Inhoudsopgave

Voorwoord

Deze masterproef is tot stand gekomen gedurende het tweede jaar van mijn opleiding tot Master in de Informatica aan de Universiteit Hasselt. Via dit proefschrift heb ik me verdiept in het domein van Relation Extraction en de DeepDive methodologie in het bijzonder. Hoewel ik hier vooraf bijzonder weinig kennis over had, raakte ik al snel geboeid door dit onderwerp en sindsdien is mijn interesse alleen maar toegenomen. Het was dan ook een bijzonder leerrijk proces dat me geen moment verveeld heeft. Deze masterproef vormde een grote uitdaging door zijn omvang en duur, maar bracht tegelijkertijd veel voldoening met zich mee.

Graag wil een aantal personen bedanken voor hun steun bij het tot stand komen van dit werk. In de eerste plaats wil ik Prof. Dr. Jan Van den Bussche, promotor van deze masterproef, bedanken. Zijn begeleiding en kritische geest waren een grote hulp bij het uitwerken van deze masterproef. Tevens wil ik Prof. Dr. Hendrik Blockeel van de KU Leuven bedanken voor de gesprekken rond het verband tussen de DeepDive methodologie en Machine Learning. Ten slotte wil ik nog mijn ouders bedanken voor hun continue ondersteuning en aanmoediging gedurende mijn opleiding.

Samenvatting

Om automatisch inzichten te verwerven in data, moet deze in een gestructureerd formaat staan (bv. een relationele database). Aangezien de overgrote meerderheid van beschikbare data echter in een ongestructureerd formaat staat (bv. Webpagina's), blijven vele waardevolle inzichten onontgonnen. Het doel van relation extraction is om relationele informatie te extraheren uit ongestructureerde data, wat de analyse van veel meer data mogelijk maakt. In deze masterproef verdiepen we ons in de DeepDive methodologie, een state-of-the-art aanpak voor relation extraction die aan de universiteit van Stanford ontwikkeld werd.

Fundamenteel steunt DeepDive op een Markov Logic Network (MLN) om een machine learning model op te stellen waarin features en extra domeinkennis gecombineerd worden. De extra domeinkennis modelleert verbanden tussen de te classificeren objecten onderling, waardoor DeepDive tot betere resultaten komt dan een zuiver feature-gebaseerd model. Dit is precies wat de DeepDive methodologie uniek maakt; we hebben vastgesteld dat de DeepDive methodologie in afwezigheid van formules met domeinkennis sterk overeenkomt met Logistic Regression. Daarnaast onderscheidt DeepDive zich ook door het ondersteunen van weight tying en User-Defined Functions, wat praktische meerwaarden zijn doordat ze de realisatie van toepassingen vereenvoudigen.

Vervolgens gebruiken we het DeepDive systeem om uit een collectie nieuwsartikels over Limburgs ondernemerschap te extraheren wie de baas is van welk bedrijf. Deze informatie kan bijvoorbeeld gebruikt worden om Google's Knowledge Panels aan te vullen. Ondanks DeepDive's goede ondersteuning blijft dit een bijzonder uitdagend probleem. Het kostte ons vele iteraties om tot 382 extracties te komen met een aanvaardbare nauwkeurigheid. Via onze implementatie leerde DeepDive waardevolle features en de formules met extra domeinkennis bevorderden het resultaat. Voor de gelabelde kandidaten zijn 94% van de extracties correct, maar voor de niet-gelabelde kandidaten slechts 66%. Er zijn dus nog verbeteringen nodig vooraleer de extracties geschikt zijn voor publicatie op het Web, maar de huidige implementatie vormt een uitstekend startpunt. We ondervonden dat DeepDive voldoende ondersteuning biedt aan de gebruiker om een (relatief) vlotte implementatie mogelijk te maken. Alleen voor de evaluatie van de resultaten schiet DeepDive momenteel te kort.

Dankzij systemen zoals DeepDive neemt de hoeveelheid gestructureerde data toe. We geloven dat ook hierin nog waardevolle informatie aanwezig is die niet rechtstreeks toegankelijk is. Daarom onderzochten we hoe de DeepDive methodologie aangepast kan worden om nieuwe relaties te extraheren uit reeds gestructureerde data. Het spreekt niet voor zich hoe we features kunnen bepalen in zulke data. We stellen voor om de data als een RDF graaf te beschouwen en de features te genereren op basis van enkele algemene graafpatronen. Via een eenvoudige Proof of Concept tonen we aan dat onze aanpak veel potentieel bezit. Een aanverwant domein is MLN Structure Learning: op basis van een collectie gestructureerde data wordt een MLN geleerd dat een high-level model vormt van die data. We stelden vast dat de feature-gebaseerde formules in ons MLN steeds vertaald kunnen worden naar predicaat-gebaseerde formules, waardoor onze aanpak tegelijk een alternatief is voor MLN Structure Learning. Ondanks dat onze aanpak eenvoudiger is dan bestaande algoritmes, behalen we kwalitatieve resultaten op een standaard benchmark: het geleerde MLN bestaat uit formules die intuïtief steek houden en sluit nauw aan bij de data.

1 Inleiding

Het centrale onderwerp van deze masterproef is de DeepDive methodologie [1], ontwikkeld aan de universiteit van Stanford. Het is een state-of-the-art benadering om relationele informatie te extraheren uit ongestructureerde data.

1.1 Big Data

Big Data [2] is één van de hot topics binnen de informatica vandaag de dag. Met de term 'Big Data' verwijst men naar de gigantische hoeveelheden data die we tegenwoordig voorhanden hebben. Meer bepaald wordt Big Data gekenmerkt door de drie V's: volume, velocity en variety.[1] *Volume* verwijst naar de extreme grootte van de collecties data waarmee men werkt; *velocity* verwijst naar de extreme snelheid waarmee data geproduceerd en verwerkt wordt; en *variety* verwijst naar de extreem grote verscheidenheid van de data (zoals tekst, matrices, tabellen of XML).

Het analyseren van Big Data, 'Big Data mining' genoemd, brengt vele opportuniteiten met zich mee. Netflix bijvoorbeeld, toont iedere klant aanbevelingen (zoals films en series die de klant waarschijnlijk interessant vindt). Door de hoge kwaliteit van de aanbevelingen zorgt Netflix dat de klant blijft gebruik maken van hun dienst, en dus voor winst blijft zorgen. Zulke aanbevelingen worden bepaald door de kijkgeschiedenis van de klant te vergelijken met die van andere klanten. Voor bedrijven als Netflix is Big Data mining dus de manier om winstgevend te zijn.[2] Ook vanuit een puur wetenschappelijk standpunt kan Big Data mining vele bijdrages leveren. Zo kunnen bijvoorbeeld grote hoeveelheden biologische en medische gegevens geanalyseerd worden om tot nieuwe inzichten te komen over het menselijk genoom [4].

De drie V's die Big Data karakteriseren, brengen tegelijkertijd heel wat uitdagingen met zich mee. In deze masterproeftekst kijken we in het bijzonder naar *variety*. Men onderscheidt gestructureerde data (b.v. een database), semigestructureerde data (b.v. een XML-bestand) en ongestructureerde data (b.v. een PDF-bestand). Het spreekt voor zich dat data eenvoudiger geanalyseerd kan worden indien er veel structuur in de data aanwezig is. Echter, naar schatting is maar liefst 80% van de data ongestructureerd, met het Web als voornaamste bron [5]. Aangezien het analyseren van ongestructureerde data zeer moeilijk is, wordt zulke data bijna nooit gebruikt voor analyse, en blijft er bijgevolg veel kennis onontgonnen. De term die hier vaak opgeplakt wordt, is 'Dark Data': de grote hoeveelheid data die er wel degelijk is, maar die we als het ware niet kunnen zien. Het is een analogie met dark matter (donkere materie) in de astronomie.

1.2 Information Extraction en Relation Extraction

Relation Extraction is een deeltaak van Information Extraction. Information Extraction [6] heeft als doel om ongestructureerde data (b.v. geschreven tekst

[1] Soms spreekt men van nog een vierde V, zijnde *veracity*, wat verwijst naar uitdagingen rond hoe waarheidsgetrouw de data is.

[2] In 2006 schreef Netflix een wedstrijd uit waarin men het huidige algoritme dat de aanbevelingen maakt met 10% moest verbeteren [3]. De prijs voor de winnaar bedroeg maar liefst één miljoen dollar, wat illustreert hoeveel financieel belang Netflix hieraan hecht.

in natuurlijke taal) of semi-gestructureerde data (b.v. in XML-formaat) om te zetten in een gestructureerd formaat (b.v. een database), waardoor de automatische analyse van die data mogelijk wordt. Figuurlijk gesproken is Information Extraction dus in staat om de enorme hoeveelheid Dark Data "aan het licht te brengen".

Het concrete doel van Information Extraction is het extraheren van de relaties tussen entiteiten, die door een collectie tekstdocumenten worden beschreven. Vaak focust men op binaire relaties, zoals $Getrouwd(p1, p2)$, waarbij entiteiten $p1$ en $p2$ verwijzen naar personen die met elkaar getrouwd zijn. Maar ook relaties van een hogere ariteit (b.v. ternaire relaties) zijn mogelijk, zoals $Opgericht(p, b, j)$, waarbij p verwijst naar de persoon die bedrijf b heeft opgericht in jaartal j. De resulterende extracties worden verzameld in wat men een 'knowledge base' noemt, waardoor Information Extraction ook wel 'Knowledge Base Construction' genoemd wordt. De kwaliteit van de geëxtraheerde informatie is tweevoudig. Enerzijds willen we dat de extracties correct zijn (zgn. 'high-precision'), en anderzijds willen we zo veel mogelijk van de door de tekst beschreven relaties effectief extraheren (zgn. 'high-recall').

Het Information Extraction proces doorloopt volgende drie stappen, waarvan 'Relation Extraction' vandaag de dag het meeste aandacht krijgt. De eerste stap is 'Named Entity Recognition', wat erin bestaat voor iedere zin de woorden te identificeren die naar belangrijke entiteiten verwijzen, zoals een persoon, plaats, bedrijf of datum. Vervolgens vindt 'coreference resolution' plaats. In deze stap worden woorden die naar eenzelfde entiteit verwijzen, geïdentificeerd. Wanneer bijvoorbeeld in de ene zin "Barack Obama" staat en in de volgende "hij", is het nuttig te weten dat beide tekstfragmenten naar dezelfde entiteit verwijzen. Tenslotte vindt 'Relation Extraction' plaats, waarin de eigenlijke relaties geëxtraheerd worden. Relation Extraction is het centrale onderwerp van deze masterproeftekst. In paragraaf 1.3 geven we een overzicht van de voornaamste benaderingen.

In 2001 kwam Tim Berners-Lee, uitvinder van het World Wide Web, met de term 'Semantic Web' op de proppen. Hij definieert het Semantic Web [7] als de nieuwe versie van het Web, waarin het Web niet langer bestaat uit documenten die alleen voor mensen begrijpbaar zijn, maar er bovendien informatie aanwezig is die het voor computersystemen mogelijk maakt om automatisch te redeneren over de inhoud van de Webpagina's. Hij erkende de nood om de informatie/kennis voor te stellen in een gestructureerd formaat, wat deel uitmaakt van een breder framework, het 'Resource Description Framework' (RDF) genaamd. In het RDF data model wordt informatie voorgesteld door triples van de vorm *(subject, predicate, object)*, zoals *(Barack Obama, birthplace, Honolulu)*.[3] Zo'n triple beschrijft dus de binaire relatie *predicate* tussen entiteiten *subject* en *object*. Hoe nobel dit idee ook is, tot de dag van vandaag is het Semantic Web nog niet doorgebroken omdat eindgebruikers zulke triples manueel moeten opstellen, wat veel te omslachtig is. Google echter is er met het project genaamd 'Knowledge Vault' [8] in geslaagd om een probabilistische knowledge base op te stellen bestaande uit 1.6 miljard triples, gebaseerd op de info op het Web en op reeds bestaande knowledge bases. Google gebruikt deze knowledge base om hun semantic search te verbeteren, wat illustreert dat het centrale idee

[3]In principe worden alle drie elementen voorgesteld a.d.h.v. URI's (Uniform Resource Identifiers), omdat deze uniek zijn en op die manier ambiguïteiten vermeden worden.

achter het Semantic Web wel degelijk leeft.

