd wordt. Aan de hand daarvan kunnen bedrijven de beste methode vinden om de datakwaliteit van hun masterdata te verbeteren.
Ongestructureerde masterdata bestaat in alle vormen en maten, daarom wordt in dit onderzoek gebruik gemaakt van drie verschillende datasets. De eerste bevat productinformatie die bestaat uit een paar woorden, codes en afmetingen. De tweede set bevat informatie over wijnen in de vorm van lange zinnen. De derde bevat opnieuw productinformatie, maar deze keer zonder codes of afmetingen, enkel bestaande woorden.
Wanneer alle mogelijke clustermethodes besproken zijn, worden de drie meest geschikte methodes uitgewerkt, namelijk K-means, dbscan en Hiërarchisch clusteren. De algoritmes K-means en dbscan verwachten de data echter in getallen in plaats van in woorden. Ook hiervoor worden verschillende manieren gezocht en de twee meest geschikte manieren hiervoor worden uitgewerkt, namelijk tf-idf en word2vec.
Hiërarchisch clusteren verwacht de data niet in getallen zoals de andere methodes, maar dit algoritme verwacht een matrix met alle afstanden tussen iedere waarde van de dataset. Om dit te bereiken zijn er opnieuw een aantal mogelijkheden. In dit onderzoek wordt de Levenshtein distance gebruikt.
Uiteindelijk zijn er vijf verschillende combinaties die uitgewerkt worden:

1. Tf-idf met K-means.
2. Tf-idf met dbscan.
3. Levenshtein distance met hiërarchisch clusteren.
4. Word2vec met K-means.
5. Word2vec met dbscan.

Het uitwerken van de algoritmes gebeurt in Python, aangezien er in Python heel wat bibliotheken bestaan die helpen bij het opstellen van deze algoritmes. Alle vijf de combinaties worden uitgevoerd met alle drie de datasets en de resultaten worden vergeleken op basis van het aantal clusters, de grootste cluster, een eventuele vuilniscluster die restwaarden bevat en het Silhoutte gemiddelde. Dit Silhoutte

gemiddelde is een manier om de accuraatheid van een ongesuperviseerd clusteralgoritme te berekenen.
Uit de resultaten van de vergelijkende studie komt tf-idf in combinatie met K-means naar voren als de beste manier om ongestructureerde masterdata te clusteren. Dit is echter enkel het geval voor de twee datasets met productdata. Voor de dataset met de lange zinnen is geen enkele methode geslaagd in het succesvol clusteren van deze data. Hopelijk kan ik aan de hand van dit onderzoek iemand op weg helpen om een geschikte manier hiervoor te vinden.

Inhoudsopgave

Lijst van figuren

1

Inleiding

Veel bedrijven hebben wat men noemt 'een vuilnistabel'. Dit is een tabel met ongestructureerde masterdata. Hierin staan allerlei gegevens door elkaar, waardoor het lastig wordt om iets te doen met deze data. In dit onderzoek zal er gekeken worden naar de verschillende mogelijkheden om data-extractie te verbeteren in zo'n tabellen. Data-extractie of gegevensextractie slaat op het ophalen van relevante gegevens uit een tabel of databank (Encyclo.nl, g.d.). Aan de hand van dit onderzoek kunnen bedrijven hun vuilnistabellen veel beter gebruiken, of zelfs sorteren, zonder alles handmatig uit te voeren.
In dit onderzoek wordt enkel masterdata onderzocht. Masterdata zijn de gegevens van een bedrijf die niet transactioneel zijn. Dit betekent dat de data niet te maken heeft met transacties zoals het plaatsen van orders bijvoorbeeld. Masterdata kan wel veranderen in de loop der tijd, maar dit gebeurt zeer langzaam. Eén van de voornaamste voorbeelden hiervan zijn productgegevens: de naam, afmetingen, prijs... Deze gegevens gaan enkel aangepast worden in uitzonderlijke gevallen (Yellowground, g.d.).
Er zijn twee mogelijkheden om deze data op te slaan: gestructureerd of ongestructureerd. Gestructureerde data is georganiseerd en gegroepeerd zodat alle verschillende gegevens in een aparte tabel zitten. Als dit niet het geval is en er bestaat slechts één tabel met daarin allerlei gegevens, is er sprake van ongestructureerde data. Een voorbeeld hiervan is een tabel met de naam, prijs, afmetingen, beschrijving van het product allemaal tezamen (Seagate, g.d.).
Het grote probleem met ongestructureerde masterdata is het feit dat het verbeteren van de datakwaliteit een stuk moeilijker wordt. Zoeken of sorteren in zo'n tabel is heel moeilijk, ook duplicaten onderscheiden is niet gemakkelijk. Daarnaast kan het soms zelfs lastig worden om de gewenste data op te halen uit zo'n tabel.
In dit onderzoek zullen er verschillende mogelijkheden vergeleken worden om gelijkaardige waarden uit een tabel te vinden, beter gekend als fuzzymatching (Silva,

2022). Er bestaan al een heleboel technieken en metrieken hiervoor, maar wat als we de tabel eerst gaan onderverdelen in verschillende groepen (Clustering) waarbij het de bedoeling is dat elke groep iets gelijkaardigs heeft zodat er een label aan iedere groep kan gegeven worden (Tagging)? De methode clustering + tagging gaat onderzocht en vergeleken worden met bestaande algoritmes die gebruik maken van andere technieken.
De vergelijkende studie zal uitgevoerd worden aan de hand van aan tabel met ongestructureerde masterdata afkomstig uit een bestaand bedrijf. Met behulp van bibliotheken in Python zal er onder andere vergeleken worden op basis van de volgende zaken: de hoeveelheid gevonden clusters, grootte van de clusters en de Silhoutte gemiddelde's (hoe goed de waarde thuishoort in zijn cluster) (Kaplan, 2022).

1.1. Probleemstelling

Masterdata is iets wat quasi alle bedrijven hebben en bijhouden. Deze data is vaak lastig om goed geordend bij te houden. Bij de opstart van een bedrijf, als er nog zeer weinig masterdata is om op te slaan, gaat dit nog gemakkelijk, maar na een aantal jaar is de controle makkelijk te verliezen en verslechterd de datakwaliteit. Zo wordt het moeilijk om in deze data aan data-extractie te doen.

1.2. Onderzoeksvraag

Dit onderzoek focust zich op het vergelijken van verschillende mogelijkheden om ongestructureerde masterdata te clusteren. Aan de hand van de resultaten van de vergelijkende studie, kunnen bedrijven een antwoord vinden op volgende onderzoeksvraag:

- Wat is de beste manier om ongestructureerde masterdata te clusteren?

De resultaten van de vergelijkende studie zullen zodanig opgesteld worden dat er voor verschillende soorten masterdata een goede manier naar voren geschoven kan worden.

1.3. Onderzoeksdoelstelling

Het doel van dit onderzoek is een goede leidraad te voorzien voor bedrijven die orde willen brengen in hun ongestructureerde masterdata en de datakwaliteit verbeteren. Aan de hand van de bekomen resultaten kunnen bedrijven de juiste methode vinden die het beste werkt voor hun data.

1.4. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie. Ook wordt er een lijst opgesteld van alle mogelijkheden die onderzocht worden.
In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvraag.
In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvraag. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2

Stand van zaken

Als eerste stap in dit onderzoek, wordt een literatuurstudie opgesteld. Hierin wordt zo veel mogelijk bestaande informatie over de onderzoeksvraag verzameld. Voor dit onderzoek zal vooral informatie vergaard worden over machine learning, de verschillende manieren om de afstand tussen strings te berekenen en de mogelijkheden om de data te clusteren op basis van die afstanden.

2.1. Machine learning

Machine learning staat centraal in dit onderzoek. Er bestaan drie belangrijke types machine learning algoritmen.

2.1.1. Gesuperviseerd leren

Als eerste is er het gesuperviseerd leren: hierbij bevat iedere waarde onmiddellijk een label met het juiste antwoord. Eenvoudig voorbeeld: als er een algoritme gesuperviseerd wordt getraind om katten te herkennen op foto's, dan bevat iedere foto die getoond wordt aan het algoritme een label met 'ja' of 'nee'. Als het algoritme dan goed genoeg getraind is, is het in staat om foto's zonder label, het juiste label te geven (Postma, 2019).

2.1.2. Ongesuperviseerd leren

Het tweede type is ongesuperviseerd leren. Dit is logischerwijs het omgekeerde van gesuperviseerd leren. In ons simpel voorbeeld van de katten, ontvangt het algoritme een grote dataset met foto's. Dit algoritme zoekt vervolgens in deze set naar iets dat overeenkomt in veel van de foto's. Deze overeenkomst wordt in de ogen van het algoritme de 'kat'. Vervolgens splitst het de foto's in twee groepen. Eén die de 'kat' wel bevat en één die hem niet bevat. Aangezien er geen labels hangen

aan de dataset, kan het algoritme zelf niet weten of het juist of fout is. Dit zorgt ervoor dat er minder controle is over de uitkomst, waardoor deze methode minder succesvol is. Zo kan het natuurlijk gebeuren dat veel foto's, naast een kat, ook een boom bevatten. Hierdoor kan het gebeuren dat het algoritme de foto's splitst in een groep met boom en zonder boom, want het algoritme kan niet weten hoe een kat eruitziet. Waarom zou ongesuperviseerd leren dan gebruikt worden als het minder succesvol is? In het voorbeeld van de katten is het inderdaad niet zo moeilijk om een label 'ja' of 'nee' aan iedere foto te geven, maar in heel veel andere gevallen kunnen er geen labels aan de data gegeven worden en kan het enorm handig zijn als het algoritme gewoon de data zonder labels in verschillende groepen verdeelt. Dit geeft al veel inzicht in de data (Postma, 2019).

2.1.3. Reinforcement learning

Het derde type is reinforcement learning. Dit wordt veel gebruikt bij situaties waarin het algoritme een juiste volgorde van acties moet uitvoeren. Iedere keer dat het algoritme correct is, wordt het beloond. Deze methode wordt ook gebruikt voor het aanleren van gedrag bij dieren. Als een hond getraind wordt om te zitten bijvoorbeeld, krijgt hij een snoepje of aaitje iedere keer hij het commando goed uitvoert (Postma, 2019).

In dit onderzoek wordt gebruik gemaakt van ongestructureerde masterdata, dit betekent dat de data geen labels bevat. De bedoeling is om de data te onderverdelen in gelijkaardige groepen. Het is dus vanzelfsprekend dat ongesuperviseerd leren gebruikt wordt in dit onderzoek.

2.2. N-grammen

N-grammen zijn allemaal substrings van een grotere string. Deze substrings kunnen opgesteld worden met karakters of met woorden, maar meestal gebeurt dit op basis van karakters zodanig dat iedere substring *N* karakters bevat. Als er dus met 2-grammen wordt gewerkt bestaan alle substrings uit twee karakters, wordt er met 4-grammen gewerkt, bestaan ze uit vier karakters.
Bijvoorbeeld voor het woord 'Hyundai' zijn alle 2-grammen: 'Hy', 'yu', 'un', 'nd', 'da' en 'ai' en alle 4-grammen zijn: 'Hyun', 'yund', 'unda' en 'ndai'. Voor een deel van van de algoritmes die in dit onderzoek gebruikt worden, worden de waarden uit de tabel eerst in *N*-grammen gesplitst (Santos e.a., 2009).

2.3. Van woorden naar getallen

Nu er beslist is dat er gebruik gemaakt wordt van ongesuperviseerd leren, komt een volgend probleem naar boven. Onze ongestructureerde masterdata bevat woorden en karakters, maar de algoritmen kunnen dit niet groeperen. Deze verwachten de

data in de vorm van getallen. Er zijn verschillende manieren om dit te doen. Een eerste manier is door de afstand tussen de woorden te berekenen. Een tweede manier is door de woorden om te zetten in betekenisvolle getallen.
Als eerste zullen een paar methoden besproken worden om de afstand tussen twee woorden te berekenen. Twee identieke waarden onderscheiden is natuurlijk niet zo moeilijk (Lievens, 2022). Het wordt een stuk lastiger om waarden te identificeren die wel gelijkend, maar niet volledig identiek zijn. Een groot voorbeeld hiervan zijn typfouten. Het is wel belangrijk om een goed algoritme te kiezen, om zo weinig mogelijk, of in het beste geval geen, vals positieve uitkomsten te krijgen (Silva, 2022). In dit onderzoek wordt enkel de Levenshtein distance gebruikt om de afstand te berekenen, maar ter verduidelijking zullen er nog een paar andere methoden besproken worden.

2.3.1. Levenshtein distance

Een eerste mogelijkheid is om gebruik te maken van de 'Levenshtein distance'. Deze metriek bepaalt de afstand tussen twee strings aan de hand van het aantal uit te voeren operaties. Deze operaties zijn: een letter toevoegen of verwijderen en een letter veranderen. Hoe meer operaties er moeten uitgevoerd worden, hoe groter de afstand tussen twee strings (Wikiversity, 2022).

		H	Y	U	N	D	A	I
	0	1	2	3	4	5	6	7
H	1	0	1	2	3	4	5	6
O	2	1	1	2	3	4	5	6
N	3	2	2	2	2	3	4	5
D	4	3	3	3	3	2	3	4
A	5	4	4	4	4	3	2	3

Figuur (2.1)
Een voorbeeld van de levenshtein distance waar Hyundai en Honda vergeleken worden (Cuelogic, 2017).

Een voorbeeld ter verduidelijking. De woorden 'Honda' en 'Hyundai' worden hier vergeleken. De letters van beide woorden worden op een x-as en y-as geplaatst. In de tweede rij en kolom wordt aan iedere letter een getal gegeven op basis van zijn positie in het woord. Voor Hyundai is dit dus van 0 tot en met 7 en voor Honda van 0 tot en met 5. Alle andere hokjes zijn nog niet ingevuld. Het invullen begint linksboven, eerst wordt de eerste rij volledig ingevuld van links naar rechts, dan pas wordt de volgende rij begonnen.
Ieder vakje wordt berekend op basis van drie voorgaande vakjes, namelijk de vakjes

die zich links van, boven en linksboven het in te vullen vakje bevinden. Voor het eerste vakje zijn dit dus de getallen 1, 0 en 1. Met ieder getal wordt een berekening gedaan: de berekening voor de getallen erboven en er links van is eenvoudigweg plus 1. Voor het getal schuin linksboven worden de letters vergeleken: als ze gelijk zijn plus 0, als ze niet gelijk zijn plus 1.
Voor het eerste vakje zijn de berekeningen dus: 1+1=2, 1+1=2 en 0+0=0. Tot slot wordt het laagste van de drie uitkomsten ingevuld en wordt opgeschoven naar het volgende vakje er rechts naast. Als de rij klaar is, wordt de volgende rij gestart. Als de laatste rij klaar is staat de uitkomst in de rechteronderhoek van de tabel. Bij dit voorbeeld is de uitkomst drie, dit betekent dat er dus drie operaties moeten uitgevoerd worden om van het ene woord, naar het andere te gaan.
Er zijn een aantal voordelen aan het gebruik van de Levenshtein distance. De input hiervoor moet geen bepaald type zijn, dit kan van alles zijn. Als er andere tekens gebruikt worden, is dit geen probleem. Dit is handig als er gewerkt wordt met ongestructureerde masterdata, aangezien deze alle mogelijke gegevens kan bevatten. Een tweede voordeel is dat het heel goed overweg kan met typfouten, als er bijvoorbeeld 100 keer 'fiets' voorkomt in de data en een keer 'fies', dan zal de Levenshtein afstand niet veel verschillen, terwijl dit met andere algoritmes wel het geval kan zijn.

