Een Uitbreiding op Adaptieve Tensor-decomposities bij het Clusteren van Tijdreeksen

Lowie Debois , Wannes Croes
{lowie.debois, wannes.croes}@student.kuleuven.be

Abstract

In deze paper introduceren we een nieuwe tensor decompositie methode voor het clusteren van tijdreeksen. We gaan na of deze zinvol is en waarom. De decompositiemethodes waar we ons op baseren zijn de matrix methode en vectoren methode geïntroduceerd door T. Vanhoof [Vanhoof, 2023]. Bij de experimenten werd steeds gebruik gemaakt van dezelfde dataset [Decroos *et al.*, 2018]. We zien dat onze decompositie meer informatie uit de tensor kan halen zonder de kost significant te vergroten. We concluderen dat clusteren met deze decompositie dezelfde kwaliteit behaald als de vorige methodes maar hiervoor minder berekeningen gebruikt.

1 Inleiding

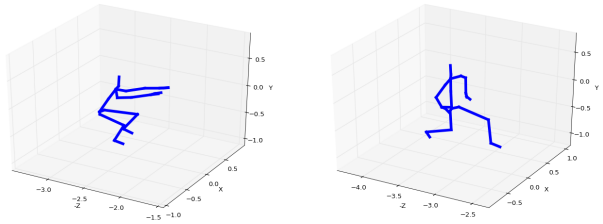
In deze paper bouwen we verder op voorgaand werk om efficiënt tijdreeksen te clusteren. De context van ons onderzoek is identiek aan dat van T. Vanhoof [Vanhoof, 2023]. Concepten zoals een tijdreeks, tensor-decomposities en de werking van aangehaalde algoritmes zijn belangrijke voorkennis en worden allemaal in zijn thesis uitgelegd. In deze inleiding wordt de context beknopt geschetst maar voor meer diepgang is zijn onderzoek aan te raden. Ander voorgaand onderzoek dat hierbij aansluit is dat van M. Pede [Pede, 2020].

1.1 De Dataset

De tijdreeksen die we clusteren komen uit de AMIE dataset [Decroos *et al.*, 2018]. Alle figuren, berekeningen en grafieken gebruiken dus enkel deze dataset. Deze dataset is interessant omdat de tijdreeksen veel mogelijkheden hebben tot classificatie. Dit staat ons toe om gemakkelijk de correctheid van de gevonden clusters, die in hoofdstuk 3.2 aan bod zullen komen, te evalueren.

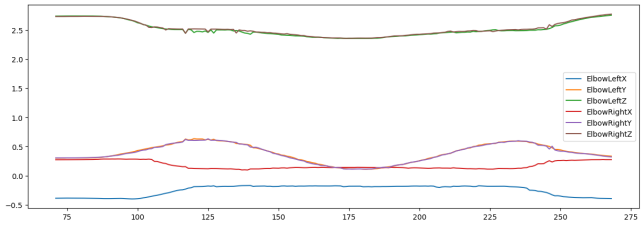
Bij het maken van de dataset heeft men aan 10 verschillende personen gevraagd om 3 oefeningen uit te voeren: een squat, een lunge en een sidelunge. Elk type oefening wordt 6 keer uitgevoerd per persoon: 3 keer correct, en 3 keer wordt er opzettelijk een fout gemaakt. Er zijn in het totaal dus 180 uitvoeringen, maar door technische fouten zijn er 186 videos gemaakt. Tijdens elke uitvoering heeft de persoon 25 verschillende sensoren op zijn lichaam. Elke sensor meet de 3D-positie van een specifiek lichaamsdeel doorheen de tijd. Deze metingen delen we op in een X-as, Y-as en Z-as. Dit komt neer op 75 tijdreeksen per uitvoering.

De tijdreeksen kunnen dus gegroepeerd worden per oefening, of per sensor aan de hand van hun locatie op de persoon (hand, knie ...).



(a) squat (b) lunge

Figuur 1: Een tijdstip in de uitvoering van een oefening door persoon 10. De posities van de sensoren zijn verbonden en vormen een skelet.



Figuur 2: De tijdreeksen van de sensoren op de elleboog tijdens het correct uitvoeren van de squat door persoon 10 in figuur 1.

1.2 Tijdreeksen Vergelijken

Clusteren van tijdreeksen houdt in dat we gelijkaardige reeksen in dezelfde cluster zetten. Om twee tijdreeksen te vergelijken gebruiken we het Dynamic Time Warping (DTW) algoritme [Pavel, 2009]. Dit algoritme definieert een afstand tussen 2 tijdreeksen, de DTW-afstand genoemd. Hoe meer de 2 tijdreeksen op elkaar lijken, hoe kleiner deze afstand.

Het grote nadeel van dit algoritme is dat de uitvoeringstijd kwadratisch groeit ten opzichte van de lengte van de tijdreeksen. De kost van een DTW-operatie is dus niet verwaarloos-

baar en daarom zullen we deze beschouwen als de significante operatie bij het analyseren van de kost van onze methode.

