KU LEUVEN

FACULTY OF
ENGINEERING SCIENCE

Adaptieve tensor factorisaties om versneld tijdreeksen te clusteren

Tuur Vanhoof

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen, hoofdoptie
Artificiële intelligentie

Promotoren:

Dr. ir. W. Meert
Prof. dr. R. Vandebril
Prof. dr. ir. H. Blockeel

Evaluatoren:

Dr. V. Vercruyssen
Prof. dr. N. Vannieuwenhoven

Academiejaar 2022 – 2023

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotoren als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail `info@cs.kuleuven.be`.

Voorafgaande schriftelijke toestemming van de promotoren is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Aan de lezer van de thesis "Adaptieve tensor factorisaties om versneld tijdreeksen te clusteren":

In wat volgt leest u het onderzoek naar en de resultaten van het werken met lage-rang benaderingen van tensoren om efficiënter tijdreeksen te gaan clusteren. Dit werk is gemaakt als opleidingsonderdeel voor het behalen van de Master in de Ingenieurswetenschappen: Computerwetenschappen.

Het maken van deze thesis heeft me voor veel nieuwe uitdagingen gesteld, van het verwerken van complexe papers tot het implementeren van algoritmes op wiskundige structuren als tensoren. Ik ben blij dat ik deze ervaringen heb mogen opdoen en uit alle uitdagingen heb kunnen leren, terwijl ik me verdiepte in het heel actuele en interessante onderwerp van het verwerken van grote hoeveelheden data. Deze thesis is er niet vanzelf gekomen en daarom wil ik een aantal mensen bedanken die mij geholpen hebben om dit tot een goed einde te brengen.

Eerst en vooral wil ik de promotoren Wannes Meert en Raf Vandebril bedanken voor de begeleiding doorheen het jaar en het helpen met het verwerven van inzicht in alle wiskundige concepten. Verder wil ik ook mijn ouders bedanken voor de kansen en motivatie die zij mij hebben gegeven doorheen de jaren aan de universiteit. Tenslotte wil ik mijn vriendin en de rest van de familie bedanken voor alle steun, en om mij steeds te aanhoren over alles omtrent matrices, tensoren en benaderingen, zelfs zonder ook maar een idee te hebben van waar het eigenlijk over ging.

Ik hoop dat u na het lezen van deze thesis wel helemaal mee bent in die concepten en het gebruik ervan om tijdreeksen te clusteren. Veel leesplezier.

Tuur Vanhoof

Inhoudsopgave

Samenvatting

Data verzamelen en opslaan is in deze tijd de normaalste zaak van de wereld. Het verwerken en analyseren van de massale hoeveelheden data daarentegen is een zeer actueel probleem. De soort data die deze thesis bekijkt, is data in de vorm van tijdreeksen. Een tijdreeks ontstaat door een variabele te meten doorheen de tijd en is te vinden in en bruikbaar voor een zeer uitgebreid aantal toepassingen.

Het verwerken van tijdreeksen gebeurt in deze thesis door ze onderling met elkaar te vergelijken en te clusteren. Op deze manier worden sterk gelijkende tijdreeksen in dezelfde cluster geplaatst. Clusteralgoritmes als spectraal clusteren en k-medoids werken op afstandsmatrices. Doordat we werken met multi-sensor data in de tijdreeksen, wordt de standaard aanpak met afstandmatrices hier vervangen door het werken met een afstandstensor. Dit resulteert in een aanpassing van het spectraal cluster algoritme, waar we rechtstreeks clusteren met de feature vectoren, bekomen door de afstandstensor te gaan benaderen. De lage-rang benadering van de tensor wordt opgesteld met twee versies van het ACA-T algoritme, Adaptive Cross Approximation voor tensoren.

De methodes baseren zich op twee verschillende decomposities van een tensor. Methode 1 werkt met frontale matrices en tube vectoren, terwijl methode 2 ontbindt in enkel vectoren. Dit geeft een afweging tussen het aantal te berekenen DTW-afstanden en de nauwkeurigheid van de benadering. Beiden zijn iteratieve algoritmes die zich focussen op het benaderen van de grootste elementen in de tensor, om zo het verschil tussen de benadering en de originele tensor zo klein mogelijk te krijgen in zo min mogelijk iteraties.

Deze thesis toont het onderzoek naar en de resultaten van de twee methodes om een tensor te gaan benaderen. We tonen aan dat de eerste methode de relatieve fout ten opzichte van de volledige tensor zeer klein kan krijgen. Hoe kleiner deze fout, hoe meer het aantal DTW berekeningen voor de benadering toeneemt, tot bijna evenveel als de volledige tensor. De tweede methode zal een grotere relatieve fout hebben, maar zal tot wel 50 keer minder DTW berekeningen nodig hebben dan de volledige tensor op te stellen. Er wordt aangetoond dat desondanks de grotere relatieve fout toch gelijkende clusters worden gevonden ten opzichte van een decompositie uit de volledige tensor.

Lijst van figuren en tabellen

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en symbolen

Afkortingen

ACA	Adaptive Cross Approximation
ACA-T	Adaptive Cross Approximation voor tensoren
ALS	Alternating Least Squares
ARI	Adjusted Rand Index
CP	Candecomp/Parafac
DTW	Dynamic Time Warping
EA	Euclidische afstand
FSTD	Fiber Sampling Tensor Decomposition
RI	Rand Index
SVD	Singular Value Decomposition

Symbolen

$\underline{\mathbf{T}}$	Tensor
$\mathbf{M}$	Matrix
$\mathbf{v}$	vector

Hoofdstuk 1

Inleiding

Tegenwoordig zijn er overal camera's, sensoren en andere meettoestellen aanwezig om alles naar wens op te meten. De data uit deze metingen kan gebruikt worden voor talloze doeleinden. Sportende personen kunnen bijvoorbeeld tijdens hun oefeningen een aantal sensoren op hun lichaam dragen. Deze sensoren kunnen de bewegingen van het lichaamsdeel waarop ze geplaatst zijn meten gedurende bepaalde oefeningen, resulterende in een tijdreeks die de beweging afzet tegen de tijd. De data die hierdoor wordt gegenereerd kan bijvoorbeeld gebruikt worden om oefeningen te herkennen of automatisch feedback te geven over de uitvoering van de oefening.

Het continu meten van alle oefeningen van verschillende personen zal helaas een probleem met zich meebrengen. Er zal namelijk een massale hoeveelheid data worden gegenereerd. De afgelopen jaren zijn de opslagcapaciteiten en manieren van opslaan zo gevorderd dat dit probleem als het ware opgelost is geraakt. Het probleem nu is dat deze grote hoeveelheid data efficiënt verwerkt en geanalyseerd moet worden alvorens ze bruikbaar is voor forecasting, patroonherkenning of dergelijke toepassingen.

Een manier om inzicht te krijgen in een grote hoeveelheid tijdreeksen is door ze te gaan clusteren. Clusteren is een techniek in machine learning waarmee de tijdreeksen worden gegroepeerd op basis van hun overeenkomsten in de dataset. Het doel van clusteren is om gelijkaardige objecten in dezelfde groep (cluster) te plaatsen, terwijl objecten die sterk van elkaar verschillen in verschillende clusters worden ondergebracht. Door bijvoorbeeld het clusteren van de tijdreeksen van sporters kunnen personen met elkaar vergeleken worden of van elkaar onderscheiden worden. Het kan ook bruikbaar zijn op vlak van oefeningen: oefeningen die sterk op elkaar gelijken kunnen zo worden gevonden, of juist afwijkingen van wat correct uitgevoerde oefeningen zouden moeten zijn.

Om te kunnen clusteren, zal deze thesis vertrekken van afstandstensoren opgebouwd uit tijdreeksen, door ze paarsgewijs met elkaar te gaan vergelijken. Deze vergelijking zal gebeuren aan de hand van een afstandsfunctie zoals de *Euclidische Afstand* (EA) of *Dynamic Time Warping* (DTW). Een volledige afstandstensor gaan

opstellen zal, zeker met behulp van DTW, veel berekeningen vereisen. DTW heeft namelijk een kwadratische tijdscomplexiteit ($\mathcal{O}(l^2)$ op de lengte l van de vergeleken tijdreeksen. Daarom zullen in deze thesis twee manieren worden besproken om de tensor te gaan benaderen zonder de volledige tensor te moeten opstellen. Deze lage-rang benadering van de tensor zal ook gebruikt kunnen worden om rechtstreeks mee te gaan clusteren, in plaats van de clustering te baseren op een afstands- of gelijkenismatrix. De fibers die de tensor benadering oplevert zullen gebruikt kunnen worden als feature vectoren voor het k-means cluster algoritme.

Voor de lage-rang benadering baseren we ons op de Adaptive Cross Approximation (ACA) voor matrices. Deze methode wordt veralgemeend naar ACA-T voor tensoren. Hiervoor worden twee nieuwe algoritmes geïtroduceerd die de tensor gaan ontbinden in combinaties van vectoren en/of matrices.

Het doel van de thesis is om te gaan kijken of deze lage-rang benaderingen gebruikt kunnen worden als vervanging voor de volledige tensor. Hiervoor wordt er gekeken naar het aantal berekeningen nodig om een goede benadering op te stellen van de tensor. Verder zal er worden gekeken of er met de benadering kan worden gewerkt om te gaan clusteren en of deze clusteringen een waardevol resultaat opleveren. Tenslotte wordt er de afweging gemaakt tussen de kwaliteit van de benadering en de clustering ten opzichte van de volledige tensor en het aantal berekeningen dat er minder moeten gebeuren door de benaderin te gebruiken.

De thesis start in Hoofdstuk 2 met de nodige achtergrond te verschaffen over tijdreeksen en de bijhorende afstandsfuncties, gevolgd door relevante notaties, opbouw van de afstandsmatrices en -tensoren en de clusteralgoritmes. Hoofdstuk 3 zal gerelateerd werk bespreken en Hoofdstuk 4 volgt daarop met de introductie van de probleemstelling en bijhorende onderzoeksvragen. De twee methodes, gebruikt om tensoren te gaan benaderen, worden uitgebreid besproken in Hoofdstuk 5, waarna de experimenten met deze algoritmes uiteen worden gezet in Hoofdstuk 6. Alles wordt tenslotte nog eens samengevat in Hoofdstuk 7, het besluit.

Hoofdstuk 2

Achtergrond

Dit hoofdstuk start de thesis met achtergrondinformatie over tijdreeksen in Sectie 2.1, gevolgd door het verwerken van tijdreeksen naar afstandsmatrices en -tensoren in Sectie 2.2. In 2.2.1 worden er eerst een aantal belangrijke notaties en begrippen toegelicht, waarna in 2.2.2 de dataset wordt toegelicht die zal resulteren in een *leading example* doorheen de thesis. Het hoofdstuk sluit af met Sectie 2.3 over manieren om vanuit de verkregen tensoren te gaan clusteren.

2.1 Tijdreeksen

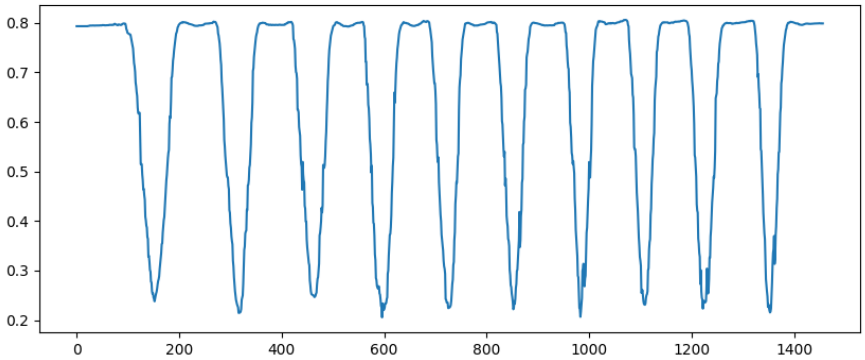
De data, gebruikt in deze thesis is afkomstig van een camera. Deze camera filmt personen terwijl ze een oefening uitvoeren, resulterend in een opeenvolgende reeks van waarden t_i, geobserveerd over een bepaalde tijd l. Dit worden tijdreeksen genoemd en zullen worden genoteerd als

$$T = t_1, t_2, ..., t_{l-1}, t_l \tag{2.1}$$

waarbij l ook de lengte van de tijdreeks aangeeft.

Een tijdreeks bevat informatie over één bepaalde waarde doorheen de tijd en kan worden gevonden in veel verschillende domeinen. Het financiële en zakelijke domein (bijvoorbeeld tijdreeksanalyse in [5]) en het medische domein (weergave van elektrocardiogram als tijdreeks [8]) zijn hier voorbeelden van. In deze thesis gebruiken we data afkomstig uit de AMIE dataset [6]. Met een Kinect camera werden 25 gewrichten van een persoon gedurende tien uitvoeringen van een bepaalde oefening gefilmd. De tijdreeks, ontstaan door het nemen van de beweging van het hoofd volgens de y-as tijdens het uitvoeren van een squat beweging, is weergegeven in Figuur 2.1. De dataset wordt verder besproken in Sectie 2.2.2.

Om dit soort tijdreeksen te gaan clusteren, hebben we een manier nodig om de tijdreeksen met elkaar te vergelijken. Dit zal gebeuren met behulp van een afstandsmaat, waarvan er hieronder twee worden besproken.



Figuur 2.1: Beweging van het hoofd volgens de y-as tijdens uitvoeren van een squat.

2.1.1 Afstandsmaten

Euclidische afstand

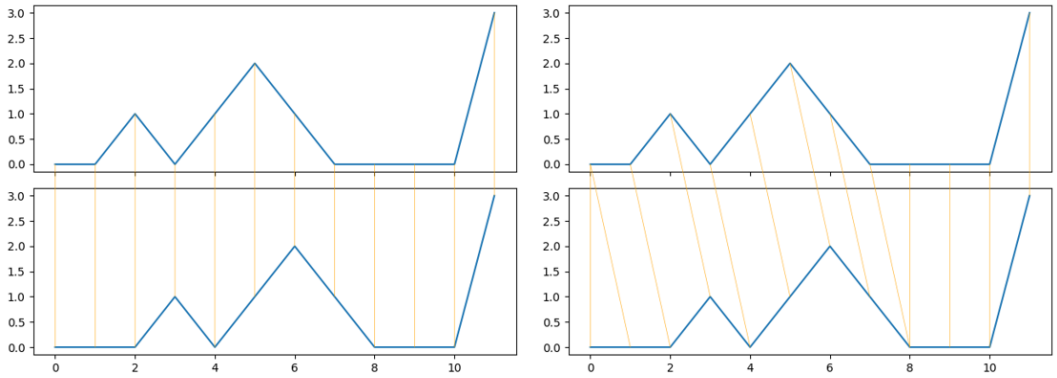
De Euclidische Afstand (EA) is een veelgebruikte afstandsmaat, zowel tussen punten als tussen vectoren. Wanneer de EA wordt berekend tussen twee tijdreeksen, worden stapsgewijs de twee punten in gelijkende tijdstappen één-op-één met elkaar vergeleken. Om al deze punten met elkaar te vergelijken, moet de volledige tijdreeks van lengte l worden doorlopen, resulterend in tijdscomplexiteit $\mathcal{O}(l)$. Echter, twee gelijkend ogende tijdreeksen kunnen toch een grote Euclidische afstand ten opzichte van elkaar hebben, zoals bijvoorbeeld in figuur 2.2. Dit komt door een kleine verschuiving in de tijd volgens de x-as, die bijvoorbeeld kan veroorzaakt worden door een kleine storing in een sensor waardoor deze één tijdsperiode later start. Om dit probleem te vermijden, wordt DTW gebruikt, dat desondanks een tijdsverschuiving er in slaagt de gelijkheid tussen twee reeksen beter te meten.

Dynamic Time Warping

Het DTW algoritme [17] zoekt naar de optimale *many-to-many* alignering tussen twee tijdreeksen, waarbij de effecten van verschuiving en vervorming worden geminimaliseerd. Gegeven twee tijdreeksen $T_1 = x_1, x_2, ..., x_{l_1-1}, x_{l_1}$ en $T_2 = y_1, y_2, ..., y_{l_2-1}, y_{l_2}$ van respectievelijke lengte l_1 en l_2 vindt DTW de optimale oplossing in $\mathcal{O}(l_1 l_2)$ tijd. Het algoritme gaat als volgt te werk:

1. Alle paarsgewijze afstanden worden berekend tussen T_1 en T_2 en deze worden opgeslagen in de kostmatrix $\mathbf{K}$ van grootte $l_1 \times l_2$. Elk element $k_{i,j}$ wordt berekend als

$$k_{i,j} = |T_2(j) - T_1(i)| + min[k_{i-1,j}, k_{i,j-1}, k_{i-1,j-1}] \tag{2.2}$$



FIGUUR 2.2: EA (links) en DTW (rechts) tussen twee gelijk ogende tijdreeksen. Gemaakt met [13]

en waarbij de eerste rij $\mathbf{k}_{1,:}$ en kolom $\mathbf{k}_{:,1}$ worden ingevuld als

$$k_{1,j} = \sum_{k=1}^{j} |y_k - x_1| \text{ en } k_{i,1} = \sum_{k=1}^{i} |y_1 - x_k|. \tag{2.3}$$

Het opstellen van de kostmatrix heeft dus een complexiteit van $\mathcal{O}(l_1 l_2)$.

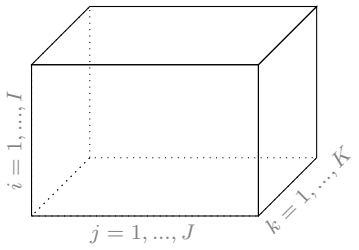
2. In de kostmatrix $\mathbf{K}$ wordt de optimale alignering gezocht. Deze kan worden gevonden door het pad te nemen doorheen de matrix met de laagste kost. Hiervoor kan een *greedy* algoritme met *backtracking* worden gebruikt dat start vanuit k_{l_1,l_2} en eindigt in $k_{1,1}$. Dit heeft een complexiteit van $\mathcal{O}(l_1 + l_2)$.