1.3 Mogelijke benaderingen

In deze paragraaf beschouwen we kort de voornaamste benaderingen voor Relation Extraction [6, 9].

De eerste technieken voor Relation Extraction extraheerden relaties door het detecteren van patronen in de tekst, b.v. gebruikmakend van reguliere expressies. Een voorbeeld is het patroon "X ... vrouw van Y", om te detecteren of X en Y met elkaar getrouwd zijn. Enerzijds kan men zulke patronen zelf opstellen, wat enorm tijdrovend is en veel domeinkennis vereist. Anderzijds bestaan meer geautomatiseerde benaderingen. DIPRE [10] is daar het bekendste voorbeeld van, en gaat als volgt te werk: (1) bepaal een kwalitatief patroon om enkele tupels te extraheren die zeker tot de relatie horen; (2) leid, op basis van deze tupels, de meest voorkomende patronen af; (3) gebruik de nieuwe patronen om meer tupels te extraheren; (4) herhaal dit proces tot er amper nieuwe tupels geëxtraheerd worden. Deze louter patroon-gebaseerde benaderingen waren niet erg kwalitatief, maar inspireerden wel vele nieuwe benaderingen.

Een tweede categorie benaderingen voor Relation Extraction noemt men 'semi-supervised'. Deze technieken gebruiken een supervised Machine Learning model om tupels te classificeren, als wel of niet tot een bepaalde relatie horend. In principe is eender welk model bruikbaar als classifier, zoals bijvoorbeeld een neuraal netwerk of een Support Vector Machine (SVM). Een tupel wordt voorgesteld door een feature-vector, bestaande uit syntactische en semantische features die uit de tekst worden geëxtraheerd. Om te weten te komen in welke mate iedere feature relevant is voor de classificatie, wordt het model getraind a.d.h.v. training data (zowel positieve als negatieve voorbeelden). In tegenstelling tot de klassieke supervised aanpak, wordt de training data niet bekomen door handlabeling, omdat dat veel te tijdrovend is. Men gebruikt een semi-automatische aanpak om snel, eenvoudig en veel training data te vergaren, waardoor men van 'semi-supervised' spreekt. Concreet gebruikt men reeds bestaande knowledge bases en/of een aantal kwalitatieve manueel opgestelde patronen. Men spreekt ook van 'weak supervision' of 'distant supervision'.

De twee voorgaande benaderingen gingen ervan uit dat de te extraheren relatie(s) op voorhand gekend is/zijn. Echter, om zo veel mogelijk informatie uit de verzameling tekstdocumenten te halen, mag men zich niet tot een vast aantal relaties beperken. Dit principe noemt men 'Open Information Extraction' en de bijhorende categorie met gebruikte benaderingen noemt men 'self-supervised'. Bekende voorbeelden van zulke systemen zijn KnowItAll [11] en TextRunner [12]. Om overweg te kunnen met eender welke relatie, worden louter algemene taalpatronen geleerd, net omdat deze relatie-onafhankelijk zijn. Een groot voordeel van deze aanpak is dat de gebruiker niet meer zelf voor training data moet zorgen; op basis van de algemene taalpatronen kan het systeem zelf kandidaten generen uit een gegeven verzameling tekstdocumenten. Deze algemeenheid maakt Relation Extraction uiteraard ook gecompliceerder, maar er wordt wel degelijk vooruitgang geboekt.

1.4 Doel van deze masterproef

Aan de universiteit van Stanford werd een state-of-the-art systeem ontwikkeld om Relation Extraction te faciliteren, DeepDive genaamd [1]. Het laat toe om op een snelle en eenvoudige manier een relatie te extraheren uit een collectie ongestructureerde data. De DeepDive methodologie valt onder de categorie van semi-supervised benaderingen.

In het eerste deel van deze masterproeftekst doorgronden we de DeepDive methodologie. De twee hoekstenen zijn Factor Graphs en Markov Logic Networks. Zodra we hier de nodige kennis rond hebben vergaard, bestuderen we de eigenlijke DeepDive methodologie. Ons doel is te weten te komen wat deze aanpak zo succesvol maakt en hoe deze zich onderscheidt van alternatieven.

In het tweede deel gebruiken we het DeepDive systeem om zelf een toepassing te realiseren. Uit een collectie nieuwsartikels proberen we te extraheren wie de baas is van welk Limburgs bedrijf; we noemen dit de baas-bedrijf relatie. Dankzij deze toepassing kunnen we ervaren hoe uitdagend relation extraction blijft, ondanks de goede ondersteuning van DeepDive.

In het derde en laatste deel gaan we na hoe we de DeepDive methodologie kunnen aanpassen zodat deze nieuwe relaties kan extraheren uit reeds gestructureerde data. We evalueren onze voorgestelde aanpak door een Proof of Concept uit te werken. Daarnaast stellen we vast dat onze aanpak ook gebruikt kan worden om een Markov Logic Network (MLN) te leren op basis van een collectie gestructureerde data; dit domein heet MLN Structure Learning. We vergelijken onze aanpak met de standaard algoritmes en evalueren deze a.d.h.v. een standaard benchmark.

Samengevat gaat we in deze masterproeftekst op zoek naar een antwoord op de volgende onderzoeksvragen:

Onderzoeksvraag 1: *Wat houdt de DeepDive methodologie in en wat maakt deze uniek?*

Onderzoeksvraag 2: *Wat zijn de sterktes en zwaktes van DeepDive toegepast op een praktijkvoorbeeld met ongestructureerde data (nieuwsartikels)?*

Onderzoeksvraag 3: *Hoe kan de DeepDive methodologie vertaald worden naar een scenario met gestructureerde data?*

Onderzoeksvraag 4: *Is de DeepDive methodologie voor gestructureerde data een bruikbare alternatieve aanpak voor MLN Structure Learning?*

		X		
		2	4	9
Y	3	0.14	0.22	0.07
	4	0.19	0.24	0.14

Figuur 1: Voorbeeld van een joint probability distribution over twee variabelen.

2 Probabilistic Graphical Models

Factor Graphs situeren zich binnen de veel bredere context van Probabilistic Graphical Models (PGMs) [13, 14]. Om een diepgaand begrip van Factor Graphs te ontwikkelen, is het noodzakelijk vanuit die ruimere context te vertrekken. Een uitstekend boek over PGMs is geschreven door Koller en Friedman [15], en vormt de voornaamste bron voor dit hoofdstuk.

2.1 Introductie

Wanneer men op een formele manier wilt redeneren, dient men steeds een model te definiëren. Dit model beschrijft de werking van datgene waar men over redeneert. We krijgen in de realiteit echter altijd en overal te maken met onzekerheden. Een model dat onzekerheden in acht kan nemen, noemt men een probabilistisch model. Om onzekerheden voor te stellen, wordt gebruikt gemaakt van variabelen, die met een bepaalde kans een bepaalde waarde aannemen. Dit noemt men 'random variables'. Op die manier wordt een probabilistisch model dus steeds voorgesteld door een kansdistributie gedefinieerd over die variabelen, wat men een 'joint probability distribution' noemt.

De eenvoudigste manier om een kansdistributie te definiëren, is d.m.v. een multi-dimensionale tabel die een kans associeert aan iedere combinatie van mogelijke waarden van de variabelen. We verduidelijken a.d.h.v. een voorbeeldje. Beschouw de kansdistributie P, gedefinieerd over de discrete random variables X en Y, kortweg genoteerd als $P(X, Y)$. Het domein van een discrete random variable V is de verzameling van waarden die V kan aannemen, genoteerd als $Dom(V)$. Stel dat $Dom(X) = \{2, 4, 9\}$ en $Dom(Y) = \{3, 4\}$. Figuur 1 toont een mogelijke definitie van $P(X, Y)$. De som van alle kansen moet steeds 1 zijn, opdat we met een geldige kansdistributie te maken hebben.

Merk op dat het aantal kansen in de tabel exponentieel toeneemt in functie van het aantal variabelen. Stel bijvoorbeeld dat P gedefinieerd is over n binaire variabelen, dan is het aantal kansen in de tabel gelijk aan 2^n. De complexe modellen die we in de praktijk nodig hebben, bestaan uit duizenden variabelen. Hierdoor wordt het al snel computationeel onhaalbaar om met zulke modellen te werken. Probabilistic Graphical Models gebruiken een graaf om de kansdistributie op compacte wijze voor te stellen. De knopen van de graaf zijn de variabelen uit de kansdistributie, en de bogen duiden afhankelijkheden tussen de knopen/variabelen aan. Uit de afhankelijkheden die de graaf voorstelt, kunnen we tevens afleiden welke *on*afhankelijkheden er tussen de variabelen zijn. Deze onafhankelijkheden laten ons toe de oorspronkelijke kansdistributie op te delen, wat tot een compactere representatie leidt. Hoe dit precies in zijn werk gaat, wordt in paragrafen 2.2 en 2.3 uitgelegd.

		M_1		
		k	m	
M_2	k	0.22	0.33	*0.55*
	m	0.18	0.27	*0.45*
		0.40	*0.60*	

Figuur 2: Mogelijke definitie van de kansdistributie over 2 biased munten.

We onderscheiden grofweg twee types van PGMs: een Bayesian Network, wat een gerichte graaf gebruikt; een Markov Network, wat een ongerichte graaf gebruikt. Voor beide types bestuderen we:

1. Representatie: hoe wordt het model voorgesteld?
2. Inference: hoe kan je probabilistische info uit het model halen?
3. Learning: hoe kan je de parameters van het model leren, gebaseerd op observaties?

2.2 Representatie van een Bayesian Network

In dit deel bespreken we de representatie van een Bayesian Network. Een Bayesian Network wordt gerepresenteerd door een verzameling conditionele kansdistributies en een gerichte graaf.

2.2.1 Onafhankelijke variabelen

Laat ons de kansdistributie opstellen voor het opwerpen van 2 munten. We noteren deze als $P(M_1, M_2)$, waarbij $Dom(M_i) = \{k, m\}$ wat respectievelijk voor 'kruis' en 'munt' staat. Laat ons ervan uitgaan dat de munten biased zijn, zodat we met een algemener scenario te maken hebben; een opgeworpen munt landt niet met 50% kans op kruis. Figuur 2 toont een mogelijke definitie van de kansdistributie. We zien duidelijk dat $P(M_1{=}m, M_2{=}k)$, verkort genoteerd als $P(m, k)$, het grootst is.

Men kan zich nu terecht de vraag stellen wat de kans is dat de eerste worp munt is; we zijn niet geïnteresseerd in de uitkomst van de tweede worp. Dit noteren we als $P(M_1{=}m)$, en noemen we een marginale kans [4]. Om deze kans te berekenen, sommeren we simpelweg alle kansen waarbij M_1 als uitkomst 'munt' heeft: $P(M_1{=}m) = P(M_1{=}m, M_2{=}k) + P(M_1{=}m, M_2{=}m) = 0.33 + 0.27 = 0.60$. Dit proces noemt men 'marginalisatie'. Wanneer we dit herhalen voor iedere mogelijke uitkomst van M_1, krijgen we de 'marginale kansdistributie' $P(M_1)$. Figuur 2 toont ook de marginale kansdistributie van $P(M_1)$ en $P(M_2)$. Aangezien een marginale kansdistributie een type kansdistributie is, moet de som van de kansen ook hier 1 zijn.