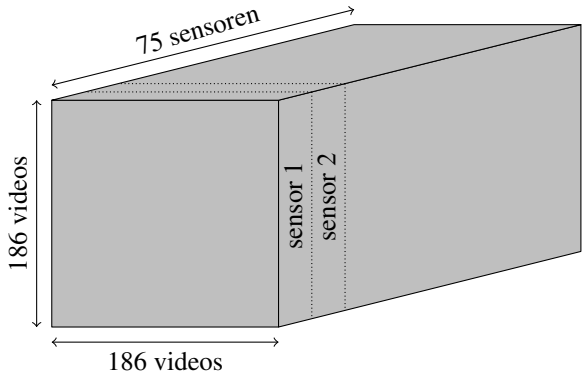
We vermelden nog het bestaan van alternatieve afstandsfuncties. Een voorbeeld is het FastDTW algoritme dat de DTW-afstand gaat benaderen in lineaire tijd en ruimte ten opzichte van de lengte van de tijdreeksen [Salvador and Chan, 2004]. Wij gebruiken echter het gewone DTW algoritme om beter te kunnen vergelijken met voorgaand onderzoek zoals dat van T. Vanhoof [Vanhoof, 2023].

1.3 De Afstandstensor

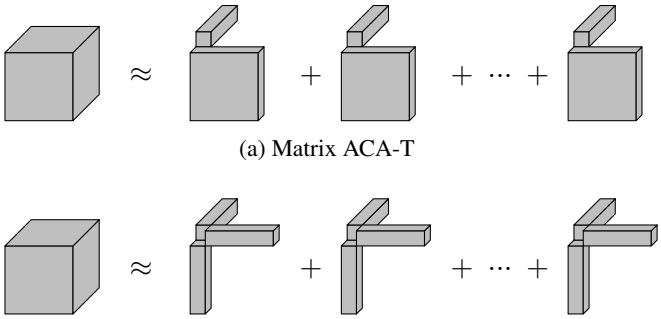
Clusteren vergt ook een clustering algoritme. Wij focussen op het efficiënt berekenen van de invoer van het cluster algoritme, maar houden ons verder niet bezig met het clusteren zelf. We gebruiken het bestaande K-means clustering algoritme [KMeans, 2007 2024]. Dit algoritme verwacht feature vectoren als invoer. Een afstandstensor opstellen uit de dataset en hiervan een decompositie berekenen zal deze feature vectoren opleveren. We leggen even uit wat de afstandstensor is en waarom een decompositie hiervan het gewenst resultaat geeft.

Afstandstensor

Wij zullen de aanpak van T. Vanhoof [Vanhoof, 2023] volgen (die ook met de AMIE dataset werkt) en een driedimensionale afstandstensor van $186 \times 186 \times 75$ opstellen. De afstandstensor is een opeenvolging van symmetrische afstandsmatrices als frontal slices. Een afstandsmatrix is een symmetrische matrix waar elke rij/kolom index een tijdreeks voorstelt. Element e_ij van deze matrix is de DTW-aftand tussen tijdreeks i en j De elementen van deze matrix zijn de paarsgewijze DTW-afstanden van de tijdreeksen. waarbij elke matrix enkel de tijdreeksen van eenzelfde sensor vergelijkt. Dit is grafisch weergegeven in figuur 3. Door tijdreeksen al te groeperen per sensor en niet alle tijdreeksen paarsgewijs te vergelijken, bevat deze afstandstensor veel minder elementen dan één grote afstandsmatrix. De uiteindelijke feature vectoren halen we dan uit een decompositie van deze tensor. Een belangrijke opmerking hierbij is dat de afstandstensor inherent informatie bevat over de dataset die bij een afstandsmatrix niet aanwezig was. De afstanden tussen tijdreeksen in de frontal slices zijn namelijk al gegroepeerd per sensor.



Figuur 3: Voorstelling van de afstandstensor.



(a) Matrix ACA-T

(b) Vector ACA-T

Figuur 4: De benadering van een tensor aan de hand van twee adaptieve decomposities.

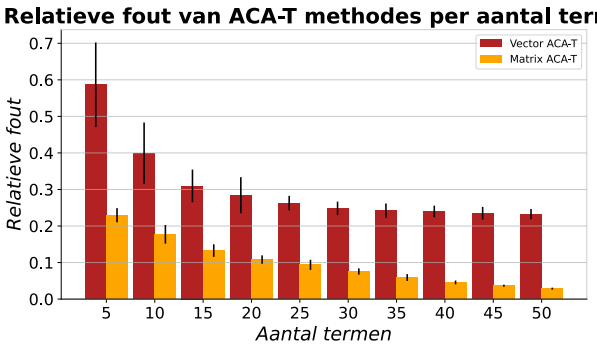
Feature vectoren

De feature vectoren in onze paper zijn vectoren van 186 (voor rijen/kolommen) of 75 (voor tubes) elementen waarvoor elk element een eigenschap voorstelt. In het geval van rij- en kolomvectoren is de eigenschap van het element op index i een maat voor de afstand van video v_i ten opzichte van de andere videos (= uitvoeringen). Dit betekent dat als video v_i en v_j op elkaar lijken (bv. beide een squat), dan zullen de elementen van de feature vectoren op indices i en j gelijkaardig zijn. Voor tubevectoren is de eigenschap van het element op index k een maat voor de afstand van sensor s_k ten opzichte van de andere sensoren. Hieruit volgt dat 2 sensoren die op elkaar lijken (bv. de X-as van de sensor op de linker en rechter knie) ook gelijkaardige waarden hebben als elementen in de feature vectoren. Merk op dat we niet meer spreken over individuele tijdreeksen maar we groeperen tijdreeksen per dimensie. In het geval van rij- en kolomvectoren stelt elk datapunt een hele video/uitvoering voor (= alle tijdreeksen van de 75 sensoren voor die uitvoering samen). Bij tubevectoren is elk datapunt een specifieke sensor (= alle tijdreeksen van de 186 uitvoeringen voor die sensor samen).