Een voorbeeld van DTW en de optimale alignering is weergegeven rechts in Figuur 2.2. De totale complexiteit van DTW is dus $\mathcal{O}(l_1 l_2)$ en stijgt evenredig met de lengte van beide tijdreeksen (kwadratische stijging wanneer $l_1 = l_2$). Aangezien tijdreeksen vaak over lange periodes worden gemeten, kan deze kost hoog oplopen en geeft dit een belangrijke factor om rekening mee te houden bij het gebruik van DTW.

Een belangrijke notie is dat DTW, zowel hier als in de meeste literatuur, een *afstand* wordt genoemd, terwijl DTW theoretisch gezien geen afstand is. Om een afstandsfunctie te zijn, moet de functie voldoen aan de volgende vier axioma's:

$$\begin{aligned}
&\text{Niet-negativiteit: } d(X,Y) \geq 0 \\
&\text{Identiteit: } d(X,X) = 0 \\
&\text{Symmetrie: } d(X,Y) = d(Y,X) \\
&\text{Driehoeksongelijkheid: } d(X,Z) \leq d(X,Y) + d(Y,Z)
\end{aligned} \tag{2.4}$$

Waar de Euclidische afstand voldoet aan alle vier en dus een metrische afstand is, voldoet dynamic time warping niet aan het axioma van de driehoeksongelijkheid. Omdat DTW wel gebruikt wordt als een afstand en overige literatuur dit ook doet, zal DTW verder in de thesis toch een afstand genoemd worden.



Figuur 2.3: Visualisatie van een derde orde tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$, gebaseerd op [9].

2.2 Matrices en Tensoren

2.2.1 Notatie en begrippen

Tensoren zijn meerdimensionale tabellen en worden in dit werk geschreven als vetgedrukte, onderlijnde hoofdletters, bijvoorbeeld $\underline{\mathbf{T}}$. Deze thesis zal voornamelijk handelen over tensoren van de derde orde (visueel voorgesteld in Figuur 2.3), waarbij *orde* (ook: *mode*) staat voor het aantal dimensies van de tensor. Een tweede orde tensor is een matrix, die wordt genoteerd met een vetgedrukte hoofdletter, bijvoorbeeld $\mathbf{A}$. Een vector, geschreven als vetgedrukte kleine letter, bijvoorbeeld $\mathbf{v}$, is een eerste orde tensor.

Om element (i, j, k) uit tensor $\underline{\mathbf{T}}$ aan te duiden, wordt de notatie $t_{i,j,k}$ gebruikt. Een gelijke notatie wordt gebruikt voor elementen uit matrices en vectoren, respectievelijk element $a_{i,j}$ voor element (i, j) uit $\mathbf{A}$ en v_i als $i$de element van $\mathbf{v}$.

Een matrix is opgebouwd uit rijen en kolommen, waarbij rij i van matrix $\mathbf{A}$ geschreven wordt als $\mathbf{a}_{i,:}$ en kolom j als $\mathbf{a}_{:,j}$. Het dubbelpunt duidt hier op het nemen van alle elementen terwijl de letter de vast gekozen rij/kolom is. Bij tensoren worden dit *fibers* genoemd, waarbij de rij- en kolomfibers overeenkomen met rijen en kolommen (notatie respectievelijk $\mathbf{t}_{i,:,k}$ en $\mathbf{t}_{:,j,k}$), en tubefibers gebruikt worden voor de bijkomende derde dimensie (notatie: $\mathbf{t}_{i,j,:}$). In termen van de mode is de kolomfiber de mode-1 fiber, de rijfiber de mode-2 fiber en de tube is dan mode-3.

Naast de ééndimensionale notie van fibers bestaat er ook de tweedimensionale opdeling van tensoren in *slices*. Dit kan bekeken worden als de verdeling van de tensor in matrices en in een derde orde tensor kan dit horizontaal ($\mathbf{X}_{i::}$), verticaal ($\mathbf{X}_{:j:}$) of frontaal ($\mathbf{X}_{::k}$). Zo is elke slice op zijn beurt een afstandsmatrix, maar deze zal niet in elke dimensie een symmetrische matrix zijn. Dit is bijvoorbeeld te zien in de tensor zoals in Figuur 2.4. Deze werd opgesteld uit frontaal geplaatste afstandsmatrices (wel symmetrisch), maar bij het nemen van horizontale slices heeft dit geen garantie meer om symmetrisch te zijn.

$\times_n$ is het **n-mode product**, dat wordt gebruikt om een tensor met een matrix te vermenigvuldigen in mode n. De formule: $\underline{\mathbf{T}} \times_1 \mathbf{X}$ betekent bijvoorbeeld dat elke mode-*1* fiber in $\underline{\mathbf{T}}$ vermenigvuldigd wordt met matrix $\mathbf{X}$.

Een ander product dat aan bod zal komen in deze thesis is het **outer product**,

genoteerd met $\circ$. Dit wordt gebruikt om een tensor van mode-3 te gaan schrijven als een product van drie vectoren, zoals bijvoorbeeld in de CP-decompositie (Sectie 3.1.2): $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$ ziet eruit als volgt:

$$\underline{\mathbf{T}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \quad . \tag{2.5}$$

Elk element van de tensor zal dus het product zijn van de overeenkomstige elementen in de vector:

$$t_{i,j,k} = a_i b_j c_k \ . \tag{2.6}$$

Het **Kronecker product** $\otimes$ tussen matrices $\mathbf{M} \in \mathbb{R}^{I \times J}$ en $\mathbf{N} \in \mathbb{R}^{K \times L}$ resulteert in een matrix van $(IK) \times (JL)$ en wordt als volgt geformuleerd:

$$\mathbf{M} \otimes \mathbf{N} = \begin{bmatrix} m_{1,1}\mathbf{N} & m_{1,2}\mathbf{N} & \cdots & m_{1,I}\mathbf{N} \\ m_{2,1}\mathbf{N} & m_{2,2}\mathbf{N} & \cdots & m_{2,I}\mathbf{N} \\ \vdots & \vdots & \ddots & \vdots \\ m_{I,1}\mathbf{N} & m_{I,2}\mathbf{N} & \cdots & m_{I,J}\mathbf{N} \end{bmatrix} . \tag{2.7}$$

Hierbij wordt elk element uit $\mathbf{M}$ vermenigvuldigd met $\mathbf{N}$, wat ook neerkomt op

$$\mathbf{M} \otimes \mathbf{N} = [\mathbf{m}_1 \otimes \mathbf{n}_1 \quad \mathbf{m}_1 \otimes \mathbf{n}_2 \quad \cdots \quad \mathbf{m}_J \otimes \mathbf{n}_{L-1} \quad \mathbf{m}_J \otimes \mathbf{n}_L]. \tag{2.8}$$

Het **Khatri-Rao product** $\odot$ tussen matrices $\mathbf{M} \in \mathbb{R}^{I \times K}$ en $\mathbf{N} \in \mathbb{R}^{J \times K}$ resulteert in een matrix van $(IJ) \times K$ en wordt als volgt geformuleerd:

$$\mathbf{M} \odot \mathbf{N} = [\mathbf{m}_1 \otimes \mathbf{n}_1 \quad \mathbf{m}_2 \otimes \mathbf{n}_2 \quad \cdots \quad \mathbf{m}_K \otimes \mathbf{n}_K]. \tag{2.9}$$

De elementen van $\mathbf{M}$ worden met behulp van het Kronecker product kolomsgewijs vermenigvuldigd met de elementen van $\mathbf{N}$.

Verder zal van tensoren ook de **norm** berekend worden, bijvoorbeeld om een kwantitatieve maat te hebben van hoe groot het verschil is tussen twee tensoren. Hiervoor wordt de tensor norm gebruikt die overeenkomt met de Frobenius norm voor matrices, namelijk de vierkantswortel van de som van het kwadraat van alle elementen in de tensor. De norm wordt genoteerd als $||.||$ en wordt voor een derde orde tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$ berekend met de volgende formule:

$$||\underline{\mathbf{T}}|| = \sqrt{\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} t_{i,j,k}^2} \tag{2.10}$$

2.2.2 Dataset

De dataset die doorheen deze thesis gebruikt wordt, is de dataset afkomstig uit *AMIE: Automatic Monitoring of Indoor Exercises* van Decroos et al., 2019 [6]. Er zal naar deze dataset verwezen worden als de **AMIE dataset**.

De data bestaat uit 186 video's, waarbij elke video gemaakt is met een Microsoft Kinect 3D camera. Deze camera filmt telkens 25 gewrichten van een persoon die

een oefening doet (*squat*, *forward lunge* of *side lunge*) gedurende tien repetities. Er werden tien personen gefilmd voor het onderzoek en iedere persoon voert elke oefening drie keer correct uit en drie keer maakt hij/zij een bepaalde fout (welke fout is niet van toepassing voor deze thesis). Door uit elke video de x-, y- en z-dimensie van elk gewricht te nemen, ontstaan er per video 75 tijdreeksen. De tijdreeksen kunnen beschouwd worden alsof ze gemaakt zijn door 75 verschillende sensoren en er zal verder ook gesproken worden over sensoren als het over deze 75 gaat. Het zijn deze tijdreeksen die met elkaar vergeleken worden en waarmee er zal worden geclusterd.

2.2.3 Afstandsmatrix

Bij het vergelijken van meerdere tijdreeksen kan er gebruik gemaakt worden van een afstandsmatrix $\mathbf{A}$. Elk van de elementen van $\mathbf{A}$ bevat de afstand tussen twee tijdreeksen T_i en T_j. Dergelijke afstand wordt geschreven als $d(T_i, T_j)$ en is hier bijvoorbeeld de DTW-afstand tussen reeksen T_i en T_j. Dit resulteert in de volgende matrix:

$$\mathbf{A} = \begin{bmatrix} d(T_1,T_1)=0 & d(T_1,T_2) & ... & d(T_1,T_{n-1}) & d(T_1,T_n) \\ d(T_2,T_1) & 0 & ... & d(T_2,T_{n-1}) & d(T_2,T_n) \\ ... & ... & ... & ... & ... \\ d(T_{n-1},T_1) & d(T_{n-1},T_2) & ... & 0 & d(T_{n-1},T_n) \\ d(T_n,T_1) & d(T_n,T_2) & ... & d(T_n,T_{n-1}) & 0 \end{bmatrix} \in \mathbb{R}^{n\times n}. \quad (2.11)$$

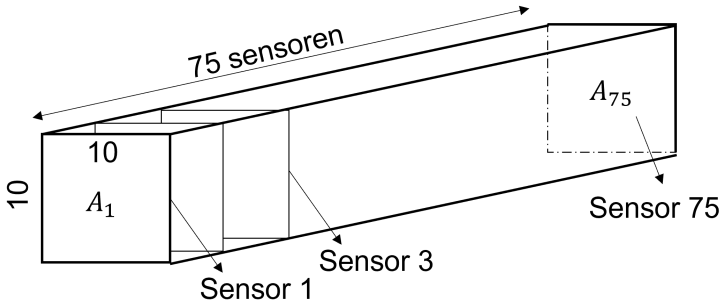
De diagonaal van $\mathbf{A}$ zal bestaan uit nullen, aangezien de afstand van een tijdreeks tot zichzelf 0 is. Verder zal bij het gebruik van EA of DTW als afstandsfunctie de afstandsmatrix ook symmetrisch zijn, aangezien beide afstandsfuncties de symmetrie eigenschap hebben dat

$$d(T_i, T_j) = d(T_j, T_i). \quad (2.12)$$

Deze afstandsmatrix is de basis voor het clusteren van tijdreeksen. Voor het opstellen van de afstandsmatrix voor de AMIE dataset, waarbij voor een oefening alle tien personen (en per persoon dus alle 75 sensoren) met elkaar worden vergeleken, geeft dit een matrix $\mathbf{A}_{AMIE} \in \mathbb{R}^{750\times750}$. Dit resulteert in meer dan 280000 DTW berekeningen nodig, wat een zeer kostelijke operatie gaat zijn. Daarom zal er in deze thesis gebruik worden gemaakt van derde orde tensoren die een tweedimensionale matrix uitbreiden met een derde dimensie om een onderscheid te kunnen maken tussen personen en sensoren, wat ook zal leiden tot minder DTW berekeningen.

2.2.4 Afstandstensor

Het gebruik van een tensor zal naast het toevoegen van een of meerdere dimensies ook kunnen zorgen dat er **minder elementen nodig** zijn dan in bijvoorbeeld een matrix. Neem hier het voorbeeld van bij de matrix, de situatie met tien personen, met elk 75 sensoren op hun lichaam (elke sensor produceert een tijdreeks): de afstandsmatrix $\mathbf{A}$ was van grootte 750×750.



FIGUUR 2.4: Een afstandstensor opgesteld uit afstandsmatrices van 10 personen voor alle 75 sensoren. Gebaseerd op de AMIE dataset.

Voor een tensor kan hier een onderscheid gemaakt worden per sensor, zodat elke persoon per sensor i met elkaar vergeleken wordt, wat telkens een matrix van $\mathbf{A_i} \in \mathbb{R}^{10\times10}$ oplevert. Door dit te doen voor elke sensor en elke $\mathbf{A_i}$ te beschouwen als frontale matrix (zie Figuur 2.4), kan er zo een tensor worden opgebouwd van $(10*10)*75 = 7500$ elementen, wat een verschil geeft van een factor $562500/7500 = 75$ in het aantal DTW berekeningen. Een tweede manier om minder berekeningen te gaan doen, is door niet deze volledige tensor te berekenen maar de benadering van de tensor te gaan opstellen. Dit zal verder worden besproken in Sectie 3.1.

Het hebben van een extra dimensie geeft de mogelijkheid om te gaan **clusteren in meer dimensies**. Zo kan er geclusterd worden per sensor (sensoren die zich gelijkend gedragen komen in dezelfde cluster) of per persoon (gelijkenissen vinden tussen personen, bijvoorbeeld verschillen man en vrouw).

De tensor zou ook kunnen worden **uitgebreid met** zogenaamde *experten informatie*. Dit is informatie die gekend is door de expert, zijnde de persoon die het onderzoek doet of de sensoren plaatst. De AMIE dataset bevat bijvoorbeeld expert informatie over waar de sensor is aangebracht, informatie die een grote rol kan spelen of extra inzicht geven bij onder meer het clusteren. In deze thesis wordt de experten informatie niet toegevoegd aan de tensor maar gebruikt om na te gaan of de algoritmes al dan niet bruikbare resultaten opleveren.

2.3 Clusteren

Zoals al werd vermeld, zullen de tijdreeksen gebruikt worden om te gaan clusteren. Het doel van clusteren is om gelijkende tijdreeksen (met een kleine afstand tussen elkaar) in eenzelfde groep (een cluster) te plaatsen. De tijdreeksen worden eerst in een tensor geplaatst, waarna met behulp van een decompositie van die tensor (zie Sectie 3.1) kan worden geclusterd in de gewenste dimensie. In het lopend voorbeeld zou dit dus in de dimensie van personen of sensoren kunnen zijn. Hieronder worden twee clusteralgoritmes besproken die in de experimenten gebruikt zullen worden, gevolgd door hoe de clusters worden geëvalueerd.

2.3.1 Spectraal clusteren

In het geval van matrices vertrekt spectraal clusteren [11] vanuit de gelijkenismatrix $\mathbf{G} \in \mathbb{R}^{n \times n}$. De eerste stap van het algoritme is de afstandsmatrix $\mathbf{A}$ om te zetten naar deze gelijkenismatrix. Er zijn veel manieren mogelijk om dit te doen (bijvoorbeeld $G_{i,j} = e^{A_{i,j}/r}$ met $r = max(A)$) maar dit is niet belangrijk voor onze toepassing.

De gelijkenismatrix wordt gebruikt om de Laplaciaan $\mathbf{L}$ op te stellen, waarbij

$$\mathbf{L} = \mathbf{D} - \mathbf{G} \,. \tag{2.13}$$

In deze formule is $\mathbf{D}$ de diagonaalmatrix met elementen $D_{i,i} = \sum_j G_{i,j}$ en dezelfde dimensies als $\mathbf{G}$. De volgende stap is het berekenen van de eerste k eigenvectoren van $\mathbf{L}$, $\{u_1, u_2, \cdots, u_k\}$, met k het gewenste aantal clusters.

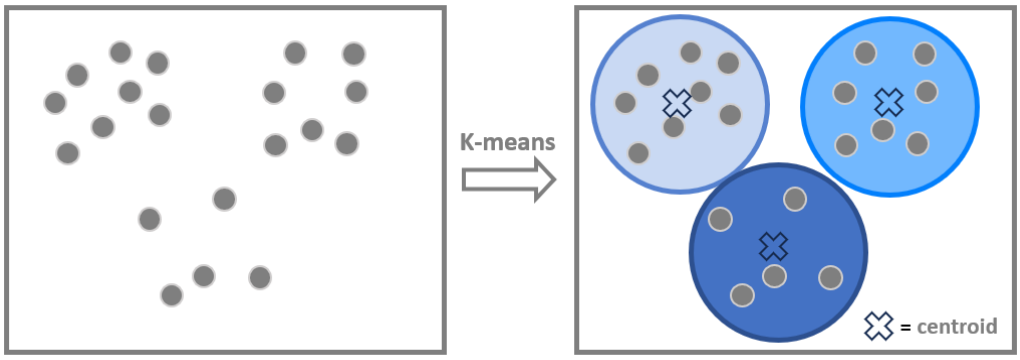
Deze eigenvectoren vormen een nieuwe representatie van de data en kunnen beschouwd worden als de *feature vectoren*, met elke waarde in de vector overeenkomend met een te clusteren object. Dit is de duurste stap in het algoritme omdat *Singular Value Decomposition* (SVD) wordt gebruikt om de eigenvectoren van $\mathbf{L}$ te berekenen, met een complexiteit van $\mathcal{O}(n^3)$. In de tweede stap wordt met behulp van het clusteralgoritme *k-means* en de feature vectoren elk van de objecten in één van de k clusters geplaatst.