Het opwerpen van munten is een klassiek voorbeeld binnen de kansrekening om onafhankelijkheid tussen variabelen te duiden. Het spreekt immers voor zich dat de uitkomst van de eerste worp geen invloed heeft op de uitkomst van de tweede worp, en omgekeerd. Deze onafhankelijkheid noteert men als $M_1 \perp M_2$. Om na te gaan of de hierboven gedefinieerde kansdistributie hier inderdaad

[4] Deze benaming komt van het feit dat zo'n kans typisch in de marge geschreven wordt.

aan voldoet, moet er voor ieder mogelijk paar uitkomsten m_1, m_2 gelden dat $P(m_1, m_2) = P(m_1) \cdot P(m_2)$. Ten eerste geldt dat $P(M_1{=}m) \cdot P(M_2{=}k) = 0.60 \cdot 0.55 = 0.33 = P(m, k)$. Ook de overige drie combinaties van uitkomsten voldoen aan deze vereiste, waardoor we concluderen dat M_1 en M_2 onafhankelijk zijn

Wanneer alle random variables in een kansdistributie onafhankelijk zijn van elkaar, is het zeer eenvoudig om een compactere representatie van de kansdistributie op te stellen. Beschouw een kansdistributie $P(X_1, X_2, \ldots, X_n)$, waarbij alle X_i's onafhankelijk van elkaar zijn. Analoog aan het voorbeeldje uit de vorige paragraaf, kunnen we de kans op een specifieke gebeurtenis berekenen door: $P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i)$. Stel dat voor alle X_i's geldt dat $|Dom(X_i)| = 2$, zoals bij het opwerpen van een munt, dan heeft deze parametrisatie slechts $2 \cdot n$ parameters. Indien we echter de kansdistributie in tabelvorm hadden voorgesteld, hadden we maar liefst 2^n parameters nodig. Deze parametrisatie reduceert dus een exponentieel aantal parameters naar een linear aantal parameters. De parametrisatie van de kansdistributie van een Bayesian Network steunt op een gelijkaardig idee.

2.2.2 Bayesian Network parametrisatie

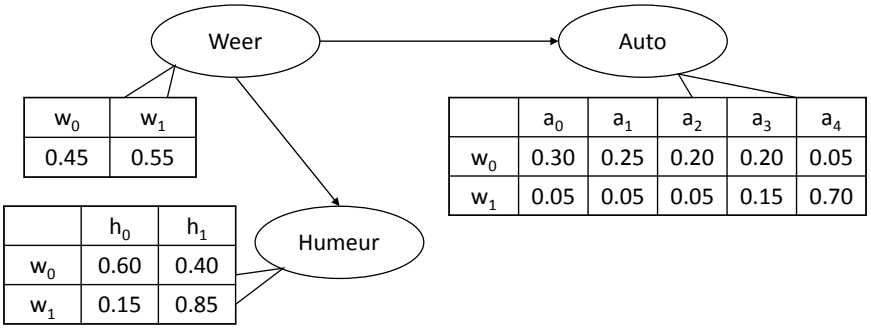
In de praktijk zijn er zeer weinig scenario's waarin alle random variables onafhankelijk van elkaar zijn. In een Bayesian Network worden de afhankelijkheden tussen variabelen voorgesteld door een gerichte acyclische graaf. De knopen van de graaf zijn de variabelen. Er is een boog van X_i naar X_j om aan te geven dat de waarde van X_i een impact heeft op de keuze van een waarde voor X_j; m.a.w. X_j is afhankelijk van X_i. Men spreekt ook van een causaal verband tussen beide variabelen. Het zou best kunnen dat X_j afhankelijk is van meerdere variabelen. Meer algemeen is X_j afhankelijk van zijn 'parents'. De parents van een variabele X in een graaf G, genoteerd als Pa_X^G, definiëren we als de verzameling van knopen Y waarvoor geldt dat er een boog van Y naar X bestaat.

Figuur 3 geeft een eenvoudig voorbeeldje; negeer voorlopig de tabellen. Deze graaf illustreert de relaties tussen het weer, het humeur van iemand en zijn keuze van wagen. Zo zien we dat het humeur van de persoon afhangt van het weer. Stel dat de persoon in het bezit is van 5 wagens. Hij kiest de auto waar hij vandaag mee zal rijden op basis van het weer (b.v. een cabrio bij goed weer, een jeep bij sneeuw).

We onderscheiden slecht en goed weer, respectievelijk als w_0 en w_1 genoteerd. De kansdistributie stellen we op zoals in deel 2.2.1. De tabel die aan de knoop 'Weer' hangt in de figuur, stelt deze kansdistributie voor.

Stel echter dat we de kansdistributie voor het humeur van de persoon willen opstellen. We onderscheiden slecht en goed humeur, respectievelijk als h_0 en h_1 genoteerd. Vermits H ('Humeur') afhankelijk is van de waarde die W ('Weer') aanneemt, spreekt men van een 'conditionele kans'. Zo noteren we bijvoorbeeld de kans op een slecht humeur (h_0) gegeven dat het slecht weer was (w_0), als $P(h_0 \mid w_0)$. Voor iedere waarde van W stellen we een conditionele kansdistributie op over H, zijnde $P(H \mid w_0)$ en $P(H \mid w_1)$. De figuur toont deze, alsook de conditionele kansdistributies voor A ('Auto'). Ook voor conditionele kansdistributies moet gelden dat de som van de kansen 1 is; de som van de kansen per rij moet 1 zijn.

Gegeven de afhankelijkheden die de graaf toont en de intuïtie uit deel 2.2.1,



Figuur 3: Bayesian Network graaf die de relaties beschrijft tussen het weer, het humeur van iemand, en zijn keuze van wagen.

kunnen we deze kansdistributie bijgevolg parametriseren als:

$$P(W, H, A) = P(W) \cdot P(H \mid W) \cdot P(A \mid W)$$

Zo kunnen we bijvoorbeeld eenvoudig de kans berekenen dat het slecht weer is, de persoon goed gehumeurd is en voor wagen a_3 koos: $P(w_0, h_1, a_3) = 0.45 \cdot 0.40 \cdot 0.20 = 0.036$. Deze parametrisatie vereist $2 + 4 + 10 = 16$ parameters, zoals de tabelletjes laten zijn, terwijl we $2 \cdot 2 \cdot 5 = 20$ parameters zouden nodig hebben wanneer we de kansdistributie als één grote tabel zouden voorstellen. Dit illustreert hoe een verzameling conditionele kansdistributies een compactere representatie is van de joint probability distribution.

Het bovenstaande voorbeeld kunnen we als volgt veralgemenen. Voor een gegeven een Bayesian Network graaf G kunnen we de bijhorende kansdistributie $P(X_1, X_2, \ldots, X_n)$ op een compacte manier parametriseren door de conditionele kansdistributie te definiëren voor iedere variabele gegeven zijn parents. De kansdistributie zelf wordt dan gegeven door:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Pa_{X_i}^G) \tag{1}$$

Dit product noemt men ook de 'factorisatie' van P volgens G; of men zegt dat P 'factoriseert' volgens G als deze gelijkheid klopt. Deze gelijkheid steunt op de kettingregel voor Bayesian Networks, die is gebaseerd op de algemenere kettingregel (chain rule). Het bewijs hiervan stellen we uit tot deel 2.2.4.

Een Bayesian Network wordt dan gedefinieerd door de verzameling conditionele kansdistributies $\mathcal{P}$ en de bijhorende gerichte graaf G, typisch genoteerd als $B = (G, \mathcal{P})$.

2.2.3 Conditionele onafhankelijkheden in Bayesian Networks

In het vorige deel zagen we hoe de graaf de basis vormde om tot de parametrisatie te komen. Meer bepaald toonde de graaf ons de afhankelijkheden tussen variabelen, en welke conditionele kansdistributies we bijgevolg moesten definiëren. Echter, zoals in deel 2.2.1 gemotiveerd, steunt de compacte representatie op de *on*afhankelijkheden in de graaf; deze laten ons toe de kansdistributie

op te delen in onafhankelijke delen.

In een typisch scenario zijn variabelen niet onafhankelijk van elkaar, maar wel conditioneel onafhankelijk (conditionally independent). We noteren $X \perp Y \mid Z$ om de conditionele onafhankelijkheid van random variables X en Y gegeven Z aan te duiden. Voor een specifieke toekenning van waarden aan de random variables, wordt conditionele onafhankelijkheid als volgt gedefinieerd: $X{=}x \perp Y{=}y \mid Z{=}z \iff P(X{=}x \mid Y{=}y, Z{=}z) = P(X{=}x \mid Z{=}z)$. In woorden betekent dit dat we, om de kans $P(X{=}x)$ te bepalen, alleen moeten weten dat $Z{=}z$; weten dat $Y{=}y$ is van geen belang. Conditionele onafhankelijkheid tussen random variables wordt gedefinieerd als: voor alle $x \in Dom(X)$, $y \in Dom(Y)$ en $z \in Dom(Z)$ geldt dat $X{=}x \perp Y{=}y \mid Z{=}z$.

Om conditionele onafhankelijkheid te verduidelijken, kijken we opnieuw naar figuur 3. Zoals verwacht geldt dat variabelen H ('Humeur') en A ('Auto') niet onafhankelijk zijn, omdat bijvoorbeeld $P(h_1, a_3) \approx 0.106$ en $P(h_1) \cdot P(a_3) \approx 0.109$, m.a.w. $P(h_1, a_3) \neq P(h_1) \cdot P(a_3)$. De conditionele onafhankelijkheid tussen H en A gegeven W geldt wel. Zo geldt o.a. dat $P(h_1 \mid a_3, w_0) = 0.40 = P(h_1 \mid w_0)$.

Dit voorbeeldje illustreert dat we de Bayesian Network graaf tevens kunnen zien als de specificatie van de conditionele onafhankelijkheden die in de kansdistributie aanwezig zijn. Meer bepaald specificeert een Bayesian Network graaf G voor iedere variabele X_i, per definitie, volgende conditionele onafhankelijkheden:

$$X_i \perp NonDescendants_{X_i}^G \mid Pa_{X_i}^G \tag{2}$$

met $NonDescendants_{X_i}^G$ de verzameling knopen Y waarvoor geen pad van X_i naar Y bestaat. Iedere knoop in de graaf is dus onafhankelijk van zijn niet-descendants gegeven zijn parents.

Het is belangrijk te benadrukken dat voorgaande definitie alleen iets zegt over de conditionele onafhankelijkheden die de graaf specificeert. De definitie zegt niet dat de kansdistributie hier gegarandeerd ook aan voldoet. In het volgende deel komen hier nog op terug.

2.2.4 Factorisatie vs Conditionele onafhankelijkheden

In deel 2.2.2 gebruikten we de Bayesian Network graaf om tot de compacte representatie van de kansdistributie te komen, de factorisatie dus. In het vorige deel gebruikten we de Bayesian Network graaf echter om de verzameling conditionele onafhankelijkheden in de kansdistributie te bepalen. Het is dan ook logisch dat beide perspectieven equivalent zijn. Meer formeel gelden volgende stellingen:

1. Als P voldoet aan de verzameling conditionele onafhankelijkheden die G specificeert, dan kan P gerepresenteerd worden als het product van de verzameling conditionele kansdistributies aan G geassocieerd (factorisatie); en omgekeerd:
2. Als P gerepresenteerd kan worden als het product van de verzameling conditionele kansdistributies aan G geassocieerd (factorisatie), dan voldoet P aan de verzameling conditionele onafhankelijkheden die G specificeert.

We introduceren de notie van een 'independency map', kortweg I-map genoemd. De I-map van een graaf G is de verzameling conditionele onafhankelijkheden die G specificeert, genoteerd als $I(G)$; de I-map van een kansdistributie P

is de verzameling conditionele onafhankelijkheden waaraan P voldoet, genoteerd als $I(P)$. Zeggen dat P voldoet aan de verzameling conditionele onafhankelijkheden die G specificeert, betekent dus dat iedere independency in $I(G)$ ook in $I(P)$ moet zitten: $I(G) \subseteq I(P)$. We lezen dit als "G is een I-map voor P". Merk op dat P nog extra conditionele onafhankelijkheden mag bezitten die niet door G gespecificeerd worden.