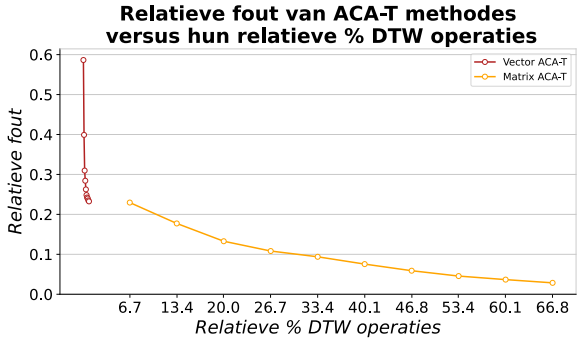
1.4 Decomposities met Adaptive Cross Approximation (voor Tensoren)

Er bestaan veel verschillende decomposities van een tensor [Vanhoof, 2023]. In deze paper focussen we op het uitbreiden van de twee adaptieve decomposities die T. Vanhoof heeft geïntroduceerd door middel van het Adaptive Cross Approximation (ACA) algoritme uit te breiden naar tensoren (ACA-T). Het ACA algoritme wordt zowel door M. Pede [Pede, 2020] als T. Vanhoof [Vanhoof, 2023] uitgelegd. De twee decomposities die we voortaan Matrix ACA-T en Vector ACA-T noemen (of de matrix en vectoren methode zoals T. Vanhoof) zijn in figuur 4 grafisch voorgesteld.

Hierbij zou de naamgeving nu duidelijk moeten worden. De decompositie van Matrix ACA-T bestaat uit een som van decompositie termen waarbij elke term een uitwendig product bevat van een matrix (of *slice*) en een vector (of *tube*) die samen een tensor van rang-1 opleveren. Bovendien bevat elke term ook nog eens een factor waarmee heel de term wordt vermenigvuldigd. Vector ACA-T is analoog maar de matrix wordt vervangen door een rij- en kolomvector. Meer detail



Figuur 5: De relatieve fout van Vector ACA-T en Matrix ACA-T ten opzichte van het aantal termen in de decompositie. De bar is het gemiddelde en de zwarte lijn geeft de standaardafwijking aan (steekproefgrootte = 50).



Figuur 6: De gemiddelde relatieve fout van Vector ACA-T en Matrix ACA-T ten opzichte van het aantal DTW-operaties nodig om die fout te bereiken (steekproefgrootte = 50). Hierbij is 100% gelijk aan het berekenen van de volledige tensor.

over deze methodes en hoe ze kunnen berekend worden, is te vinden in de al vermelde thesis [Vanhoof, 2023]. Meer specifiek in hoofdstuk 5.

2 Onze Decompositie Methode en de Probleemstelling

2.1 Motivatie uit Voorgaand Onderzoek

Om onze nieuwe methode te introduceren zullen we eerst wat context en motivatie geven. Eerder werd al vermeld dat onze voorgestelde methode zijn inspiratie haalt uit Matrix ACA-T en Vector ACA-T van T. Vanhoof. Wat volgt zijn enkele experimenten die door Vanhoof zijn uitgevoerd over de relatieve fout van de benadering van de twee methodes en hun respectievelijke kost (het aantal DTW-operaties). Figuren 5 en 6 tonen een succesvolle reproductie van de resultaten van Vanhoof met onze eigen implementatie. Deze is te vinden op de github repository van deze paper. [Croes and Debois, 2024].

De relatieve fout ϵ ten opzichte van de afstandstensor zoals beschreven in 1.3 wordt als volgt berekend:

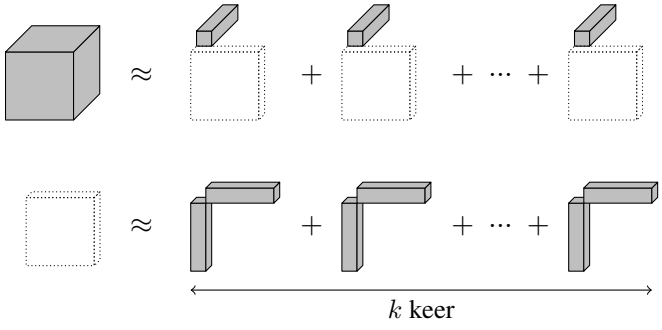
$$\epsilon = \frac{||T - \tilde{T}||}{||T||}$$

Hierbij is T de tensor en $\tilde{T}$ de benadering van de tensor, dit is de som van alle termen in de decompositie. De norm $||.||$ staat voor de Frobeniusnorm [Vanhoof, 2023].