Door te werken met tensoren, kunnen we de hele eerste stap van het algoritme overslaan. Er zal geen afstandsmatrix zijn en er moeten bijgevolg geen gelijkenismatrix en Laplaciaan opgesteld worden. Het belangrijkste is echter dat de dure SVD stap kan worden overgeslagen. De eerste k eigenvectoren zullen voor een matrix de belangrijkste informatie bevatten. Een tensor daarentegen heeft geen eigenvectoren, maar werkt in de plaats daarvan met decomposities (besproken in Sectie 3.1). Deze decompositie zullen we opstellen om de tensor te gaan benaderen. Zo worden niet alleen het aantal kostelijke DTW berekeningen verminderd, maar zullen we dus ook de eigenvectoren gaan vervangen door feature vectoren rechtstreeks bekomen uit de tensor decompositie. De manier waarop de decompositie wordt gemaakt, wordt besproken in Hoofdstuk 5. In deze thesis wordt dus een eigen versie van spectraal clusteren gebruikt als toepassing voor tensoren, met in stap één het opstellen van een decompositie om de feature vectoren te verkrijgen en in stap twee het opstellen van een k-means clustering met behulp van de verkregen feature vectoren.

2.3.2 K-means

K-means is een iteratief clusteralgoritme dat gebruikt wordt om een dataset te verdelen in k verschillende clusters, waarbij elk van de clusters een groep van gelijkende datapunten voorstelt [3]. Het algoritme gaat na initialisatie itereren over twee stappen: de toewijzing stap en de update stap.

Voor de initialisatie worden de eerste k centroids gekozen met behulp van *K-means++* [1]. Deze initialisatie gaat de kans vergroten dat de gekozen centroids



FIGUUR 2.5: Illustratie van k-means clustering met behulp van centroids als cluster representatie.

gelijkmatig verspreid zijn over de dataset en ver van elkaar verwijderd. Het algoritme heeft zo een grotere kans om sneller te convergeren naar een goede oplossing. K-means++ start met het kiezen van een willekeurige centroid in de dataset. Daarna wordt voor elk ander punt van de data de afstand berekend naar de dichtstbijzijnde (reeds gekozen) centroid. Datapunten met een grotere afstand hebben een grotere kans om te worden gekozen als volgende centroid. Op basis van deze afstanden wordt een kansverdeling opgesteld en het element met de grootste kans wordt dan als centroid genomen. Het proces van het berekenen van de afstanden, opstellen van de kansverdeling en het kiezen van centroids wordt dan herhaald tot er k centroids zijn geselecteerd.

Na de initialisatie volgt de toewijzing. In deze stap wordt elk van de datapunten toegewezen aan een cluster. Dit gebeurt door de afstand tussen het datapunt en elke centroid te berekenen. Iedere centroid zal de representatie zijn van één van de k clusters en alle datapunten worden toegewezen aan de cluster van de dichstbijzijnde centroid.

In de update stap zal dan de centroid van elke cluster worden herberekend. Een centroid in k-means zal het gemiddelde zijn van alle datapunten in de cluster. De nieuwe centroids worden dus berekend door het gemiddelde te nemen van elk element dat in deze iteratie in de cluster van de centroid zit. Zo zullen de centroids zich steeds dichter in het centrum van hun cluster bevinden en bijgevolg meer representatief zijn voor die cluster. Deze stappen blijven zich herhalen tot het algoritme is geconvergeerd naar stabiele clusters. De clusters zijn stabiel wanneer er in opeenvolgende iteraties geen veranderingen zijn in de toewijzing en update stap. De implementatie wordt gedaan met behulp van de *scikit-learn library* in Python [15].

De keuze van het aantal clusters k is een belangrijke factor voor een goede interpretatie van de resultaten van het algoritme. Hiervoor kan de experten informatie gebruikt worden. Voor de AMIE dataset is namelijk voor elke tijdreeks geweten door

welke persoon en sensor ze is gemaakt, tijdens het uitvoeren van welke oefening. Zo kan het aantal clusters gekozen worden aan de hand van de toepassing die we willen bekijken. We zullen bijvoorbeeld gaan clusteren op het aantal personen of op de verschillende oefeningen.

2.3.3 Evaluatie

Om de kwaliteit van de clusters te beoordelen, wordt de **Adjusted Rand Index** (ARI) gebruikt [7]. De ARI-score is gebaseerd op de gelijkenis tussen de toekenning van tijdreeksen aan clusters door het gebruikte clusteralgoritme en de gekende *true labels* van de tijdreeksen. Deze twee *verdelingen* van tijdreeksen worden verder partities genoemd. Er is gekozen voor deze methode aangezien de true labels van een tijdreeks voorhanden zijn, namelijk omdat voor elke tijdreeks geweten is door welke sensor ze gemaakt is en tot welke persoon ze behoort (experten informatie).

De ARI is een verbetering van de Rand Index (RI), die wordt berekend op basis van het tellen van het aantal paren van tijdreeksen dat in beide partities ofwel in dezelfde cluster zit ofwel in verschillende clusters. RI berekent dan de verhouding tussen de som van de True Positives (TP) en de True Negatives (TN) en het totale aantal tijdreeks-paren:

$$\frac{TP + TN}{\frac{n(n+1)}{2}} \tag{2.14}$$

met n het aantal tijdreeksen. Dit resulteert in een score tussen 0 en 1, waarbij 1 een perfecte match aangeeft. De RI-score zal echter zelden 0 zijn, aangezien zelfs een random toekenning van tijdreeksen aan clusters een bepaalde overeenkomst zal hebben met de true labels. Daarom wordt de ARI score gebruikt die ook de False Positives en Negatives (FP en FN) in rekening brengt en zo de RI score *corrigeert* zodat de score kan gaan van -1 tot 1. Hier duidt 1 weer een perfecte overeenkomst aan tussen partities, 0 duidt op random toewijzingen van tijdreeksen aan clusters en er zijn nu ook negatieve waardes tot -1 die aangeven dat er minder overeenkomsten zijn dan verwacht.

Hoofdstuk 3

Voorgaand werk

In dit hoofdstuk wordt de belangrijkste literatuur besproken die gerelateerd is aan of gebruikt wordt in deze thesis. Er wordt gestart met de paper over tensoren en factorisaties in Sectie 3.1, gevolgd door een paper over Adaptive Cross Approximation en een thesis die dit ACA algoritme gebruikt om matrices te factoriseren in Sectie 3.2. Het ACA algoritme zal later in de thesis uitgebreid worden naar een versie die werkt met tensoren, gelijkend aan het algoritme besproken in Sectie 3.3 maar dan met meerdere implementaties en resulterende in een andere decompositie.

3.1 Tensor factorisaties

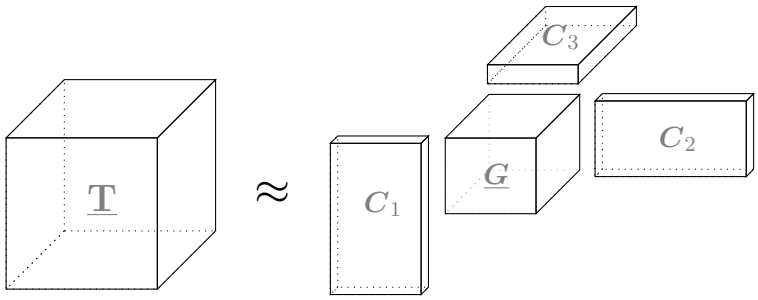
De paper *Tensor Decompositions and Applications* van Kolda en Bader [9] is de basis voor het gebruiken van tensoren en factorisaties. Factoriseren (ook wel *decompositie maken*) van een tensor is de tensor reduceren tot kleinere onderdelen die samen een benadering van de tensor kunnen genereren. De twee meest bekende voorbeelden van tensor decomposities zijn de Tucker factorisatie en de Candecomp/Parafac (CP) decompositie. Gebaseerd op de paper van Kolda en Bader, en de paper van Phan en Cichocki [16] worden hieronder deze twee decomposities besproken.

3.1.1 Tucker decompositie

De Tucker decompositie gaat een hogere-orde tensor $\underline{\mathbf{T}}$ opdelen in een kern tensor $\underline{\mathbf{G}}$ en een factor matrix $\mathbf{C_i}$ aan elke mode van de tensor. Voor een derde orde tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$ is dit dus als volgt:

$$\underline{\mathbf{T}} \approx \underline{\mathbf{G}} \times_1 \mathbf{C_1} \times_2 \mathbf{C}_2 \times_3 \mathbf{C_3} \quad . \tag{3.1}$$

Een grafische weergave van deze decompositie is weergegeven in Figuur 3.1 en krijgt voor derde orde tensoren soms de naam *Tucker3*. De elementen in de kern tensor $\underline{\mathbf{G}}$ geven de hoeveelheid interactie aan tussen de verschillende factor matrices $\mathbf{C}_i$. Met $\underline{\mathbf{G}} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ zijn de factor matrices van grootte $\mathbf{C}_1 \in \mathbb{R}^{I \times r_1}$, $\mathbf{C}_2 \in \mathbb{R}^{J \times r_2}$ en $\mathbf{C}_3 \in \mathbb{R}^{K \times r_3}$. Een compacte notatie voor Formule 3.1 is $\underline{\mathbf{T}} \approx [\![\underline{\mathbf{G}}\,; \mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}]\!]$. Het opstellen van een Tucker decompositie kan met behulp van *Hogere Orde Singular Value Decompositie* of HOSVD [2], hier niet verder besproken.



Figuur 3.1: Visualisatie van de Tucker decompositie van $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$.

3.1.2 CP decompositie

De Candecomp/Parafac decompositie gaat een hogere-orde tensor opdelen in een som van eerste rang tensoren (vectoren), zoals weergegeven in figuur 3.2. Voor de tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$ ziet de CP factorizatie eruit als volgt:

$$\underline{\mathbf{T}} \approx \sum_{i=1}^{N} \mathbf{a_i} \circ \mathbf{b_i} \circ \mathbf{c_i} \tag{3.2}$$

waarbij $\circ$ het tensor product is zoals uitgelegd in 2.2.1. Hierbij zijn de vectoren $\mathbf{a_i}$, $\mathbf{b_i}$ en $\mathbf{c_i}$ respectievelijk elementen van $\mathbb{R}^I$, $\mathbb{R}^J$ en $\mathbb{R}^K$. Ook hier is een compacte notatie mogelijk, namelijk $\underline{\mathbf{T}} \approx [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$, waarbij $\mathbf{A}$, $\mathbf{B}$ en $\mathbf{C}$ de factor matrices zijn. Een factor matrix is opgebouwd uit de vectoren in dezelfde mode, bijvoorbeeld $\mathbf{A} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_N] \in \mathbb{R}^{I \times N}$.

CP decompositie opstellen

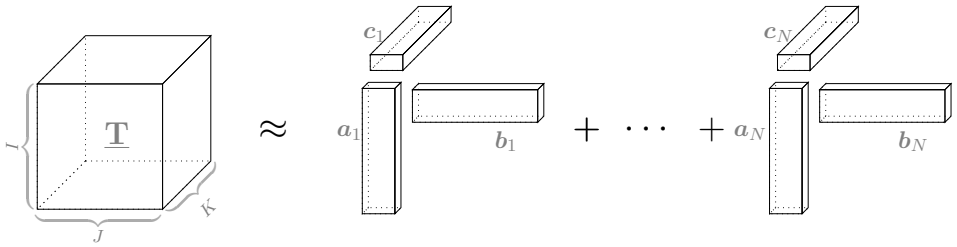
Een veelgebruikte manier om de CP decompositie op te stellen is met behulp van het *Alternating Least Squares* algoritme (ALS). Het doel van ALS is om, gegeven de derde orde tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$, een CP decompositie te gaan berekenen met N producten van rang-1 tensoren die $\underline{\mathbf{T}}$ het best benadert. In formulevorm komt dit neer op

$$\underset{\underline{\hat{\mathbf{T}}}}{\text{minimize}} \, ||\underline{\mathbf{T}} - \underline{\hat{\mathbf{T}}}|| \quad \text{waarbij} \quad \underline{\hat{\mathbf{T}}} = \sum_{i=1}^{N} \mathbf{a_i} \circ \mathbf{b_i} \circ \mathbf{c_i} \; = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]. \tag{3.3}$$

Het algoritme gaat starten met initiële willekeurige waardes voor $\mathbf{A}$, $\mathbf{B}$ en $\mathbf{C}$ waarna ALS iteratief en één per één deze matrices gaat verbeteren tot aan een bepaald (gekozen) convergentie criterium wordt voldaan.

Per iteratie worden, telkens om de beurt, twee van de matrices vastgezet en wordt er opgelost naar de andere matrix, bijvoorbeeld ALS neemt $\mathbf{B}$ en $\mathbf{C}$ vast en lost op naar $\mathbf{A}$. Daarna neemt ALS $\mathbf{A}$ en $\mathbf{C}$ vast en lost op naar $\mathbf{B}$ en uiteindelijk ook naar $\mathbf{C}$ met $\mathbf{A}$ en $\mathbf{B}$ vast. Door twee matrices vast te nemen, resulteert het probleem zich tijdelijk tot een lineair *least-squares* probleem. Bijvoorbeeld met $\mathbf{A}$ en $\mathbf{C}$ vast, lost het algoritme het volgend minimalisatie probleem op:

$$\min_{\hat{\mathbf{B}}} ||\mathbf{T}_{(\mathbf{2})} - \hat{\mathbf{B}}(\mathbf{C} \odot \mathbf{A})^T|| \, . \tag{3.4}$$



FIGUUR 3.2: Visualisatie van de CP decompositie van $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$.

Hierbij is $\mathbf{T}_{(\mathbf{2})}$ de mode-2 matricizatie van tensor $\underline{\mathbf{T}}$ (uitleg hieronder in 3.1.3), $||\cdot||$ de norm en $\odot$ het Khatri-Roa product (besproken in Sectie 2.2.1). $\hat{\mathbf{B}}$ is de benadering van $\mathbf{B}$ uit de vorige iteratie en er zijn gelijkende formules voor het verbeteren van $\mathbf{A}$ en $\mathbf{C}$. Afhankelijk van de initieel gekozen waardes voor $\mathbf{A}$, $\mathbf{B}$ en $\mathbf{C}$ zal het algoritme kunnen resulteren in verschillende matrices wanneer het convergentie criterium wordt bereikt en kan dit meer of minder iteraties nodig hebben.

3.1.3 Matricizatie

Een derde mogelijke optie is om de tensor als matrix te gaan beschouwen door *matricizatie* te doen. Hiervoor wordt de tensor *ontvouwen* volgens een mode (bijvoorbeeld mode-2), wat inhoudt dat alle fibers van die mode op zulk een manier worden herschikt zodat ze de kolommen van de matrix worden (notatie: $\mathbf{T}_{(\mathbf{2})}$). Dat zal extra overhead kosten omdat de fibers van de tensor niet noodzakelijk in een gestructureerde manier worden overlopen. Mogelijks gaat er tijdens het ontvouwen ook belangrijke informatie verloren omdat de fibers van eenzelfde afstandsmatrix (slice) niet meer bij elkaar terug te vinden zijn. Het lijkt voor deze thesis dus beter om te gaan werken met voorheen besproken tensor decomposities.

Voor het opstellen van deze decomposities is het nodig om de volledige tensor voorhanden te hebben. Omdat een hele afstandstensor opstellen een kostelijke zaak is, zal deze thesis zich richten op het gaan opstellen van dergelijke decomposities zonder te vertrekken vanuit de volledige tensor, en deze juist te gaan gebruiken als benadering van de volledige afstandstensor waarmee dan bijvoorbeeld geclusterd kan worden.

3.2 Adaptive Cross Approximation

Een algoritme voor het benaderen van een matrix $\mathbf{A}$ met behulp van een decompositie is uitgebreid besproken door Mach et al., [12]. Deze techniek heet *Adaptive Cross Approximation* (ACA) en steunt op het herhaaldelijk nemen van een *cross* of *skelet* (bestaande uit een rij en kolom) uit de matrix tot daarmee een lage rang benadering $\mathbf{A}_k$ kan worden gemaakt die $\mathbf{A}$ genoeg benadert. Hierbij is k de rang van de benadering en deze komt overeen met het aantal rijen en kolommen die geselecteerd worden uit $\mathbf{A}$.

Ook in het werk van Pede [14] wordt ACA gebruikt in zijn onderzoek naar het snel clusteren van tijdreeksen met lage-rang benaderingen van matrices. Hierin vertrekt de auteur vanuit het idee van *Singular Value Decompositie* (SVD) en hoe deze gebruikt kan worden door het nemen van de k grootste singuliere waarden en de bijhorende vectoren om een rang-k benadering $\mathbf{A}_k$ van een afstandsmatrix $\mathbf{A}$ op te stellen. SVD vertrekt vanuit de volledige afstandsmatrix. Hierdoor versnelt dit de clusterprocedure niet en wordt ACA voorgesteld als een manier om niet alle elementen in een grote afstandsmatrix te moeten berekenen, maar te gaan werken met een benadering van de matrix.

Adaptive Cross Approximation voor de matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ start met het nemen van een rij. Dit kan gebeuren door willekeurig een rij te nemen of door het samplen van een waarde uit elk van de rijen, en de rij met de grootste waarde te kiezen als de initële rij. In het voorbeeld in Figuur 3.3 is dit rij 2. Daarna verloopt het algoritme iteratief tot de benadering goed genoeg is (threshold te kiezen) of tot er k rijen en kolommen zijn gekozen.