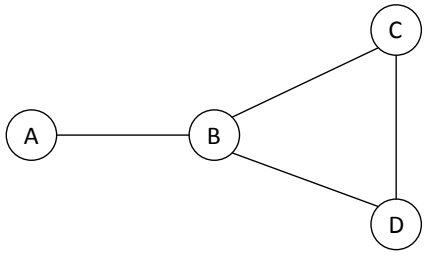
Zoals we aan het einde van deel 2.2.3 opmerkten, voldoet P niet noodzakelijk aan de conditionele onafhankelijkheden die G specificeert. Nu we de notie van een I-map geïntroduceerd hebben, komt dat dus neer op: definitie 2 definieert alleen de I-map van G, ze zegt niet dat G noodzakelijk ook een I-map is voor P. Als we er echter van uitgaan dat P factoriseert volgens G, geldt dat wel. Dit komt neer op stelling 2 van hierboven. Op het bewijs gaan we echter niet in, omdat dat te ver zou leiden.

Herinner dat we stelden dat de onafhankelijkheden in een Bayesian Network graaf de basis vormen om tot een compacte representatie van de kansdistributie te komen. Deze redenering volgt dus stelling 1 van hierboven. Om tot vergelijking 1 te komen, in deel 2.2.2, gingen we er inderdaad (impliciet) van uit dat P voldeed aan de onafhankelijkheden die G specificeert. In dit deel stellen we het bewijs op voor deze stelling.

We introduceren eerst nog de definitie van 'topologische ordening'. Een ordening van variabelen $X_1, \ldots, X_n$ is een topologische ordening t.o.v. een gerichte graaf G, indien voor iedere boog van X_i naar X_j geldt dat X_i vóór X_j staat in de ordening. Bijgevolg geldt voor een topologische ordening steeds dat de parents (en meer algemeen 'ancestors') van een knoop vóór de knoop zelf staan; de children (en meer algemeen 'decendants') van een knoop staan achter de knoop zelf.

De stelling in puntje 1 kunnen we nu herschrijven als: Als G een I-map is voor P, dan factoriseert P volgens G. Beschouw een willekeurige kansdistributie $P(X_1, X_2, X_3, \ldots, X_n)$. Het bewijs gaat als volgt:

1. Ga ervan uit dat $X_n, \ldots, X_3, X_2, X_1$ een topologische ordening is t.o.v. G.
2. Pas de kettingregel (chain rule) toe op P:
   $P(X_1, X_2, X_3, \ldots, X_n) = P(X_1 \mid X_2, X_3, \ldots, X_n) \cdot P(X_2, X_3, \ldots, X_n)$.
3. Pas nu de kettingregel toe op $P(X_2, X_3, \ldots, X_n)$, waardoor:
   $P(X_1, X_2, X_3, \ldots, X_n) = P(X_1 \mid X_2, X_3, \ldots, X_n) \cdot P(X_2 \mid X_3, \ldots, X_n) \cdot P(X_3, \ldots, X_n)$.
4. Wanneer we dit blijven herhalen, bekomen we:
   $P(X_1, X_2, X_3, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_{i+1}, \ldots, X_n)$.
5. Beschouw een willekeurige factor in dit product, $P(X_i \mid X_{i+1}, \ldots, X_n)$. Vermits G een I-map is voor P en op basis van vergelijking 2, geldt dat: $X_i \perp NonDescendants^G_{X_i} \mid Pa^G_{X_i} \subseteq I(P)$.
6. Ten eerste geldt wegens de topologische ordening (uit stap 1) dat $Pa^G_{X_i} \subseteq \{X_{i+1}, \ldots, X_n\}$. Ten tweede dat de descendants van X_i hier geen deel van kunnen uitmaken; de $NonDescendants^G_{X_i}$ kunnen hier wel deel van uitmaken. Bijgevolg weten we dat:
   $\{X_{i+1}, \ldots, X_n\} = Pa^G_{X_i} \cup \mathcal{Z}$, met $\mathcal{Z} \subseteq NonDescendants^G_{X_i}$.
7. Aangezien $X_i \perp NonDescendants^G_{X_i} \mid Pa^G_{X_i}$, is X_i ook onafhankelijk van



(a)

ϕ_1(A,B)

a_0	b_0	1
a_0	b_1	5
a_1	b_0	100
a_1	b_1	30
a_2	b_0	50
a_2	b_1	120

(b)

Figuur 4: (a) Eenvoudige Markov Network graaf. (b) Mogelijke invulling van $\phi_1(A, B)$.

iedere deelverzameling van $NonDescendants_{X_i}^G$ (gegeven $Pa_{X_i}^G$), waaronder $\mathcal{Z}$: $X_i \perp \mathcal{Z} \mid Pa_{X_i}^G$.

8. Nu, $P(X_i \mid X_{i+1}, \ldots, X_n) \overset{\text{(stap 6)}}{=} P(X_i \mid Pa_{X_i}^G \cup \mathcal{Z}) \overset{\text{(stap 7)}}{=} P(X_i \mid Pa_{X_i}^G)$.
9. Door iedere factor in het product op die manier te schrijven, bekomen we formule 1. Q.E.D.

2.3 Representatie van een Markov Network

In dit deel bespreken we de representatie van een Markov Network. Andere populaire benamingen voor een Markov Network zijn 'Undirected Graphical Model' en 'Markov Random Field'. Een Markov Network wordt gerepresenteerd door een verzameling factoren en een ongerichte graaf. Een factor laat toe een willekeurig probabilistisch verband te beschrijven tussen een aantal variabelen, wat Markov Networks algemener (en bijgevolg "krachtiger") maakt dan Bayesian Networks. We hanteren dezelfde aanpak als bij Bayesian Networks (deel 2.2): eerst bespreken we de parametrisatie en relateren deze aan de graafstructuur; daarna bespreken we welke conditionele onafhankelijkheden de graafstructuur specificeert; tot slot bespreken we de relatie tussen beide.

2.3.1 Markov Network parametrisatie

Aangezien een Markov Network een type PGM is, is het doel van dit model opnieuw om een kansdistributie op een compacte manier te representeren, steunend op de onafhankelijkheden tussen variabelen. In een Markov Network gebruiken we een ongerichte graaf om de afhankelijkheden/relaties tussen variabelen voor te stellen. Een ongerichte boog stelt een algemeen probabilistisch verband voor tussen het paar variabelen. Hierdoor zijn we niet langer beperkt tot het uitdrukken van causale verbanden tussen de variabelen, wat Markov Networks algemener en krachtiger maakt dan Bayesian Networks. Figuur 4(a) geeft een voorbeeld van een Markov Network graaf, en toont dat er o.m. tussen A en B een verband is.

Zo'n verband wordt beschreven door een 'factor', wat men typisch noteert als $\phi_1(A, B)$, die de compatibiliteit tussen iedere combinatie van waarden van

de variabelen definieert. Figuur 4(b) toont een mogelijke invulling van die compatibiliteiten, in de veronderstelling dat $Dom(A) = \{a_0, a_1, a_2\}$ en $Dom(B) = \{b_0, b_1\}$. Zo zien we bijvoorbeeld dat (a_1, b_0) een veel hogere compatibiliteit heeft dan (a_0, b_0).

Formeel is een factor $\phi(\mathcal{S})$ een functie die een positief reëel getal ($\mathbb{R}_0^+$) associeert aan iedere combinatie van waarden die een verzameling variabelen $\mathcal{S} \subseteq \mathcal{X}$ kan aannemen. Bij Bayesian Networks vormden de conditionele kansdistributies de basis voor de parametrisatie; bij Markov Networks zijn dat de factoren. Merk bovendien op dat we een conditionele kansdistributie kunnen zien als een factor; in deel 2.3.4 komen we hier nog op terug.

Laat ons nu ook factoren definiëren voor de overige verbanden in figuur 4(a): $\phi_2(B, C)$, $\phi_3(B, D)$ en $\phi_4(C, D)$. De concrete invulling is niet van belang. Deze verzameling factoren vormt dus de parametrisatie van de kansdistributie. Wanneer we, naar analogie met Bayesian Networks, de factoren met elkaar vermenigvuldigen, bekomen we: $\widetilde{P}(A, B, C, D) = \phi_1(A, B) \cdot \phi_2(B, C) \cdot \phi_3(B, D) \cdot \phi_4(C, D)$. Merk op dat dit geen geldige kansdistributie is, omdat al deze compatibiliteiten niet noodzakelijk tussen 0 en 1 liggen. We noemen $\widetilde{P}$ de niet-genormaliseerde maatstaf of kans. Om tot een kansdistributie te komen, moeten we $\widetilde{P}$ nog delen door de som van alle niet-genormaliseerde kansen, die we Z noemen. De constante Z wordt de 'normalisatie-constante' genoemd. Bijgevolg: $P(A, B, C, D) = 1/Z \cdot \widetilde{P}(A, B, C, D)$.

Voor een willekeurige kansdistributie $P(X_1, X_2, \ldots, X_n)$ geparametriseerd door de verzameling factoren $\Phi = \{\phi_1(\mathcal{S}_1), \phi_2(\mathcal{S}_2), \ldots, \phi_m(\mathcal{S}_m)\}$, wordt de factorisatie gegeven door:

$$
\begin{aligned}
P(X_1, X_2, \ldots, X_n) &= \frac{1}{Z}\widetilde{P}(X_1, X_2, \ldots, X_n), \text{ met} \\
\widetilde{P}(X_1, X_2, \ldots, X_n) &= \prod_{i=1}^{m} \phi_i(\mathcal{S}_i) \quad \text{en} \\
Z &= \sum_{(x_1, x_2, \ldots, x_n) \in Dom(\mathcal{X})} \widetilde{P}(x_1, x_2, \ldots, x_n).
\end{aligned}
\tag{3}
$$

De normalisatie-constante, Z, wordt ook de 'partition function' genoemd. Deze som gaat over alle mogelijke combinaties van waarden die de variabelen kunnen aannemen. Een Markov Network wordt dan gedefinieerd door de verzameling factoren Φ en de bijhorende ongerichte graaf H, typisch genoteerd als $M = (H, \Phi)$.

Door de notie van 'compatibiliteit' zijn Markov Networks veel minder intuïtief dan Bayesian Networks; het is veel moeilijker om een factor aan een kans te koppelen. We zien alleen dat een relatief hoge compatibiliteit tussen waarden voor een bepaalde factor, overeenkomt met een verhoogde kans.

In ons voorbeeld gingen we ervan uit dat er één factor gedefinieerd was per boog, en iedere factor dus twee parameters had. Echter, vergelijking 3 laat zien dat een factor over een willekeurig aantal parameters gedefinieerd mag worden. Het is inderdaad logisch dat een bepaalde interactie beschreven wordt door de combinatie van meer dan twee variabelen (of door slechts één variabele). Daarom dient er per factor $\phi(\mathcal{X})$ in de parametrisatie, een complete deelgraaf, bestaande uit de variabelen $\mathcal{X}$, in de graaf aanwezig te zijn. Een deelverzameling variabelen die een complete deelgraaf vormen, wordt ook een 'clique' genoemd.

Op basis van de graaf in figuur 4(a) zouden we dus een factor $\phi(B, C, D)$ kunnen definiëren; de bijhorende complete deelgraaf visualiseert de afhankelijkheid tussen de waarden van deze drie variabelen. De parametrisatie zou dan uit twee factoren bestaan: $\phi_1(A, B)$ en $\phi_2(B, C, D)$. We hebben nu dus twee mogelijke parametrisaties gezien voor dezelfde graaf. In deel 2.3.5 komen we terug op dit "probleem".