Zoals te zien op figuur 6 bevindt Matrix ACA-T zich vooral boven 20% van de DTW-operaties vergeleken met het berekenen van de volledige tensor. Ondanks de aantrekkelijke relatieve fout, zijn deze percentages natuurlijk te hoog. De kost van Vector ACA-T daarentegen gaat niet boven de 2%. Echter bereikt de relatieve fout een plateau rond een 30-tal termen. De vraag is dus of er een combinatie bestaat van deze twee methodes die zowel een lage relatieve fout behaald (zoals Matrix ACA-T) als het aantal DTW-operaties laag houdt (zoals Vector ACA-T). Dit is wat onze nieuwe methode tracht te behalen. We noemen deze methode **Vector ACA-T type** k. De rede achter deze naamgeving zal in hoofdstuk 2.2 duidelijk worden.

2.2 Vector ACA-T type k

Onze nieuwe methode Vector ACA-T type k zal Vector ACA-T verder uitbreiden door meer vectoren toe te voegen per term. Met vectoren werken zoals Vector ACA-T in plaats van volledige matrices zoals Matrix ACA-T zal het aantal DTW-operaties laag houden. De manier waarop onze methode dit bereikt, is door de kostelijke frontale slices in elke term van Matrix ACA-T te gaan benaderen met ACA. Theoretisch starten we dus vanuit de matrix methode, maar we berekenen de slices niet volledig. Op deze manier hopen we de belangrijkste informatie uit elke slice te halen zonder te veel elementen van die slice te moeten berekenen. We kunnen dan het aantal termen k kiezen bij elke matrix-decompositie. Indien we $k = 1$ stellen, wordt elke frontale slice benaderd met een product van 1 rij- en kolomvector. Dit komt natuurlijk overeen met de originele Vector ACA-T. Vector ACA-T type 1 is dus identiek aan de vectoren methode van T. Vanhoof. De algemene decompositie wordt in figuur 7 grafisch voorgesteld.



Figuur 7: De benadering van een tensor aan de hand van Vector ACA-T type k.

Algorithm 1 Vector ACA-T type k

$decomp \leftarrow newTensorDecomp(K, N, M, n)$
$S \leftarrow sample\_tensor()$
$\delta \leftarrow max\_abs(S)$
while $decomp.length() < n$ **do**
  $matrix\_residu \leftarrow tensor.matrix\_at(\delta) - decomp.matrix\_at(\delta)$
  $aca\_decomp \leftarrow aca(matrix\_residu, k, \delta)$
  $tube\_residu \leftarrow tensor.tube\_at(\delta) - decomp.tube\_at(\delta)$
  $decomp.add(\delta, tube\_residu, aca\_decomp)$
  $\delta \leftarrow max\_abs(tube\_residu)$
  $update\_samples(S)$
  $max \leftarrow max\_abs(S)$
  if $max < \delta$ **then**
    $\delta \leftarrow max$
  end if
end while

Verband tussen Kost, Aantal Elementen en DTW-Operaties

In de pseudocode van het algoritme 1 worden samples gebruikt die elke iteratie geüpdatet worden om slechte randomisatie te vermijden. Eerder in sectie 1.2 werd vermeld dat het aantal DTW-operaties de kost van een decompositie aangeeft. Dit betekent dat de DTW-operaties bij het initieel samplen de kost verhogen. Echter is dit een kleine en constante hoeveelheid. Bovendien kunnen we DTW-operaties besparen in latere iteraties omdat er elementen zullen overlappen met vorige iteraties. Door zowel sampling als het voorkomen van overlappende elementen zal het aantal elementen in de decompositie niet exact overeenkomen met het theoretische minimum aantal DTW-operaties nodig om de decompositie te berekenen. Echter zijn deze aantallen in grootte-orde wel gelijk. Om deze rede zullen we het aantal elementen en nodige DTW-operaties door elkaar gebruiken om de kost van een decompositie aan te duiden.

2.3 Onderzoeksvragen

Nu dat we Vector ACA-T type k geïntroduceerd hebben, kunnen we onze onderzoeksvragen formuleren. Het werk van T. Vanhoof breidde ACA uit voor tensoren en ging na of de clustering met de benadering van Vector ACA-T gelijkaardig was aan de clustering gemaakt met de volledige tensor ter beschikking [Vanhoof, 2023].

T. Vanhoof vroeg zich dus af of dat we wel de volledige tensor nodig hadden voor een goede clustering. Het antwoord is *nee* aangezien Vector ACA-T ook voor een goede clustering zorgt. Deze paper zet een stap verder. We breiden Vector ACA-T uit met meer vectoren en vragen ons af: **Is deze uitbreiding zinvol? Is Vector ACA-T type k een goede decompositie? Kunnen we dezelfde resultaten verkrijgen als Vector ACA-T met minder rekenwerk?**. Of meer specifiek: **wat is de invloed van de parameter k in Vector ACA-T type k op zijn decompositie en verkregen clustering?** Het volgende hoofdstuk tracht aan de hand van veel berekeningen en grafieken een antwoord te vinden op deze vragen.