In elke iteratie gaat ACA alle afstanden in de gekozen rij berekenen en hiervan het grootste element nemen als δ_i. In het voorbeeld is dit het vierde element in rij 2, gelabeld met δ_1. Van deze grootste waarde wordt nu de volledige kolom berekend, waarna ook hier het grootste element wordt gezocht, het zesde element in kolom 4 in Figuur 3.3. Dit resulteert in een eerste rang-1 benadering van de matrix $\mathbf{A}$, namelijk

$$\mathbf{A}_1 \approx \delta_1^{-1} * \text{kolom}_4 * (\text{rij}_2)^T \ . \tag{3.5}$$

Deze rang-1 benadering wordt dan afgetrokken van de originele matrix ($\mathbf{R} = \mathbf{A} - \mathbf{A_1}$) en creëert het residu $\mathbf{R}$. Dit wordt gedaan om te voorkomen dat rijen en kolommen worden hergebruikt. Een belangrijke notie hier is dat niet de volledige residu matrix moet worden berekend, maar enkel de in de volgende iteratie gekozen rij en kolom, aangezien er telkens wordt gezocht naar het maximum in één rij of kolom. Dit geeft een complexiteit lineair met de grootte van de matrix $\mathbf{A}$, $O(n)$.

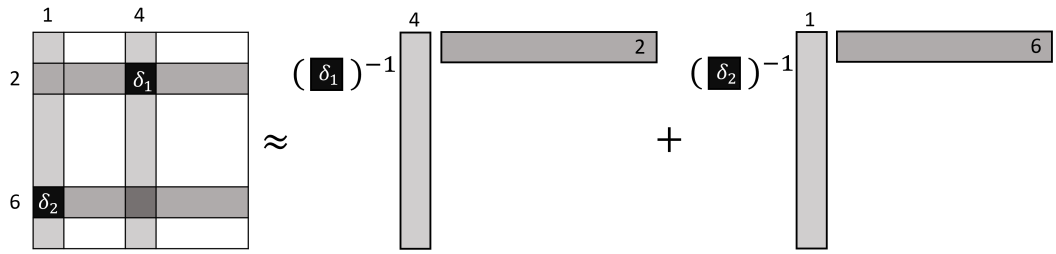
De volgende iteratie berekent in dit residu weer de rij (van het grootste element in de kolom van de vorige iteratie, nu dus rij 6), zoekt het grootste element, berekent daarvan de kolom, enzovoort. Dit resulteert uiteindelijk in de rang-k benadering van $\mathbf{A}$, waarbij

$$\mathbf{A} \approx \mathbf{A}_k = \sum_{i}^{k} \delta_i^{-1} * \text{kolom}_i * (\text{rij}_i)^T \ . \tag{3.6}$$

Met $k < n$ zal de benadering dus met minder afstandsberekeningen kunnen gebeuren, resulterend in snellere resultaten, maar die resultaten zijn slechts een benadering.

3.3 ACA voor tensoren

ACA blijkt een goede methode voor het benaderen van matrices, maar deze thesis richt zich op het benaderen van tensoren. Daarom kan ACA gebruikt worden als



FIGUUR 3.3: Visualisatie van het ACA algoritme, voor een rang van $k = 2$.

basis voor andere algoritmes, die meer van toepassing zijn voor tensoren. Twee voorbeelden hiervan zijn *TreeCUR* en *FSTD* uit de paper van Caiafa en Cichocki, 2009 [4].

3.3.1 TreeCUR

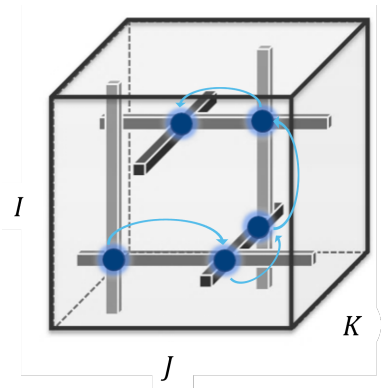
De eerste techniek is TreeCUR, waarbij de tensor ontvouwen wordt volgens één van zijn modes. Door het ontvouwen van de tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times J \times K}$ volgens mode-1 ontstaat er een matrix $\mathbf{T}' \in \mathbb{R}^{I \times JK}$ die gebruikt kan worden als input voor het ACA algoritme. De rijen in deze matrix komen dan overeen met de horizontale slices van de tensor.

De slices van de tensor zijn eigenlijk op zich al matrices. Als deze op hun beurt benaderd worden met ACA voordat ze in de matrix $\mathbf{T}'$ worden geplaatst, moeten er weer minder berekeningen gebeuren. Dit geeft een tijdscomplexiteit van $\mathcal{O}(J \times K)$ en resulteert in een boomstructuur, met in elk blad een n-mode fiber. Vandaar komt ook de naam *TreeCUR*, met CUR een alternatieve benaming voor het ACA-algoritme. Zoals vermeld in Sectie 3.1.3, kan er nuttige informatie verloren gaan na het ontvouwen van een tensor. Ook heeft het algoritme hieronder een betere tijdscomplexiteit, waardoor in de thesis het idee van FSTD verkozen wordt boven TreeCUR.

3.3.2 FSTD

Een tweede uitbreiding van ACA is *Fiber Sampling Tensor Decomposition* (FSTD). Hier wordt, in plaats van te gaan kijken naar een matrix, de ACA techniek uitgebreid zodat deze werkt met rij, kolom en tube fibers. Het algoritme start gelijkend aan de matrix versie, namelijk met het kiezen van een initiële kolom fiber $\mathbf{t}_{:,j_1,k_1}$ en de gewenste rang k (het aantal te selecteren fibers). Het algoritme wordt visueel voorgesteld in Figuur 3.4. FSTD berekent dan de elementen in de kolom en de index i_1 van het grootste element wordt opgeslagen. Daarna worden de volgende drie stappen herhaald tot de gewenste rang k bereikt is:

1. Bereken het residu van de rij fiber $\mathbf{t}_{i_{p-1},:,k_{p-1}}$ (rij in originele tensor min de rij in de benaderde tensor). Vind hierin de maximale waarde en sla de index j_p van de maximale waarde op.



Figuur 3.4: Visualisatie van het FSTD algoritme, voor een rang van $k = 2$.

2. Neem van deze index de tube fiber ($\mathbf{t}_{i_{p-1},j_p,:}$) en bereken hiervan het residu. Zoek de grootste waarde in de tube en sla hiervan de index k_p op.

3. Bereken tenslotte het residu van de kolom fiber $\mathbf{t}_{:,j_p,k_p}$, vind de maximale waarde en bewaar de index i_p.

Ook hier moet telkens het residu van slechts één fiber berekend worden, resulterende in een lineaire complexiteit $\mathcal{O}(I)$. De opgeslagen indices kunnen dan worden gebruikt om de Tucker decompositie op te stellen, bestaande uit een nieuw gevormde kern tensor $\underline{\mathbf{U}}$ en een matrix $\mathbf{C}_i$ aan elk van de modes:

$$\underline{\mathbf{T}} \approx \hat{\underline{\mathbf{T}}}_k = \underline{\mathbf{G}} \times_1 \mathbf{C}_1 \times_2 \mathbf{C}_2 \times_3 \mathbf{C}_3 \,. \tag{3.7}$$

De matrix $\mathbf{C}_1$ bevat dan alle door FSTD geselecteerde kolommen, $\mathbf{C}_2$ de rijen en de tube fibers staan in $\mathbf{C}_3$.

Hoofdstuk 4

Probleemstelling

Om de personen of sensoren uit het onderzoek te kunnen gaan clusteren, wordt er eerst een afstandstensor opgesteld. Wanneer hier de DTW afstand gebruikt wordt om de paarsgewijze afstanden te berekenen, geeft dit voor elk element in de tensor een $\mathcal{O}(l_1 l_2)$ complexiteit (met l_1 en l_2 de lengtes van de te vergelijken tijdreeksen). Wanneer het aantal personen en/of het aantal sensoren groter wordt, neemt de grootte van de tensor toe en evenredig daarmee ook het aantal kostelijke DTW berekeningen.

Om deze kost te beperken, focust de thesis zich op het benaderen van de afstandstensor, zodat niet alle elementen van de volledige tensor berekend moeten worden. Hierdoor zullen er minder berekeningen moeten gebeuren, resulterende in een goedkopere en snellere uitvoering. Daar staat tegenover dat de waardes in de tensor slechts een benadering zullen zijn van de originele afstandstensor. Het belangrijkste is hier om een goede *tradeoff* te vinden tussen de kwaliteit van de benadering en de versnelling van het algoritme. Dit leidt rechtstreeks tot de onderzoeksvraag van deze thesis.

4.1 Onderzoeksvragen

De hoofdvraag luidt als volgt: **geeft de benadering van een tensor, met behulp van een tensor factorisatie, een clustering die gelijkend is aan de clustering gemaakt met de volledige tensor?**
Een tweede vraag die hier dan meteen bij gesteld kan worden, is: in welke mate versnelt werken met een benadering het hele procedure?

Om een antwoord op de hoofdvraag te vinden, worden er algoritmes opgesteld gebaseerd op het ACA algoritme voor matrices. De algoritmes zullen gebruikt worden om een lage-rang benadering van de afstandstensor op te stellen. Werkende met een tensor zijn er een aantal mogelijkheden om deze lage-rang benadering te gaan voorstellen. Voor deze thesis zijn hier twee methodes uitgekozen, namelijk de benadering als de som van een product van vectoren in drie modes en de benadering

als som van het product van een vector in één mode en een matrix in de twee andere. Beide methodes worden besproken in Hoofdstuk 5, en leiden ook tot onze derde en laatste onderzoeksvraag, namelijk: welke van de implementaties geeft de beste afweging tussen kwaliteit van benadering en het aantal nodige berekeningen.

Voor de snelheid van de algoritmes zal telkens gekeken worden naar het aantal DTW berekeningen nodig (zijnde de zwaarste operatie) om de tensor op te stellen. De rest van de *cluster pipeline* (van data naar tensor naar clustering) zal telkens identiek gebeuren en als even grote kost worden beschouwd. Het clusteren zelf zal gebeuren met de methodes voorgesteld in Sectie 2.3 en ook op die manier geëvalueerd worden.

De implementatie van de algoritmes en de experimenten (in Python) zijn terug te vinden in de softwarebibliotheek op Github [18].

Hoofdstuk 5

Tensoren benaderen

Zoals besproken in Sectie 3.1, gaat het ontbinden van een tensor resulteren in een combinatie van lagere rang componenten. Het doel van deze thesis is nu om, aan de hand van deze lage-rang componenten, de oorspronkelijke tensor zo goed mogelijk te gaan benaderen, zonder te starten vanuit de volledig opgestelde tensor. Dit zal zowel het geheugengebruik als het aantal benodigde berekeningen ten opzichte van het werken met een volledige tensor verlagen.

Als basis voor het opstellen van de CP factorisatie zal gebruik worden gemaakt van skelet decomposities. Zoals besproken in Sectie 3.2, kan met behulp van skeletten een benadering van matrices worden opgesteld. Hieronder worden twee manieren besproken waarop deze skeletten kunnen worden veralgemeend naar tensoren, door het gebruik van fibers en/of slices.

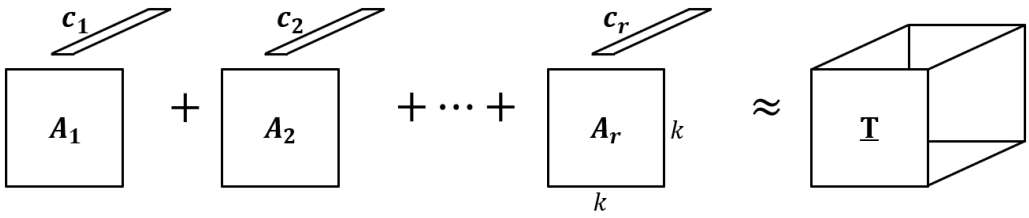
5.1 Methode 1: Matrix en vector

De eerste methode om de tensor te gaan benaderen, is gebaseerd op hoe de tensor wordt opgebouwd voor de AMIE dataset. Voor het maken van de volledige tensor worden alle personen met elkaar vergeleken per sensor, resulterende in symmetrische afstandsmatrices als frontale slices. Het idee dat hier uit voortkomt, is de tensor te gaan opstellen door een tweedimensionale matrix te vermenigvuldigen met een vector in de derde dimensie.

Om de vergelijking te maken met het ACA algoritme voor matrices, zal het *cross* nu bestaan uit een matrix en een vector in plaats van twee vectoren (visueel voorbeeld: $\mathbf{A_1}$ en $\mathbf{c_1}$ in Figuur 5.1). Omdat er op een zelfde manier adaptief een cross gekozen wordt om de tensor te gaan benaderen, zullen we de methode *Adaptive Cross Approximation voor tensoren* noemen, afgekort **ACA-T**. De rang-één benadering van de tensor wordt dus

$$\underline{\mathbf{T}} \approx \frac{1}{t_{i,j,k}} (\mathbf{A}_{::k} \circ \mathbf{t}_{i,j,:}) \tag{5.1}$$

waarbij $t_{i,j,k}$ het gezamenlijke element is van matrix $\mathbf{A}_{::k}$ en vector $\mathbf{t}_{i,j,:}$. Verder kan van de tensor dan het residu $\underline{\mathbf{R}}$ worden berekend als het verschil tussen de



Figuur 5.1: ACA voor tensoren, benadering op basis van een frontale matrix en fiber in derde mode (rang R).

oorspronkelijke tensor en de huidige benadering

$$\underline{\mathbf{R}} \approx \underline{\mathbf{T}} - \frac{1}{t_{i,j,k}} (\mathbf{A}_{::k} \circ \mathbf{t}_{i,j,:}) \; . \tag{5.2}$$

Het residu kan telkens gebruikt worden in de volgende iteratie, resulterende in een steeds nauwkeuriger wordende benadering. Hoe kleiner de elementen worden in het residu, des te beter de benadering (met als perfecte benadering alle elementen van $\underline{\mathbf{R}}$ gelijk aan 0). Het gebruik van het residu verzekert de kans dat reeds correct benaderde elementen niet opnieuw worden gekozen, aangezien deze elementen in het residu nul zullen zijn en dus (zolang de tensor niet perfect benaderd is) nooit gevonden zullen worden als maximum. Een belangrijke notie is hier dat dit residu niet volledig uitgerekend moet worden, omdat er telkens gezocht zal worden in één matrix en vector in plaats van in de volledige tensor.

Per iteratie wordt er een matrix (of *slice*) en een vector (ook *tube*) toegevoegd aan de benadering. Om efficiënt deze benadering te verbeteren, zal het kiezen van deze structuren op een adaptieve manier gebeuren. We zoeken namelijk de elementen die de grootste impact hebben op het residu van de benadering. Het toevoegen van de slice of tube waar dit element zich in bevindt, zal bijgevolg de grootste verbetering van de benadering opleveren. Zoals aangehaald in de papers van ACA en de toepassingen van ACA op tensoren [12, 4] zal het element met de grootste absolute waarde de grootste impact hebben. De theoretische uitvoer van het algoritme wordt hieronder besproken. Het algoritme zelf wordt in pseudocode weergegeven in Algoritme 1 en er wordt bij de beschrijving verwezen naar de lijnen in kwestie.

We starten met rang $r = 1$ en berekenen x aantal willekeurige elementen uit de tensor. In deze lijst van elementen S wordt de grootste waarde gezocht, waarvan dan de tube fiber of de matrix slice wordt genomen als startpunt voor het algoritme. Er kan ook gestart worden vanuit een willekeurig gekozen fiber of slice, maar door het berekenen van een aantal elementen is de kans groter om te starten met een betere benadering.

In de implementatie wordt gestart met de slice van het gevonden grootste element. De slice wordt ingevuld door alle DTW-afstanden te berekenen en hier wordt opnieuw

het element met de grootste absolute waarde genomen (in Algoritme 1: regel 5-8). Dit wordt δ_1 en van dit element wordt nu ook de volledige tube fiber uitgerekend. In deze tube wordt het grootste element gezocht en daarvan wordt de index bijgehouden voor de volgende iteratie (Algoritme 1: regel 10-12). We hebben nu de eerste slice en fiber, waarmee de rang-één benadering $\underline{\mathbf{B}}_1$ van tensor $\underline{\mathbf{T}}$ kan worden opgesteld, namelijk

$$\underline{\mathbf{T}}_1 \approx \underline{\mathbf{B}}_1 = \delta_1^{-1}(\mathbf{A_1} \circ \mathbf{c_1}) \ . \tag{5.3}$$

Met deze benadering $\underline{\mathbf{B}}_1$ kan zoals in formule 5.2 het residu $\underline{\mathbf{R}}_1$ worden berekend als $\underline{\mathbf{R}}_1 = \underline{\mathbf{T}} - \underline{\mathbf{B}}_1$. In de volgende iteratie wordt dezelfde procedure herhaald, nu startend vanuit het residu en met als initiële slice $\mathbf{A}_{::index}$, met de index van het gevonden grootste element uit de vorige iteratie. Hier wordt duidelijk dat in elke iteratie de volledige residu tensor niet moet worden uitgerekend, maar enkel de slice en de fiber ervan die worden gekozen per iteratie. In elke stap wordt het residu dus gebaseerd op het vorige residu en de huidige benadering:

$$\underline{\mathbf{R}}_k \approx \underline{\mathbf{R}}_{k-1} - \delta_k^{-1}(\mathbf{A_k} \circ \mathbf{c_k}) \ . \tag{5.4}$$

Wanneer het algoritme r iteraties heeft afgelegd, zal de tensor een rang-r benadering hebben die overeenkomt met de formule:

$$\underline{\mathbf{T}}_r \approx \sum_{i=1}^{r} \delta_i^{-1}(\mathbf{A_i} \circ \mathbf{c_i}) \ . \tag{5.5}$$

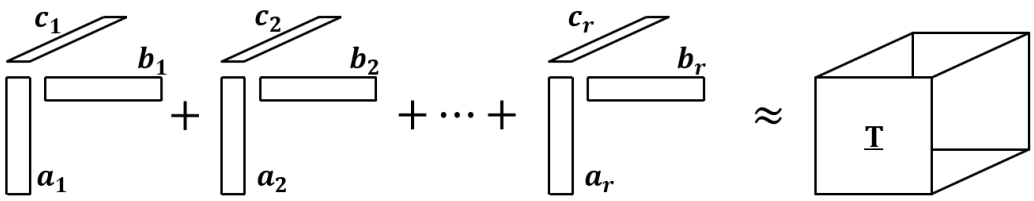
Deze formule wordt visueel weergegeven in Figuur 5.1. In het lopend voorbeeld zou de matrix $\mathbf{A}_i$ dimensie $\mathbb{R}^{10\times10}$ hebben ($k = 10$) en de tube fiber $\mathbf{c}_i$ een lengte van 75.