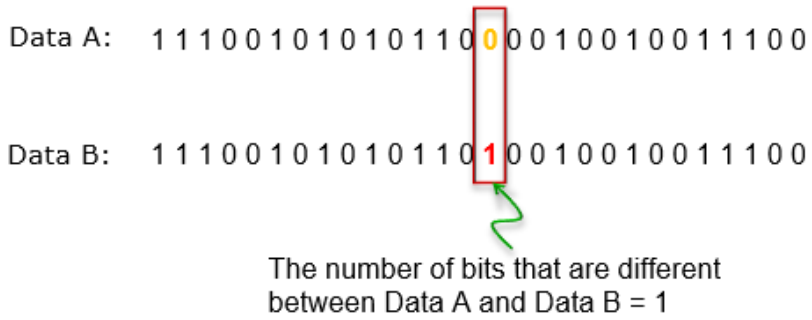
		F	I	E	T	S
	0	1	2	3	4	5
F	1	0	1	2	3	4
I	2	1	0	1	2	3
E	3	2	1	0	1	2
S	4	3	2	1	1	1

Figuur (2.2)
Een voorbeeld van de levenshtein distance waar fiets en fies vergeleken worden.

2.3.2. Hamming distance

Een tweede metriek om de afstand te bepalen is de 'Hamming distance'. Dit kan op twee manieren. Een eerste manier is heel gelijkaardig aan de Levenshtein distance, alleen zonder het toevoegen of verwijderen van karakters. Er wordt dus enkel gekeken hoeveel karakters verschillen. Een tweede manier transformeert eerst alle karakters in beide strings in hun binaire vorm. Bijvoorbeeld: 'a' wordt '1100001'

en 'A' wordt '1000001' (Includehelp, g.d.). Vervolgens doet deze hetzelfde als de eerste manier. De afstand tussen beide strings wordt dus bepaald aan de hand van het aantal verschillen in de binaire codes van de karakters. De afstand tussen 'a' en 'A' is dus 1, aangezien er slechts één binair karakter verschilt (Silva, 2022). Op onderstaande figuur is een voorbeeld te zien waar de strings 'Data A' en 'Data B' vergeleken worden.

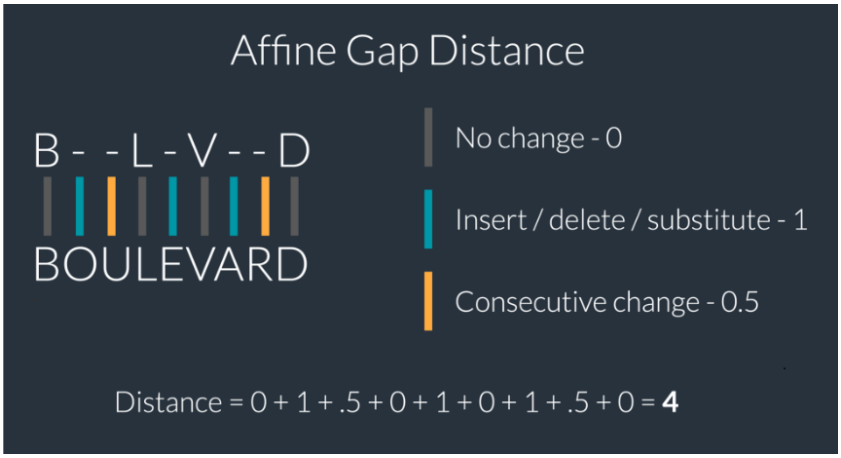


Figuur (2.3)
Een voorbeeld van Hamming distance waar Data A en Data B vergeleken worden (ShareTechnote, g.d.).

Er zijn twee grote nadelen aan de hamming distance. Het eerste is het feit dat deze metriek enkel de afstand kan bepalen van strings die exact even lang zijn. In ongestructureerde masterdata is dit meestal niet het geval. Een mogelijke oplossing hiervoor is spaties toevoegen aan het einde van de kortste string zodat ze even lang worden, maar per spatie die wordt toegevoegd zal de afstand groter worden, terwijl de waarden misschien heel gelijkend zijn.
Een tweede nadeel van Hamming distance is het feit dat dit enkel de exact gelijke waarden eruit haalt. Als bijvoorbeeld 'fiets' en 'de fiets' worden vergeleken zullen, na drie spaties toe te voegen achter 'fiets', de letters 'f' en 'd' vergeleken worden enzovoort, waardoor deze twee strings heel ver uit elkaar zullen liggen, terwijl er in onze use case verwacht wordt dat beide waardes eigenlijk in dezelfde cluster terecht komen.

2.3.3. Affine gap distance

Een volgende metriek is de 'Affine gap distance'. Hierbij wordt er rekening gehouden met afkortingen, dit komt voornamelijk voor bij namen: 'K. Dehandschutter' in plaats van 'Kobe Dehandschutter'. Met vorige metrieken is de afstand hiertussen zeer groot, aangezien er drie letters toegevoegd moeten worden, terwijl de twee strings op zich heel gelijkaardig zijn (Walgran, 2019). Qua scoring werkt het als volgt: voor elk eerste karakter dat moet aangepast worden, ook wel de opening van de gap genaamd, wordt er plus één gedaan en voor elke opvolger in de gap plus een halfje (Lievens, 2022). Op onderstaande figuur is een voorbeeld te zien waarin de afstand tussen 'blvd' en 'boulevard' berekend wordt.



Figuur (2.4)
Een voorbeeld van affine gap distance waar blvd en boulevard vergeleken worden (Azavea, 2019).

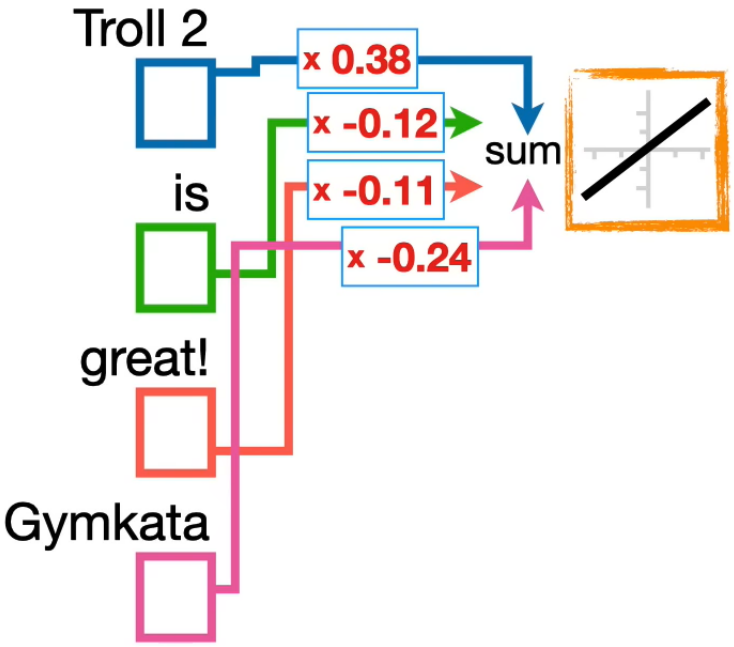
2.3.4. Natural Language Processing

Natural language processing, beter gekend als NLP, ontstond als kruispunt tussen artificiële intelligentie en taalkunde. NLP bestaat uit een grote hoeveelheid technieken die computers gebruiken om menselijke tekst te begrijpen zodat ze dit kunnen gebruiken om verschillende taken uit te voeren (Nadkarni e.a., 2011).
Het feit dat natuurlijke taal zo immens groot is en zo onvoorspelbaar en dubbelzinnig kan zijn, zorgde voor enkele grote problemen. Als eerste moest NLP de betekenis van de tekst kunnen achterhalen om zo de relaties tussen verschillende woorden te kunnen specificeren, denk bijvoorbeeld aan werkwoord, bijwoord, bijvoeglijk naamwoord... Daarnaast is er ook nog die dubbelzinnigheid en intonatie, als mens is het makkelijker woordmopjes of sarcasme te begrijpen, als AI is dit een enorme uitdaging (Nadkarni e.a., 2011).
Er zijn natuurlijk een heleboel verschillende manieren van NLP. Zo bestaat er fonologie om de klank en geluiden te herkennen van verschillende woorden. Dit wordt gebruikt om gesproken tekst te begrijpen. Ook is er morfologie, dit kijkt naar de vorm van de woorden. Hiermee kunnen prefix en suffix onderscheiden worden. In het woord 'Oneerlijk' zal bijvoorbeeld de 'on' als prefix uitkomen, wat meestal 'niet' betekent en dan blijft 'eerlijk' over, waardoor de NLP kan achterhalen dat 'oneerlijk' 'niet eerlijk' betekent. Zo zijn er nog een aantal methodes: semantisch, syntactisch, lexicaal, discourse en pragmatisch (Liddy, 2001).

2.3.5. Word2vec

Bij de Levenshtein distance wordt nog geen rekening gehouden met de semantiek van de woorden. 'Frigo' en 'Koelkast' gelijken qua karakters totaal niet op elkaar, maar betekenen op zich wel hetzelfde. Om de similariteit van deze strings te berekenen, wordt elk woord omgezet in een vector via het word2vec algoritme (Lievens, 2022).

Word2vec is op 2 verschillende manieren getraind. Als eerste is er het skip-gram algoritme. Dit wordt gebruikt om de semantische relaties tussen woorden te modelleren door aan de hand van een woord, zijn mogelijke context te voorspellen. Als tweede is er het continuous bag-of-words (CBOW) algoritme. In tegenstelling tot het skip-gram-algoritme, gebruikt CBOW de context van een woord om het woord zelf te voorspellen. Om genoeg data te hebben om voor alle woorden een goede vector te berekenen, gebruikt word2vec het volledige wikipedia als vocabulaire (Starmer, 2023).
Deze twee algoritmes werken zeer gelijkaardig. Alle unieke woorden uit de vocabulaire, in onderstaand voorbeeld: 'Troll2', 'is', 'great' en 'Gymkata' krijgen een inputwaarde. Deze inputs zijn verbonden met een activation function, deze neemt de som van de inputs en berekent voor iedere input een output. Aan de connecties tussen de inputs en de activation function worden gewichten gehangen, zoals te zien is op onderstaande figuur. Deze gewichten zullen bij het eindresultaat de waarde zijn die geassocieerd wordt met het woord. In het begin worden deze gewichten willekeurig gekozen. Via backpropagation zullen die gewichten aangepast worden tot ze de gewenste waarde hebben (Starmer, 2023).

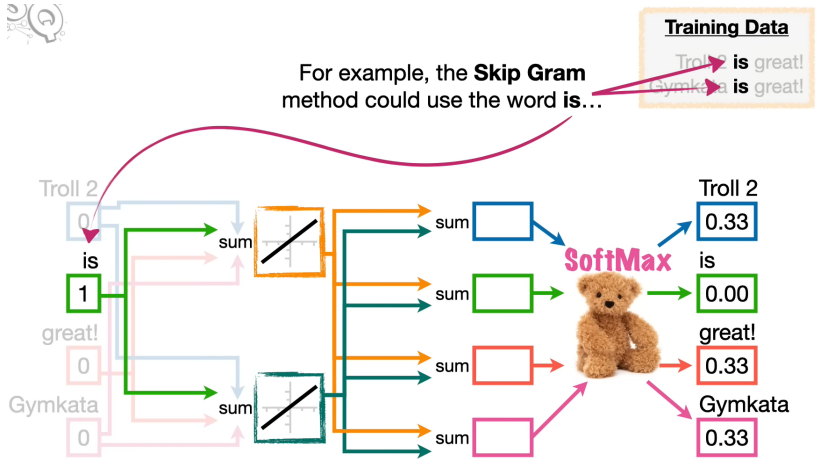


Figuur (2.5)
Een grafische voorstelling van de activation functie met willekeurige gewichten (Starmer, 2023).

Backpropagation, ook wel 'achterwaartse propagatie' genoemd, is een algoritme dat wordt gebruikt voor het trainen van neurale netwerken. Backpropagation werkt door het berekenen van de afgeleiden van de verliesfunctie ten opzichte van elke gewichtsparameter in het netwerk, en vervolgens het aanpassen van deze gewichten om de fout te verminderen. De verliesfunctie wordt gebruikt om het verschil

tussen de voorspelde uitkomst en de gewenste uitkomst te bepalen.
Het proces begint met het doorsturen van input door het netwerk om een voorspelling te genereren. Vervolgens wordt de fout tussen de voorspelde output en de werkelijke output berekend. Met behulp van de afgeleiden van de verliesfunctie worden de gewichten in het netwerk aangepast, zodat de volgende keer dat hetzelfde voorbeeld wordt doorgevoerd, de fout kleiner wordt (Starmer, 2023).
Het skip-gram-algoritme werkt door een doelwoord te nemen en vervolgens te berekenen hoe groot de kans is dat elk ander woord uit de vocabulaire in de context van het doelwoord voorkomt (Starmer, 2023). Het skip-gram-algoritme begint met willekeurige gewichten aan ieder woord te geven, vervolgens voorspelt het aan de hand van die waarden welke woorden vaak in de context voorkomen. Het kijkt in de vocabulaire of deze voorspelling degelijk is, en past dan de gewichten aan om een beter resultaat te krijgen. Het blijft dit herhalen tot er voor ieder woord een goede waarde is waarvoor zijn context voorspeld kan worden. Het skip-gram-algoritme gebruikt ook niet één activation function, maar meestal zelfs honderd of nog meer. Aan iedere functie wordt dan een ander gewicht meegegeven per woord, en hoe meer iteraties er gebeuren, hoe dichter die gewichten bij elkaar zullen liggen. Ieder woord heeft bijgevolg een honderdtal uitkomsten en deze vectoren worden bij elkaar opgeteld. Alle opgetelde vectoren van alle woorden worden meegegeven met een softmax function. Deze functie zet de vectoren om in een getal tussen 0 en 1 zodat de som van alle getallen 1 is en hoe hoger dit getal, hoe groter de kans dat het woord in de context voorkomt, hoe kleiner dit getal, hoe kleiner de kans dat het woord in de context voorkomt (Starmer, 2023).
In dit onderstaande figuur wordt er voorspeld wat de context van 'is' zou kunnen zijn, door bij 'is' als input 1 mee te geven en alle andere inputs op 0 te zetten. Op de figuur is te zien dat hier twee activation functions gebruikt worden, dus ieder woord heeft twee connecties met twee gewichten. Als eindresultaat zien we dat 'is' een waarde 0 heeft en de andere drie een waarde van 0.33, want ze hebben alledrie evenveel kans om in de context van 'is' voor te komen. Door de trainingsdata rechtsboven te bekijken, is dit een goede uitkomst (Starmer, 2023).

Het CBOW-algoritme werkt gelijkaardig, het neemt twee woorden en probeert vervolgens te voorspellen welk woord in het midden van deze woorden voorkomt. Meerdere woorden krijgen een 1 als input. Het CBOW-algoritme begint ook met willekeurige waarden aan ieder woord te geven, vervolgens voorspelt het aan de hand van die waarden welke woorden in het midden kunnen staan. Het kijkt in de vocabulaire of deze voorspelling degelijk is, en past dan de gewichten aan om een beter resultaat te krijgen. Het blijft dit herhalen tot er voor iedere context een goede waarde is waarvoor het woord in het midden voorspeld kan worden. De rest van het algoritme verloopt op dezelfde manier als bij het skip-gram-algoritme (Starmer, 2023).