3 Resultaten en Experimenten

3.1 Relatieve Fout & Kost

Relatieve Fout

We beginnen bij het herproduceren van de grafieken uit sectie 2.1. Daar hebben we Vector ACA-T en Matrix ACA-T vergeleken aan de hand van hun relatieve fout ten opzichte van de originele tensor uit 1.3. In deze sectie zien we hoe Vector ACA-T type k zich daartussen bevindt. Concreet maken we identiek dezelfde grafiek als in figuur 5 maar dan met extra types tussen de bar van Vector ACA-T (rood) en Matrix ACA-T (oranje) toegevoegd. Dit experiment is te zien in figuur 8.

Zoals te zien is in figuur 8 zorgt een hogere k voor een lagere relatieve fout bij eenzelfde aantal termen. Dit is ook wat we verwachtten aangezien een hogere k meer vectoren toevoegt en dus meer DTW-operaties uitvoert. Opmerkelijk is de vergelijking van de verschillende types met de originele Vector ACA-T (type 1) en Matrix ACA-T. We zien dat hogere types (vooral 8 en 15) bij een laag aantal termen een relatieve fout hebben gelijkaardig aan die van Matrix ACA-T. Terwijl bij een hoger aantal termen een plateau bereikt wordt dichter bij de relatieve fout van Vector ACA-T (type 1).

Relatieve Kost

Dat de relatieve fout van hogere types kleiner is voor eenzelfde aantal termen is goed nieuws, maar de kost om deze relatieve fouten te bereiken is ook belangrijk. In figuur 9 herhalen we daarom het experiment in figuur 6 maar weer met de extra types bij Vector ACA-T (type 1) toegevoegd. Hier is Matrix ACA-T weggelaten om de schaal van de x-as overzichtelijk te houden.

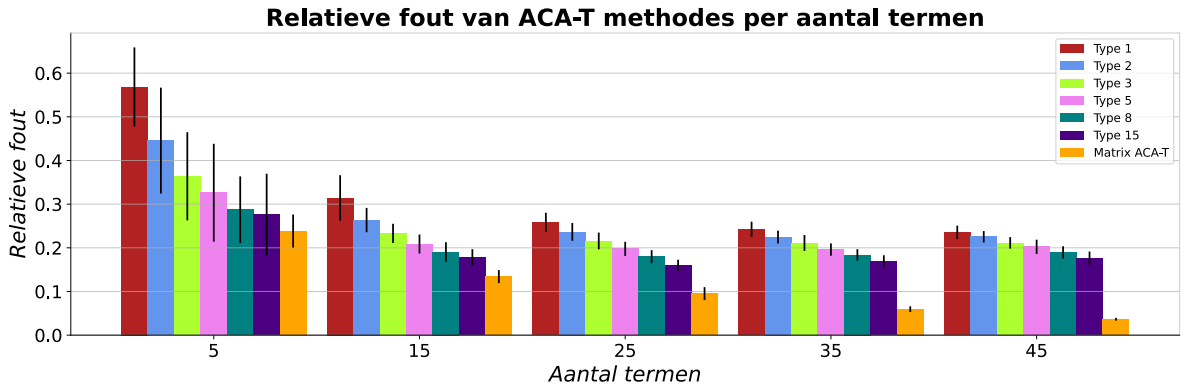
In dit experiment van figuur 9 zijn de plateaus waar daarnet over werd gesproken duidelijk te zien in de staart van de L-curves. De relatieve fouten van type 15 vormen bijna een horizontale lijn. We zien ook dat type 8 en 15 hun plateau sneller bereiken dan type 1 en 2. Dit wijst aan dat extra termen toevoegen aan een Vector ACA-T type k decompositie minder effect heeft wanner k groot is. De informatie die een decompositie van type k uit de tensor kan halen, wordt voor hogere k dus bij een lager aantal termen bereikt. Met andere woorden: een decompositie met hogere k komt vroeger termen met ruis tegen.

Optimale k

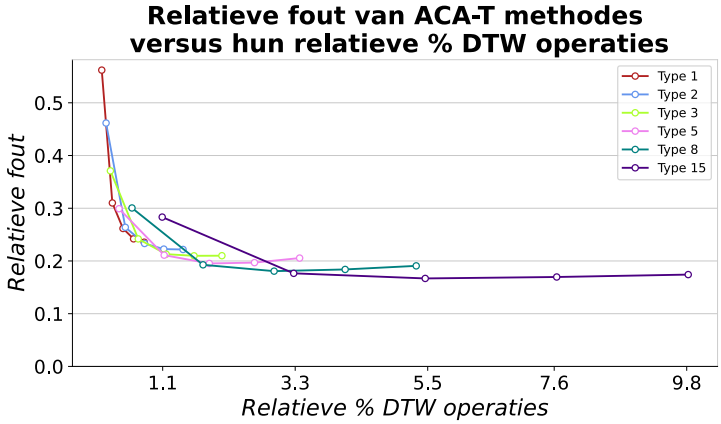
Een goede balans tussen de relatieve fout en de kost voor het experiment in figuur 9 is te bepalen door het oppervlak van de rechthoek van het datapunt te minimaliseren. De rechthoek van een datapunt is de rechthoek met als linkeronderhoek de oorsprong en rechterbovenhoek het datapunt zelf. We hebben het optimaal punt voor figuur 9 niet exact bepaald maar dit punt zal ergens rond 1.0% DTW operaties liggen met een relatieve fout van ongeveer 0.21. In deze regio liggen enkele datapunten van verschillende types dicht bij elkaar. Zo vallen datapunten van type 1, 2, 3 en 5 (met elk een verschillend aantal termen) in deze regio. Men zou dus een van deze punten kunnen kiezen aan de hand van andere voorwaarden zoals de kwaliteit van de clustering met zowel de rij als tube vectoren zoals in 3.2 wordt uitgelegd.