Algorithm 1 : ACA-T voor matrix × vector benadering van tensor

Input: Onberekende tensor $\underline{\mathbf{T}}$, *max_iters*, *max_r*

1: Bereken samples: $S =$ **generate_samples**$(\underline{\mathbf{T}})$
2: $k_0 = max(S)$
3: $r = 1$
4: **while** $r < max\_r$ **do**
5: bereken matrix $\underline{\mathbf{T}}_{::k_r}$
6: $\mathbf{A}_r = \underline{\mathbf{T}}_{::k_r} - \sum_{i=1}^{r} \delta_i^{-1}(\mathbf{A_i}(\mathbf{c_i})_{k_r})$
7: $max\_idx = (i_r, j_r) = argmax(abs(\mathbf{A}_r))$
8: $\delta_r = \mathbf{A}_r[max\_idx]$
9:
10: bereken tube $\underline{\mathbf{T}}_{i_r j_r :}$
11: $\mathbf{c}_r = \underline{\mathbf{T}}_{i_r j_r :} - \sum_{i=1}^{r} \delta_i^{-1}((\mathbf{A_i})_{i_r, j_r}\mathbf{c_i}$
12: $k_{r+1} = max(abs(\mathbf{c}_r))$
13: $r + 1$
14: **end while**

Result: $\underline{\mathbf{T}}_{benaderd} = \sum_{i=1}^{r} \delta_i^{-1}(\mathbf{A_i} \circ \mathbf{c_i})$



Figuur 5.2: ACA voor tensoren, benadering op basis van een vector in elke mode (rang R).

Voor het lopend voorbeeld, waar de tensor $\underline{\mathbf{T}}$ van dimensie $\mathbb{R}^{i\times i\times k}$ is, zal elke iteratie van het ACA-T algoritme één symmetrische matrix van $i \times i$ en één tube van lengte k gaan berekenen, wat neerkomt op een complexiteit per iteratie van $\mathcal{O}((i * (i+1))/2 + k)$.

5.2 Methode 2: Vectoren

Het idee van de tweede methode is om de matrix in de twee modes te gaan vervangen door een vector in elk van de modes. De volledige tensor zal dus benaderd worden door de som van rang-één tensoren, zoals een CP decompositie. Een rang-één tensor van mode-3 is een tensor die geschreven kan worden als het *outer product* van drie vectoren $\underline{\mathbf{T}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$. Deze decompositie sluit aan bij de rang-1 skelet decompositie van Adaptive Cross Approximation voor matrices. Dit skelet met twee vectoren wordt nu voor tensoren uitgebreid met een vector in de derde mode, de tube fiber. De manier waarop deze vectoren worden gekozen, volgt de manier van ACA, in lijn met de manier van FSTD besproken in Sectie 3.3.

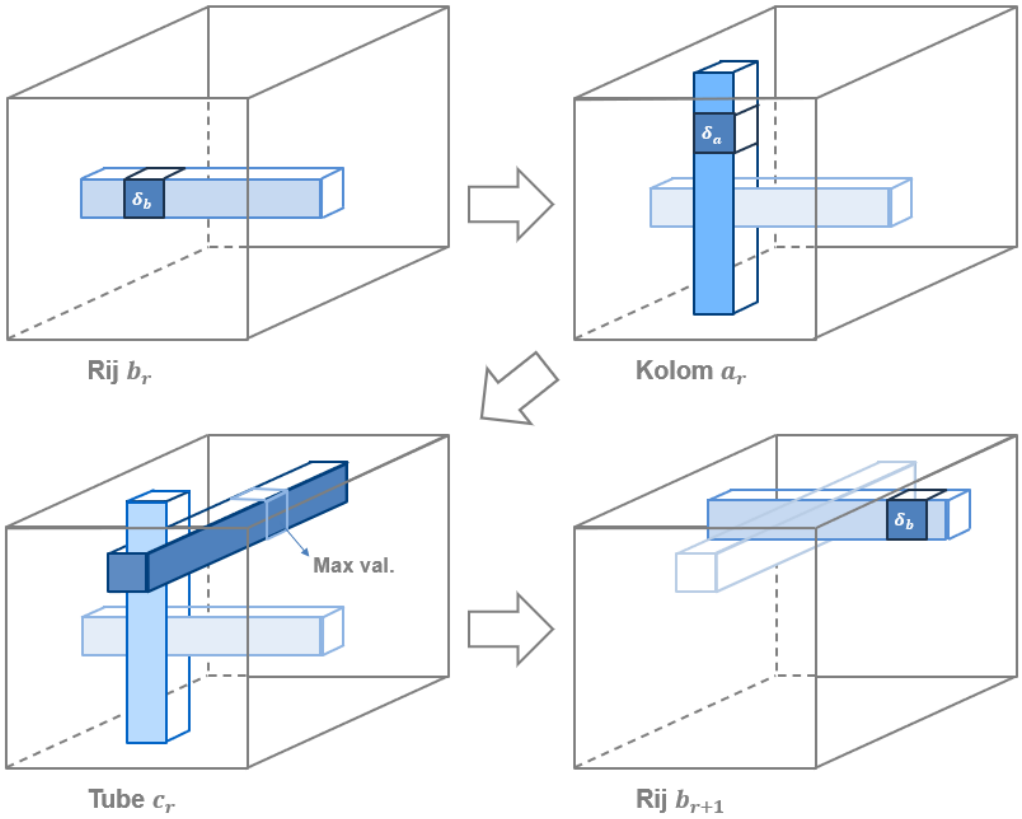
De motivatie voor deze methode is dat werken met vectoren in plaats van een volledige matrix per iteratie minder opslag en minder berekeningen vereist. Door het berekenen van slechts twee vectoren (een rij en een kolom vector) zal in ons geval (tensor $\underline{\mathbf{T}} \in \mathbb{R}^{i\times i\times k}$) de complexiteit per iteratie afnemen van $\mathcal{O}((i * (i+1))/2 + k)$ naar $\mathcal{O}((i+i) + k)$. De hypothese is dat het algoritme meer iteraties zal nodig hebben om een zelfde fout van benadering te bekomen. Het algoritme zal mogelijks een minder nauwkeurigere benadering kunnen bereiken door minder informatie te hebben wanneer de volledige matrix niet exact wordt berekend.

Door het nemen van een kolom fiber $\mathbf{a}$, een rij fiber $\mathbf{b}$ en een tube fiber $\mathbf{c}$ kan er telkens een rang-1 benadering van de tensor $\underline{\mathbf{T}}$ worden opgesteld, namelijk:

$$\underline{\mathbf{T}} \approx \frac{\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}}{\delta_a \delta_b} \tag{5.6}$$

waarbij δ_a en δ_b respectievelijk de grootste waarde zijn uit de gekozen kolom en rij fiber. Dit geeft dan de residu tensor

$$\underline{\mathbf{R}} = \underline{\mathbf{T}} - \frac{\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}}{\delta_a \delta_b} \ . \tag{5.7}$$



FIGUUR 5.3: Een visuele uitwerking van een iteratie van het ACA-T algoritme met vectoren.

Het algoritme start met het nemen van s aantal willekeurige samples uit de tensor $\underline{\mathbf{T}}$. Deze samples worden opgeslagen in lijst S. In lijst S wordt de sample $t_{i,j,k}$ met grootste waarde gezocht. De positie (i, j, k) van de sample in de tensor wordt gebruikt als startpunt voor de eerste iteratie. Figuur 5.3 geeft een iteratie weer zoals hieronder besproken. Elke iteratie r verloopt als volgt:

- De rij $\mathbf{b_r}$ wordt berekend, als zijnde rij $\mathbf{t}_{i,:,k}$ uit residu tensor $\underline{\mathbf{R}}$. In de eerste iteratie is de residu tensor gelijk aan de originele tensor $\underline{\mathbf{T}}$. In deze rij wordt het element met de grootste absolute waarde gezocht, δ_b.

- Van element δ_b wordt nu de kolom $\mathbf{a_r}$ berekend in $\underline{\mathbf{R}}$, kolom $\mathbf{t}_{:,j,k}$. Opnieuw wordt in deze fiber het grootste element gezocht. Dit wordt δ_a en wordt gebruikt om de tube fiber te selecteren.

- Alle elementen in de tube fiber $\mathbf{c_r}$ van element δ_a worden berekend en het element met de grootste absolute waarde wordt gevonden. Van dit element wordt in de volgende iteratie de rij genomen.

- Daarna worden alle samples in lijst S opnieuw geëvalueerd. Dit wordt gedaan door voor elke sample de nieuwe waarde te berekenen en die af te trekken van hun vorige waarde. Samples die nu nul zijn, zijn dus perfect benaderd.
- In de laatste stap wordt er gekeken naar de nieuwe samples. De grootste waarde in de lijst komt overeen met de sample die (tot deze iteratie) het slechtst benaderd is. Wanneer deze waarde groter is dan de maximale waarde gevonden in de tube, zullen we in de volgende iteratie starten in de rij van die sample. Dit wordt gedaan om de belangrijkste skeletten eerst te selecteren, namelijk de rijen, kolommen en tubes waar de benadering nog het slechtst is.

De samples worden verder gebruikt wanneer een rij, kolom of tube van het residu al perfect benaderd is. Wanneer de grootste waarde gevonden in een fiber gelijk is aan nul, weten we dat alle andere elementen ook nul moeten zijn. Dit wijst dan op een reeds perfecte benadering. Het gaan benaderen van deze fiber heeft geen zin aangezien de fiber reeds perfect benaderd is. In de plaats daarvan wordt de grootste sample gekozen en de iteratie opnieuw gestart in de rij van deze sample.

Het algoritme resulteert uiteindelijk in r rij, kolom en tube fibers die de belangrijkste fibers van de tensor representeren. Deze fibers vormen de lage-rang benadering van tensor $\underline{\mathbf{T}}$ en kunnen worden gebruikt om mee te gaan clusteren. Verder kunnen ze gebruikt worden om de benadering van de tensor op te stellen als

$$\underline{\mathbf{T}} \approx \hat{\underline{\mathbf{T}}} = \sum_{r=1}^{R} \frac{1}{\delta_a} \mathbf{a_r} \circ \frac{1}{\delta_b} \mathbf{b_r} \circ \mathbf{c_r} \ . \tag{5.8}$$

Hoofdstuk 6

Experimenten

In dit hoofdstuk worden de experimenten besproken die zijn gebuikt om de onderzoeksvragen te beantwoorden. In sectie 6.1 wordt de AMIE dataset opnieuw kort besproken, met nu meer detail over de inhoud en het gebruik van de set. Daarna wordt er in Sectie 6.2 opnieuw gekeken naar de derde onderzoeksvraag en hiervoor worden de twee methodes met elkaar vergeleken. Uit deze methodes zullen we de vectoren methode kiezen om de cluster experimenten uit te voeren. In Sectie 6.3 wordt er gekeken naar de clusters voor verschillende k-waardes van k-means en of deze clusteringen een bepaalde betekenis hebben. Tenslotte worden die resultaten in Sectie 6.4 gebruikt om het clusteren met behulp van een lage-rang benadering te gaan evalueren.

6.1 Dataset

Zoals reeds aangehaald in Sectie 2.2.2, is de dataset gebruikt voor deze thesis de AMIE dataset. Deze is gemaakt door Decroos et al voor het onderzoek *AMIE: Automatic Monitoring of Indoor Exercises* [6]. De dataset bestaat uit 186 *skeletons*. Elk skeleton komt overeen met een video van één van de tien personen die een bepaalde oefening uitvoert. Normaal zouden er dus per persoon 18 video's zijn (drie soorten oefening, zes mogelijke uitvoeringen per oefening) maar zes video's zijn in twee delen gefilmd door technische problemen en bestaan dus uit minder dan tien repetities. Vandaar dat er 186 video's zijn in plaats van de verwachte 180. Dit vormt geen probleem voor onze toepassing.

Als er wordt gesproken over de volledige tensor, is dit de tensor $\underline{\mathbf{T}} \in \mathbb{R}^{186\times186\times75}$. Hierin wordt dus per sensor elke video (van een uitvoering van een oefening per persoon) met elkaar vergeleken. Elk van deze video's bevat de observatie van 75 verschillende variabelen, de sensoren. Een observatie van een sensor resulteert in de tijdreeks, zoals de tijdreeks van Figuur 2.1. We weten dus van elke tijdreeks bij welke persoon ze hoort en door het uitvoeren van welke oefening ze ontstaan is. Een voorbeeld van de informatie die we voor een persoon hebben, is weergegeven in Tabel 6.1. De getallen van **"Uitvoering"** hebben de volgende betekenis: 1 staat voor de

Video	Persoon	Oefening	Uitvoering
132	person2	squat	1
133	person2	squat	1
134	person2	squat	1
135	person2	squat	2
136	person2	squat	3
137	person2	squat	4
138	person2	lunge	1
139	person2	lunge	1
140	person2	lunge	1
141	person2	lunge	2
142	person2	lunge	3
143	person2	lunge	4
144	person2	sidelunge	1
145	person2	sidelunge	1
146	person2	sidelunge	1
147	person2	sidelunge	4
148	person2	sidelunge	4
149	person2	sidelunge	4

Tabel 6.1: Een voorbeeld van de data beschikbaar voor een persoon (person2).

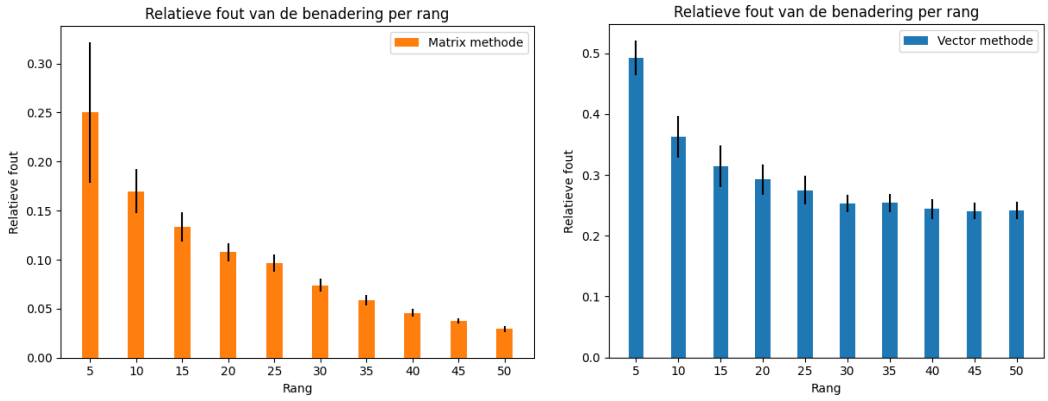
correcte uitvoering, en 2, 3 en 4 zijn de drie verschillende fouten. Voor de sidelunge is er dus maar één type fout dat wordt gemaakt. Deze informatie zal gebruikt worden bij het clusteren, voor het vinden van de echte labels of het verklaren van mogelijke clusteringen.

6.2 Vergelijking methodes

Het eerste experiment dat we zullen doen, is het vergelijken van de twee methodes geïntroduceerd in Hoofdstuk 5. In wat volgt wordt methode 1 de "matrix methode" genoemd en methode 2 de "vectoren methode". Om verwarring te vermijden, zullen we het in de experimenten niet meer hebben over ACA voor matrices (zoals in Sectie 3.2) en verwijst de matrix-methode dus altijd naar methode 1 voor tensoren.

Om de vergelijking te maken, wordt er eerst gekeken naar hoe de relatieve fout bij beide algoritmes afneemt wanneer de rang toeneemt. Het toenemen van de rang resulteert in het gelijktijdig toenemen van het aantal iteraties. Bijgevolg neemt dus ook het aantal gekozen matrices en tube fibers (voor methode 1) en het aantal rijen, kolommen en tubes (voor methode 2) evenredig toe. Elk van deze gekozen structuren zal gebruikt worden voor het opstellen van de benadering van de tensor.

Beide algoritmes worden 50 keer uitgevoerd op de volledige tensor, waarbij de rang r toeneemt van 5 tot 50, in stappen van 5.



(A) Matrix methode (B) Vectoren methode

FIGUUR 6.1: De figuren tonen de relatieve fout per rang, met links (A) het resultaat voor de matrix methode en rechts in (B) hetzelfde voor de vectoren methode. De balkjes zijn telkens het gemiddelde van de uitvoeringen, met de zwarte strepen de standaardafwijking.