Figuur (2.6)
Een grafische voorstelling van het eindresultaat wanneer is als input wordt gegeven (Starmer, 2023).

Word2Vec gebruikt deze twee modellen en neemt vervolgens het gemiddelde van de twee vectoren zodat de uitkomst zo perfect mogelijk is (Starmer, 2023).

2.3.6. Tf-idf

Een derde mogelijkheid is tf-idf, dit staat voor term frequency-inverse document frequency. Eerst en vooral hebben we term frequency. Dit is het aantal keer dat een woord voorkomt in het document. Dit kan berekend worden door gewoonweg te tellen hoeveel keer het voorkomt, maar als de documenten maar een paar worden bevatten, zoals in ongestructureerde masterdata toch vaak het geval is, kan dit ook berekend worden aan de hand van een boolean. Het wordt dus 1 als het document het woord bevat en 0 als het woord er niet in voorkomt. Vervolgens is er de inverse document frequency. Dit is een berekening van hoe zeldzaam het woord is ten opzichte van alle documenten in de dataset. Dit is noodzakelijk, want anders komen woorden zoals 'het', 'een', 'is' of 'er' altijd naar boven als veelgebruikte woorden. De bedoeling is dus om de woorden die niet veel voorkomen in alle documenten, maar wel regelmatig in een bepaald document voorkomen, een hoge score te geven. De idf voor een bepaald woord (t) in een dataset (D) wordt als volgt berekend: de logaritme van N, waarbij N gelijk is aan het aantal documenten (d), gedeeld door het aantal documenten waarin het woord (t) voorkomt (Simha, 2021).

$$\text{idf}(t, D) = \log\left(\frac{N}{\text{count}(\text{d} \in \text{D}: \text{t} \in \text{D})}\right)$$

Wanneer de tf en de idf berekend zijn, moeten deze nog samengevoegd worden natuurlijk. Dit wordt simpelweg gedaan door de twee waarden te vermenigvuldigen. Hoe hoger de uitkomst is, hoe belangrijker het woord is. Als de uitkomst dicht bij 0 ligt is dit woord totaal niet belangrijk (Simha, 2021).

2.4. Clustering

Nu alle data uit de datasets omgezet zijn in vectoren, kan er begonnen worden met clusteren. Hiervoor worden drie verschillende methoden gebruikt. K-means, hiërarchisch en dbscan.

2.4.1. K-means

Het K-means clustering algoritme wordt gebruikt om te clusteren in een tabel met ongestructureerde data. Er moet op voorhand opgegeven worden hoeveel clusters er nodig zijn, dit is het natuurlijke getal K. Vervolgens worden er K zwaartepunten (centers) geïnitialiseerd op een random positie. Daarna wordt voor elke waarden berekend bij welk zwaartepunt ze het dichtste liggen en aan die cluster toegevoegd. Daaropvolgend start een iteratief proces waarbij voor iedere cluster het zwaartepunt wordt verlegd naar het middelpunt. Nu wordt opnieuw voor iedere waarde in de tabel berekend bij welk zwaartepunt ze het dichtst ligt en zo worden opnieuw clusters gevormd.
Dit iteratief proces blijft doorgaan totdat de zwaartepunten niet meer verplaatsen en de uiteindelijke clusters gevormd zijn. De dataset is dan in K clusters verdeeld. Een groot nadeel van dit algoritme is dat het getal K op voorhand moet vastliggen, terwijl het juist zeer belangrijk is dat dit getal goed gekozen wordt.
Een oplossing hiervoor is gebruik maken van canopy clustering. Dit is ook een clustering algoritme dat zeer snel uitgevoerd wordt, maar niet zo accuraat is. Wat wel positief is aan dit algoritme, is dat hier K niet op voorhand moet bepaald worden. Hierdoor kunnen we eerst de canopy clustering uitvoeren en aan de hand van de K die hier gebruikt wordt, het K-means algoritme uitvoeren (Vandenbussche, 2016). Dit algoritme heeft wel twee andere parameters nodig: T1 ofwel 'loose distance' en T2 ofwel 'tight distance' waarbij geldt dat T1 > T2. Canopy clustering begint door een willekeurige waarde te verwijderen uit de dataset en in een aparte cluster te steken, vervolgens worden alle waarden waarvan de afstand kleiner is dan T1 aan dezelfde cluster toegevoegd. Als voor deze waarden ook geldt dat de afstand kleiner is dan T2, wordt de waarde ook verwijderd uit de dataset zodat deze niet meer in andere clusters kan komen. Als de afstand tussen T1 en T2 ligt, betekent het dat de waarde wel aan de cluster wordt toegevoegd, maar dat deze ver genoeg van het centerpunt verwijderd is zodat deze waarde ook tot andere clusters kan behoren. Als deze eerste cluster klaar is, wordt opnieuw een willekeurige waarde uit de dataset verwijderd en dezelfde stappen worden uitgevoerd. Dit blijft doorgaan totdat alle waarden uit de dataset zijn verwijderd (Mahout, g.d.).

2.4.2. Dbscan

Dbscan is de afkorting voor Density-based spatial clustering of applications with noise. Dit is een clustering algoritme dat ontstaan is om nested clusters goed te kunnen berekenen. Dit zijn clusters die gedeeltelijk of zelfs helemaal omringd zijn

door een andere cluster. Dit algoritme komt het dichtst in de buurt met hoe het menselijk oog clustert.
Dbscan gaat als volgt te werk: eerst telt het voor iedere waarde, het aantal punten die dicht in de buurt liggen. Dit wordt simpelweg gedaan door een cirkel te trekken rond het punt en vervolgens alle punten die in de cirkel liggen te tellen. De straal van de cirkel is één van de twee parameters die op voorhand moet worden meegegeven. De tweede parameter is het aantal dichte buren een punt nodig heeft om een Core Point te worden. Stel dat dit aantal vier is, dan worden alle punten die vier of meer punten in hun cirkel hebben een Core Point. Dit zijn meestal de punten die in het midden van een cluster liggen.
Vervolgens start het algoritme bij een willekeurig Core Point en dit wordt aan de eerste cluster toegevoegd. Daarna worden zijn buren bekeken en alle Core Points daarvan worden ook aan de cluster toegevoegd. Voor ieder toegevoegd punt, wordt deze stap uitgevoerd, tot er geen meer bij kunnen. Vervolgens worden voor ieder punt uit de cluster, zijn buren die geen Core Points zijn ook toegevoegd, maar dit zijn de laatste, hun buren worden niet meer toegevoegd aan deze cluster. Dit is de eerste cluster, nu wordt er terug gestart bij een willekeurig Core Point die nog niet tot een cluster behoort. Op het einde, als er geen Core Points meer overblijven, kan het zijn dat er nog punten zijn die niet tot een cluster behoren. Deze zijn de uitschieters (Starmer, 2022).

2.4.3. Hierarchisch

Bij hierarchisch clusteren moet de data niet omgezet worden in getallen zoals bij K-Means of dbscan wel nodig was. Hierbij wordt gebruik gemaakt van de afstand tussen de strings. Als eerste stap wordt de eerste waarde vergeleken met alle andere waarden om te berekenen welke meest gelijkend is aan waarde 1. Deze stap wordt uitgevoerd bij iedere waarde in de dataset zodat voor elke waarde, de afstand tot zijn meest gelijkende waarde geweten is. Vervolgens wordt de kleinste afstand genomen en deze twee waarden vormen een eerste cluster (Starmer, 2017).
Nu worden deze stappen opnieuw uitgevoerd, maar de cluster wordt vergeleken alsof het één waarde is. Op deze manier kan blijven doorgegaan worden totdat alle waarden in één cluster zitten (Starmer, 2017).

2.5. MoSCoW

Om het meest geschikte algoritme te vinden zal gebruik gemaakt worden van de MoSCoW methode. Dit is een populaire methode om de vereiste eigenschappen te beheren en te prioriteren. MoSCoW is een acroniem en staat voor Must-have, Should-have, Could-have en Won't have (ProductPlan, g.d.).

1. Must have: Eigenschappen die verplicht zijn, niet onderhandelbaar.
2. Should have: Belangrijke eigenschappen die een grote waarde bijdragen, maar

niet essentieel zijn.

3. Could have: Dit zijn eigenschappen die wel een kleine bijdrage leveren, maar eerder bijzaak zijn.
4. Won't have: Eigenschappen die voor deze situatie of dit onderzoek geen meerwaarde brengen.

3

Methodologie

Zoals vele onderzoeken wordt hier ook begonnen met een requirements analyse. Op die manier is het duidelijk welke eigenschappen van de algoritmes belangrijk zijn voor deze onderzoeksvraag.

3.1. Requirements analyse

- 1. Must have: Gemiddelde aantal waarden per cluster > 5
- 2. Should have: Grootste cluster bevat minder dan 5% van het aantal waarden
- 3. Could have: Silhoutte average > 0.25
- 4. Won't have: Een vuilniscluster die meer dan 20% van alle waarden bevat.

Het is moeilijk een ideale voorspelling te doen voor het gemiddelde aantal waarden per cluster. In het geval dat de data perfect in twee clusters kan verdeeld worden, zal het gemiddelde heel hoog liggen terwijl de clusters ook succesvol zijn. Er is dus geen maximum op dit gemiddelde. Een minimum is echter wel verplicht. Wanneer dit gemiddelde lager dan vijf zou liggen, betekent dat dat de clusters veel te klein zijn om als succesvol beschouwt te worden.
Als tweede zou de grootste cluster best minder dan 5% van alle waarden bevatten. Het is een should have, want er kunnen op zich gevallen zijn waar meer dan vijf procent wel degelijk in dezelfde cluster hoort, maar in de meeste gevallen en in de datasets die in dit onderzoek gebruikt worden, is dit niet het geval.
Als derde is er de Silhoutte average. Dit is een score tussen -1 en 1 en deze wordt berekend door voor iedere waarde te kijken of deze in de juiste cluster zit, of dichter aanleunt bij een andere cluster. Hoe hoger het gemiddelde, hoe beter de clusters. Dit is een could have, want dit kan soms bedrieglijk overkomen. Wanneer er bijvoorbeeld maar één cluster is en alle waarden zitten in diezelfde cluster, dan is dit

gemiddelde 1 aangezien geen elke waarde beter in een andere cluster thuishoort, want er zijn geen andere clusters.
Als vierde is er ook een won't have, in de meeste gevallen zal er een vuilniscluster aangemaakt worden waar waarden in belanden die nergens anders bijpassen. Het is echter zeker niet de bedoeling dat deze cluster te groot wordt, want dan worden de resultaten niet meer nuttig. De maximumgrootte voor deze vuilniscluster wordt dus op 20% gelegd. Als er dus 1000 waarden zijn, mogen er maximum 200 zijn die niet tot een goede clusters behoren.

3.2. Short-list

Uit alle algoritmes die besproken geweest zijn in de literatuurstudie, zijn er vijf finale combinaties overgebleven die onderzocht en vergeleken zullen worden. Hiërarchisch clusteren heeft een methode nodig om de afstand tussen twee waarden te berekenen. Dit wordt gecombineerd met de levenshtein distance, wat een goede en eenvoudige manier is om dit te bepalen. De twee andere clustermethoden, K-means en dbscan hebben een methode nodig die de tekst kan omzetten in nummers. Hiervoor zijn er twee geschikte methoden gevonden, die op een heel verschillende manier te werk gaan, namelijk tf-idf en word2vec. De vijf combinaties zijn als volgt:

- 1. Tf-idf en K-means
- 2. Tf-idf en dbscan
- 3. Word2vec en K-means
- 4. Word2vec en dbScan
- 5. Levenshtein en hiërarchical clustering

3.3. Gebruikte data

Dit onderzoek maakt gebruik van drie datasets met ongestructureerde data. De eerste dataset komt rechtstreeks uit de praktijk, heeft 4146 waarden en bevat productdata. Een voorbeeld van een waarde uit deze dataset is: '100mm Knauf Rocksilk Flexi Slab (4.32m^2)'. Bij veel waarden in deze set zijn ook de afmetingen vermeld. Deze informatie is echter niet interessant, aangezien stukken hout van 100mm en van 140mm tot dezelfde cluster behoren. Om een beter resultaat te verkrijgen worden alle getallen in deze set op 0 gezet. Zo blijft het duidelijk dat de data een afmeting bevat, maar de grootte van die afmeting is irrelevant.
De tweede dataset bevat 4000 waardes omtrent wijn en is afkomstig van 'Kaggle' (Iv, 2018). Een voorbeeld van een waarde uit deze set: 'The variety is unmistakable, with notes of dried herb, dark cherry and espresso. The jammy fruit flavors are rich, with lightly grainy tannins and coffee notes on the finish. It shows a fine sense of

balance.'. De waardes uit deze set zijn duidelijk een stuk langer en bevatten geen afkortingen of getallen.
De derde dataset is gevonden op 'UCI' en bevat 5000 waarden (Chen e.a., 2012). Deze set bevat opnieuw productdata, maar ook deze heeft geen getallen of afkortingen. Een voorbeeld van een waarde uit deze set: 'GREY HEART HOT WATER BOTTLE'.

```
df_original = pd.read_csv("data.csv", delimiter=';')
chosen_column = "Material Description EN"

df_original[chosen_column] = df_original[chosen_column].str.replace('[0-9]',
    '0', regex=True)
```

Om deze datasets te kunnen gebruiken in Python, wordt gebruik gemaakt van pandas. Pandas is een Python bibliotheek die heel veel kan, onder andere csv bestanden inlezen. De delimiter die wordt meegegeven is de aanduiding waar een waarde in de kolom eindigt en de volgende waarde begint. De chosen_column waarde is de naam van de kolom die wordt gebruikt, in dit geval is dit de kolom met materiaal beschrijvingen. Door 'df_original[chosen_column]' op te roepen, wordt de juiste kolom uit de dataset verkregen. Tot slot worden, zoals hiervoor besproken, alle getallen tussen 0 en 9 vervangen door een 0.

3.4. Opmaken algoritmes in Python

3.4.1. tf-idf

Nu de short-list is opgesteld en de data ingelezen is, is de volgende stap het uitwerken van de algoritmen in Python. Er wordt begonnen met tf-idf. De TfidfVectorizer wordt geïmporteerd van sklearn. Tf-idf wordt uitgevoerd met n-grammen, maar deze kunnen op basis van karakters of volledige woorden zijn. De variabele chosen_ngrams bepaalt dit: als de waarde 'char' is, zijn het karakters, is de waarde 'word' dan is het met woorden.

```
from sklearn.feature_extraction.text import TfidfVectorizer

chosen_ngrams = "char"
chosen_ngram_range = (2, 4)
tfidf_vectorizer=TfidfVectorizer(use_idf=True, analyzer = chosen_ngrams,
    ngram_range=chosen_ngram_range)
tfidf_vectorizer_vectors =
    tfidf_vectorizer.fit_transform(df_original[chosen_column].to_list())
```

De volgende variabele bepaalt de n-gram range, in dit geval wordt gewerkt met 2-grammen, 3-grammen en 4-grammen. Vervolgens wordt een vectorizer aange-

maakt aan de hand van deze twee variabelen. Deze vectorizer wordt gebruikt op de kolom met data zodat tfidf_vectorizer_vectors een lijst met alle vectoren van de waarden bevat.