3.2 Het Clusteren

Het uiteindelijke doel van al deze benaderingen is natuurlijk om de tijdreeksen uit de dataset te clusteren. De AMIE dataset is een gelabelde dataset. Dit maakt het mogelijk om de clusters die we bekomen uit de bovengenoemde methodes eenvoudig te interpreteren en evalueren.



Figuur 8: De relatieve fout van Vector ACA-T type k voor verschillende k ten opzichte van het aantal termen in de decompositie. De bar is het gemiddelde en de zwarte lijn geeft de standaardafwijking aan (steekproefgrootte = 50).



Figuur 9: De gemiddelde relatieve fout van Vector ACA-T type k voor verschillende k ten opzichte van het aantal DTW-operaties nodig om die fout te bereiken (steekproefgrootte = 50). Hierbij is 100% gelijk aan het berekenen van de volledige tensor.

Zoals eerder vermeld maken wij gebruik van het KMeans cluster algoritme. Dit algoritme neemt feature vectoren als input, en zal dan clusters als resultaten geven. Om de data te clusteren maken we rechtstreeks gebruik van de decompositie van de afstandstensor. Dit doen we door vectoren uit de decompositie te selecteren en te gebruiken als invoer voor het KMeans cluster algoritme. We hebben hierbij een zekere keuze: we kunnen kiezen in welke dimensie we clusteren. Als we bijvoorbeeld de rijen als feature vectoren kiezen, zullen we uit elke term van de decompositie de rijvectoren selecteren. Deze keuze zal bepalen op wat we clusteren. Met de rijen als feature vectoren, zullen we clusteren op de specifieke uitvoering van de oefening. Als we de tubevectoren als feature vectoren selecteren, zullen we clusteren op de sensoren zoals in 1.3 werd vermeld.

Aangezien de data gelabeld is, kunnen aan de clusters een betekenis gegeven worden. In figuur 10a zien we een deel van de resultaten van clusteren in 3 clusters met de rijen als feature vectoren. We hebben hierbij vector-ACA type 3 met 30 termen gebruikt. We zien dat de tijdreeksen gegroepeerd worden volgens het soort uitvoering. We zien echter ook dat deze groepering niet helemaal correct is. In de vierde rij zien we dat een squat in de foute groep wordt ingedeeld. We willen dus nagaan hoe correct deze clustering is. We komen hier dadelijk op terug. In figuur 10b zien we een deel van de resultaten van clusteren in 3 clusters met de tubes als feature vectoren. We zien dat de tijdreeksen gegroepeerd worden volgens de X-, Y- en Z-as.

Van zodra we een interpretatie van de clusters hebben, kunnen we kwantificeren hoe correct de clustering is volgens die interpretatie. We doen dit aan de hand van de Adjusted Rand Index (ARI). De Adjusted Rand Index zal de overeenkomst tussen de bekomen resultaten en de 'true labels' kwantificeren. We gebruiken de true labels die overeenkomen met onze interpretatie van de clusters. De true labels zijn dus eigenlijk de resultaten die we hopen te bereiken. Bijvoorbeeld: Bij het eerste experiment zagen we dat de uitvoeringen gegroepeerd werden per soort. We kunnen dan als true labels de soort uitvoering nemen, en vervolgens meten hoe correct de groepering per soort effectief is. We zien de ARI-scores van dit experiment in figuur 11. We hebben dit experiment uitgevoerd met verschillende types van onze uitgebreide methode. Op de x-as van deze figuur zien we het aantal feature vectoren uit de decompositie dat we gebruikt hebben als invoer

Person	Exercise	Cluster
person8	squat	2
person8	squat	2
person8	squat	2
person8	squat	0
person8	squat	2
person8	squat	2
person8	lunge	0
person8	lunge	0
person8	lunge	0
person8	lunge	0
person8	lunge	0
person8	lunge	0
person8	lunge	0
person8	sidelunge	1
person8	sidelunge	1
person8	sidelunge	1
person8	sidelunge	1
person8	sidelunge	1
person8	sidelunge	1
...	...	...
person5	sidelunge	1

(a) Rijen als feature vectoren

Sensor	Cluster
AnkleLeftX	1
AnkleLeftY	2
AnkleLeftZ	0
AnkleRightX	1
AnkleRightY	2
AnkleRightZ	0
ElbowLeftX	1
ElbowLeftY	2
ElbowLeftZ	0
ElbowRightX	1
ElbowRightY	2
ElbowRightZ	0
FootLeftX	1
FootLeftY	2
FootLeftZ	0
FootRightX	1
FootRightY	2
FootRightZ	0
HandLeftX	1
...	...
WristRightZ	0