Het resultaat is weergegeven in Figuur 6.1. Op de x-as staat de rang geplot in de stappen van 5. Op de y-as geven we de relatieve fout weer. De relatieve fout ϵ wordt berekend als

$$\epsilon = \frac{||\underline{\mathbf{T}} - \hat{\underline{\mathbf{T}}}||}{||\underline{\mathbf{T}}||} \tag{6.1}$$

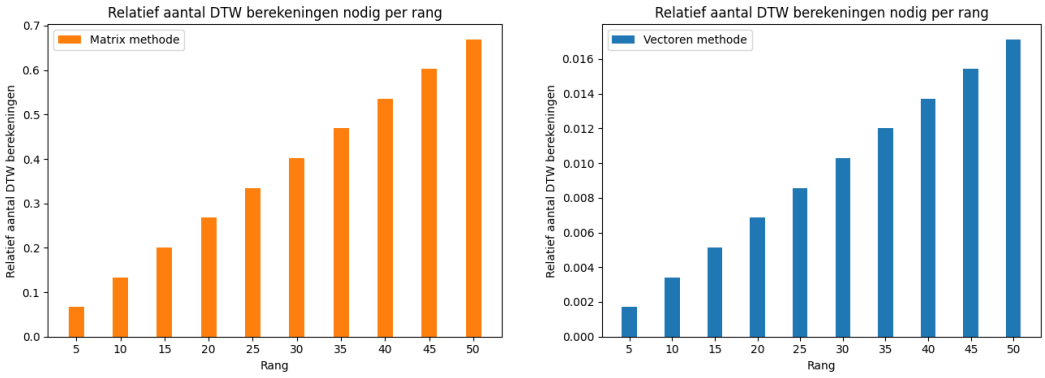
waarbij $\hat{\underline{\mathbf{T}}}$ de benadering is van tensor $\underline{\mathbf{T}}$ en $||X||$ de norm van X zoals geformuleerd in Sectie 2.2.1.

Wat hier meteen opvalt, is dat in beide methodes de relatieve fout ϵ het sterkst daalt in de eerste toenames van de rang r. Dit is te verklaren doordat beide methodes telkens de matrix/vectoren gaan selecteren waar nog de grootste fout in zit (hoogste waardes in het residu) en zo de benadering maximaal kunnen verbeteren (en dus de relatieve fout verlagen). Daarna gaat de methode stagneren. Bij de vectoren methode (6.1b) zal dit sneller gebeuren en zal de relatieve fout gemiddeld rond 0.24 blijven. De matrix methode daarentegen zal een betere benadering opleveren waarbij de relatieve fout bij rang 50 zelfs daalt tot ongeveer 0.03.

De verklaring voor dit grote verschil is te vinden in het aantal elementen dat effectief berekend wordt bij beide methodes. Dit verschil wordt weergegeven in Figuur 6.2. In deze afbeeldingen staat er op de x-as opnieuw de rang in de stappen van 5. Op de y-as staat er nu het relatief aantal DTW berekeningen. Het relatief aantal DTW berekeningen wordt berekend als

$$\#DTW_{relatief} = \frac{\#DTW_{method}}{\#DTW_{volledige\_tensor}} \,. \tag{6.2}$$

In deze formule is $\#DTW_{method}$ het aantal DTW berekeningen nodig om met de methode rang r te bereiken. Dit wordt gedeeld door het aantal DTW berekeningen



(A) Matrix methode (B) Vectoren methode

Figuur 6.2: De figuren tonen het relatief aantal DTW berekeningen per rang, met links (A) het resultaat voor de matrix methode en rechts in (B) hetzelfde voor de vectoren methode.

dat nodig zou zijn om de volledige tensor op te stellen $\#DTW_{volledige\_tensor}$. In het geval van de tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times I \times K}$ zal dit gelijk zijn aan

$$\#DTW_{volledige\_tensor} = \frac{I * (I+1)}{2} * K \ . \tag{6.3}$$

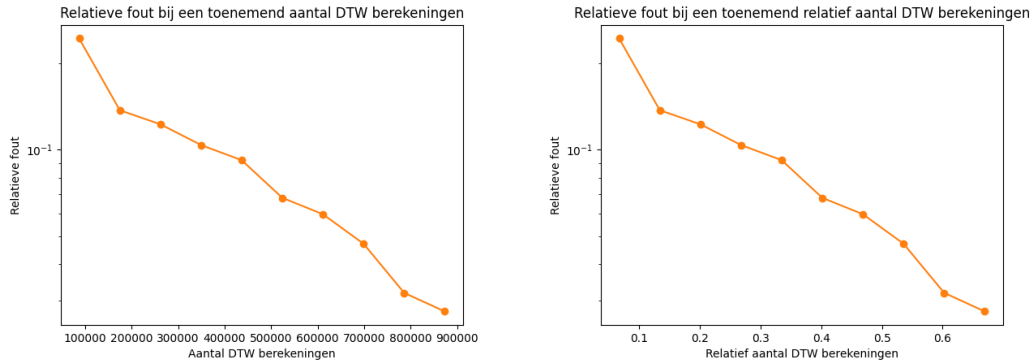
Merk hier op dat de formule niet $I * I * K$ is, omdat elk van de frontale slices een symmetrische afstandsmatrix is. Deze worden telkens berekend door enkel de bovendriehoeksmatrix te berekenen en daarmee de symmetrische matrix op te stellen. Dit resulteert per matrix dus in $\frac{I*(I+1)}{2}$ aantal DTW berekeningen, voor K matrices.

Voor de benadering van rang $r = 5$ gebruikt de matrix methode 6% van de berekeningen die zouden worden gebruikt om de volledige tensor op te stellen. Voor rang 50 gaat dit naar 68%. Als we dan de vectoren methode bekijken, is het belangrijk om te kijken naar het verschil van de schaal op de y-as. Een benadering van rang $r = 5$ gebruikt hier slechts 0.2% van het totaal aantal berekening voor de volledige tensor op te stellen. Dit stijgt naar slechts 1.7% voor rang 50.

Als laatste geven we nog een overzicht door de relatieve fout te plotten in functie van het absolute aantal DTW berekeningen en het relatieve aantal DTW berekeningen voor beide methodes. Voor de matrix methode is dit weergegeven in Figuur 6.3. De resultaten van de vector methode staan eronder in Figuur 6.4.

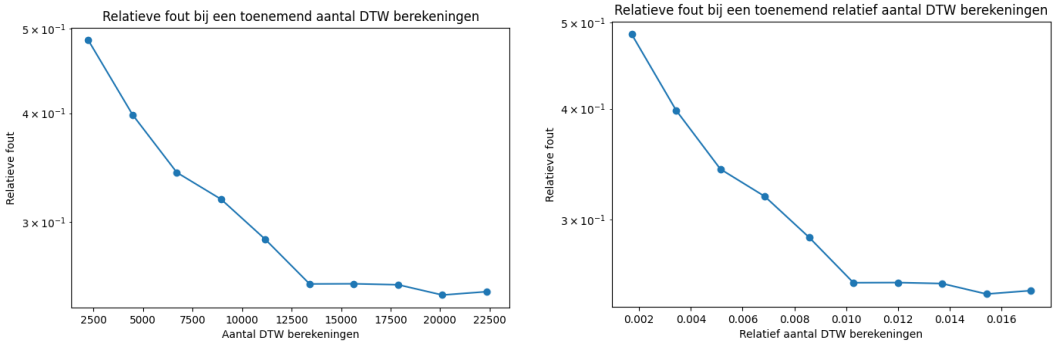
Het belangrijkste om hierbij te vermelden is het verschil in schaal op de assen. Het aantal DTW berekeningen ligt significant hoger bij de matrix methode dan bij de vector methode. Voor een relatieve fout van 0.25 heeft de vector methode 20000 DTW berekeningen nodig. Voor een zelfde relatieve fout heeft de matrix methode bijna 100000 berekeningen nodig, een factor van bijna 5 keer meer.

Dit resultaat volgt de hypothese die we stelden in Sectie 5.2. Methode 2 heeft meer iteraties nodig voor een zelfde fout te kunnen bekomen als methode 1 en



(A) Aantal DTW berekeningen (B) Relatief aantal DTW berekeningen

FIGUUR 6.3: De figuur toont in (A) het aantal DTW berekeningen dat de matrix methode nodig heeft om een bepaalde relatieve fout ϵ te bekomen. Rechts in (B) wordt dit weergegeven met het relatief aantal DTW berekeningen door te delen door het totaal aantal te berekenen elementen voor de volledige tensor.



(A) Aantal DTW berekeningen (B) Relatief aantal DTW berekeningen

FIGUUR 6.4: In (A) wordt het aantal DTW berekeningen dat de matrix methode nodig heeft om een bepaalde relatieve fout ϵ te bekomen getoond. Rechts in (B) wordt dit weergegeven met het relatief aantal DTW berekeningen door te delen door het totaal aantal te berekenen elementen voor de volledige tensor.

resulteert in een minder nauwkeurige benadering. Daar staat natuurlijk tegenover dat methode 2 een significant aantal minder berekeningen moet doen per iteratie. Voor grote datasets is de tweede methode dus zeker meer geschikt. Daarom zullen we deze methode gebruiken voor de cluster experimenten, om te kijken of er met dit niveau van benadering een bruikbare clustering kan worden bekomen.

6.3 Interpretatie clusters

De belangrijkste parameter om de resultaten van het k-means algoritme te kunnen interpreteren, is de waarde van k. De hoeveelheid clusters die wordt gezocht in de

feature vectoren zal tot verschillende mogelijke clusteringen kunnen leiden. Daarom zullen we in de volgende experimenten een aantal k-waardes kiezen en kijken wat het resultaat van de clustering oplevert. Met behulp van de informatie die gekend is over de AMIE dataset kan dan gekeken worden of de clustering al dan niet waardevolle informatie oplevert.

Het k-means algoritme zal, zoals besproken in Sectie 2.3.2, lopen op de feature vectoren bekomen door een lage-rang benadering te maken van de afstandstensor $\underline{\mathbf{T}}$. Om rechtstreeks te kunnen werken met de rijen en tubes wordt ACA-T van methode 2 gebruikt, de vectoren methode. Het algoritme wordt uitgevoerd met rang $r = 30$ en de resultaten worden opgeslagen en volledig weergegeven in Appendix A.

Als feature vectoren kunnen we zowel de rij fibers als de tube fibers gebruiken. Het verschil is dat we voor de rij fibers gaan clusteren op personen en oefeningen, terwijl we met behulp van de tube fibers de clustering gaan maken op de sensoren. Omdat ze beide interessante use-cases kunnen hebben, worden de beide opties als feature vectoren besproken.

De experimenten worden gedaan op de volledige tensor $\underline{\mathbf{T}} \in \mathbb{R}^{186\times186\times75}$. Bij het gebruik van de rijen zullen de feature vectoren dus van lengte 186 zijn, voor de tubes hebben ze een lengte van 75. Elk van de elementen in de feature vectoren komt overeen met een te clusteren object. Het aantal feature vectoren wordt bepaald door het aantal gekozen rijen, dat gelijk is aan de rang r (hier $r = 30$). Hieronder worden beide opties besproken voor een aantal waardes van k. De waardes die niet worden besproken, zijn weggelaten omdat ze gelijkende interpretaties hebben met de andere waardes of omdat er geen bruikbare interpretatie gevonden werd.

6.3.1 Rijen als feature vectoren

Als de feature vectoren bestaan uit de rijen verkregen na ACA-T, zullen we gaan clusteren in de dimensie van de personen en oefeningen. De gevormde clusters na het uitvoeren van k-means met verschillende waardes k is weergegeven in Tabel A.1. In de eerste kolom staat welke video het is (afgekort naar "Vid."). Voor elke video weten we dus over welke persoon, oefeningen en uitvoering van de oefening het gaat. Dit zullen we proberen te gebruiken om aan de gevonden clusters een betekenis te geven.

k = 2: Wanneer de video's worden verdeeld in twee clusters, zien we dat de *squat* en *lunge* oefening samen geclusterd worden. Deze oefeningen zullen dus meer op elkaar gelijken dan op de *sidelunge*. Dit zullen we nog zien terugkomen in andere k-waardes, waar soms foutief squats als lunges worden geclusterd en omgekeerd.

k = 3: Bij drie clusters zien we meteen de onderverdeling in de drie verschillende oefeningen. Elk van de clusters omvat één oefening, op een paar fouten na. In Tabel 6.2 is een deel van de volledige tabel uit Appendix A.1 weergegeven. Hier is te zien voor $k = 3$ dat de squat oefening gelabeld wordt met 2, de lunge met 1 en de

TABEL 6.2: De cluster resultaten van k-means voor verschillende k-waardes, snippet uit de volledige Tabel A.1 met alleen de video's van person1.

Vid.	Persoon	Oefening	Uitv.	k=2	k=3	k=6	k=9	k=12
20	person1	squat	1	1	2	2	7	9
21	person1	squat	1	1	2	2	2	11
22	person1	squat	1	1	2	2	7	9
23	person1	squat	2	1	2	2	2	11
24	person1	squat	3	1	2	2	7	9
25	person1	squat	4	1	2	2	7	9
26	person1	lunge	1	1	1	3	3	10
27	person1	lunge	1	1	1	3	3	10
28	person1	lunge	1	1	1	3	3	10
29	person1	lunge	2	1	1	3	3	3
30	person1	lunge	3	1	1	3	3	10
31	person1	lunge	4	1	1	3	3	10
32	person1	sidelunge	1	0	0	0	0	0
33	person1	sidelunge	1	0	0	0	0	0
34	person1	sidelunge	1	0	0	0	0	0
35	person1	sidelunge	4	0	0	0	0	0
36	person1	sidelunge	4	0	0	0	0	0
37	person1	sidelunge	4	0	0	0	0	0
38	person1	sidelunge	4	0	0	0	0	0

sidelunge krijgt cluster 0.

k = 6: Voor hogere waardes van k wordt het al snel moeilijker om een bepaalde interpretatie te geven aan de clusters. Zo zien we dat in de meeste gevallen alle vier de mogelijke uitvoeringen van een oefening nog steeds in dezelfde cluster terecht komen. Het onderscheid wordt gemaakt op niveau van personen, waar bijvoorbeeld de sidelunge van person1 in cluster 0 terecht komt, maar de sidelunge van person10 in cluster 4 wordt geplaatst.

k = 9: Bij $k = 9$ zien we voor het eerst ook onderverdelingen binnen een oefening. Dit gebeurt voornamelijk bij lunges en squats. De verdeling gebeurt echter niet volgens bepaalde uitvoeringen. De clustering leidt niet tot verdere interpretaties.

k = 12: K-means met 12 clusters geeft een clustering die erg gelijkt op de clustering met $k = 9$. Over het algemeen wordt er geclusterd op oefeningen. Er lijkt geen verder verband te zijn tussen de personen en de clusters.

Informatie halen uit de clusters ontstaan door k groter te nemen dan 3 is zeer moeilijk. Vanaf $k = 3$ zal de clustering gebaseerd worden op de soort oefening, en niet zozeer op de verschillende personen. Deze informatie zal worden gebruikt om de

Tabel 6.3: De cluster resultaten van k-means voor verschillende k-waardes, snippet uit de volledige Tabel A.2 met slechts 6 van de 75 sensoren.

Sensor	**k=3**	**k=4**	**k = 6**	**k =7**	**k=9**	**k=15**
FootRightX	2	2	2	2	2	13
FootRightY	0	0	0	0	0	12
FootRightZ	1	1	4	4	4	9
HandLeftX	2	2	2	2	7	7
HandLeftY	0	0	0	6	5	5
HandLeftZ	1	3	3	3	3	11

clusters te gaan evalueren die worden bekomen met de lage-rang benadering in het volgende experiment (Sectie 6.4).

6.3.2 Tubes als feature vectoren

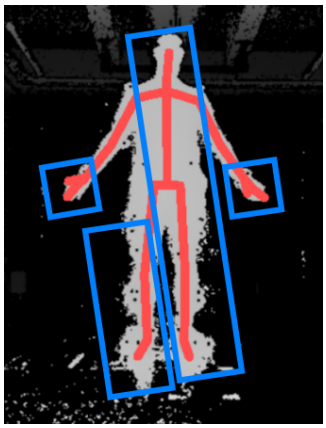
Door het nemen van de tubes als feature vectoren wordt er geclusterd in de dimensie van de sensoren. De resultaten van de clusteringen zijn weergegeven in Appendix A.2. In Tabel A.2 staan in de eerste kolom de verschillende sensoren. De naam bestaat telkens uit de plaats van de sensor (bijvoorbeeld "AnkleLeft") en de as volgens welke de sensor de beweging meet (X-, Y- of Z-as). Aan de hand van deze twee kenmerken proberen we hieronder de gevonden clusteringen te verklaren.

k = 3: De clustering bekomen na k-means met $k = 3$ heeft een duidelijke betekenis. Elke sensor die meet volgens de y-as zit in cluster 0, de sensoren die meten volgens de z-as zitten in cluster 1 en cluster 2 is gevuld met de sensoren metend volgens de x-as. Dit is te zien in Tabel 6.3.

k = 4: Bij het clusteren in vier clusters blijven cluster 0 en 2 hetzelfde (sensoren volgens y- en x-as) maar de sensoren die meten volgens de z-as worden gesplitst. In cluster 3 worden de sensoren op en rond de hand geplaatst ("Hand", "HandTip", "Thumb" en "Wrist") terwijl de rest in cluster 1 blijft.

k = 6: Voor $k = 6$ zullen ook de sensoren volgens de x-as worden gesplitst. Op het eerste zicht lijkt hier geen bepaalde verklaring voor te zijn, omdat cluster 2 en 5 door elkaar worden gebruikt (zie bijvoorbeeld in Tabel 6.3). Bij grondigere observatie van de volledige tabel valt het op dat er 7 van de 10 "Left" sensoren (op de linkerkant van het lichaam) in cluster 2 vallen, en 7 van de 10 "Right" sensoren in cluster 5. De centrale sensoren (zonder "Left" of "Right", bijvoorbeeld "Head") worden voornamelijk bij de "Left" categorie geclusterd.

De z-as sensoren worden onderverdeeld in drie verschillende clusters. Er is een cluster voor de sensoren op het onderlichaam (voet tot heup en ruggengraat, cluster 4) en een voor het bovenlichaam (schouders tot hoofd, cluster 1). De armen worden apart geclusterd (elleboog tot vingertop, cluster 3).