Herinner dat de Markov Network representatie als doel had een compactere representatie van de kansdistributie te vormen. Laat ons terug naar het voorbeeld in figuur 4 kijken. Veronderstel dat $|Dom(A)| = 3$ en $|Dom(B)| = 2$, zoals voorheen, en $|Dom(C)| = |Dom(D)| = 2$. Wanneer we de kansdistributie in tabelvorm zouden voorstellen, hebben we dus $3 \cdot 2 \cdot 2 \cdot 2 = 24$ parameters nodig. Als we de parametrisatie met vier factoren gebruiken, hebben we $6 + 4 + 4 + 4 = 18$ parameters nodig. Indien we van de parametrisatie met twee factoren uitgaan, hebben we $6+8 = 14$ parameters nodig. Dit voorbeeld illustreert hoe de Markov Network representatie inderdaad voor een compactere representatie kan zorgen.

2.3.2 Conditionele onafhankelijkheden in Markov Networks

Net als bij Bayesian Networks gebruiken we de Markov Network graaf om de afhankelijkheden tussen variabelen duidelijk te maken. Opnieuw steunt de compacte representatie echter op de conditionele onafhankelijkheden tussen variabelen. In dit deel bespreken we welke conditionele onafhankelijkheden een Markov Network graaf specificeert.

Om het redeneren over onafhankelijkheden te vereenvoudigen, introduceren we volgende definitie van 'scheidbaar': Gegeven een Markov Network graaf H, zijn twee deelverzamelingen knopen $\mathcal{X}$ en $\mathcal{Y}$ scheidbaar volgens een derde deelverzameling knopen $\mathcal{Z}$, indien ieder mogelijk pad tussen een $X \in \mathcal{X}$ en een $Y \in \mathcal{Y}$ gebruik maakt van een $Z \in \mathcal{Z}$. Het is makkelijker om deze scheidbaarheid visueel na te gaan, als volgt. Verwijder de knopen in $\mathcal{Z}$ uit de graaf, tezamen met de bogen die eraan hingen. Nu zijn $\mathcal{X}$ en $\mathcal{Y}$ scheidbaar volgens $\mathcal{Z}$ indien er geen enkel pad bestaat tussen een $X \in \mathcal{X}$ en een $Y \in \mathcal{Y}$. We noteren deze scheidbaarheid als $sep_H(\mathcal{X}; \mathcal{Y} \mid \mathcal{Z})$. De link tussen conditionele onafhankelijkheid en scheidbaarheid is de volgende: $sep_H(\mathcal{X}; \mathcal{Y} \mid \mathcal{Z}) \Leftrightarrow (\mathcal{X} \perp \mathcal{Y} \mid \mathcal{Z})$.

Beschouw een Markov Network $M = (H, \Phi)$ en de onderliggende kansdistributie $P(X_1, X_2, \ldots, X_n)$. Noem de verzameling van alle variabelen $\mathcal{X}$. De conditionele onafhankelijkheden die de graaf specificeert, worden gedefinieerd door drie eigenschappen, de 'Markov properties' genaamd:

De eerste eigenschap wordt de 'paarsgewijze Markov eigenschap' genoemd: voor ieder niet-naburig paar variabelen (X_i, X_j) geldt dat ze conditioneel onafhankelijk zijn gegeven alle andere variabelen:

$$X_i \perp X_j \mid \mathcal{X} - \{X, Y\} \tag{4a}$$

De tweede eigenschap heet de 'lokale Markov eigenschap', en is het equivalent van hoe we conditionele onafhankelijkheden in een Bayesian Network graaf beschreven (vgl. 2). Een variabele X_i is conditioneel onafhankelijk van alle andere knopen in de graaf gegeven zijn buren. We noteren de buren van een knoop X in een graaf G als $Neighbours_X^G$. De eigenschap is dan als volgt gedefinieerd:

$$X_i \perp (\mathcal{X} - \{X\} - Neighbours_{X_i}^G) \mid Neighbours_{X_i}^G \tag{4b}$$

De derde eigenschap heet de 'globale Markov eigenschap'. Deze steunt rechtstreeks op de scheidbaarheid en hadden we eigenlijk al gedefinieerd: twee deelverzamelingen variabelen $\mathcal{X}_A$ en $\mathcal{X}_B$ zijn conditioneel onafhankelijk gegeven een derde deelverzameling variabelen $\mathcal{X}_Z$, op voorwaarde dat ze scheidbaar zijn volgens $\mathcal{X}_Z$. Meer formeel geldt:

$$sep_H(\mathcal{X}_A; \mathcal{X}_B \mid \mathcal{X}_Z) \Leftrightarrow (\mathcal{X}_A \perp \mathcal{X}_B \mid \mathcal{X}_Z) \tag{4c}$$

Op het eerste zicht lijken deze drie eigenschappen equivalent, in die zin dat ze dezelfde verzameling conditionele onafhankelijkheden definiëren. Voor positieve kansdistributies kan men dit inderdaad aantonen (voor het bewijs verwijzen we naar Koller en Friedman [15]). Een kansdistributie wordt een 'positieve kansdistributie' genoemd indien alle kansen strikt groter dan 0 zijn. Voor niet-positieve kansdistributies is de globale Markov eigenschap sterker dan de lokale, die op haar beurt sterker is dan de paarsgewijze. Met 'sterker' bedoelen we dat de sterkere meer conditionele onafhankelijkheden definieert dan de zwakkere. Wanneer we in het algemeen over de conditionele onafhankelijkheden van een kansdistributie spreken, bedoelen we die opgelegd door de globale Markov eigenschap (tenzij expliciet anders vermeld).

2.3.3 Factorisatie vs Conditionele onafhankelijkheden

Net als bij Bayesian Networks is er een verband tussen de factorisatie van de kansdistributie en de conditionele onafhankelijkheden die de graaf specificeert. We gebruiken opnieuw de notie van een I-map. Beschouw een Markov Network graaf H, een willekeurige kansdistributie P en een willekeurige positieve kansdistributie P^+. Volgende stellingen beschrijven het verband (naar analogie met deel 2.2.4):

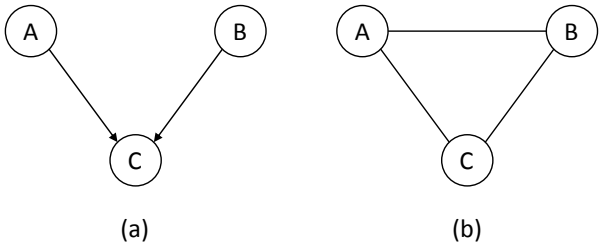
1. Als H een I-map is voor P^+, dan factoriseert P^+ volgens H;
2. Als P factoriseert volgens H, dan is H een I-map voor P.

Merk op dat stelling 1 een positieve kansdistributie vereist. Deze stelling, ook wel 'de stelling van Hammersley-Clifford' genoemd, geldt dus niet voor een niet-positieve kansdistributie. Voor het bewijs van beide stellingen verwijzen we naar Koller en Friedman [15].

2.3.4 Bayesian Networks vs Markov Networks

We hebben nu twee verschillende representaties gezien voor een kansdistributie. We zagen hoe Bayesian Networks causale verbanden tussen variabelen kunnen uitdrukken, en hoe Markov Networks meer algemene verbanden uit kunnen drukken. In dit deel gaan we dieper in op het verschil tussen beide.

Van een Bayesian Network naar een Markov Network:
We beginnen met het herleiden van een Bayesian Network naar het overeenkomstige Markov Network. Zoals we in deel 2.3.1 al opmerkten, kunnen we een conditionele kansdistributie $P(X_i \mid Pa^G_{X_i})$ gewoon zien als een factor $\phi_i(X_i, Pa^G_{X_i})$. Door voor iedere conditionele kansdistributie de overeenkomstige factor te definiëren, bekomen we dus de factorisatie die dezelfde kansdistributie voorstelt als het Bayesian Network. Merk op dat de partition function $Z = 1$.



Figuur 5: (a) Gerichte graaf G. (b) Bijhorende moral graph $\mathcal{M}[G]$.

Om te bepalen hoe de bijhorende ongerichte graaf er uitziet, moeten we kijken naar de verzameling factoren die we gedefinieerd hebben. Bijgevolg bevat deze graaf niet alleen een boog tussen X_i en zijn parents, maar ook tussen de parents onderling. De graaf die we op deze manier bekomen, noemt men een 'moral graph', waarbij het er dus op neerkomt dat er een (ongerichte) boog is tussen knopen X en Y als er een (gerichte) boog tussen hen liep (in eender welke richting), of X en Y ouder zijn van eenzelfde kind[5]. De moral graph van een gerichte graaf G noteren we als $\mathcal{M}[G]$. Figuur 5 toont een zeer eenvoudig voorbeeld.

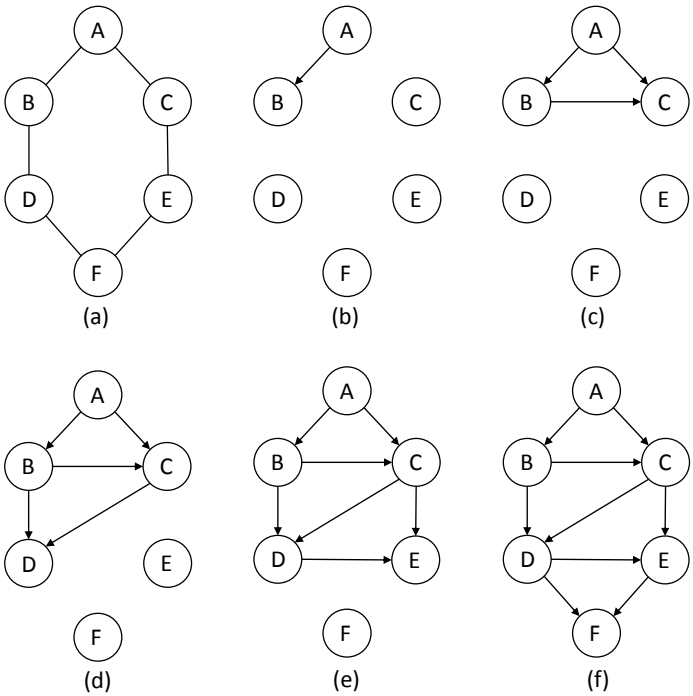
Nu we een Markov Network hebben dat dezelfde kansdistributie voorstelt als het Bayesian Network en we weten hoe de bijhorende graaf er uitziet, kunnen we nagaan of de verzamelingen conditionele onafhankelijkheden die ze specificeren, overeenkomen. Wegens stelling 2 uit deel 2.3.3 weten we dat $\mathcal{M}[G]$ een I-map is voor P, net zoals G dat is; hun I-maps zijn allebei deelverzamelingen van $I(P)$. Nu willen we dus weten of $I(\mathcal{M}[G]) = I(G)$. Intuïtief zorgt het toevoegen van bogen voor het verdwijnen van conditionele onafhankelijkheden. Figuur 5 illustreert dit: in de gerichte graaf was A conditioneel onafhankelijk van B (zonder dat er iets gegeven was), maar in de moral graph geldt deze conditionele onafhankelijkheid niet meer. Dit voorbeeld ontkracht de algemene gelijkheid, en toont aan dat $I(\mathcal{M}[G]) \subseteq I(G)$.

Een gerichte graaf G kan zelf al 'moral' zijn, wat betekent dat er een boog is tussen ieder paar ouders van een gemeenschappelijk kind. Dan geldt er logischerwijze wel dat $I(\mathcal{M}[G]) = I(G)$. Voor het bewijs verwijzen we naar Koller en Friedman [15].

Van een Markov Network naar een Bayesian Network:
We beginnen met de relatief eenvoudige opdracht om een (zinvolle) gerichte graaf G te definiëren die een I-map is voor een ongerichte graaf H. Het algoritme om zo'n gerichte graaf op te stellen, bestaat erin ten eerste een willekeurige ordening van de variabelen te kiezen. Vervolgens dient men voor iedere variabele X_i in de ordening de verzameling ouders $\mathcal{U}$ te vinden zodat: $\mathcal{U}$ is de kleinst mogelijke deelverzameling van $\{X_1, \ldots, X_{i-1}\}$ waarvoor in de ongerichte graaf geldt dat $X_i \perp (\{X_1, \ldots, X_{i-1}\} - \mathcal{U}) \mid \mathcal{U}$.