3.4.2. K-means

Om K-means te kunnen gebruiken, moet het aantal clusters op voorhand beslist worden. De manier die hiervoor gebruikt wordt in dit onderzoek, wordt in het volgende hoofdstuk genaamd 'Canopy clustering' uitgelegd. Voorlopig wordt met een waarde van 20 gewerkt.
De KMeans() functie wordt geïmporteerd van sklearn, het aantal clusters en de vectoren van tf-idf worden hier als parameter meegegeven. Vervolgens wordt de pandas bibliotheek opnieuw gebruikt om een nieuwe dataframe aan te maken die twee kolommen bevat. De eerste kolom bevat de labels die de KMeans functie teruggegeven heeft, de tweede kolom bevat de productbeschrijvingen. Dit ziet er als volgt uit:
5 , 000mm F/F Cladding Roll 0000x0000
18 , 000mm Kingspan TP00 PIR (0.00m²)
18 , 000mm Kingspan TR00 PIR (0.00m²)
De Cladding Roll zit in cluster nummer vijf, de twee kingspannen, waarbij de afmetingen op 0 geplaatst zijn om identieke waardes to bekomen, zitten in cluster nummer achttien.

```
kmeans = KMeans(20).fit(tfidf_vectorizer_vectors)
df_clustered = pd.DataFrame({'cluster': kmeans.labels_, 'value':
    df_original[chosen_column].to_list()})
```

3.4.3. Canopy clustering

Om K-means te kunnen gebruiken, moet het aantal clusters op voorhand beslist worden. In dit onderzoek wordt dit aantal bepaald door canopy clustering. Zoals besproken in de literatuurstudie is deze manier van clusteren niet zo accuraat, maar berekent deze wel zelf het aantal clusters. Hieronder is de functie te zien die gebruikt wordt om het aantal clusters te berekenen. Deze functie heeft vier parameters:

1. X: De dataset in de vorm van een lijst met vectoren.
2. T1: Loose distance.
3. T2: Tight distance.
4. Distance metric: De manier om de afstand te berekenen, deze staat standaard op de euclidean distance.

```
def canopy(X, T1, T2, distance_metric='euclidean'):
canopies = dict()
X1_dist = pairwise_distances(X, metric=distance_metric)
canopy_points = set(range(X.shape[0]))
while canopy_points:
point = canopy_points.pop()
i = len(canopies)
canopies[i] = {"c":point, "points": list(np.where(X1_dist[point] < T1)[0])}
canopy_points = canopy_points.difference(set(np.where(X1_dist[point] < T2)[0]))
return len(canopies)
```

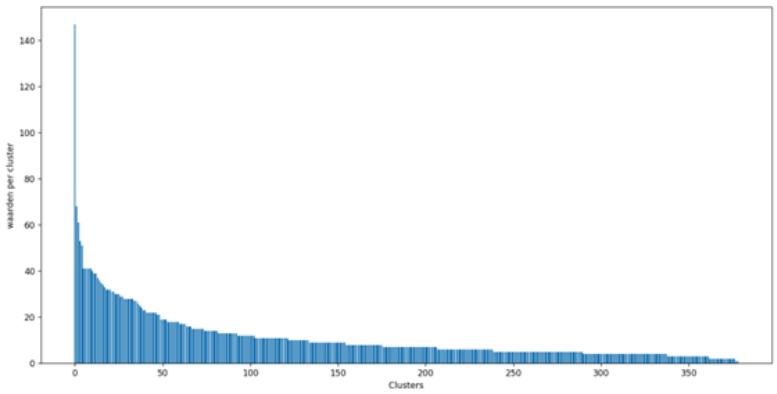
De functie begint met het aanmaken van een lege dictionary om hier uiteindelijk alle clusters in op te nemen. Vervolgens wordt de pairwise_distances functie gebruikt van sklearn. Deze geeft een matrix terug met alle afstanden tussen alle waarden van de array, zodat geldt dat X1_dist[i,j] de afstand is tussen de waarden i en j uit de originele array van vectoren. Hierna maakt de functie een set aan die alle indices van de array van vectoren bevat. Zolang er waardes in de set zitten, worden volgende acties doorlopen:

- De laatste index wordt verwijderd, dit wordt het centerpunt van de nieuwe cluster.
- Er wordt een nieuwe cluster aangemaakt in de dictonairy.
- Aan die nieuwe cluster worden alle resterende waarden toegevoegd waarvan hun afstand tot het centerpunt kleiner is dan T1.
- Alle indices waarvoor deze afstand ook kleiner is dan T2 worden uit de canopy_points set verwijderd.

Van zodra alle waardes in de set verwijderd zijn, wordt het aantal clusters teruggegeven.
Aangezien enkel het aantal clusters belangrijk is en niet wat er precies in de cluster zit, is T1 in deze situatie irrelevant. T2 is wel belangrijk, als deze waarde te klein is, wordt voor iedere waarde een aparte cluster aangemaakt, is de waarde te groot, dan ontstaat er slechts één cluster die alle waarden bevat.
Om de ideale waarde voor T2 te bepalen, werd eerst wat logisch nagedacht. Er zijn ongeveer 4000 waarden in de dataset en volgens de requirements moet er minstens een gemiddelde van vijf waarden per cluster zijn. Een maximum van 800 clusters dus. Door T2 een waarde van 1,2 te geven, werden er 557 clusters verkregen, maar door het resultaat te bekijken bleek dit nog te veel, volgende producten zitten in een aparte cluster, terwijl enkel de kleur verschilt:
473 , Ridge Tee Black
344 , Ridge Tee Dark Brown

317 , Ridge Tee Terracotta

Voor dit onderzoek worden deze waarden in dezelfde cluster verwacht. Door T2 een waarde van 1,35 toe te kennen, resulteert dit slechts in 120 clusters. Aangezien de grootste cluster meer dan 500 waarden bevat terwijl de tweede grootste cluster slechts 100 waarden bevat, blijkt dat 1,35 te veel is. Na verder onderzoek komt 1,26 als goede waarde naar boven. Dit resulteert in 378 clusters zodat er gemiddeld ongeveer tien waarden in een cluster zitten. Er wordt nog steeds één cluster opgemerkt die opmerkelijk groter is dan de anderen met 147 waarden, maar na deze cluster in detail te bekijken blijkt dit grotendeels data die inderdaad niet tot een andere cluster behoort.



Figuur (3.1)
Een staafdiagram van de resultaten van tf-idf in combinatie met K-means.

3.4.4. Word2vec

De word2vec functie wordt ingeladen met spacy. Dit verwacht een string parameter die bestaat uit vier delen, deze bepaalt welke versie van het word2vec algoritme teruggegeven wordt. Aangezien word2vec de afstand bepaalt aan de hand van de semantiek, bestaat er voor elke taal een ander algoritme natuurlijk. De parameter bestaat uit:

- De taal: de datasets die gebruikt worden in dit onderzoek zijn in het Engels, dus wordt 'en' gebruikt. Dit kan natuurlijk ook in andere talen zoals Nederlands en Frans.
- Het type: 'core' is de enige mogelijkheid in het Engels.
- De data waarmee het algoritme getraind is: 'web' is de enige mogelijkheid in het Engels. In het Nederlands moet deze waard verplicht 'news' zijn, het Nederlandse algoritme is dus enkel getraind met data van nieuws en media.

- De size: de mogelijkheden zijn 'sm', 'md', 'lg' en 'trf'. Hoe smaller de size, hoe sneller en efficiënter het algoritme gebruikt kan worden, hoe groter de size, hoe accurater de resultaten zijn.

Word2vec gebruikt de betekenis van een woord om het daarna een vector toe te wijzen, maar als in onze data verschillende code of afmetingen staan zoals '454mm', dan kan het gebeuren dat word2vec daar niet mee overweg kan. Om die reden zijn verschillende tekens aangepast in de data. Zo zijn bijvoorbeeld alle / tekens vervangen door een 9, alle $*$ tekens door een 8.
Vervolgens maakt numpy een lege array aan. Aan deze array wordt zijn shape meegegeven met als *x* de shape van de originele dataset en als *y* moet het eruitzien zoals een uitkomst van het word2vec algoritme. Dit wordt verkregen door het algoritme aan te roepen met als waarde 'test' en daarvan zijn shape te gebruiken. Daarna wordt geïtereerd over alle rijen van de dataset. Voor iedere rij wordt de tekst omgezet naar een vector met behulp van het word2vec algoritme. Deze vector wordt aan de aangemaakte array op de juiste plaats toegevoegd.

```
word2vec = spacy.load('en_core_web_md')
doc= np.empty(shape=(df_original.shape[0], word2vec("test").vector.shape[0]))

for idx, line in df_original.iterrows():
    vector = word2vec(line[chosen_column]).vector
    doc[idx] =vector
```

3.4.5. Dbscan

De DBSCAN() functie wordt net zoals K-means geïmporteerd van sklearn. Deze heeft twee parameters nodig, een eps en een minimum samples. De eps is de straal van de cirkel die rond ieder punt getrokken wordt en de minimum samples is het aantal punten er in die cirkel moeten liggen om een Core Point te worden. Om de beste waarden te vinden, werd gewerkt met een dubbele for lus zodat allerlei combinaties geprobeerd konden worden.
Al snel werd duidelijk dat er voor deze doelstelling slechts twee mogelijkheden zijn voor het minimum aantal samples, namelijk 1 of 2. De uitkomst is echter gelijk voor deze twee mogelijkheden. Het enige verschil is dat bij een minimum van 1 alle waarden die niet dicht bij andere waarden liggen helemaal alleen in een aparte cluster zitten en bij een minimum van twee zitten deze waarden simpelweg niet in een cluster. Het label van hun cluster staat dan op -1, dit kan op zich gezien worden als een aparte cluster waar alle restwaarden in zitten. In dit onderzoek wordt gewerkt met een minimum van 2 zodat het aantal succesvolle clusters gekend is.
De straal van de cirkel was iets lastiger om te bepalen. Bij het proberen van waarde 1 had één cluster 1600 waarden. Toen 1 te veel bleek, werd 0.5 geprobeerd. Dit bleek echter te weinig, want 1100 waarden van de 4000 waren niet aan een cluster

toegevoegd. Als volgende werd 0.75 geprobeerd, met nog steeds meer dan 800 waarden die niet tot een cluster behoorden. De ideale waarde lag hierdoor duidelijk tussen 0.75 en 1. Er werd niet direct een perfecte waarde gevonden aangezien er nog altijd veel waarden niet aan een cluster werden toegevoegd en tegelijkertijd werd de grootste succesvolle cluster maar groter en groter. Toen werd er gezocht naar een waarde waarbij de het aantal waarden in die twee clusters ongeveer gelijk was. Uiteindelijk werd tot de conclusie gekomen dat 0.92 als beste straal aangeduid, aangezien zo'n 660 waarden niet tot een cluster behoorden en de grootste succesvolle cluster zo'n 670 waarden bevatte.

```
    db = DBSCAN(eps=0.92, min_samples=2).fit(tfidf_vectorizer_vectors)
```

3.4.6. Levenshtein

Om hiërarchisch te kunnen clusteren, zijn alle afstanden nodig tussen alle waarden. Eerst wordt de dataset omgezet in een lijst. Daarna wordt een lege array aangemaakt en de lengte van de lijst wordt gebruikt als het aantal rijen en het aantal kolommen. Zo wordt het een twee-dimensionale array ook wel matrix genoemd. Ook wordt het datatype van deze array op integer gezet. Vervolgens wordt gewerkt met een dubbele for lus zodat alle combinaties tussen alle waarden overlopen worden. Voor elke combinatie wordt de distance() functie aangeroepen, deze is geïmporteerd van Levenshtein en zal de levenshtein distance tussen de twee waarden teruggeven. Deze afstand wordt dan op de juiste plaats in de matrix geplaatst.

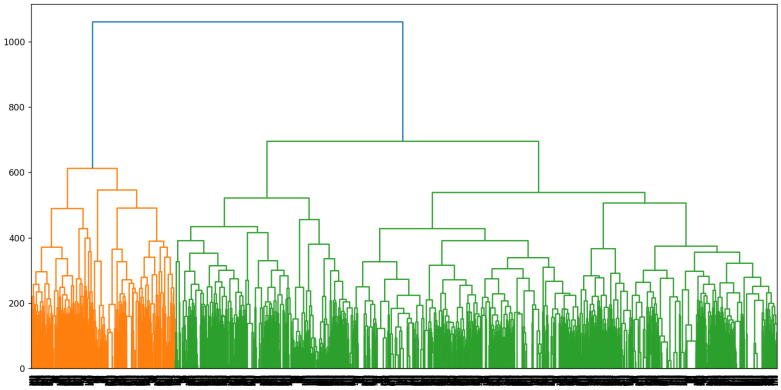
```
data_list = df_original[chosen_column].to_list()

Matrix = np.zeros((len(data_list),len(data_list)),dtype=np.int)

for i in range(0,len(data_list)):
    for j in range(0,len(data_list)):
        Matrix[i,j] = distance(data_list[i],data_list[j])
```

3.4.7. Hiërarchisch clusteren

Het hiërarchisch clusteren wordt gedaan met behulp van Scipy, dit is een wetenschappelijk rekenpakket voor Python. Van Scipy worden de functies linkage(), dendrogram() en fcluster() gebruikt. De linkage functie is de functie die clustert, hij krijgt de matrix met afstanden mee en een methode. Deze methode bepaalt hoe de afstand tussen een waarde en een cluster berekend wordt. Bij *complete* wordt de verste afstand gebruikt, bij *single* de kortste afstand en bij *average* het gemiddelde

van alle afstanden. Vervolgens kan een dendrogram gemaakt worden van deze data. Bij het tonen van het plot, wordt de onderstaande figuur weergegeven.

```
linkage_data = linkage(Matrix, method='complete')
dendrogram(linkage_data)
plt.show()
```



Figuur (3.2)
Dendrogram van het hiërarchisch clusteren.

Uit het dendrogram alleen is weinig af te leiden aangezien dit blijft doorgaan totdat alle waarden in één cluster zitten. Het dendrogram moet als het ware horizontaal doorgesneden worden op de juiste plaats zodat er kan berekend worden welke waarden samen in een cluster zitten. Om dit uit te voeren wordt gebruik gemaakt van de fcluster functie. Deze functie bepaalt waar de horizontale scheiding ligt door te kijken naar de afstand tussen twee clusters die samengevoegd worden. Als deze afstand groter is dan de treshold, dan worden de clusters niet meer samengevoegd.

```
threshold = 200
final_clusters = fcluster(linkage_data, threshold, criterion='distance')
```

3.5. Vergelijkende studie

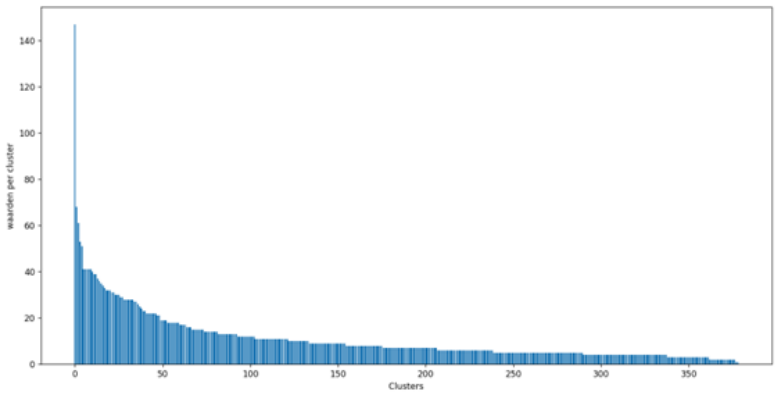
In deze sectie worden de resultaten van alle algoritmes besproken. Ook worden deze resultaten vergeleken met elkaar, zodat kan achterhaald worden welke manieren het beste werken voor verschillende situaties.