FIGUUR 6.5: De verdeling van de clusters volgens de x-as, op de persoon gegenereerd door de Kinect camera uit het AMIE onderzoek [6].

k = 7: De enige verandering bij $k = 7$ is een onderverdeling in de y-as sensoren. Hier worden nu ook de sensoren op en rond de handen samen in een aparte cluster geplaatst.

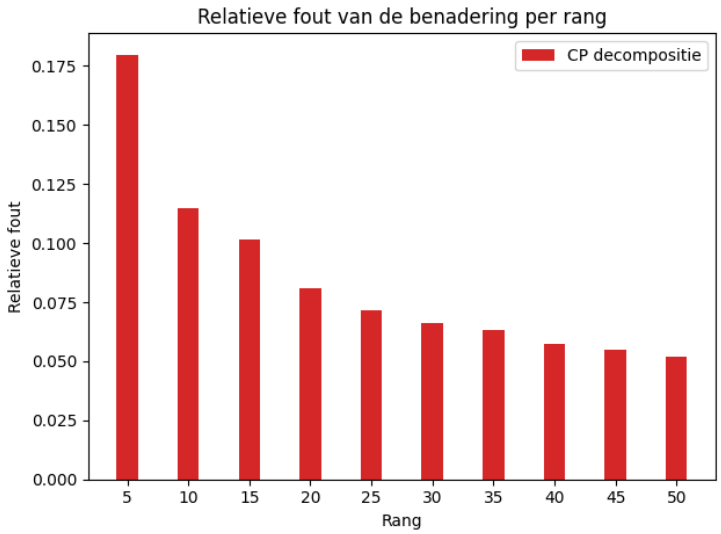
k = 9: Bij negen clusters wordt de onderverdeling minder duidelijk. Volgens elk van de assen is er een aparte cluster voor de handen. Voor de x-as is de cluster verdeling weergegeven in Figuur 6.5.

k = 15: Als laatste testen we een run met veel clusters ($k = 15$). De interpretatie van deze resultaten is veel moeilijker. Het lijkt er voornamelijk op dat de clusteringen zich meer en meer rond een bepaalde zone van het lichaam gaan bevinden, en dit voor elk van de assen.

Het interpreteren van de clusteringen, opgeleverd door k-means te laten lopen op de tube fibers, is dus meer voor de hand liggend dan met de rij fibers. De betekenis van de verschillende clusters zal toe te wijzen zijn aan de as volgens welke de sensor beweegt en de plaats van de sensoren op het lichaam. Vanaf $k = 3$ wordt er het onderscheid gemaakt tussen de drie verschillende assen. Bij een toenemende aantal clusters k zullen er daarna volgens die assen onderscheid gemaakt worden tussen de plaatsen waar de sensoren zijn aangebracht op het lichaam. Dit kan gaan van de linker- of rechterkant ("Left" en "Right") van het lichaam tot clusters op en rond specifieke lichaamsdelen (bijvoorbeeld de handen en polsen of de voeten en enkels).
Beide

6.4 Clusteren vanuit lage-rang benadering

In dit experiment komen we terug op onze hoofdvraag, namelijk of het mogelijk is om met de lage-rang benadering een clustering te verkrijgen die gelijkend is aan de clustering gemaakt met de volledige tensor. Op het einde van dit experiment zal met



Figuur 6.6: De figuur toont de relatieve fout per rang voor de benadering met behulp van de CP decompositie.

behulp van het experiment uit 6.2 ook de tweede vraag beantwoord worden.

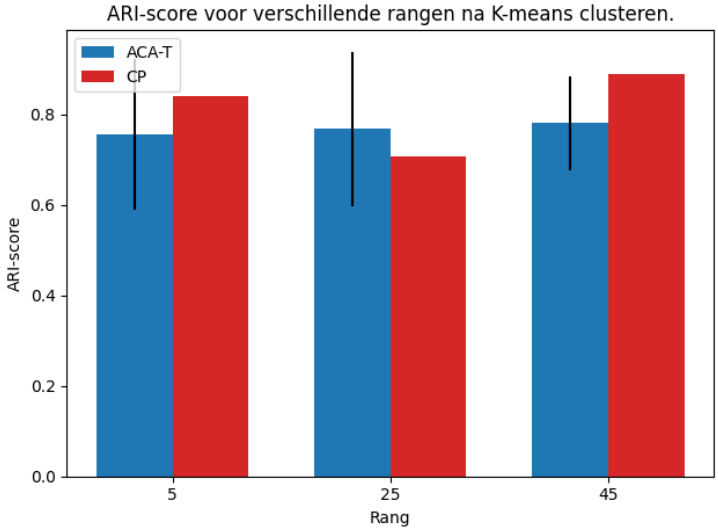
Om de verkregen clustering vanuit de lage-rang benadering te kunnen vergelijken, hebben we een methode nodig die vertrekt vanuit de volledige tensor. Hiervoor kiezen we de Candecomp/Parafac decompositie via het ALS (Alternating Least Squares) algoritme, geïmplementeerd in de TensorLy bibliotheek [10]. Dit algoritme zal bij een toenemende rang een evenredig aantal fibers hebben in elk van de modes (bijvoorbeeld voor rang $r = 5$ zal de CP decompositie bestaan uit 5 rij fibers, 5 kolom fiber en 5 tube fibers). In Figuur 6.6 is te zien hoe de relatieve fout afneemt naarmate de rang toeneemt. Deze vertrekt echter vanuit de volledige tensor $\underline{\mathbf{T}} \in \mathbb{R}^{I \times I \times K}$ en heeft dus, onafhankelijk van de rang, altijd

$$\mathcal{O}(\frac{(i * (i + 1))}{2} * k) \tag{6.4}$$

DTW berekeningen nodig omdat eerst de volledige tensor wordt opgesteld.

Voor het experiment zullen we de ARI-score gebruiken als kwantitatieve methode om de overeenkomst van de clusters met de *ground truth* te vergelijken. De ground truth zijn de echte labels die de te clusteren objecten zouden moeten krijgen. Deze worden gebaseerd op de resultaten in het experiment 6.3. Zo zullen we voor k-means met $k = 3$ met de rijen als feature vectoren de echte labels gelijkstellen aan de drie types van oefeningen. Met de tubes als feature vectoren en $k = 3$ nemen we de verschillende assen (X, Y, Z) als cluster labels.

In het eerste experiment zullen we de clustering op de rijen met elkaar vergelijken. Hiervoor laten we beide algoritmes lopen met de rang gaande van 5 tot 45, in stappen



FIGUUR 6.7: De ARI-scores van het ACA-T algoritme met vectoren naast die van de CP decompositie, weergegeven in functie van het aantal **rijen** gebruikt als feature vectoren (= rang).

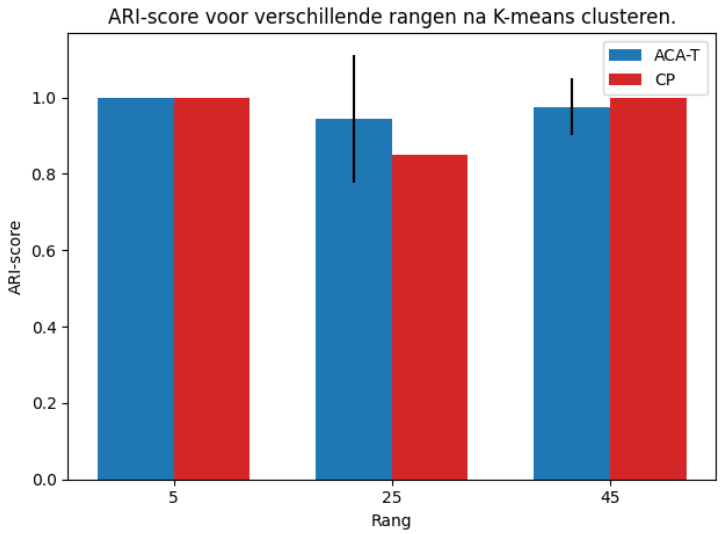
van 20. Voor beide algoritmes wordt de tensor ontbonden en daarna worden de rijen gebruikt als input (feature vectoren) voor het k-means algoritme, met $k = 3$. Voor het ACA-T algoritme nemen we het gemiddelde van 10 uitvoeringen. De resultaten zijn weergegeven in Figuur 6.7.

Het eerste dat opvalt is dat de ARI-score van het ACA-T algoritme lichtjes stijgt wanneer de rang toeneemt. De verklaring hiervoor is dat wanneer de rang toeneemt, er meer feature vectoren zullen zijn en het clusteralgoritme meer informatie heeft om te vergelijken. Ook zal bij toenemnde rang de benadering verbeteren (de relatieve fout verkleint, zie Figuur 6.1). De score ligt ook telkens op 0.75 of hoger, wat erop wijst dat het clusteren met de rijen als feature vectoren veel van de video's correct clustert en dus een redelijk goed resultaat geeft.

Daarnaast zien we dat de kwaliteit van de clusters (met behulp van de ARI-score) van het ACA-T algoritme niet veel verschilt met die van de CP decompositie. Voor rang 5 en 45 zit er een verschil op de ARI-score van ongeveer 0.05. Voor rang 25 lijkt het ACA-T algoritme gemiddeld zelfs beter te presteren.

In Figuur 6.8 zien we het resultaat van hetzelfde experiment als hierboven, maar nu op de tubes in plaats van de rijen. Opvallend is hoe beide methodes voor lage rang ($r = 5$) een perfecte clustering maken met ARI-score van 1. Ook voor rang 25 en 45 is deze score hoog, met ACA-T telkens boven 0.95. De scores liggen voor deze feature vectoren weer dicht bij elkaar.

Tenslotte willen we nog kijken naar een "moeilijkere" clustering. Voor de rijen als

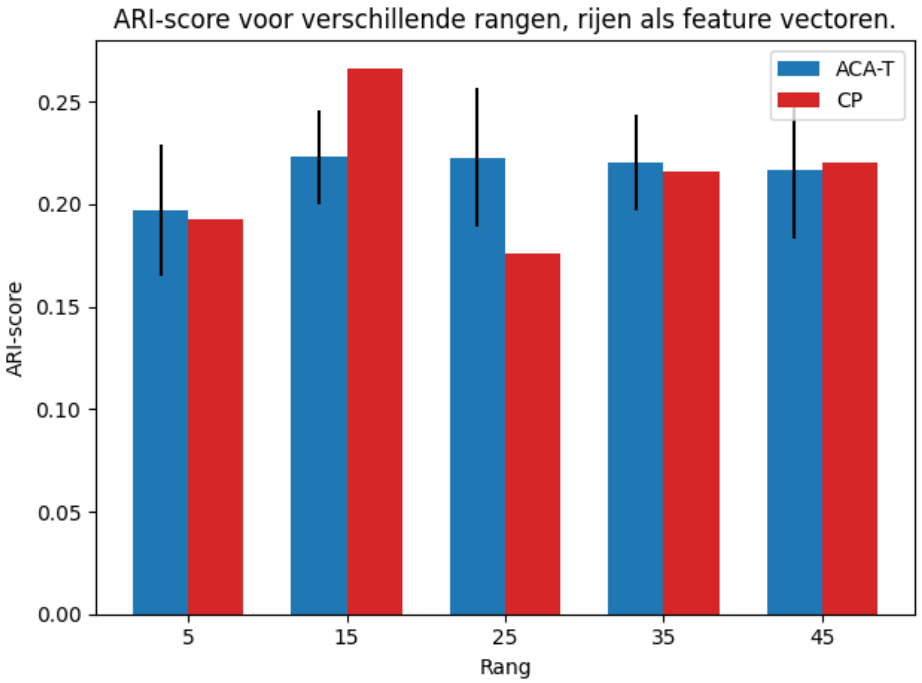


Figuur 6.8: De ARI-scores van het ACA-T algoritme met vectoren naast die van de CP decompositie, weergegeven in functie van het aantal **tubes** gebruikt als feature vectoren (= rang).

feature vectoren nemen we de echte labels overeenkomend met de soort uitvoering van de fout. We zullen de correcte uitvoering van elke oefening apart nemen, en de type fout telkens bundelen. Zo krijgen we zes clusters: correcte uitvoering van squat, correcte uitvoering van lunge, correcte uitvoering van sidelunge, en foute uitvoering type 2, type 3 en type 4. We weten uit het experiment in 6.3 dat deze clustering niet de meest voor de hand liggende clusters geeft, maar we willen toch even kijken wat de beide methodes hier mee doen.

De resultaten zijn weergegeven in Figuur 6.9. Hier is te zien dat voor elke rang de score van het ACA-T algoritme schommelt tussen 0.19 en 0.23. Zoals verwacht is dit een eerder lage score. We toonden eerder aan in het vorige experiment dat het type oefening makkelijker te onderscheiden was dan het type uitvoering. Hier wordt dit nog eens duidelijk weergegeven door het verschil in ARI-scores tussen Figuur 6.9 en 6.7. De kwaliteit van de clusters komt weer sterk overeen tussen ACA-T en CP. Dit is te zien in de kleine verschillen tussen de scores van beide methodes.

In de experimenten hierboven wordt aangetoond dat het ACA-T algoritme goede feature vectoren kan vinden om k-means op toe te passen. Dit is te zien aan de ARI-scores die richting 1 gaan. Hoge ARI-scores duiden erop dat clusters die worden gevonden komen sterk overeen met de ground truth clusters. Ook zien we in deze experimenten dat de kwaliteit van de clusters op de lage-rang benadering van ACA-T sterk overeen komt met de kwaliteit van de clusters van de CP decompositie. Met deze informatie kunnen we de hoofdvraag beantwoorden. De clustering met behulp van de tensor factorisatie gaat gelijkend zijn aan de clustering vanuit de volledige



FIGUUR 6.9: De ARI-scores na k-means clustering met $k = 6$. Vergelijking tussen de rijen van de ACA-T en de CP decomposities als feature vectoren.

tensor.

Een belangrijke kanttekening om hierbij te maken is dat de kwaliteit van de clusters sterk afhangt van wat we proberen te clusteren. Clusters op basis van verschillende types oefening en de verschillende assen waar de sensor volgens meet, geven goede resultaten. Een andere clustering zoals bijvoorbeeld het type uitvoering van de oefening resulteert in een veel lagere ARI-score. Deze clusters zullen dus niet zo goed gevonden worden met behulp van tensor factorisaties. Dit is het geval voor zowel de ACA-T methode als de CP decompositie.

Met deze bevindingen en de resultaten uit 6.2, kan ook de tweede onderzoeksvraag worden beantwoord. Een clustering opstellen voor de volledige tensor $\underline{\mathbf{T}} \in \mathbb{R}^{186\times186\times75}$ zal voor rang 45 van het ACA-T algoritme met vectoren slechts 1.6% van het aantal DTW berekeningen nodig hebben ten opzichte van de volledige tensor. Voor een kostelijke berekening als DTW (kwadratisch in de lengte l van de tijdreeksen, $\mathcal{O}(l^2)$) zal deze factor van meer dan 50 keer minder dus een zeer grote versnelling geven.

Hoofdstuk 7

Besluit

In dit hoofdstuk volgen nog eens kort de conclusies die getrokken kunnen worden uit de thesis, gevolgd door een paar suggesties voor mogelijk verder onderzoek.

7.1 Conclusies

Het opbouwen van een tensor $\underline{\mathbf{T}}$ van grootte $\mathbb{R}^{I \times I \times K}$ met symmetrische afstandsmatrices als frontale slices zal

$$\mathcal{O}(\frac{(I * (I+1))}{2} * K) \tag{7.1}$$

DTW berekeningen nodig hebben. Door in de plaats daarvan de tensor te gaan benaderen zonder deze volledig uit te rekenen, kunnen we deze kost verlagen naar $\mathcal{O}(((I * (I+1))/2 + K) * R)$ door te werken met een matrix $\times$ vector ontbinding (matrix methode, methode 1). Deze kost kan zelfs nog verder verlaagd worden tot $\mathcal{O}((I + I) + K) * R)$ door te werken met een ontbinding in drie vectoren (vectoren methode, methode 2). In beide formules is de hoeveelheid DTW berekeningen afhankelijk van de gekozen rang R.

De twee voorgestelde methodes gaan de volledige tensor benaderen met behulp van een generalisatie van het Adaptive Cross Approximation algoritme. Beide methodes gaan per iteratie de elementen met de grootste fout zoeken in het residu van de tensor. Het residu is telkens de originele tensor min de huidige benadering. Methode 1 doet dit door van het grootste element telkens de frontale matrix en de tube fiber te nemen. Methode 2 zal de grootste elementen gebruiken voor het kiezen van een rij, kolom en tube fiber in elke iteratie. Het belangrijkste voor deze methodes is dat het volledige residu niet wordt uitgerekend, maar enkel de volgende matrix/fiber die wordt gekozen. Dit resulteert in de sterke afname van het aantal nodige berekeningen (volledige tensor tegenover slechts een frontale slice, of slechts een fiber). Beide implementaties zijn terug te vinden als Python implementatie in de softwarebibliotheek [18].

In de experimenten wordt vervolgens aangetoond dat beide methodes de volledige tensor goed gaan benaderen. Voor de matrix methode gaat bij toenemende rang

deze relatieve fout erg klein worden. Bij een hoge rang ($r = 50$) zullen er echter al bijna 70% van de hoeveelheid DTW berekeningen van de totale tensor nodig zijn. Bij de vectoren methode zal dit voor rang 50 slechts 1.7% zijn. De relatieve fout ligt bij deze methode wel hoger. Door het lagere aantal berekeningen en de mogelijkheid om rechtstreeks te gaan clusteren met de vectoren in de ontbinding, wordt de tweede methode gebruikt om verder experimenten op de clusteringen te gaan doen.