We nemen het voorbeeld uit Koller en Friedman [15] over, getoond in figuur 6(a). We passen het algoritme hierop toe voor de ordening A, B, C, D, E, F, en doorlopen dus volgende stappen:

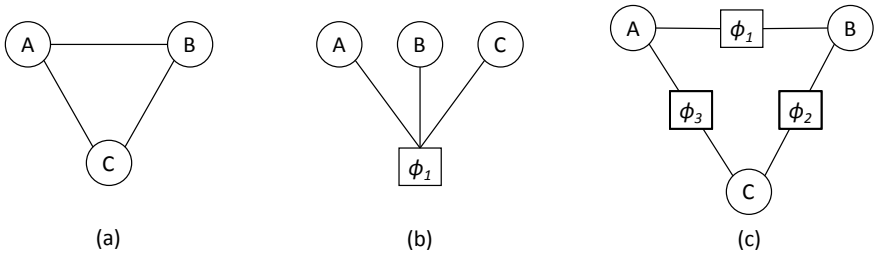
[5]De verklaring voor de naam 'moral graph' is de volgende. De boog tussen de ouders duidt aan dat ze getrouwd zijn, wat het ouderschap moreel verantwoord maakt.



Figuur 6: Van ongerichte graaf naar gerichte graaf.

1. Beschouw A: A heeft uiteraard geen ouders, omdat we nog geen andere variabelen beschouwden.
2. Beschouw B: vermits er in de ongerichte graaf een boog tussen A en B was, zijn ze afhankelijk, en moet A dus een ouder zijn van B. Zie figuur 6(b).
3. Beschouw C: door de boog tussen A en C in de ongerichte graaf is A al zeker een ouder van C. Op dit moment gaan we dus uit van $C \perp B \mid A$. Maar we zien dat C en B gegeven A niet conditioneel onafhankelijk zijn in de ongerichte graaf (wegens het pad C, E, F, D, B), waardoor ook B een ouder van C moet worden. Zie figuur 6(c).
4. Beschouw D: door de boog tussen B en D is B al zeker een ouder van D. Aangezien D niet onafhankelijk is van A en C gegeven B (wegens het pad D, F, E, C), moeten nog minstens één van beide variabelen een ouder worden van D. Als we voor A zouden kiezen, blijven D en C afhankelijk (wegens het pad D, F, E, C). Kiezen we voor C, dan geldt dat $D \perp A \mid B, C$. Zie figuur 6(d).
5. Beschouw E: dit verhaal is analoog aan voorgaande stap. Door C en D als ouders van E te kiezen, is E onafhankelijk van A en B. Zie figuur 6(e).
6. Beschouw F: vermits D en E noodzakelijkerwijs ouders zijn van F, is F onafhankelijk van de overige variabelen. We krijgen dus de gerichte graaf getoond in figuur deelfiguur 6(f).

De resulterende graaf is als het ware een triangulatie van de oorspronkelijke



Figuur 7: (a) Markov Network graaf. (b) Factor Graph met één factor. (c) Factor Graph met drie factoren.

graaf, en wordt een 'chordal graph' genoemd[6]. Men kan aantonen dat een gerichte graaf G die een I-map is voor een ongerichte graaf H, steeds een chordal graph is; voor het bewijs verwijzen we naar Koller en Friedman [15]. We hebben nu dus dat $I(G) \subseteq I(H)$ en stellen ons de vraag of $I(G) = I(H)$. Er zijn echter conditionele onafhankelijkheden verloren gegaan door het toevoegen van de vele bogen. Zo waren in het voorbeeld op figuur 6(a) variabelen C en D conditioneel onafhankelijk gegeven A en F, maar in de gerichte graaf is dat niet langer het geval. Hierdoor besluiten we dat het niet mogelijk is om een gerichte graaf G op te stellen die dezelfde conditionele onafhankelijkheden specificeert als een willekeurige ongerichte graaf H.

Net als bij de omvorming van een Bayesian Network naar een Markov Network, is er ook hier één uitzondering, waarvoor dus wel geldt dat $I(G) = I(H)$. Opnieuw zou het logisch moeten klinken dat dit het geval is wanneer G zelf reeds chordal was, wat dus betekent dat G getrianguleerd was.

2.3.5 Factor Graphs en Log-Linear Model

In dit deel beschrijven we twee varianten van het Markov Network die in de praktijk zeer vaak gebruikt worden. We beginnen met de Factor Graph, daarna bespreken we het Log-Linear Model.

Factor Graph:
Beschouw de Markov Network graaf in figuur 7(a). Herinner dat een Markov Network graaf een clique bevat per factor. Door louter naar deze graaf te kijken, kunnen we onmogelijk weten welke factoren er gedefinieerd zijn. Meer bepaald zijn er twee mogelijkheden: één factor $\phi(A, B, C)$; of drie factoren $\phi_1(A, B)$, $\phi_2(B, C)$ en $\phi_3(A, C)$. Een Factor Graph toont deze structuur expliciet, en werkt die ambiguïteit zo dus weg.

De Factor Graph representatie van een Markov Network bevat twee soorten knopen. Ten eerste bevat deze een gewone variabele-knoop voor iedere variabele in het Markov Network, zoals voorheen. Ten tweede wordt er per factor een factor-knoop toegevoegd, voorgesteld door een vierkantje. Een factor-knoop F is met een variabele-knoop X verbonden op voorwaarde dat X één van de parameters is van F. Voor ons voorbeeldje krijgen we bijgevolg de Factor Graphs getoond in figuur 7(b) en (c). Merk op dat een Factor Graph een bipartiete

[6]Deze benaming komt van het feit dat in iedere lus van de graaf als het ware een koorde gespannen werd om de lus weg te werken

graaf is.

Log-Linear Model:
Het Log-Linear Model is een alternatieve parametrisatie voor Markov Networks, gedefinieerd als volgt:

$$\begin{aligned} P(X_1, \ldots, X_n) &= \frac{1}{Z} \exp\left(\sum_{i=1}^{m} w_i f_i(\mathcal{S}_i)\right), \text{ met} \\ Z &= \sum_{X_1, X_2, \ldots, X_n} \exp\left(\sum_{i=1}^{m} w_i f_i(\mathcal{S}_i)\right) \end{aligned} \tag{5}$$

waarbij de w_i's en de $f_i(\mathcal{S}_i)$'s respectievelijk de 'gewichten' en 'features' worden genoemd. Een gewicht is een reëel getal, en een feature associeert een reëel getal aan iedere combinatie van waarden van zijn parameters. Zoals voorheen is Z de normalisatie-constante. De exponentiële functie is steeds positief, waardoor het Log-Linear Model dus een geldige kansdistributie definieert.

De naam 'Log-Linear' komt van het feit dat het natuurlijk logaritme van de niet-genormaliseerde maatstaf een lineaire combinatie (gewogen som) van parameters is. In de praktijk werkt men vaak met binaire features, zoals bijvoorbeeld bij DeepDive. Typisch worden ze door de gebruiker gedefinieerd, terwijl de gewichten geleerd worden (uit training data) – hier komen we in deel 2.8 op terug. De rol van een gewicht is uiteraard om de invloed van de bijhorende feature te specificeren. Merk op dat een positief gewicht voor een verhoging van de kans zorgt, en een negatief gewicht voor een verlaging van de kans (in de veronderstelling dat de feature naar een positief getal evalueerde, b.v. naar 1).

Om een Log-Linear Model te visualiseren, herschrijft men de factorisatie van hierboven typisch als een product van factoren (zoals de factorisatie van een Markov Network dus):

$$P = \frac{1}{Z} \exp\left(\sum_{i=1}^{n} w_i f_i(\mathcal{S}_i)\right) = \frac{1}{Z} \prod_{i=1}^{n} \exp(w_i f_i(\mathcal{S}_i)) = \frac{1}{Z} \prod_{i=1}^{n} \phi_i(\mathcal{S}_i) \tag{6}$$

Voor iedere feature dient men dus een factor $\phi_i(\mathcal{S}_i) = \exp(w_i f_i(\mathcal{S}_i))$ te definiëren. Dit model kunnen we visualiseren m.b.v. een Factor Graph (of de standaard ongerichte graaf). In de Factor Graph wordt ook steeds het gewicht bij een factor expliciet geschreven, b.v. onderaan in het vierkantje. Het Log-Linear Model met de bijhorende Factor Graph is een zeer vaak gebruikte representatie van een Markov Network, die bijvoorbeeld door DeepDive gebruikt wordt.

Tot slot beschrijven we nog hoe men een Markov Network (geparametriseerd volgens vergelijking 3) om kan zetten in een equivalent Log-Linear Model. Veronderstel dat de parametrisatie van het Markov Network uit slechts één factor $\phi(\mathcal{S})$ bestaat. Herinner dat deze factor een getal associeert aan iedere mogelijke toekenning $\mathcal{S}{=}s$. We definiëren nu een binaire feature voor alle mogelijke toekenningen, $f_s(\mathcal{S})$, die 1 teruggeeft als $S{=}s$. Bijgevolg zal voor een concrete $\phi(\mathcal{S}{=}s)$ telkens precies één van de features 1 teruggeven, namelijk die feature die gedefinieerd werd voor die specifieke s. Zo houden we dus over: $1/Z \cdot \exp(w_s)$, wat gelijk moet zijn aan $1/Z \cdot \phi(s)$. Bijgevolg moeten we het gewicht definiëren

als: $w_s = \log(\phi(s))$. Voor een parametrisatie bestaande uit meer dan één factor moeten we gewoon het proces herhalen voor iedere factor.

2.4 Exact Inference: Variable Elimination

Inference in een PGM betekent dat we probabilistische informatie over de onderliggende kansdistributie $P(X_1, \ldots, X_n)$ willen opvragen. In de praktijk is men meestal geïnteresseerd in een marginale kans $P(\mathcal{Y}{=}y)$, kortweg $P(y)$ genoteerd, met $\mathcal{Y} \subset \mathcal{X}$.

Nog een vaak voorkomende query is het bepalen van een conditionele kans. Beschouw allereerst de kansdistributie die we bekomen door het observeren van de waarde van een deelverzameling variabelen $\mathcal{E} \subset \mathcal{X}$. Zo'n observatie wordt ook wel 'evidence' genoemd (wat de keuze voor de letter $\mathcal{E}$ verklaart). Deze specifieke conditionele kansdistributie wordt de 'a-posteriori kansdistributie' genoemd, genoteerd als $P((\mathcal{X}{-}\mathcal{E}) \mid \mathcal{E}{=}e)$.[7] Een conditionele kans noteren we als $P(\mathcal{Y}{=}y \mid \mathcal{E}{=}e)$, of kortweg $P(y|e)$, met $\mathcal{E} \subset \mathcal{X}$ en $\mathcal{Y} \subseteq (\mathcal{X} - \mathcal{E})$. Een conditionele kans is dus een (al dan niet marginale) kans, gegeven de observatie van een aantal variabelen. We kunnen deze kans berekenen door $P(y|e) = P(y, e)/P(e)$, waardoor opnieuw de interesse in het berekenen van marginale kansen duidelijk wordt.

2.4.1 Naïeve aanpak

Het berekenen van een marginale kans $P(\mathcal{Y}{=}y)$ kan zeer eenvoudig, a.d.h.v. volgende naïeve aanpak. We genereren de volledige kansdistributie in tabelvorm, op basis van de factorisatie. Vervolgens sommeren we alle entries die voldoen aan $\mathcal{Y}{=}y$ (marginalisatie), zoals in deel 2.2.1 geïllustreerd werd. Meer formeel doen we dus het volgende. Noem $\mathcal{X}$ de volledige verzameling variabelen, $\mathcal{X} = \{X_1, \ldots, X_n\}$. De variabelen die we willen elimineren, noemen we $\mathcal{Z} = \mathcal{X} - \mathcal{Y}$. Dan berekenen we $P(\mathcal{Y}{=}y) = \sum_{\mathcal{Z}} P(\mathcal{Y}{=}y, \mathcal{Z})$.