Eerste dataset

Als eerste worden de vijf mogelijkheden uitgevoerd met de dataset die uit de praktijk komt. Hierbij wordt gekeken naar de grootste cluster, aantal clusters, gemiddelde aantal waarden per cluster, een eventuele vuilniscluster en het Silhoutte gemiddelde. Dit Silhoutte gemiddelde geeft weer hoe goed elke waarde in zijn respectievelijke cluster thuishoort in vergelijking met de andere clusters. Dit is een score tussen min één en één. Hoe hoger de score, hoe beter de clusters gevormd zijn. Als laatste wordt ook een kort willekeurig stuk uit de data getoond.

1. Tf-idf en K-means:

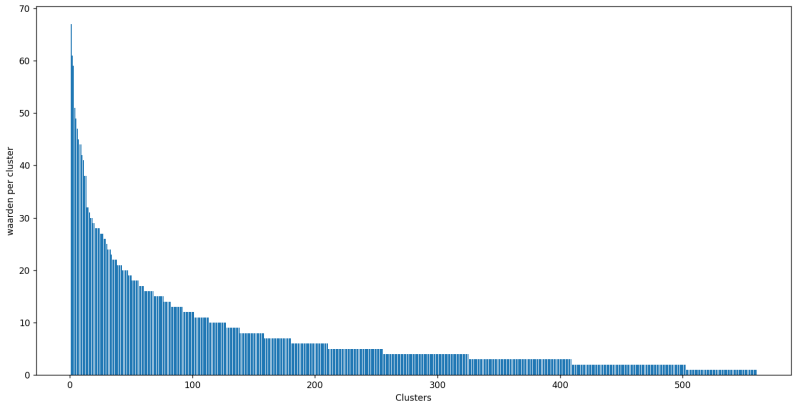
    - Grootste cluster: 147
    - Grootte vuilniscluster: 147
    - Aantal clusters: 378
    - Gemiddeld aantal waarden per cluster: 10,96 => quasi 11
    - Silhoutte avg: 0.47
    - 110 , Perfix T.B 0.0x00
      110 , "Perfix TB/Zn 0,0X00"
      285 , Pin Router Head Hitachi
      83 , PIN THREADS 00 X 000
      83 , PIN THREADS 0 X 00
      169 , PINCE DEMONTAGE RS-EVO
      176 , Pipe clip CUPRA D00 (CR00)
      73 , Pipe Clip Diameter 000 B
      73 , Pipe Clip Diameter 000 G



Figuur (3.3)
Resultaten tf-idf met K-means voor de eerste dataset

2. Levenshtein distance en hiërarchisch clusteren:

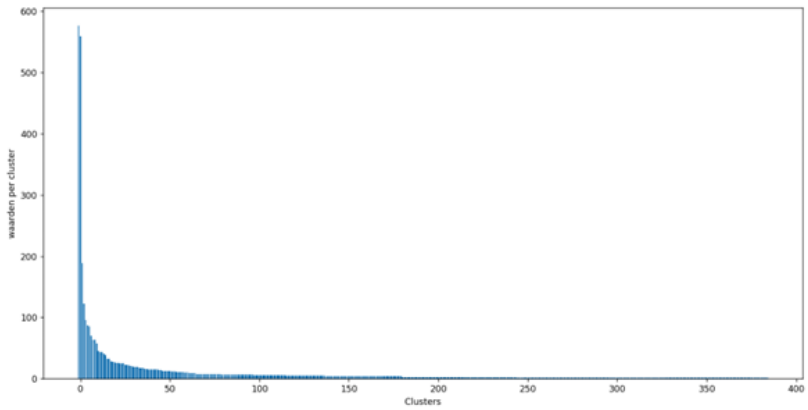
    - Grootste cluster: 67
    - Grootte vuilniscluster: Niet gevonden
    - Aantal clusters: 560
    - Gemiddeld aantal waarden per cluster: 7,40
    - Silhoutte avg: 0.44
    - 361 , Perfix T.B 0.0x00
      361 , "Perfix TB/Zn 0,0X00"
      442 , Pin Router Head Hitachi
      377 , PIN THREADS 00 X 000
      377 , PIN THREADS 0 X 00
      456 , PINCE DEMONTAGE RS-EVO
      393 , Pipe clip CUPRA D00 (CR00)
      397 , Pipe Clip Diameter 000 B
      397 , Pipe Clip Diameter 000 G



Figuur (3.4)
Resultaten hiërarchisch clusteren voor de eerste dataset

3. Tf-idf en dbscan:

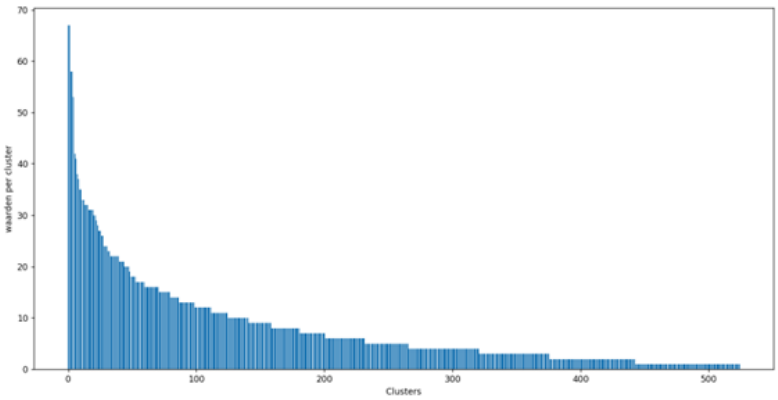
    - Grootste cluster: 577
    - Grootte vuilniscluster: 559
    - Aantal clusters: 384
    - Gemiddeld aantal waarden per cluster: 10,80
    - Silhoutte avg: 0.36
    - -1 , Perfix T.B 0.0x00
      -1 , "Perfix TB/Zn 0,0X00"
      -1 , Pin Router Head Hitachi
      248 , PIN THREADS 00 X 000
      248 , PIN THREADS 0 X 00
      -1 , PINCE DEMONTAGE RS-EVO
      -1 , Pipe clip CUPRA D00 (CR00)
      249 , Pipe Clip Diameter 000 B
      249 , Pipe Clip Diameter 000 G



Figuur (3.5)
Resultaten tf-idf met dbscan voor de eerste dataset

4. Word2vec en K-means:

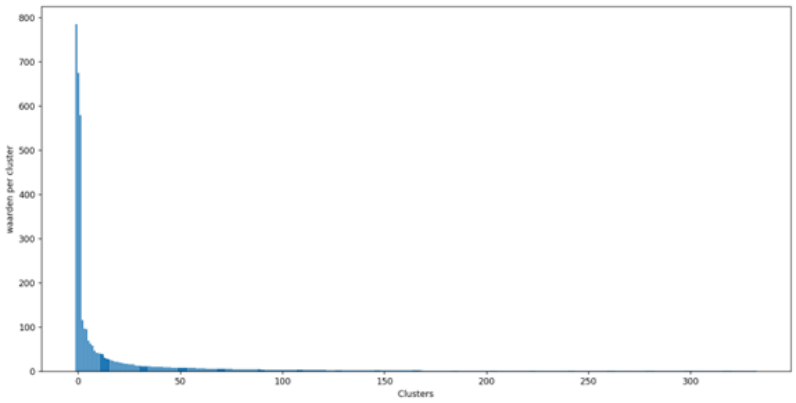
   - Grootste cluster: 67
   - Grootte vuilniscluster: Niet gevonden
   - Aantal clusters: 524
   - Gemiddeld aantal waarden per cluster: 7,91
   - Silhoutte avg: 0.64
   - 377 , Perfix T.B 0.000 0
     443 , Perfix TB 9 Zn 0 7 000 0
     506 , Pin Router Head Hitachi
     415 , PIN THREADS 00 X 000
     415 , PIN THREADS 0 X 00
     113 , PINCE DEMONTAGE RS-EVO
     347 , Pipe clip CUPRA D00 (C000)
     427 , Pipe Clip Diameter 000 B
     384 , Pipe Clip Diameter 000 G



Figuur (3.6)
Resultaten word2vec met K-means voor de eerste dataset

5. Word2vec en dbscan:

    - Grootste cluster: 785
    - Grootte vuilniscluster: 675
    - Aantal clusters: 332
    - Gemiddeld aantal waarden per cluster: 12,94
    - Silhoutte avg: 0.16
    - -1 , Perfix T.B 0.000 0
      -1 , Perfix TB 9 Zn 0 7 000 0
      -1 , Pin Router Head Hitachi
      -1 , PIN THREADS 00 X 000
      -1 , PIN THREADS 0 X 00
      66 , PINCE DEMONTAGE RS-EVO
      216 , Pipe clip CUPRA D00 (C000)
      223 , Pipe Clip Diameter 000 B
      224 , Pipe Clip Diameter 000 G



Figuur (3.7)
Resultaten word2vec met dbscan voor de eerste dataset

Als laatste wordt de gelijkenis tussen de vijf resultaten berekend. Dit gebeurt door de adjusted_rand_score() functie die afkomstig is van sklearn. Deze functie vergelijkt de cluster resultaten en geeft een getal tussen nul en één terug. Hoe hoger dit getal is, hoe dichter de resultaten bij elkaar liggen.

(Word2vec met Kmeans) - (Hiërarchisch):	ARI = 0.51
(Tf-idf met Kmeans) - (Hiërarchisch):	ARI = 0.46
(Tf-idf met Kmeans) - (Word2vec met Kmeans):	ARI = 0.43
(Tf-idf met dbscan) - (Word2vec met dbscan):	ARI = 0.21
(Tf-idf met Kmeans) - (Tf-idf met dbscan):	ARI = 0.18
(Hiërarchisch) - (Tf-idf met dbscan):	ARI = 0.12

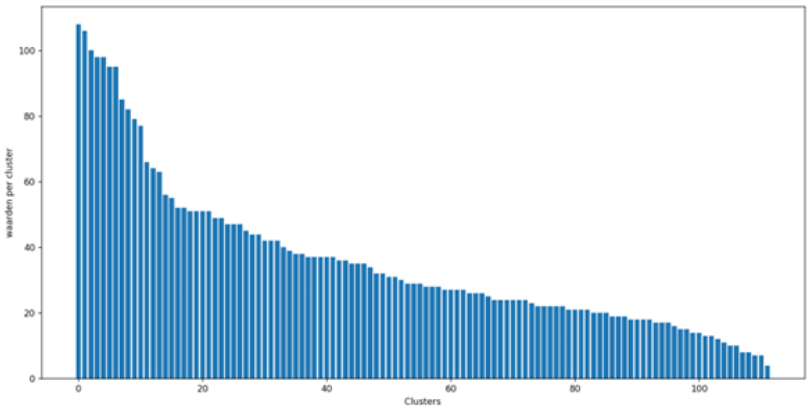
(Word2vec met Kmeans) - (Tf-idf met dbscan):	ARI = 0.11
(Tf-idf met Kmeans) - (Word2vec met dbscan):	ARI = 0.07
(Word2vec met Kmeans) - (Word2vec met dbscan):	ARI = 0.07
(Hiërarchisch) - (Word2vec met dbscan):	ARI = 0.06

De resultaten worden gerangschikt van groot naar klein en door deze scores te bekijken valt op dat er drie scores hoger zijn dan de rest. Dit zijn de combinaties tussen Kmeans en Hiërarchisch clusteren. Hieruit kan afgeleid worden dat de resultaten van het clusteren met dbscan heel verschillend zijn van de rest. Dit wil nog niet perse zeggen dat het slecht is, maar het is geen goed teken.
Als tweede worden de silhoutte gemiddelde's vergeleken. 0.64, 0.47, 0.44, 0.36 en 0.16. Ook hier valt op dat de gemiddelde's bij het clusteren met dbscan de slechtste resultaten weergeeft. Kmeans in combinatie met Word2vec scoort hier duidelijk het hoogst.
Als derde wordt de grafiek bekeken. Ook hier valt dbscan direct op. Door alle toppen van de staven te verbinden kan een variatie op de 1/x grafiek ingebeeld worden. Bij de twee dbscan grafieken valt op dat die lijn enorm snel verticaal naar beneden gaat en vervolgens nog heel lang quasi horizontaal doorgaat. Dit betekent dat er een paar clusters heel veel waarden bevatten en dat er enorm veel clusters zijn die slechts een paar waarden hebben.
Bij de drie andere grafieken gebeurt de overgang van verticaal naar horizontaal veel trager, wat positief is. Dit wil zeggen dat de clusters beter verdeeld zijn.
Wat nog opvalt bij dbscan is de hoeveelheid waarden die niet tot een cluster behoren. Het feit dat ongeveer 600 van de 4000 waarden niet aan een cluster zijn toegewezen, is zeker niet de bedoeling bij deze onderzoeksvraag.

Tweede dataset

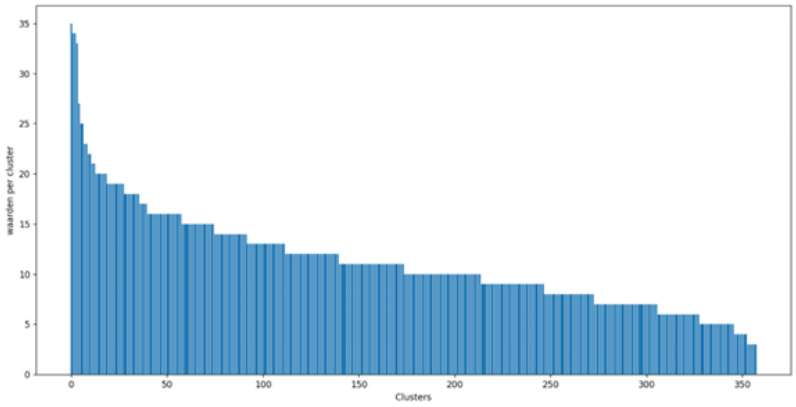
Als tweede dataset wordt de data met informatie over wijnen gebruikt. Dezelfde kenmerken worden vergeleken.