(b) Tubes als feature vectoren

Figuur 10: Deel van de resultaten van clusteren met vector-ACA type 3

voor het cluster algoritme. Dit is evenredig met de grootte van onze decompositie, en dus de kost. We zien dat de ARI-score verhoogt als we meer feature vectoren uit de decompositie nemen om als invoer te gebruiken voor het cluster algoritme. Dit is logisch: door meer feature vectoren uit de decompositie te nemen, halen we meer informatie uit de dataset. Bijgevolg zal de resulterende clustering dus meer accuraat zijn. We merken op dat de verschillende types soortgelijke ARI-scores behalen. Enkel bij een laag aantal feature vectoren zal type 8 iets slechter presteren. We kunnen dus concluderen dat deze types even goed clusteren. Er is echter een subtiel verschil tussen de types. Herinner dat type 8 per term in de decompositie 8 rij- en kolomvectoren bevat, en maar 1 tubevector. Om een specifiek aantal rijvectoren te bekomen, zullen we voor type 8 dus minder termen in onze decompositie nodig hebben dan bijvoorbeeld bij type 1. Bijgevolg hebben we bij type 8 minder tubevectoren in de gehele decompositie dan bij type 1. Aangezien we deze tubevectoren niet gebruiken in ons experiment, is dit een verloren kost. Type 8 zal dus een kleinere kost hebben dan type 1, en toch even goed clusteren. Concreet: stel dat we n feature vectoren willen berekenen. Dan is het totaal aantal berekende elementen bij type 8:

$$c_8(n) = 186 * 2 * n + 75 * \frac{n}{8} = \frac{3051}{8}n. \quad (1)$$

Bij type 1 is dit:

$$c_1(n) = 186 * 2 * n + 75 * n = 447n \quad (2)$$

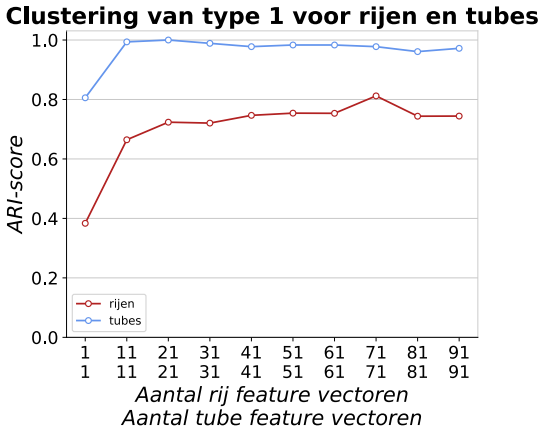
Hieruit volgt:

$$\frac{c8(n)}{c1(n)} = \frac{\frac{3051}{8}n}{447n} = \frac{1017}{1192} \approx 0.8532 \quad (3)$$

We zien dat de decompositie van type 8 slechts 85.32% van het aantal elementen van de decompositie van type 1 bevat. Aangezien we voor elk element een dure DTW-operatie moeten uitvoeren, is dit een goede maat voor de kost. Type 8 is dus ongeveer 15% goedkoper dan type 1, terwijl ze beiden een even goede clustering bekomen.

Clusteren in beide dimensies

Als laatste gaan we uit eenzelfde decompositie clusteren in zowel de rij- als tubedimensie. Onze decompositie bevat ten slotte vectoren in alle dimensies[1]. We zullen decomposities van verschillende types opstellen en deze met elkaar vergelijken. We doen dit door voor elk type te clusteren met zowel de rijvectoren als de tubevectoren. Beide clusteringen evalueren we terug met de ARI-score. In figuur 11 zien we dat voor alle types de clusteringen een plateau bereiken. We gaan nu voor de verschillende types nagaan hoe goed de clustering in de tube dimensie is op het punt dat dit plateau begint. In figuren 12, 14 en 13 zien we de ari score van het clusteren in de rij- en tubedimensies voor de types 1, 3 en 5. Op de x-as staan het aantal rij feature vectoren en daaronder het aantal tube feature vectoren. Bij figuur 12 zien we dat de tube clustering eerder een plateau bereikt dan de rij clustering. Dit wil zeggen dat we, als we een goede clustering in beide dimensies willen, een overschot aan tubevectoren zullen hebben.