Het probleem van deze naïeve aanpak is dat het aantal sommaties exponentieel is, aangezien we op de exponentieel grote tabelvorm terugvallen. Dit gaat regelrecht in tegen de compactheid die Probabilistic Graphical Models nastreven. We zijn dus op zoek naar een algoritme dat een marginale kans kan berekenen zonder de volledige tabel op te stellen, en bijgevolg niet exponentieel veel tijd nodig heeft. In het volgend deel beschrijven we zo'n algoritme, Variable Elimination genaamd, en in dit deel introduceren we de inzichten die de basis voor dit algoritme vormen[8].

Om in te zien waar de naïeve aanpak overal ruimte voor verbetering biedt, werken we een concreet voorbeeld uit. Beschouw de gerichte graaf $A \to B \to C \to D$, wat een kettinggraaf van lengte 4 genoemd wordt. Veronderstel dat al deze variabelen binair zijn, en dat we de marginale kans $P(d_1)$ willen berekenen. Op basis van de factorisatie voor Bayesian Networks (vgl. 1), kunnen we $P(d_1) = \sum_{A,B,C} P(A, B, C, d_1)$ reeds herschrijven als $P(d_1) = \sum_C \sum_B \sum_A P(A) \cdot P(B|A) \cdot P(C|B) \cdot P(d_1|C)$, waardoor we volledige kansdistributie niet langer moeten opstellen; we werken rechtstreeks met de conditionele kansdistributies.

[7] De benaming *a-posteriori* kansdistributie komt van het feit dat het de kansdistributie is die we *achteraf* bekomen, na de observatie.

[8] Het inference-probleem is NP-hard [15], waardoor eender welk algoritme worst-case exponentieel veel tijd nodig heeft. In de praktijk zijn de algoritmes wel degelijk bruikbaar.

Er zijn nog verbeteringen mogelijk, wat we inzien door voorgaande sommaties volledig uit te schrijven:

$$\begin{aligned}
P(d_1) = {} & P(a_0) \cdot P(b_0|a_0) \cdot P(c_0|b_0) \cdot P(d_1|c_0) + \\
& P(a_1) \cdot P(b_0|a_1) \cdot P(c_0|b_0) \cdot P(d_1|c_0) + \\
& P(a_0) \cdot P(b_1|a_0) \cdot P(c_0|b_1) \cdot P(d_1|c_0) + \\
& P(a_1) \cdot P(b_1|a_1) \cdot P(c_0|b_1) \cdot P(d_1|c_0) + \\
& P(a_0) \cdot P(b_0|a_0) \cdot P(c_1|b_0) \cdot P(d_1|c_1) + \\
& P(a_1) \cdot P(b_0|a_1) \cdot P(c_1|b_0) \cdot P(d_1|c_1) + \\
& P(a_0) \cdot P(b_1|a_0) \cdot P(c_1|b_1) \cdot P(d_1|c_1) + \\
& P(a_1) \cdot P(b_1|a_1) \cdot P(c_1|b_1) \cdot P(d_1|c_1).
\end{aligned}$$

We zien dat ieder opeenvolgend paar regels eindigt met dezelfde $P(c|b) \cdot P(d|c)$. Bijgevolg kunnen we deze herberekening vermijden door de distributiviteitsregel toe te passen ($B \cdot A + C \cdot A = (B+C) \cdot A$). We wijzigen dus de volgorde van product en som; i.p.v. eerst het product en dan de som te berekenen, berekenen we eerst de som en dan het product. Dankzij dit eerste belangrijke inzicht bekomen we:

$$\begin{aligned}
P(d_1) = {} & [P(a_0) \cdot P(b_0|a_0) + P(a_1) \cdot P(b_0|a_1)] \cdot P(c_0|b_0) \cdot P(d_1|c_0) + \\
& [P(a_0) \cdot P(b_1|a_0) + P(a_1) \cdot P(b_1|a_1)] \cdot P(c_0|b_1) \cdot P(d_1|c_0) + \\
& [P(a_0) \cdot P(b_0|a_0) + P(a_1) \cdot P(b_0|a_1)] \cdot P(c_1|b_0) \cdot P(d_1|c_1) + \\
& [P(a_0) \cdot P(b_1|a_0) + P(a_1) \cdot P(b_1|a_1)] \cdot P(c_1|b_1) \cdot P(d_1|c_1).
\end{aligned}$$

Verder zien we dat de $[P(a_0) \cdot P(b_0|a_0) + P(a_1) \cdot P(b_0|a_1)]$ en $[P(a_0) \cdot P(b_1|a_0) + P(a_1) \cdot P(b_1|a_1)]$ ieder twee keer gebruikt worden. Merk op dat deze sommen verkort kunnen geschreven worden als $P(b_0)$ en $P(b_1)$ respectievelijk, wat tezamen de kansdistributie $P(B)$ voorstelt. Hierdoor zien we in dat we herberekeningen kunnen vermijden door $P(B)$ eenmalig te berekenen, en daarna gewoon te gebruiken waar nodig. Dit idee volgt de 'dynamic programming' aanpak: breek een probleem op in deelproblemen en sla de resultaten van de deelproblemen op om ze later te kunnen hergebruiken. Dankzij dit tweede belangrijke inzicht bekomen we:

$$\begin{aligned}
P(d_1) = {} & P(b_0) \cdot P(c_0|b_0) \cdot P(d_1|c_0) + \\
& P(b_1) \cdot P(c_0|b_1) \cdot P(d_1|c_0) + \\
& P(b_0) \cdot P(c_1|b_0) \cdot P(d_1|c_1) + \\
& P(b_1) \cdot P(c_1|b_1) \cdot P(d_1|c_1).
\end{aligned}$$

Deze twee stappen kunnen we nu nog eens herhalen, waarna we $P(d_1)$ zeer eenvoudig kunnen bepalen. Het is interessant om te zien wat deze verbeteringen precies doen, gezien vanuit de notatie met sommatie-symbolen. We beginnen dus opnieuw met:

$$P(d_1) = \sum_C \sum_B \sum_A P(A) \cdot P(B|A) \cdot P(C|B) \cdot P(d_1|C).$$

Door de volgorde van product en som om te wisselen, bekomen we:

$$P(d_1) = \sum_C P(d_1|C) \cdot \sum_B P(C|B) \cdot \sum_A P(A) \cdot P(B|A), \tag{7}$$

wat mag omdat bijvoorbeeld $P(C|B)$ geen gebruik maakt van A. Als we nu de sommaties van binnen naar buiten toe uitwerken, bekomen we:

$$\begin{aligned} P(d_1) &= \sum_C P(d_1|C) \cdot \sum_B P(C|B) \cdot \sum_A P(A) \cdot P(B|A) \\ &= \sum_C P(d_1|C) \cdot \sum_B P(C|B) \cdot P(B) \\ &= \sum_C P(d_1|C) \cdot P(C) \end{aligned}$$

Merk op dat $P(B) = \sum_A P(A) \cdot P(B|A)$, analoog voor $P(C)$. Zo doen we in essentie aan dynamic programming. We vermijden namelijk dat $P(B)$ twee keer zou worden berekend: voor c_0 én voor c_1.

We hebben nu de twee essentiële ideeën besproken die een efficiënter algoritme toelaten. Laat ons nu eens kijken hoe efficiënt onze aanpak is voor een kettinggraaf van lengte n (met n knopen dus). De berekening van een tussentijdse kansdistributie (b.v. $P(B)$), vereist 4 vermenigvuldigingen (één per entry in $P(B|A)$). Vermits we orde n tussentijdse kansdistributies opstellen, is de complexiteit van dit algoritme linear (in n).

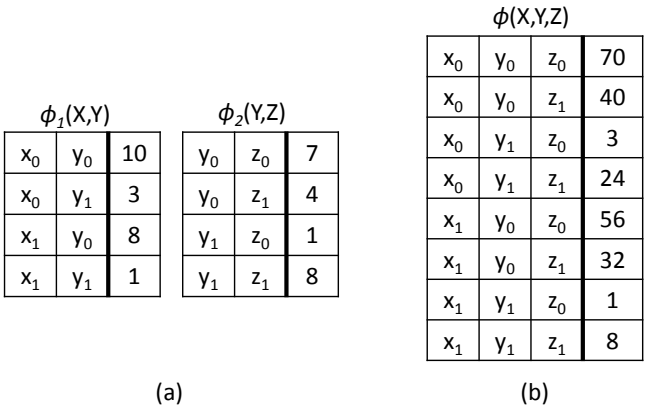
2.4.2 Variable Elimination algoritme

Een concreet algoritme om een marginale kansdistributie te berekenen, is het 'Variable Elimination' algoritme, ook wel het 'Sum-Product' algoritme genoemd. Dit algoritme steunt op de twee inzichten uit het vorige deel: (1) het omkeren van de volgorde van product en som, en (2) het opslaan van tussentijdse resultaten. We focussen ons op Markov Networks, omdat factoren algemener zijn dan conditionele kansdistributies – herinner uit deel 2.3.4 dat een Bayesian Network factorisatie eenvoudig herleid kan worden naar een Markov Network factorisatie.

Zoals we in voorgaand deel zagen, zal marginalisatie (uiteraard) een belangrijke taak van het algoritme zijn. We definiëren nu wat marginalisatie voor factoren betekent. Beschouw een factor $\phi(\mathcal{X}, Y)$, met $\mathcal{X}$ een verzameling variabelen en $Y \notin \mathcal{X}$ een variabele. Veronderstel dat we de 'marginale factor' $\psi(\mathcal{X})$ willen berekenen; we willen Y elimineren. Voor iedere mogelijke toekenning $\mathcal{X}{=}x$, definiëren we $\psi(\mathcal{X}{=}x) = \sum_Y \phi(\mathcal{X}{=}x, Y)$.

Het Variable Elimination algoritme gebruikt de notie van een 'product factor', zijnde het resultaat van de vermenigvuldiging van factoren. We leggen dit uit a.d.h.v. een voorbeeldje. Beschouw factoren $\phi_1(X, Y)$ en $\phi_2(Y, Z)$, gedefinieerd zoals in figuur 8(a) – het zijn alle drie binaire variabelen. De product factor $\phi(X, Y, Z)$ definiëren we door iedere combinatie x, y, z in te vullen als $\phi(x, y, z) = \phi_1(x, y) \cdot \phi_2(y, z)$. Figuur 8(b) toont het resultaat. Dit kan men triviaal veralgemenen naar een willekeurig aantal factoren.

Nu kunnen we de werking van het Variable Elimination algoritme toelichten. Beschouw een willekeurig Markov Network, geparametriseerd volgens de verzameling factoren Φ. Veronderstel dat de kansdistributie gedefinieerd is over



Figuur 8: (a) Definitie van $\phi_1(X, Y)$ en $\phi_2(Y, Z)$. (b) Product factor $\phi(X, Y, Z)$.

een verzameling variabelen $\mathcal{X}$, en we op zoek zijn naar de marginale kans $P(\mathcal{Y})$, met $\mathcal{Y} \subset \mathcal{X}$. De verzameling variabelen die we moeten elimineren, noemen we $\mathcal{Z} = \mathcal{X} - \mathcal{Y}$. We doorlopen de te elimineren variabelen in een willekeurige volgorde, en voor iedere variabele $Z \in \mathcal{Z}$ doen we het volgende:

1. Definieer Φ' als de deelverzameling van Φ, bestaande uit de factoren die Z als parameter hebben.
2. Definieer ψ als de product factor van de factoren in Φ'.
3. We elimineren Z uit ψ (marginalisatie), en noemen de resulterende tussentijdse factor τ.
4. Geef $\tau \cup (\Phi - \Phi')$ terug als output – we hebben dus Φ' vervangen door de gemarginaliseerde product factor τ en geven ook de onaangeroerde factoren terug.