1. Tf-idf met n-grammen op basis van karakters en K-means:
    - Grootste cluster: 108
    - Grootte vuilniscluster: Niet gevonden
    - Aantal clusters: 111
    - Gemiddeld aantal waarden per cluster:36,04
    - Silhoutte avg: -0.0004



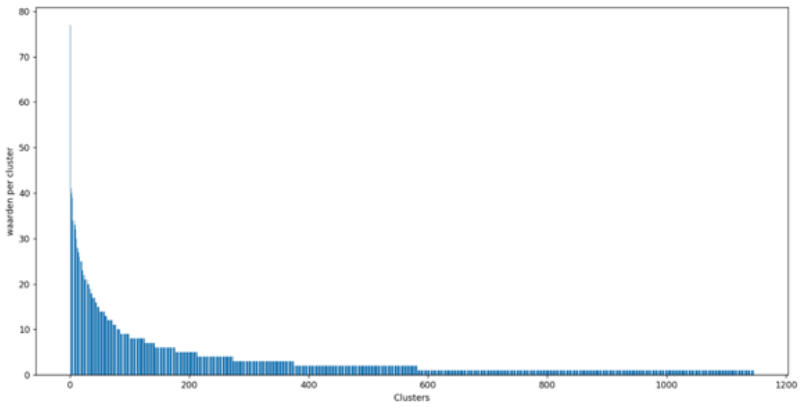
Figuur (3.8)
Resultaten tf-idf, op basis van karakters, met K-means voor de tweede dataset

2. Tf-idf met n-grammen op basis van woorden en K-means:
    - Grootste cluster: 35
    - Grootte vuilniscluster: Niet gevonden
    - Aantal clusters: 358
    - Gemiddeld aantal waarden per cluster: 11,17
    - Silhoutte avg: 0.0019



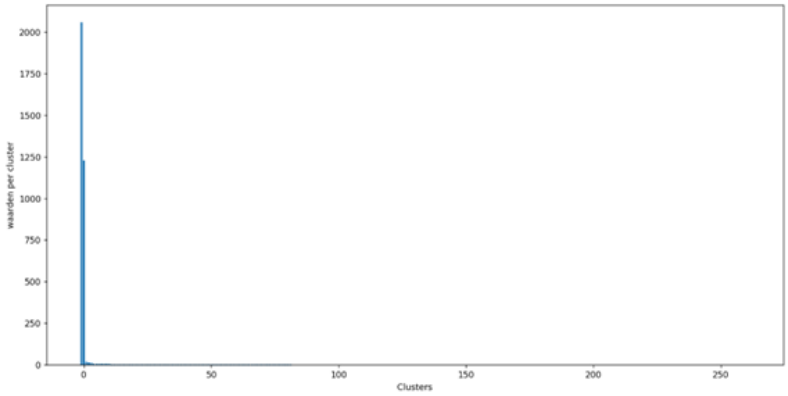
Figuur (3.9)
Resultaten tf-idf, op basis van woorden, met K-means voor de tweede dataset

3. Levenshtein distance en hiërarchisch clusteren:
    - Grootste cluster: 77
    - Grootte vuilniscluster: Niet gevonden
    - Aantal clusters: 1146
    - Gemiddeld aantal waarden per cluster: 3,49
    - Silhoutte avg: 0.02



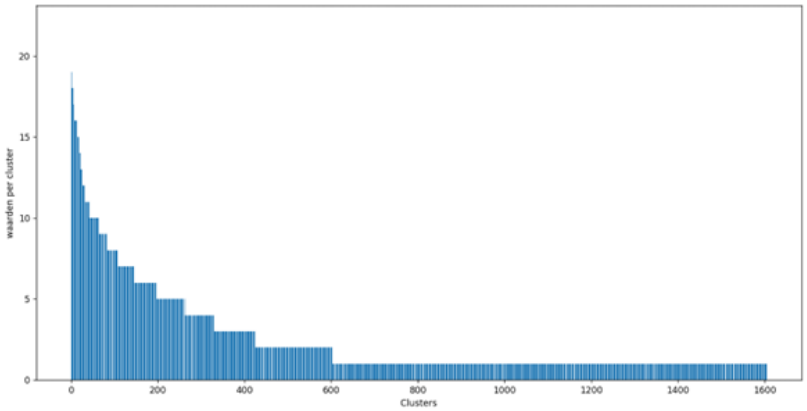
Figuur (3.10)
Resultaten hiërarchisch clusteren voor de tweede dataset

4. Tf-idf en dbscan:
    - Grootste cluster: 2060
    - Grootte vuilniscluster: 2060
    - Aantal clusters: 261
    - Gemiddeld aantal waarden per cluster: 15,3
    - Silhoutte avg: 0.0017



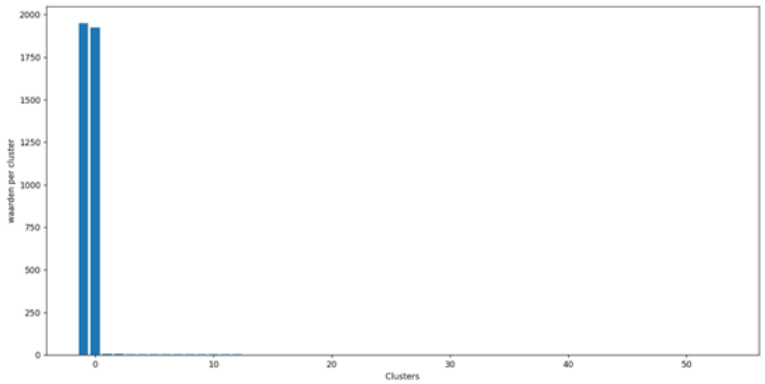
Figuur (3.11)
Resultaten tf-idf met dbscan voor de tweede dataset

5. Word2vec en K-means:
    - Grootste cluster: 18
    - Grootte vuilniscluster: Niet gevonden
    - Aantal clusters: 1606
    - Gemiddeld aantal waarden per cluster: 2,49
    - Silhoutte avg: 0.007



Figuur (3.12)
Resultaten word2vec met K-means voor de tweede dataset

6. Word2vec en dbscan:
    - Grootste cluster: 1963
    - Grootte vuilniscluster: 1925
    - Aantal clusters: 53
    - Gemiddeld aantal waarden per cluster: 75,47
    - Silhoutte avg: -0.23



Figuur (3.13)
Resultaten word2vec met dbscan voor de tweede dataset

Ook voor deze dataset worden alle adjusted rand scores berekend:
(Tf-idf, op basis van char, met Kmeans) - (Tf-idf, op basis van word, met Kmeans):ARI 0.029
(Tf-idf met dbscan) - (Word2vec met dbscan): ARI 0.013
(Word2vec met Kmeans) - (Tf-idf, op basis van word, met Kmeans): ARI 0.008
(Tf-idf, op basis van word, met Kmeans) - (Hiërarchisch): ARI 0.006
(Tf-idf, op basis van char, met Kmeans) - (Word2vec met Kmeans): ARI 0.005
(Tf-idf, op basis van char, met Kmeans) - (Hiërarchisch): ARI 0.005
(Tf-idf, op basis van char, met Kmeans) - (Tf-idf met dbscan): ARI 0.003
(Word2vec met Kmeans) - (Hiërarchisch): ARI 0.003
(Word2vec met Kmeans) - (Tf-idf met dbscan): ARI 0.0003
(Tf-idf, op basis van char, met Kmeans) - (Word2vec met dbscan): ARI 0.002
(Hiërarchisch) - (Tf-idf met dbscan): ARI 0.002
(Tf-idf, op basis van word, met Kmeans) - (Tf-idf met dbscan): ARI 0.001
(Tf-idf, op basis van word, met Kmeans) - (Word2vec met dbscan): ARI 0.001
(Word2vec met Kmeans) - (Word2vec met dbscan): ARI 0.001
(Hiërarchisch) - (Word2vec met dbscan): ARI 0.001

Uit deze scores blijkt direct dat geen van de resultaten op elkaar lijken. Ook de silhoutte gemiddelde's liggen tegen de nul en sommigen zijn zelfs negatief. Het ziet er naar uit dat geen enkele cluster methode overweg kan met data in de vorm

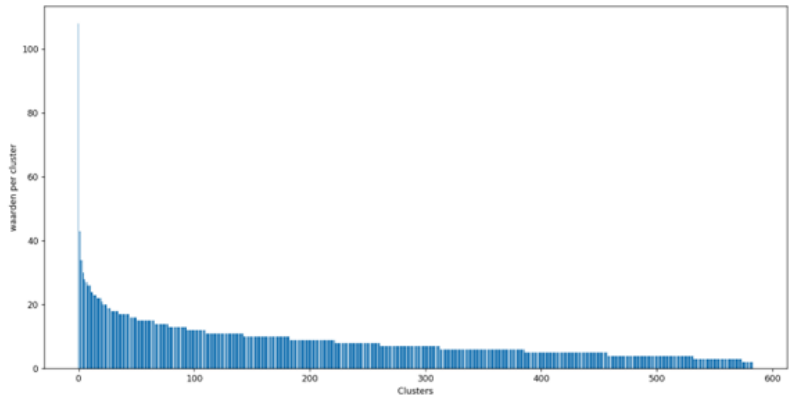
van lange zinnen. De grafieken van Kmeans zien er op het eerste ogenblik nog wel positief uit, maar na een stuk van de geclusterde data te bekijken, lijkt dit enorm willekeurig. Zo zitten de eerste en vierde waarde die hieronder getoond worden in dezelfde cluster, terwijl dit de woorden zijn die overeenkomen in de beschrijvingen: 'aromas of', 'this tastes', 'palate', 'finish' en 'fruits'. Dit zijn niet de belangrijke woorden die voorkomen in de beschrijvingen. Er wordt te veel geclusterd op hoe de beschrijving is opgesteld in plaats van wat de beschrijving precies inhoud.

88,"Reedy green-leaning aromas of plum and raspberry feed into a creamy palate. This tastes primarily of herbal red-berry fruits that finish oaky, dry and spicy."
66,"This shows some pungent, boxwood-like scents up front, followed by modest passion fruit and a bit of citrus, falling away on the finish. Drink now."
82,"Lisboa, with its cool climate, lends itself to crisp rosés like this. This lively fruity wine is touched by hints of vanilla and layers of red-fruit flavors. It is light and ready to drink."
88,"Dry earthy herbal aromas of tomato, red-berry fruits and oak set up a full palate with pounding tannins. This tastes roasted and herbal, with notes of bell pepper, tomato and creamy oak. A hard-tannin finish confirms that this Merlot skipped charm school."
96,"This is a neutral, papery white wine, with a flat, rather than lively mouthfeel. The apple flavored fruit is there, along with a touch of licorice, but the freshness seems to have been lost along the way."
74,"Mula Velha, "the old donkey," is a younger wine than its name suggests. Brisk and fruity, it offers firm tannins and layers of blackberry fruit. With a touch of oak aging, the wine is developing well and will be ready to drink from late 2017."
82,"This wine is ripe, full of black fruits and soft tannins. With plenty of acidity to keep it crisp, the wine has a cool, flavorful character topped off by berry fruits. It's ready to drink."
111,"This wine is light in tannins and ripe in fruit, with a delicious red-berry character. Drink this attractive wine from 2019."
111,"This is a soft, ripe and fruity wine. It has gentle tannins and attractive red-berry fruits. There is just a touch of dryness that gives this easygoing wine its structure. Drink from early 2017."

Derde dataset

Als derde en laatste dataset wordt de data met productinformatie over retail producten gebruikt. Dezelfde kenmerken worden vergeleken.

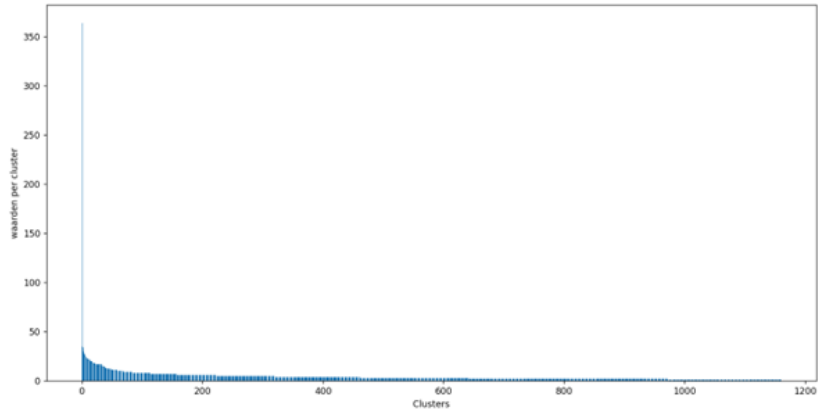
1. Tf-idf met n-grammen op basis van karakters en K-means:
    - Grootste cluster: 108
    - Grootte vuilniscluster: 108
    - Aantal clusters: 583
    - Gemiddeld aantal waarden per cluster: 8,55
    - Silhoutte avg: 0.62
    - 400 , BLACK CHRISTMAS TREE
      400 , PINK AND WHITE CHRISTMAS TREE
      387 , IVORY STRING CURTAIN WITH POLE
      18 , BLUE FELT HANGING HEART W FLOWER
      18 , PINK FELT HANGING HEART W FLOWER
      479 , SMALLFOLKART BAUBLE CHRISTMAS DEC
      388 , FOLKART ZINC HEART CHRISTMAS DEC
      388 , FOLKART CLIP ON STARS



Figuur (3.14)
Resultaten tf-idf, op basis van karakters, met K-means voor de derde dataset

2. Tf-idf met n-grammen op basis van woorden en K-means:

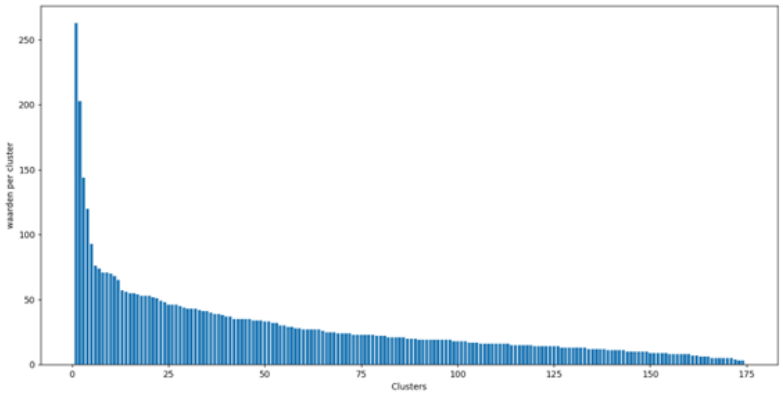
    - Grootste cluster: 364
    - Grootte vuilniscluster: 364
    - Aantal clusters: 1161
    - Gemiddeld aantal waarden per cluster: 4,30
    - Silhoutte avg: 0.87
    - 773 , BLACK CHRISTMAS TREE
      1025 , PINK AND WHITE CHRISTMAS TREE
      1128 , IVORY STRING CURTAIN WITH POLE
      909 , BLUE FELT HANGING HEART W FLOWER
      909 , PINK FELT HANGING HEART W FLOWER
      376 , SMALLFOLKART BAUBLE CHRISTMAS DEC
      410 , FOLKART ZINC HEART CHRISTMAS DEC
      148 , FOLKART CLIP ON STARS



Figuur (3.15)
Resultaten tf-idf, op basis van woorden, met K-means voor de derde dataset

3. Levenshtein distance en hiërarchisch clusteren:

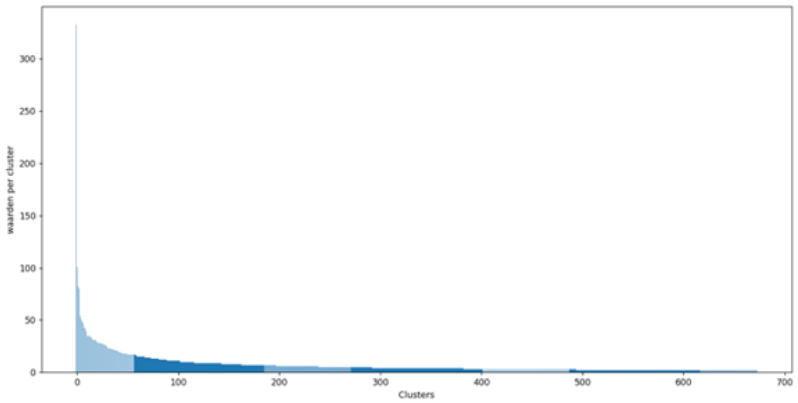
   - Grootste cluster: 263
   - Grootte vuilniscluster: 263
   - Aantal clusters: 174
   - Gemiddeld aantal waarden per cluster: 28,66
   - Silhoutte avg: 0.28
   - 152 , BLACK CHRISTMAS TREE
     63 , PINK AND WHITE CHRISTMAS TREE
     21 , IVORY STRING CURTAIN WITH POLE
     21 , BLUE FELT HANGING HEART W FLOWER
     21 , PINK FELT HANGING HEART W FLOWER
     24 , SMALLFOLKART BAUBLE CHRISTMAS DEC
     24 , FOLKART ZINC HEART CHRISTMAS DEC
     80 , FOLKART CLIP ON STARS