Alvorens een vergelijking te kunnen maken tussen clusters gegenereerd door een lage-rang benadering of een decompositie vanuit de volledige tensor, hebben we eerst gekeken naar de betekenis achter clusters. Hier ondervonden we dat k-means clusteren het best ging werken met de waarde van $k = 3$. Bij het verhogen van die waarde k waren de clusters niet meer duidelijk te interpreteren. Als input voor het clusteren gebruikten we dan zowel de rijen als de tubes van de ontbindingen, met verschillende clusteringen als resultaat. We konden concluderen dat, zolang er rekening wordt gehouden met wat er precies wordt geclusterd, de clustering met een lage-rang benadering van de tensor goede resultaten opleverde.

7.2 Verder onderzoek

Tijdens het zoeken naar literatuur en het implementeren van de methodes had ik de indruk dat lage-rang benaderingen van tensoren en methodes om dit te doen nog maar zeer beperkt aanwezig zijn. Er is dus nog veel onderzoek te doen op dit vlak van tensor benaderingen en deze thesis raakt slechts een klein stuk van alle mogelijke toepassingen.

In deze thesis hebben we twee manieren uitgekozen om met het ACA-T algoritme de tensor te gaan ontbinden. Hier zijn nog vele andere opties mogelijk. Het aantal berekeningen van de matrix methode kan bijvoorbeeld worden verminderd door de frontale matrices op hun beurt te gaan benaderen met ACA voor matrices. Deze ontbinding kan dan op zijn beurt mogelijks gebruikt worden om ook met de matrix manier een rechtstreekse clustering op te gaan stellen.

Verder kan er ook een uitgebreider onderzoek worden gedaan naar welke clusteringen er al dan niet mogelijk zijn met de tensor factorisaties. De methodes kunnen misschien ook verder geoptimaliseerd worden om sneller betere benaderingen te bekomen.

Bijlagen

Bijlage A

Resultaten voor interpretaties clustering

In deze bijlage staan de resultaten die gebruikt zijn om de interpretaties van de clustering te maken voor verschillende k-waardes.

A.1 ACA-T met rijen als feature vectoren

Tabel A.1: De resultaten van de k-means clustering met de rijen als feature vectoren en verschillende waardes van k.

Vid.	**Persoon**	**Oefening**	**Uitv.**	**k=2**	**k=3**	**k=6**	**k=9**	**k=12**
1	person8	squat	1	1	2	5	2	11
2	person8	squat	1	1	2	5	2	11
3	person8	squat	1	1	2	5	6	6
4	person8	squat	2	1	1	5	6	6
5	person8	squat	3	1	2	5	6	6
6	person8	squat	4	1	2	5	6	6
7	person8	lunge	1	1	1	3	8	8
8	person8	lunge	1	1	1	3	8	8
9	person8	lunge	1	1	1	3	8	8
10	person8	lunge	1	1	1	3	8	8
11	person8	lunge	2	1	1	3	8	8
12	person8	lunge	3	1	1	3	8	8
13	person8	lunge	4	1	1	3	8	8
14	person8	sidelunge	1	0	1	1	1	1
15	person8	sidelunge	1	0	1	1	1	1
16	person8	sidelunge	1	0	1	1	1	1
17	person8	sidelunge	4	0	1	1	1	1
18	person8	sidelunge	4	0	1	1	1	1
19	person8	sidelunge	4	1	1	1	1	1

20	person1	squat	1	1	2	2	7	9
21	person1	squat	1	1	2	2	2	11
22	person1	squat	1	1	2	2	7	9
23	person1	squat	2	1	2	2	2	11
24	person1	squat	3	1	2	2	7	9
25	person1	squat	4	1	2	2	7	9
26	person1	lunge	1	1	1	3	3	10
27	person1	lunge	1	1	1	3	3	10
28	person1	lunge	1	1	1	3	3	10
29	person1	lunge	2	1	1	3	3	3
30	person1	lunge	3	1	1	3	3	10
31	person1	lunge	4	1	1	3	3	10
32	person1	sidelunge	1	0	0	0	0	0
33	person1	sidelunge	1	0	0	0	0	0
34	person1	sidelunge	1	0	0	0	0	0
35	person1	sidelunge	4	0	0	0	0	0
36	person1	sidelunge	4	0	0	0	0	0
37	person1	sidelunge	4	0	0	0	0	0
38	person1	sidelunge	4	0	0	0	0	0
39	person6	squat	1	1	2	5	6	6
40	person6	squat	1	1	2	5	6	6
41	person6	squat	1	1	2	5	6	6
42	person6	squat	2	1	2	5	6	6
43	person6	squat	3	1	2	5	6	6
44	person6	squat	4	1	2	5	6	6
45	person6	lunge	1	1	1	3	8	8
46	person6	lunge	1	1	1	3	3	3
47	person6	lunge	1	1	1	3	3	3
48	person6	lunge	1	1	1	3	3	3
49	person6	lunge	2	1	1	3	3	3
50	person6	lunge	3	1	1	3	3	3
51	person6	lunge	4	1	1	3	3	3
52	person6	sidelunge	1	0	0	0	5	5
53	person6	sidelunge	1	0	0	0	5	5
54	person6	sidelunge	1	0	0	0	5	5
55	person6	sidelunge	4	0	0	0	5	5
56	person6	sidelunge	4	0	0	0	5	5
57	person6	sidelunge	4	0	0	0	5	5
58	person9	squat	1	1	2	2	2	11
59	person9	squat	1	1	2	2	2	11
60	person9	squat	1	1	2	2	2	11
61	person9	squat	2	1	2	5	2	6

62	person9	squat	3	1	2	5	2	11
63	person9	squat	4	1	2	2	2	2
64	person9	lunge	1	1	1	3	8	8
65	person9	lunge	1	1	1	3	8	10
66	person9	lunge	1	1	1	3	8	10
67	person9	lunge	2	1	1	3	3	10
68	person9	lunge	3	1	1	3	8	10
69	person9	lunge	4	1	1	3	3	10
70	person9	sidelunge	1	0	0	0	0	0
71	person9	sidelunge	1	0	0	0	5	5
72	person9	sidelunge	1	0	0	0	5	5
73	person9	sidelunge	1	0	0	0	5	5
74	person9	sidelunge	4	0	0	0	5	5
75	person9	sidelunge	4	0	0	0	5	5
76	person9	sidelunge	4	0	0	0	5	5
77	person4	squat	1	1	2	2	2	2
78	person4	squat	1	1	2	2	2	2
79	person4	squat	1	1	2	2	2	2
80	person4	squat	2	1	2	2	2	2
81	person4	squat	3	1	2	2	2	2
82	person4	squat	4	1	2	2	2	2
83	person4	lunge	1	1	1	3	8	10
84	person4	lunge	1	1	1	3	8	10
85	person4	lunge	1	1	1	3	8	10
86	person4	lunge	2	1	1	3	3	10
87	person4	lunge	3	1	1	3	8	10
88	person4	lunge	4	1	1	3	3	10
89	person4	lunge	4	1	1	3	8	10
90	person4	sidelunge	1	0	0	0	0	0
91	person4	sidelunge	1	0	0	0	0	0
92	person4	sidelunge	1	0	0	0	0	0
93	person4	sidelunge	4	0	0	0	0	0
94	person4	sidelunge	4	0	0	0	0	0
95	person4	sidelunge	4	0	0	0	0	0
96	person10	squat	1	1	2	5	6	6
97	person10	squat	1	1	2	5	6	6
98	person10	squat	1	1	2	5	6	6
99	person10	squat	2	1	2	5	6	6
100	person10	squat	3	1	2	5	6	6
101	person10	squat	4	1	2	5	6	6
102	person10	lunge	1	1	1	3	8	8
103	person10	lunge	1	1	1	3	8	8

104	person10	lunge	1	1	1	3	3	3
105	person10	lunge	2	1	1	3	3	3
106	person10	lunge	3	1	1	3	3	3
107	person10	lunge	4	1	1	3	3	3
108	person10	sidelunge	1	0	0	4	4	4
109	person10	sidelunge	1	0	0	4	4	4
110	person10	sidelunge	1	0	0	4	4	4
111	person10	sidelunge	4	0	0	4	4	4
112	person10	sidelunge	4	0	0	4	4	4
113	person10	sidelunge	4	0	0	4	4	4
114	person3	squat	1	1	2	2	7	7
115	person3	squat	1	1	2	2	7	7
116	person3	squat	1	1	2	2	7	9
117	person3	squat	2	1	2	2	7	7
118	person3	squat	3	1	2	2	7	7
119	person3	squat	4	1	2	2	7	7
120	person3	lunge	1	1	1	3	3	3
121	person3	lunge	1	1	1	3	3	3
122	person3	lunge	1	1	1	3	3	3
123	person3	lunge	2	1	1	3	3	3
124	person3	lunge	3	1	1	3	3	3
125	person3	lunge	4	1	1	3	3	3
126	person3	sidelunge	1	0	0	4	4	4
127	person3	sidelunge	1	0	0	4	4	4
128	person3	sidelunge	1	0	0	4	4	4
129	person3	sidelunge	4	0	0	4	4	4
130	person3	sidelunge	4	0	0	4	4	4
131	person3	sidelunge	4	0	0	4	4	4
132	person2	squat	1	1	2	2	7	7
133	person2	squat	1	1	2	2	7	7
134	person2	squat	1	1	2	2	7	7
135	person2	squat	2	1	2	2	7	7
136	person2	squat	3	1	2	2	7	7
137	person2	squat	4	1	2	2	7	7
138	person2	lunge	1	1	1	3	3	3
139	person2	lunge	1	1	1	5	6	6
140	person2	lunge	1	1	1	5	6	6
141	person2	lunge	2	1	1	5	6	6
142	person2	lunge	3	1	1	5	6	6
143	person2	lunge	4	1	1	5	6	6
144	person2	sidelunge	1	0	0	4	4	4
145	person2	sidelunge	1	0	0	4	4	4

146	person2	sidelunge	1	0	0	4	4	4
147	person2	sidelunge	4	0	0	4	4	4
148	person2	sidelunge	4	0	0	4	4	4
149	person2	sidelunge	4	0	0	4	4	4
150	person7	squat	1	1	2	2	7	7
151	person7	squat	1	1	2	2	7	7
152	person7	squat	1	1	2	2	7	9
153	person7	squat	1	1	2	2	7	7
154	person7	squat	2	1	2	2	7	7
155	person7	squat	3	1	2	2	7	9
156	person7	squat	4	1	2	2	7	7
157	person7	lunge	1	1	1	3	3	3
158	person7	lunge	1	1	1	3	3	3
159	person7	lunge	1	1	1	3	3	10
160	person7	lunge	2	1	1	3	3	3
161	person7	lunge	3	1	1	3	3	3
162	person7	lunge	4	1	1	3	3	3
163	person7	sidelunge	1	0	0	4	4	4
164	person7	sidelunge	1	0	0	4	4	4
165	person7	sidelunge	1	0	0	4	4	4
166	person7	sidelunge	4	0	0	4	4	4
167	person7	sidelunge	4	0	0	4	4	4
168	person7	sidelunge	4	0	0	4	4	4
169	person5	squat	1	1	2	2	7	9
170	person5	squat	1	1	2	2	2	11
171	person5	squat	1	1	2	2	2	11
172	person5	squat	2	1	2	2	2	11
173	person5	squat	3	1	2	2	2	11
174	person5	squat	4	1	2	2	2	11
175	person5	lunge	1	1	1	3	3	3
176	person5	lunge	1	1	1	3	3	3
177	person5	lunge	1	1	1	3	3	3
178	person5	lunge	2	1	1	3	3	3
179	person5	lunge	3	1	1	3	3	3
180	person5	lunge	4	1	1	3	3	3
181	person5	sidelunge	1	0	0	0	5	5
182	person5	sidelunge	1	0	0	0	5	5
183	person5	sidelunge	1	0	0	0	5	5
184	person5	sidelunge	4	0	0	0	5	5
185	person5	sidelunge	4	0	0	0	5	5
186	person5	sidelunge	4	0	0	0	5	5

A.2 ACA-T met tubes als feature vectoren

Tabel A.2: De resultaten van de k-means clustering met de tubes als feature vectoren en verschillende waardes van k.

Sensor	**k=3**	**k=4**	**k = 6**	**k =7**	**k=9**	**k=15**
AnkleLeftX	2	2	5	5	8	8
AnkleLeftY	0	0	0	0	0	12
AnkleLeftZ	1	1	4	4	4	9
AnkleRightX	2	2	2	2	2	13
AnkleRightY	0	0	0	0	0	12
AnkleRightZ	1	1	4	4	4	9
ElbowLeftX	2	2	2	2	2	2
ElbowLeftY	0	0	0	0	0	0
ElbowLeftZ	1	1	1	1	1	1
ElbowRightX	2	2	5	5	6	6
ElbowRightY	0	0	0	0	0	0
ElbowRightZ	1	3	3	3	1	1
FootLeftX	2	2	5	5	8	8
FootLeftY	0	0	0	0	0	12
FootLeftZ	1	1	4	4	4	9
FootRightX	2	2	2	2	2	13
FootRightY	0	0	0	0	0	12
FootRightZ	1	1	4	4	4	9
HandLeftX	2	2	2	2	7	7
HandLeftY	0	0	0	6	5	5
HandLeftZ	1	3	3	3	3	11
HandRightX	2	2	5	5	6	6
HandRightY	0	0	0	6	5	5
HandRightZ	1	3	3	3	3	3
HandTipLeftX	2	2	2	2	7	7
HandTipLeftY	0	0	0	6	5	5
HandTipLeftZ	1	3	3	3	3	11
HandTipRightX	2	2	5	5	6	6
HandTipRightY	0	0	0	6	5	5
HandTipRightZ	1	3	3	3	3	3
HeadX	2	2	2	2	2	2
HeadY	0	0	0	0	0	0
HeadZ	1	1	1	1	4	10
HipLeftX	2	2	2	2	2	2
HipLeftY	0	0	0	0	0	0
HipLeftZ	1	1	4	4	4	4
HipRightX	2	2	5	5	2	2

HipRightY	0	0	0	0	0	0
HipRightZ	1	1	4	4	4	4
KneeLeftX	2	2	5	5	8	8
KneeLeftY	0	0	0	0	0	12
KneeLeftZ	1	1	4	4	4	14
KneeRightX	2	2	2	2	2	2
KneeRightY	0	0	0	0	0	0
KneeRightZ	1	1	4	4	4	14
NeckX	2	2	2	2	2	2
NeckY	0	0	0	0	0	0
NeckZ	1	1	1	1	4	10
ShoulderLeftX	2	2	2	2	2	2
ShoulderLeftY	0	0	0	0	0	0
ShoulderLeftZ	1	1	1	1	4	10
ShoulderRightX	2	2	5	5	2	2
ShoulderRightY	0	0	0	0	0	0
ShoulderRightZ	1	1	1	1	4	10
SpineBaseX	2	2	5	5	2	2
SpineBaseY	0	0	0	0	0	0
SpineBaseZ	1	1	4	4	4	4
SpineMidX	2	2	2	2	2	2
SpineMidY	0	0	0	0	0	0
SpineMidZ	1	1	1	1	4	4
SpineShoulderX	2	2	2	2	2	2
SpineShoulderY	0	0	0	0	0	0
SpineShoulderZ	1	1	1	1	4	10
ThumbLeftX	2	2	2	2	7	7
ThumbLeftY	0	0	0	6	5	5
ThumbLeftZ	1	3	3	3	3	11
ThumbRightX	2	2	5	5	6	6
ThumbRightY	0	0	0	6	5	5
ThumbRightZ	1	3	3	3	3	3
WristLeftX	2	2	2	2	7	7
WristLeftY	0	0	0	6	5	5
WristLeftZ	1	3	3	3	1	11
WristRightX	2	2	5	5	6	6
WristRightY	0	0	0	6	5	5
WristRightZ	1	3	3	3	3	11

Bibliografie

[1] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++, 2012.

[2] G. Bergqvist and E. G. Larsson. The higher-order singular value decomposition: Theory and an application [lecture notes]. *IEEE Signal Processing Magazine*, 27(3):151–154, 2010.

[3] J. Burkardt. K-means clustering. *Virginia Tech, Advanced Research Computing, Interdisciplinary Center for Applied Mathematics*, 2009.

[4] C. Caiafa and A. Cichocki. Methods for factorization and approximation of tensors by partial fiber sampling. *CAMSAP 2009 - 2009 3rd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, pages 73 – 76, 01 2010.

[5] A. Chakraborti, M. Patriarca, and M. Santhanam. Financial time-series analysis: a brief overview. *arXiv.org, Quantitative Finance Papers*, 5, 04 2007.

[6] T. Decroos, K. Schutte, T. Beéck, B. Vanwanseele, and J. Davis. *AMIE: Automatic Monitoring of Indoor Exercises: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part III*, pages 424–439. 01 2019.

[7] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec. 1985.

[8] P. Kanani and M. Padole. Ecg heartbeat arrhythmia classification using time-series augmented signals and deep learning approach. *Procedia Computer Science*, 171:524–531, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

[9] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455 – 500, 2009.

[10] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

[11] U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 01 2004.

[12] T. Mach, L. Reichel, M. Van Barel, and R. Vandebril. Adaptive cross approximation for ill-posed problems. *Journal of Computational and Applied Mathematics*, 303:206–217, 2016.

[13] W. Meert, K. Hendrickx, T. Van Craenendonck, and P. Robberechts. Dtaidistance, Aug. 2020.

[14] M. Pede. Snel clusteren van tijdreeksen via lage-rang benaderingen. *Masterthesis computerwetenschappen*, 2019-2020.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] A.-H. Phan and A. Cichocki. Tensor decompositions for feature extraction and classification of high dimensional datasets. *Nonlinear Theory and Its Applications, IEICE*, 1, 10 2010.

[17] P. Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

[18] T. Vanhoof. Adaptieve tensor factorisaties om versneld tijdreeksen te clusteren. https://github.com/TuurVh/Thesis_Code, 2023.