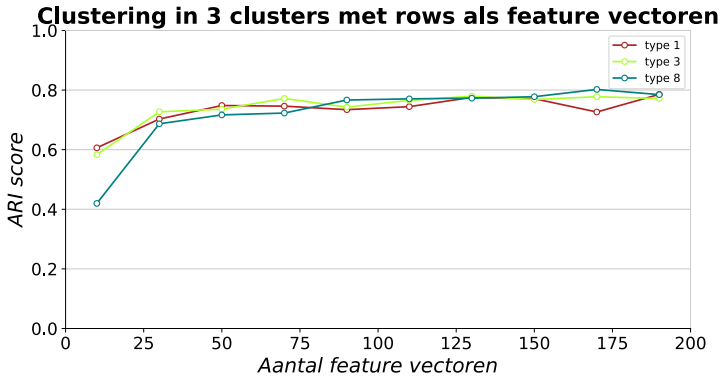


Figuur 12: Clustering met type 1 in 3 clusters in beide dimensies

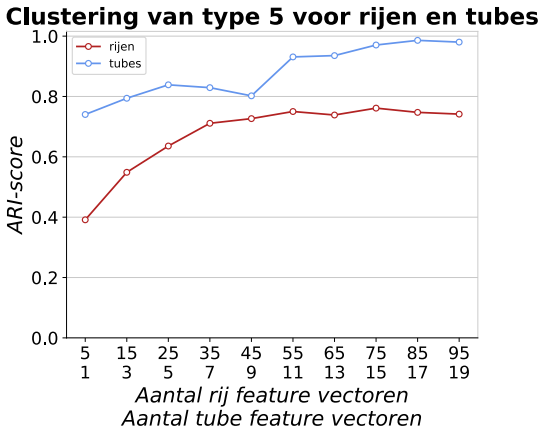
We kunnen nu eens kijken naar type 3 in figuur 14. Hier zien we dat beide curves op hetzelfde moment een plateau bereiken. Hier zullen we dus geen overschot aan vectoren hebben.

Bij type 5 in figuur **??** zien we het tegenovergestelde als type 1. Hierbij bereikt de rode curve eerder een plateau dan de blauwe curve. We hebben dus te veel rijvectoren. Hieruit kunnen we concluderen dat type 3 het meest efficiënte aantal vectoren berekent om een goede clustering te behalen in zowel de rij- als tubedimensie.

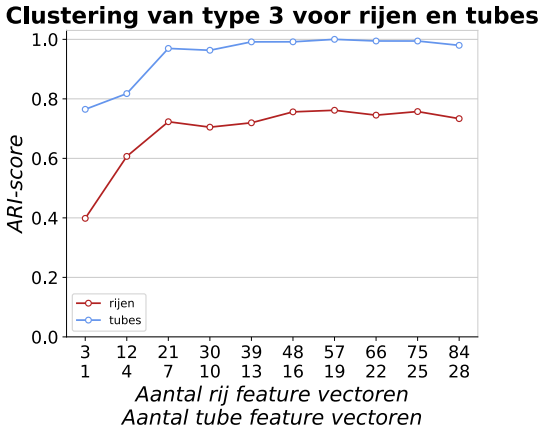
[1]De kolomdimensie is eruit gelaten aangezien symmetrische slices voor eenzelfde clustering in rijen als kolommen zorgen.



Figuur 11: Clustering in 3 clusters met rijen als feature vectoren



Figuur 13: Clustering met type 5 in 3 clusters in beide dimensies



Figuur 14: Clustering met type 3 in 3 clusters in beide dimensies

4 Conclusie

Na hoofdstuk 3 grijpen we terug naar de onderzoeksvragen van sectie 2.3. De belangrijkste vraag „Is Vector ACA-T type k een zinvolle decompositie van een tensor?" heeft een definitieve *ja* als antwoord. We zien dat de relatieve fout van hogere types verder daalt terwijl de relatieve kost aanvaardbaar blijft. De L-curves in figuur 9 geven aan dat veel verschillende types rond de optimale regio liggen. Daarnaast gaf het clusteren ook belovende resultaten. Hogere types zoals type 8 behalen evenwaardige ARI-scores terwijl ze rekenwerk besparen vergeleken met Vector ACA-T. Tot slot beïnvloed het type ook de ARI-score-plateaus van de rij en tube clustering wanneer beide dimensies van eenzelfde decompositie gebruikt worden om te clusteren. Hogere types geven meer rekenwerk aan de rijen waarbij type 3 zijn vectoren het efficiëntst gebruikt. We vermelden hierbij dat dit allemaal specifiek naar de AMIE dataset gericht is en dus een algemene methode voor het bepalen van welk type decompositie optimaal is, laten we voor verder onderzoek.

Referenties

[Croes and Debois, 2024] Wannes Croes and Lowie Debois. Tensortimeseriesclustering. https://github.com/lolgameboy/TensorTimeSeriesClustering, 2024.

[Decroos *et al.*, 2018] Tom Decroos, Kurt Schütte, Tim Op De Beéck, Benedicte Vanwanseele, and Jesse Davis. Amie: Automatic monitoring of indoor exercises. In *ECMLPKDD'18*, 2018.

[KMeans, 2007 2024] scikitlearn clustering. https://scikit-learn.org/stable/modules/clustering.html, 2007-2024. Bezocht in maart 2024.

[Pavel, 2009] Senin Pavel. Dynamic time warping algorithm review. *ResearchGate*, 01 2009.

[Pede, 2020] Mathias Pede. Snel clusteren van tijdreeksen via lage-rang benaderingen, 2020.

[Salvador and Chan, 2004] Stan Salvador and Philip K. Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 2004.

[Vanhoof, 2023] Tuur Vanhoof. Adaptieve tensor factorisaties om versneld tijdreeksen te clusteren, 2023.