Figuur (3.16)
Resultaten hiërarchisch clusteren voor de derde dataset

4. Tf-idf en dbscan:

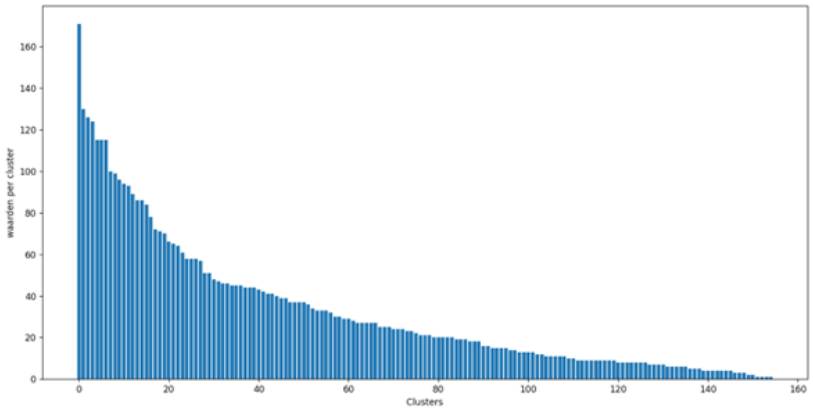
    - Grootste cluster: 333
    - Grootte vuilniscluster: 333
    - Aantal clusters: 673
    - Gemiddeld aantal waarden per cluster: 7,41
    - Silhoutte avg: 0.62
    - 506 , BLACK CHRISTMAS TREE
      506 , PINK AND WHITE CHRISTMAS TREE
      -1 , I VORY STRING CURTAIN WITH POLE
      620 , BLUE FELT HANGING HEART W FLOWER
      620 , PINK FELT HANGING HEART W FLOWER
      507 , SMALLFOLKART BAUBLE CHRISTMAS DEC
      508 , FOLKART ZINC HEART CHRISTMAS DEC
      -1 , FOLKART CLIP ON STARS



Figuur (3.17)
Resultaten tf-idf met dbscan voor de derde dataset

5. Word2vec en K-means:

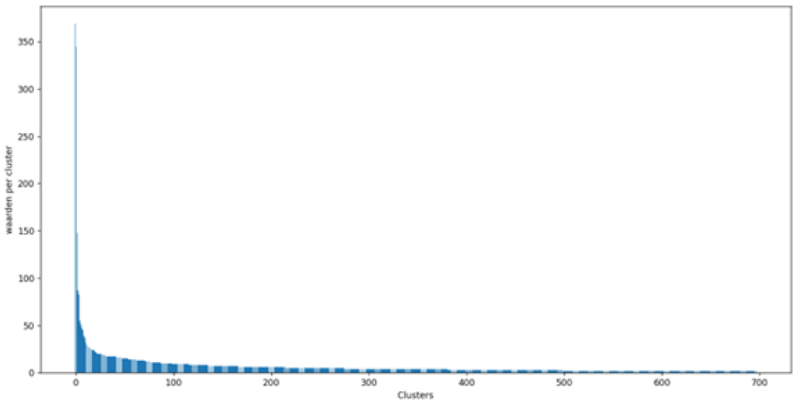
   - Grootste cluster: 168
   - Grootte vuilniscluster: 168
   - Aantal clusters: 155
   - Gemiddeld aantal waarden per cluster: 32,17
   - Silhoutte avg: 0.31
   - 63 , BLACK CHRISTMAS TREE
     24 , PINK AND WHITE CHRISTMAS TREE
     84 , IVORY STRING CURTAIN WITH POLE
     131 , BLUE FELT HANGING HEART W FLOWER
     131 , PINK FELT HANGING HEART W FLOWER
     80 , SMALLFOLKART BAUBLE CHRISTMAS DEC
     80 , FOLKART ZINC HEART CHRISTMAS DEC
     149 , FOLKART CLIP ON STARS



Figuur (3.18)
Resultaten word2vec met K-means voor de derde dataset

6. Word2vec en dbscan:

    - Grootste cluster: 369
    - Grootte vuilniscluster: 369
    - Aantal clusters: 697
    - Gemiddeld aantal waarden per cluster: 7,16
    - Silhoutte avg: 0.64
    - 618 , BLACK CHRISTMAS TREE
      532 , PINK AND WHITE CHRISTMAS TREE
      -1 , IVORY STRING CURTAIN WITH POLE
      655 , BLUE FELT HANGING HEART W FLOWER
      655 , PINK FELT HANGING HEART W FLOWER
      533 , SMALLFOLKART BAUBLE CHRISTMAS DEC
      534 , FOLKART ZINC HEART CHRISTMAS DEC
      -1 , FOLKART CLIP ON STARS



Figuur (3.19)
Resultaten word2vec met dbscan voor de derde dataset

Ook voor de laatste dataset worden alle resultaten met elkaar vergeleken aan de hand van de adjusted rand score.
(Tf-idf, op basis van word, met Kmeans) - (Tf-idf met dbscan): ARI = 0.44
(Tf-idf, op basis van char, met Kmeans) - (Tf-idf met dbscan): ARI = 0.37
(Tf-idf met dbscan) - (Word2vec met dbscan): ARI = 0.33
(Tf-idf, op basis van word, met Kmeans) - (Word2vec met dbscan) ARI = : 0.31
(Tf-idf, op basis van char, met Kmeans) - (Tf-idf, op basis van word, met Kmeans):ARI = 0.23
(Word2vec met Kmeans) - (Word2vec met dbscan): ARI = 0.23
(Tf-idf, op basis van char, met Kmeans) - (Hiërarchisch): ARI = 0.22
(Tf-idf, op basis van char, met Kmeans) - (Word2vec met Kmeans): ARI = 0.21
(Hiërarchisch) - (Tf-idf met dbscan): ARI = 0.21

(Tf-idf, op basis van char, met Kmeans) - (Word2vec met dbscan): ARI = 0.19
(Word2vec met Kmeans) - (Tf-idf met dbscan): ARI = 0.18
(Word2vec met Kmeans) - (Hiërarchisch): ARI = 0.15
(Tf-idf, op basis van word, met Kmeans) - (Hiërarchisch): ARI = 0.14
(Tf-idf, op basis van word, met Kmeans) - (Word2vec met Kmeans): ARI = 0.13
(Hiërarchisch) - (Word2vec met dbscan): ARI = 0.13

Deze gemiddelde's liggen gelukkig een stuk hoger dan die bij de tweede dataset. In tegenstelling tot de vorige datasets, valt er te zien dat dbscan in combinatie met tf-idf hier redelijk goed scoort. De Silhoutte gedmiddelde's zijn: 0.87, 0.64, 0.62, 0.62, 0.31 en 0.28. Dbscan presteert hier op zich wel beter, maar het feit dat meer dan 300 waarden niet aan een cluster kunnen toegevoegd worden, blijft een probleem.
Het valt ook op dat de tf-idf op basis van woorden het beter doet dan diegene op basis van karakters. Dit is echter bedrieglijk, want dit komt voornamelijk omdat hij 1161 clusters gebruikt. Het Silhoutte gemiddelde duidt aan of er veel waarden eerder tot een andere cluster behoren dan degene waarin ze zitten. Aangezien er 1161 clusters zijn, en er in de data redelijk wat identieke waarden zijn, bestaan veel clusters enkel uit waarden die identiek zijn. Dit betekent inderdaad dat de meest waarden in de juiste cluster zitten, maar er zouden eigenlijk meer clusters samengevoegd moeten worden die dicht bij elkaar liggen.

3.6. Resultaten

Voor data in de vorm van lange zinnen, zoals de dataset met informatie over wijnen, is er jammer genoeg geen goede clustermethode gevonden. Deze zinnen bevatten te veel woorden waar eigenlijk geen rekening mee mag gehouden worden. Woorden zoals 'Sauvignon' of 'Merlot', die toch respectievelijk 1160 en 420 keer in de dataset voorkomen, zouden als belangrijker aanschouwd moeten worden. Een goed resultaat zou bijvoorbeeld zijn als alle Merlot wijnen in dezelfde cluster zaten. Dit is echter niet geslaagd.

Data die een stuk korter is en bestaat uit een paar woorden, afmetingen en codes, zoals de dataset met productinformatie die verkregen is uit de praktijk, is een veel beter voorbeeld van hoe ongestructureerde data er meestal uitziet. Voor dit soort data zijn er wel goede methodes gevonden. Zowel Levenshtein distance in combinatie met het hiërarchisch clusteren als tf-idf in combinatie met K-means zijn twee methodes die aangeraden worden.
De resultaten van beide methodes zijn heel gelijkend. Bij K-means valt op dat er een stuk minder clusters zijn waardoor waarden die minder gelijkend zijn, sneller in dezelfde cluster belanden dan bij hiërarchisch. Aangezien bij hiërarchisch clusteren met Levenshtein distance gewerkt wordt, gaan waarden die hetzelfde zijn, maar

met een code of afmeting die een ander aantal karakters heeft, niet meer in dezelfde cluster zitten.
Onderstaande waarden bijvoorbeeld. Bij hiërarchisch zitten ze in verschillende clusters, maar bij K-means behoren ze tot dezelfde cluster.
74 , 0000000 Sinusoidal Filler Pair (Mixed)
73 , 0000000 Sinusoidal Filler Pair WHT
147, 0000000 Sinusoidal Filler WHT

1. K-means: Let hierbij zeker op het gebruik van tf-idf op basis van karakters, anders geeft deze methode in deze situatie heel verschillende resultaten.
   Voordelen:

   - Minder clusters.
   - Waarden moeten maar een beetje gelijkend zijn om in dezelfde cluster te komen.

   Nadelen:

   - Een duidelijke vuilniscluster.
   - Het is wat zoeken naar een goede waarde voor T2 tijdens de canopy clustering.

2. Hiërarchisch clusteren:
   Voordelen:

   - Deze clustering blijft doorgaan tot er één cluster overblijft waarbij het een stuk eenvoudiger is om uit te zoeken wat de beste plaats is om een horizontale streep te trekken in het dendrogram.
   - Waarden in dezelfde cluster zijn steeds zeer gelijkend
   - Geen vuilniscluster: waarden die apart horen zitten apart in een cluster

   Nadelen:

   - Veel clusters.
   - Veel waarden die wel samen horen, maar niet gelijkend genoeg zijn, waardoor ze in verschillende clusters belanden.

Als er geen codes of afmetingen in de data voorkomen, zoals in de dataset met productinformatie uit de retail, zijn er ook een paar succesvolle methodes gevonden. Dbscan was in deze situatie op zich nog bruikbaar. Het slaagde erin heel gelijkende waarden samen te clusteren, maar de andere methodes slaagden daar even goed in en deze konden zelfs minder gelijkende waarden ook in dezelfde cluster steken.

Dbscan komt hierdoor uit dit onderzoek als de minst succesvolle methode om ongestructureerde masterdata te clusteren.
Vervolgens blijven er nog drie methodes over. Wanneer naar de resultaten gekeken wordt, zouden de *black christmas tree* en de *pink and white christmas tree* liefst in dezelfde cluster zitten. Dit is slechts in één van de drie methodes geslaagd. K-means in combinatie met tf-idf komt ook hier naar boven als de beste cluster methode. De data is goed geclusterd, de clusters zijn relatief klein, gemiddeld slechts 8.55 waarden per cluster, maar de waarden erin passen beter bij elkaar dan in de clusters van het hiërarchisch clusteren of die van K-means in combinatie met word2vec.
Er is ook nog steeds een vuilniscluster met 108 waarden in. Dit is jammer, maar aan de andere kant is dit waarschijnlijk wel normaal, er zijn altijd waarden die totaal verschillend zijn van al de rest. Bij de andere twee methoden bevatte de vuilniscluster 168 en 263 waarden, dit zijn er dus nog meer. Tijdens de requirements analyse werd bepaald dat de vuilniscluster maximum 10% van alle waarden mocht bevatten, in dit geval dus maximum 500 waarden. Dit is voldaan bij alle drie deze methodes.

4

Conclusie

Uit de resultaten van de vergelijkende studie kan geconcludeerd worden dat K-means op alle vlakken de beste clustermethode is voor ongestructureerde master-data. De methode werkt door eerst de data om te zetten in vectoren aan de hand van tf-idf, hierbij worden de strings eerst in n-grammen gesplitst met een range van 2 tot 4 karakters. Vervolgens wordt een canopy clustering uitgevoerd op deze lijst van vectoren zodat er kan bepaald worden hoeveel clusters er nodig zijn. Tot slot clustert het K-means algoritme de dataset op basis van dit aantal clusters en de vectoren van tf-idf.

Uit dit onderzoek blijkt ook dat dbscan helemaal niet geschikt is om aan de hand van tf-idf of word2vec ongestructureerde masterdata te clusteren. Hiërarchisch clusteren of K-means combineren met word2vec bleek dan weer wel in staat de data succesvol te clusteren, maar toch niet zo goed als de combinatie tussen K-means en tf-idf.

Er is echter toch nog een grote vraag die overblijft, aangezien er geen manier gevonden is om ongestructureerde masterdata in de vorm van lange zinnen succesvol te clusteren. Zijn er misschien andere methodes die over het hoofd gekeken zijn? Ook is er in dit onderzoek niet dieper ingegaan op het taggen van de clusters zodat er kan geweten zijn waarom de waarden in een cluster samen horen. Dit is bij deze een uitnodiging tot verder onderzoek.

A

Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

Samenvatting

Het beheren van ongestructureerde masterdata is een probleem waar veel bedrijven mee kampen. Het is heel lastig om structuur te brengen in zulke tabellen. In dit onderzoek wordt een nieuwe manier vergeleken met bestaande manieren. Door middel van clustering en tagging zal gelijkaardige data gegroepeerd worden waarna het eenvoudiger moet zijn om aan data-extractie te doen. Er zullen requirements opgesteld worden voor de algoritmes om zo te kijken welke het best geschikt is voor deze probleemstelling. Er zal vooral vergeleken worden op vlak van accuraatheid, recall, precisie en snelheid. Aan de hand van de resultaten zal er een proof-of-concept opgesteld worden om op de efficiëntste manier aan data-extractie te doen in ongestructureerde masterdata. Er wordt verwacht dat de nieuwe methode, die gebruikt maakt van clustering en tagging, als meest geschikte algoritme voor deze doelstelling uit deze studie naar voor zal komen.

Veel bedrijven hebben wat men noemt 'een vuilnistabel'. Dit is een tabel met ongestructureerde masterdata. Hierin staan allerlei gegevens door elkaar, waardoor het lastig wordt om iets te doen met deze data. In dit onderzoek zal er gekeken worden naar de verschillende mogelijkheden om data-extractie te verbeteren in zo'n tabellen. Data-extractie of gegevensextractie slaat op het ophalen van relevante gegevens uit een tabel of databank (Encyclo.nl, g.d.). Aan de hand van dit onderzoek zullen bedrijven hun vuilnistabellen veel beter kun gebruiken, of zelfs sorteren, zonder alles handmatig uit te voeren.

In dit onderzoek zal enkel master data onderzocht worden. Master data zijn de

gegevens van een bedrijf die niet transactioneel zijn. Dit betekent dat de data niet te maken heeft met transacties zoals het plaatsen van orders bijvoorbeeld. Master data zal wel veranderen in de loop der tijd, maar dit gebeurd zeer langzaam. Eén van de voornaamste voorbeelden hiervan zijn productgegevens: de naam, afmetingen, prijs... Deze gegevens gaan enkel aangepast worden in uitzonderlijke gevallen (Yellowground, g.d.).
Er zijn twee mogelijkheden om deze data op te slaan: gestructureerd of ongestructureerd. Gestructureerde data is georganiseerd en gegroepeerd zodat alle verschillende gegevens in een aparte tabel zitten. Als dit niet het geval is en er bestaat slechts één tabel met daarin allerlei gegevens, is er sprake van ongestructureerde data. Een voorbeeld hiervan is een tabel met de naam, prijs, afmetingen, beschrijving van het product allemaal tezamen (Seagate, g.d.).
Het grote probleem met ongestructureerde master data is het feit dat het verbeteren van de datakwaliteit een stuk moeilijker wordt. Zoeken of sorteren in zo'n tabel is heel moeilijk, ook duplicaten onderscheiden is niet gemakkelijk. Daarnaast kan het soms zelfs lastig worden om de gewenste data op te halen uit zo'n tabel.
In dit onderzoek zullen er verschillende mogelijkheden vergeleken worden om gelijkaardige waarden uit een tabel te vinden, beter gekend als fuzzymatching (Silva, 2022). Er bestaan al een heleboel technieken en metrieken hiervoor, maar wat als we de tabel eerst gaan onderverdelen in verschillende groepen (Clustering) waarbij het de bedoeling is dat elke groep iets gelijkaardigs heeft zodat er een label aan iedere groep kan gegeven worden (Tagging). De methode clustering + tagging zal worden onderzocht en vergeleken met bestaande algoritmes die gebruik maken van andere technieken.
De vergelijkende studie zal uitgevoerd worden aan de hand van aan tabel met ongestructureerde masterdata afkomstig uit een bestaand bedrijf. Met behulp van libraries in Python zal er vergeleken worden op basis van de volgende zaken: de accuraatheid, de snelheid, de precisie (hoeveelheid van de gevonden duplicaten die correct zijn), de recall (hoeveelheid van de aanwezige duplicaten is gevonden) (GoogleDevelopers, 2022).

A.1. State-of-the-art

Dit onderzoek zal uitgevoerd worden aan de hand van machine learning. Er bestaan drie verschillende types machinelearning-algoritmen. Als eerste is er het gesuperviseerd leren: hierbij zal er aan iedere waarde, de identiteit al worden meegegeven. Simpel voorbeeld: als er een algoritme gesuperviseerd wordt getraind om katten te herkennen op foto's, dan zal iedere foto die wordt getoond aan het algoritme een label bevatten met 'ja' of 'nee'. Als het algoritme dan goed genoeg getraind is, zal het in staat zijn om foto's zonder label, het juiste label te geven. Het tweede type is ongesuperviseerd leren. Dit is logischerwijs het omgekeerde van gesuperviseerd leren. In ons simpel voorbeeld van de katten, zal het algoritme een grote dataset

met foto's ontvangen. Deze zal dan in die set naar iets zoeken dat overeenkomt in veel van de foto's. Deze overeenkomst wordt in de ogen van het algoritme de 'kat'. Vervolgens splitst het de foto's in twee groepen. Eén die de 'kat' wel bevat en één die hem niet bevat. Aangezien er geen labels hangen aan de dataset, kan het algoritme zelf niet weten of het juist of fout is. Dit zorgt ervoor dat er minder controle is over de uitkomst, waardoor deze methode minder succesvol is. Zo kan het natuurlijk gebeuren dat veel foto's, naast een kat, ook een boom bevatten. Hierdoor kan het gebeuren dat het algoritme de foto's splitst in een groep met boom en zonder boom, want het algoritme kan niet weten hoe een kat eruitziet, hierdoor denkt het misschien dat een boom eigenlijk een kat is. Waarom zou ongesuperviseerd leren dan gebruikt worden als het minder succesvol is? In het voorbeeld van de katten is inderdaad niet zo moeilijk om een label 'ja' of 'nee' aan iedere foto te geven, maar in heel veel andere gevallen kunnen er geen labels aan de data gegeven worden en zou het enorm handig zijn als het algoritme gewoon de data zonder labels in verschillende groepen zou verdelen. Dit geeft al veel inzicht in de data. Het derde type is reinforcement leren. Dit wordt veel gebruikt bij situaties waarin het algoritme een juiste volgorde van acties moet uitvoeren. Iedere keer dat het algoritme correct is, wordt het beloont. Deze methode wordt ook gebruikt voor het aanleren van gedrag bij dieren. Als een hond getraind wordt om te zitten bijvoorbeeld, krijgt hij een snoepje of aaitje iedere keer hij het commando goed uitvoert.
In dit onderzoek wordt gebruik gemaakt van een ongestructureerde masterdata. Dit betekent dat de data geen labels bevat. De bedoeling is om de data te onderverdelen in gelijkaardige groepen. Het is dus vanzelfsprekend dat dit onderzoek zal gebeuren aan de hand van ongesuperviseerd leren.
Nu er beslist is dat er gebruik gemaakt zal worden van ongesuperviseerd leren, komt een volgend probleem naar boven. Onze ongestructureerde masterdata bevat woorden en karakters, maar de algoritmen kunnen dit niet groeperen. Deze verwachten de data in getallen. Een eerste stap in dit onderzoek, is dus het omzetten van de data in getallen of vectoren, dit zijn enorm grote reeksen getallen. Er zijn verschillende manieren om dit te doen.

A.1.1. Levenshtein distance

Een eerste mogelijkheid is om gebruik te maken van de 'Levenshtein distance'. Deze metriek bepaalt de afstand tussen twee strings aan de hand van het aantal uit te voeren operaties. Deze operaties zijn: een letter toevoegen of verwijderen en een letter veranderen. Hoe meer operaties er moeten uitgevoerd worden, hoe groter de afstand tussen twee strings

A.1.2. word2vec

Waar met de Levenshtein distance nog geen rekening mee wordt gehouden, is de semantiek van de woorden. 'Frigo' en 'Koelkast' bijvoorbeeld, gelijken qua karakters

totaal niet op elkaar, maar betekenen eigenlijk wel hetzelfde. Om de similariteit van deze strings te berekenen, wordt elk woord omgezet in een vector via het algoritme word2vec. Vervolgens wordt de cosinusgelijkheid van de vectoren berekend om de afstand tussen de twee woorden te bepalen

A.1.3. tf-idf

Een derde mogelijkheid is tf-idf, dit staat voor term frequency-inverse document frequency. Eerst en vooral hebben we term frequency. Dit is het aantal keer dat een woord voorkomt in het document. Dit kan berekend worden door gewoonweg te tellen hoeveel keer het voorkomt, maar als de documenten maar een paar worden bevatten, zoals in ongestructureerde masterdata toch vaak het geval is, kan dit ook berekend worden aan de hand van een boolean. Het wordt dus 1 als het document het woord bevat en 0 als het woord er niet in voorkomt. Vervolgens is er de inverse document frequency. Dit is een berekening van hoe zeldzaam het woord is ten opzichte van alle documenten in de dataset. Dit is noodzakelijk, want anders zullen woorden zoals 'het', 'een', 'is' of 'er' altijd naar boven komen als veelgebruikte woorden. De bedoeling is dus om de woorden die niet veel voorkomen in alle documenten, maar wel regelmatig in een bepaald document voorkomen, een hoge score te geven. De idf voor een bepaald woord (t) in een dataset (D) wordt als volgt berekend: de logaritme van N, waarbij N gelijk is aan het aantal documenten (d), gedeeld door het aantal documenten waarin het woord (t) voorkomt.

Figuur (A.1)

$$idf(t, D) = log\left(\frac{N}{count(d \in D : t \in d)}\right)$$

Wanneer de tf en de idf berekend zijn, moeten deze nog samengevoegd worden natuurlijk. Dit wordt simpelweg gedaan door de twee waarden te vermenigvuldigen. Hoe hoger de uitkomst is, hoe belangrijker het woord is, als de uitkomst dicht bij 0 ligt is dit woord totaal niet belangrijk.

A.1.4. Clustering

Nu alle data uit de dataset omgezet zijn in vectoren, kan er begonnen worden met clusteren. Hiervoor zullen drie verschillende methoden gebruikt worden. K-means, Hierarchisch en dbscan.

K-means

Het K-means clustering algoritme wordt gebruikt om te clusteren in een tabel met ongestructureerde data. Er moet op voorhand opgegeven worden hoeveel clusters er nodig zijn, dit is het natuurlijke getal K. Vervolgens worden er K zwaartepunten (centers) geïnitialiseerd op een random positie. Daarna wordt er voor alle waarden

berekend bij welk zwaartepunt ze het dichtste liggen en aan die cluster toegevoegd. Vervolgens start een iteratief proces waarbij voor iedere cluster het zwaartepunt wordt verlegt naar het middelpunt. Nu wordt opnieuw voor iedere waarde in de tabel berekend bij welk zwaartepunt ze het dichtst ligt en zo worden opnieuw clusters gevormd. Dit iteratief proces blijft doorgaan totdat de zwaartepunten niet meer verplaatsen en de uiteindelijke clusters gevormd zijn. De dataset zal in K clusters verdeeld zijn. Een groot nadeel van dit algoritme is dat het getal K op voorhand moet vastliggen, terwijl het juist zeer belangrijk is dat dit getal goed gekozen wordt. Hiervoor is er ook een oplossing, namelijk gebruik maken van canopy clustering. Dit is ook een clustering algoritme dat zeer snel uitgevoerd wordt, maar niet zo accuraat is. Wat wel positief is aan dit algoritme, is dat hier K niet op voorhand moet bepaald worden. Hierdoor kunnen we eerst de canopy clustering uitvoeren en aan de hand van de K die hier gebruikt wordt, het K-means algoritme uitvoeren

dbscan

Dbscan is de afkorting voor Density-based spatial clustering of applications with noise en dit is een clustering algoritme dat ontstaan is om nested clusters, goed te kunnen berekenen. Dit zijn clusters die gedeeltelijk of zelfs helemaal omringt zijn door een andere cluster. Dit is het algoritme dat het dichtst in de buurt komt met hoe het menselijk oog clustert. Dbscan gaat als volgt te werk: eerst telt het voor iedere waarde, het aantal punten die dicht in de buurt liggen. Dit wordt simpelweg gedaan door een cirkel te trekken rond het punt en vervolgens alle punten die volledig of gedeeltelijk in de cirkel liggen te tellen. De straal van de cirkel is één van de twee parameters die op voorhand zal moeten meegegeven worden. De tweede parameter is het aantal dichte buren een punt nodig heeft om een Core Point te worden. Stel dat dit aantal vier is, dan worden alle punten die vier of meer punten in hun cirkel hebben een Core Point. Dit zijn meestal de punten die in het midden van een cluster liggen. Vervolgens start het algoritme bij een willekeurig Core Point en dit wordt aan de eerste cluster toegevoegd. Daarna worden zijn buren bekeken en alle Core Points daarvan worden ook aan de cluster toegevoegd. Voor ieder toegevoegd punt, wordt deze stap uitgevoerd, tot er geen meer bij kunnen. Vervolgens worden voor ieder punt uit de cluster, zijn buren die geen Core Points zijn ook toegevoegd, maar hun buren, die niet meer. Dit is de eerste cluster, nu wordt er terug gestart bij een willekeurig Core Point die nog niet tot een cluster behoort. Op het einde, als er geen Core Points meer overblijven, kan het zijn dat er nog punten zijn die niet tot een cluster behoren. Deze zijn de uitschieters.

Bibliografie

Azavea (Red.). (2019, augustus 26). *affine-gap-distance-example*. https://www.azavea.com/blog/2019/08/30/using-the-dedupe-machine-learning-library-for-cleaning-and-matching-data/affine-gap-distance-example/

Chen, D., Sain, S. L., & Guo, K. (2012, augustus 27). *Online Retail Data Set: Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining* (UCI, Red.). Database Marketing en Customer Strategy Management.

Cuelogic (Red.). (2017, januari 25). *The Levenshtein Algorithm*. https://www.cuelogic.com/blog/the-levenshtein-algorithm

Encyclo.nl (Red.). (g.d.). *Data extractie definities*. https://www.encyclo.nl/begrip/data_extractie

GoogleDevelopers (Red.). (2022, juli 18). *feedbackClassification: Precision and Recall*. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Includehelp (Red.). (g.d.). *ASCII Character Table*. https://www.includehelp.com/ascii-table.aspx/

Iv, C. (2018, augustus 7). *winemag-data-130k* (Kaggle, Red.). https://www.kaggle.com/datasets/christopheiv/winemagdata130k

Kaplan, D. (2022, augustus 6). *K-Means Accuracy Python With Silhouette Method* (EML, Red.). https://enjoymachinelearning.com/blog/k-means-accuracy-python-silhouette/?utm_content=cmp-true

Liddy, E. D. (2001). Natural Language Processing. *In Encyclopedia of Library and Information Science*. https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub

Lievens, S. (2022, mei 30). *Achterliggende uitleg bij Duplicaatdetectie* (AI-Assisted, Red.). http://ai-assisted-mdm.be/node/32

Mahout (Red.). (g.d.). *canopy*. https://mahout.apache.org/docs/latest/algorithms/clustering/canopy/

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, *18*, 544–551. https://doi.org/10.1136/amiajnl-2011-000464

Postma, E. (2019). *AI in de digitalesamenleving: Kwaliteit van algoritmenen besluitvorming*. https://ericpostma.nl/publications/AI_Postma2019NL.pdf

ProductPlan (Red.). (g.d.). *MoSCoW Prioritization*. https://www.productplan.com/glossary/moscow-prioritization/

Santos, I., Penya, Y. K., Devesa, J., & Bringas, P. G. (2009). *N-grams-based file signatures for malwaredetection* [onderzoeksrap.]. S3Lab, Deusto Technological Foundation, Bilbao, Spain. https://www.scitepress.org/PublishedPapers/2009/18636/18636.pdf

Seagate (Red.). (g.d.). *Gestructureerde of ongestructureerde gegevens*. https://www.seagate.com/be/nl/blog/structured-vs-unstructured-data/

ShareTechnote (Red.). (g.d.). *Communication Technology*. https://www.sharetechnote.com/html/Handbook_Communication_HammingDistance.html

Silva, E. (2022, juli 15). *What is Fuzzy Matching?* (Redis, Red.). https://redis.com/blog/what-is-fuzzy-matching/

Simha, A. (2021, oktober 6). *Understanding TF-IDF for Machine Learning* (CapitalOne, Red.). https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/

Starmer, J. (2017, juni 20). *StatQuest: Hierarchical Clustering*. https://www.youtube.com/watch?v=7xHsRkOdVwo

Starmer, J. (2022, januari 10). *Clustering with DBSCAN Clearly Explained!!!* https://www.youtube.com/watch?v=RDZUdRSDOok

Starmer, J. (2023, maart 13). *Word Embedding and Word2Vec, Clearly Explained!!!* https://www.youtube.com/watch?v=viZrOnJclY0

Vandenbussche, B. (2016, juni 1). *Verbeterde aanbevelingssystemen op basis van content herkenning in tekst* [masterscriptie, UGent].

Walgran, J. (2019, augustus 30). *Using the Dedupe Machine Learning Library for Cleaning and Matching Data* (Azavea, Red.). https://www.azavea.com/blog/2019/08/30/using-the-dedupe-machine-learning-library-for-cleaning-and-matching-data/

Wikiversity (Red.). (2022, maart 5). *Duplicate record detection*. https://en.wikiversity.org/wiki/Duplicate_record_detection

Yellowground (Red.). (g.d.). *Wat is masterdata?* https://yellowground.eu/wat-is-master-data/