Nadat we op deze manier alle $Z \in \mathcal{Z}$ hebben geëlimineerd, eindigt het algoritme nog niet. Het kan namelijk zijn dat we meerdere factoren overhouden. De eigenlijke output, de marginale kansdistributie $P(\mathcal{Y})$ dus, bekomen we door de product factor van die factoren te bepalen en deze ten slotte te normaliseren.

Laat ons het algoritme nu eens toepassen op het voorbeeld met de kettinggraaf $A \to B \to C \to D$ uit het vorige deel. De factoren zijn simpelweg $\phi_A(A)$, $\phi_B(B|A)$, $\phi_C(C|B)$ en $\phi_D(D|C)$. Veronderstel dat we de variabelen in de volgorde A, B, C elimineren, om zo de marginale kansdistributie $P(D)$ te bepalen.

In stap 1 van het algoritme gebruiken we het inzicht dat we de volgorde van product en som mogen omkeren. Daardoor kunnen we namelijk de factoren ϕ_A en ϕ_B afzonderlijk beschouwen, zoals we dat deden in vergelijking 7 uit het vorige deel. In stap 2 stellen we $\psi_1(A, B)$ op. In stap 3 bekomen we dan de tussentijdse factor $\tau_1(B)$, die we in stap 4 samen met de onaangeroerde factoren teruggeven. Stap 4 komt rechtstreeks neer op ons tweede inzicht, i.e. het opslaan van tussentijdse resultaten voor hergebruik. Bij het elimineren van B, in de volgende iteratie, zal $\tau_1(B)$ namelijk hergebruikt worden. Het verdere verloop van het algoritme spreekt voor zich.

In ons voorbeeldje bestaan de tussentijdse factoren steeds uit slechts één parameter. Het is echter uitermate belangrijk om in te zien dat dat zeker niet altijd het geval is. Het aantal parameters hangt namelijk af van de structuur van de graaf en de volgorde waarin we de variabelen elimineren. Stel dat we in

ons voorbeeldje waren begonnen met het elimineren van B. In stap 1 selecteren we de factoren ϕ_B en ϕ_C. Hun product factor is $\psi_1(A, B, C)$, en de tussentijdse factor is dan $\tau_1(A, C)$ – deze bestaat dus uit twee parameters.

Bij een groot Markov Network kan zo'n tussentijdse factor uit een aanzienlijk aantal parameters bestaan. Vermits het aantal entries in een factor exponentieel toeneemt in functie van het aantal parameters, kan de marginalisatie een significante hoeveelheid rekenwerk met zich meebrengen. Het Variable Elimination algoritme wordt dan onbruikbaar in de praktijk, waardoor men op benaderende methoden dient terug te vallen. Hier gaan we in deel 2.5 dieper op in.

2.5 Approximate Inference door Sampling

Door de grote Markov Networks waar we in de praktijk mee te maken krijgen, is exact inference vaak niet haalbaar. Bijgevolg moeten we overschakelen op benaderende methoden. Deze zijn een benadering in die zin dat ze inference toepassen op een benadering van de eigenlijke kansdistributie.

2.5.1 Forward Sampling

Net als bij exact inference, willen we een marginale kans $P(\mathcal{Y}{=}y)$ bepalen. Approximate inference door sampling benadert die kans d.m.v. 'sampling'. Sampling betekent 'het genereren van samples', waarbij een 'sample' een waarschijnlijke toekenning van een waarde aan alle variabelen is. Het centrale idee achter deze kansbenaderingstechniek is dat we een kans kunnen benaderen door de fractie van samples waarvoor geldt dat $\mathcal{Y}{=}y$, mits we voldoende samples gebruiken. De samples kunnen volgens verschillende technieken gegenereerd worden. In dit deel beschouwen we Forward Sampling, een sampling-techniek die alleen op Bayesian Networks toepasbaar is; in deel 2.5.4 bespreken we Gibbs Sampling, wat zowel voor Bayesian Networks als voor Markov Networks bruikbaar is.

We verduidelijken dit idee verder door naar een voorbeeldje te kijken. Beschouw het Bayesian Network in figuur 3. Veronderstel dat we de marginale kans $P(h_1)$ willen benaderen door sampling. Een sample is een triple van de vorm $\langle w, h, a\rangle$, door de onderliggende kansdistributie gegenereerd. Forward Sampling gaat als volgt te werk om zo'n sample te genereren. We kiezen een willekeurige waarde voor w, maar met 45% kans is het w_0 en met 55% kans is het w_1. Daartoe laten we de computer een uniforme random waarde s genereren (in het interval $[0; 1[$), en als $s < 0.45$ kiezen we $w = w_0$ en anders $w = w_1$. Conceptueel komt dit erop neer dat we een balkje van lengte 1 opdelen in twee intervallen; het ene is 0.45 breed en het andere 0.55. Vervolgens kiezen we een willekeurige waarde in dit balkje en kijken in welk interval we ons bevinden; stel dat het resultaat $w = w_1$ is. Vervolgens kunnen we op analoge wijze een waarde voor h bepalen, in de rij met w_1; stel dat $h = h_1$. Voor a delen we het balkje op volgens de 5 waarden die we in de rij met w_1 aflezen; stel dat $a = a_4$. We hebben nu dus het sample $\langle w_1, h_1, a_4\rangle$. We benadrukken dat de volgorde waarin we de afzonderlijke waarden samplen van belang is: vooraleer we de waarde van een bepaalde variabele kunnen samplen, moeten we de waarden van zijn parents kennen. Anders weten we immers niet welke conditionele kansdistributie we moeten beschouwen.

Stel dat we in totaal 6 samples genereren, zijnde $\{\langle w_1, h_1, a_4\rangle$; $\langle w_0, h_0, a_1\rangle$; $\langle w_0, h_1, a_0\rangle$; $\langle w_1, h_1, a_4\rangle$; $\langle w_1, h_0, a_0\rangle$; $\langle w_0, h_1, a_0\rangle\}$. Aangezien deze samples uit

de kansdistributie zelf afkomstig zijn, kunnen we $P(h_1)$ benaderen door de fractie samples waarin $H{=}h_1$, zijnde $\widehat{P}(h_1) = 4/6 \approx 0.66$. We gebruiken de notatie met het hoedje om te benadrukken dat het om een benadering van de eigenlijke kans gaat. Doordat we met een zeer eenvoudig Bayesian Network te maken hebben, konden we $P(h_1)$ eigenlijk exact bepalen. Meer bepaald is $P(h_1) = P(w_0) \cdot P(h_1|w_0) + P(w_1) \cdot P(h_1|w_1) \approx 0.65$. We zien dus dat $\widehat{P}(h_1)$ een uitstekende benadering is, gegeven het zeer beperkte aantal samples dat we gebruikten.

Algemeen geldt dus dat we, ongeacht de gebruikte sampling-techniek, een marginale kans kunnen benaderen door:

$$P(y) \approx \widehat{P}(y) = \frac{1}{M} \; |\{s \in \mathcal{S} : s[\mathcal{Y}]{=}y\}| \tag{8}$$

met $\mathcal{S}$ de verzameling samples van de vorm $\langle x_1, \ldots, x_n \rangle$, en M het aantal samples. Het spreekt voor zich dat de benadering alsmaar beter wordt naarmate we meer samples gebruiken.

2.5.2 Rejection Sampling

Herinner uit deel 2.4 dat de conditionele kans $P(\mathcal{Y}{=}y|\mathcal{E}{=}e)$, kortweg $P(y|e)$ genoteerd, een andere veel voorkomende query is. Herinner tevens dat $P(y|e) = P(y,e)/P(e)$. Gebaseerd op Forward Sampling (vergelijking 8), kunnen we deze verhouding dus schrijven als:

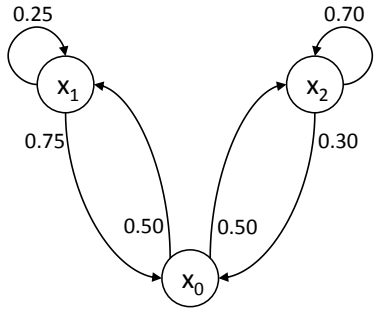
$$P(y|e) = \frac{P(y,e)}{P(e)} \approx \frac{\widehat{P}(y,e)}{\widehat{P}(e)} = \frac{|\{s \in \mathcal{S} \;:\; s[\mathcal{Y},\mathcal{E}]{=}(y,e)\}|}{|\{s \in \mathcal{S} : s[\mathcal{E}]{=}e\}|}$$

Deze aanpak noemt men 'Rejection Sampling'. De benaming komt van het feit dat we eigenlijk alleen kijken naar de samples die aan de evidence voldoen; samples die hier niet aan voldoen, negeren we bij het bepalen van de conditionele kans. Stel bijvoorbeeld dat $M{=}1000$, $|\{s{\in}\mathcal{S} : s[\mathcal{Y},\mathcal{E}]{=}(y,e)\}| = 40$ en $|\{s{\in}\mathcal{S} : s[\mathcal{E}]{=}e\}| = 50$. Het resultaat zou dan $40/50 = 0.8$ zijn. Bijgevolg hebben we ons eigenlijk niet op 1000 samples gebaseerd, maar slechts op de 50 samples uit de a-posteriori kansdistributie $P((\mathcal{X}{-}\mathcal{E}) \mid \mathcal{E}{=}e)$, om de conditionele kans te benaderen.

Dit inzicht maakt meteen de beperking van Rejection Sampling duidelijk. Als $P(e)$ klein is, wat typisch het geval is voor grote kansdistributies, negeren we zeer veel samples. Bijgevolg hebben we extreem veel samples nodig om er nog voldoende over te houden om een nauwkeurige benadering van de conditionele kans te bepalen. Idealiter zouden alle gegenereerde samples gebruikt worden, wat zou betekenen dat ieder gegenereerd sample afkomstig is uit de a-posteriori kansdistributie $P((\mathcal{X}{-}\mathcal{E}) \mid \mathcal{E}{=}e)$. Gebruikmakend van zo een verzameling samples, zouden we de conditionele kans dan kunnen benaderen door vergelijking 8 toe te passen.

2.5.3 Markov Chain Monte Carlo Sampling

Zoals we bij Rejection Sampling zagen, spreekt het niet voor zich hoe we rechtstreeks uit de a-posteriori kansdistributie kunnen samplen. In deel 2.5.4 zullen we een algemenere sampling-techniek bespreken, Gibbs Sampling genaamd,



Figuur 9: Voorbeeld van een Markov Chain.

die op efficiënte wijze samples kan genereren uit eender welke kansdistributie. We kunnen Gibbs Sampling dus gebruiken om (op efficiënte wijze) uit een a-posteriori kansdistributie te samplen. Bovendien kan Gibbs Sampling, i.t.t. Forward Sampling (zie deel 2.5.1), ook op Markov Networks toegepast worden. Meer bepaald genereert Gibbs Sampling een opeenvolging samples, waarbij ieder sample afkomstig is uit een kansdistributie die korter bij de gewenste kansdistributie ligt dan het vorige sample. Gibbs Sampling is gebaseerd op het algemenere Markov Chain Monte Carlo (MCMC) Sampling, wat we in dit deel bespreken.

Een Markov Chain (Markovketen) wordt beschreven door (1) een verzameling toestanden $\mathcal{X}$, en (2) een transitiemodel dat voor ieder paar toestanden x, x' beschrijft wat de kans is om van x naar x' te gaan, genoteerd als $\mathcal{T}(x \to x')$. We kunnen een concrete Markov Chain het eenvoudigst voorstellen d.m.v. een graaf. Figuur 9 geeft een voorbeeld van een Markov Chain voor een variabele X; de verzameling toestanden is $\mathcal{X} = Dom(X) = \{x_0, x_1, x_2\}$ en het transitiemodel wordt door de pijlen met hun kansen beschreven.[9]

We kunnen deze Markov Chain nu gebruiken om een 'random walk' te genereren. Dat is een willekeurige opeenvolging van toestanden waarin X zich zou kunnen bevinden, doorheen de tijd. Stel bijvoorbeeld dat X zich initieel in toestand x_0 bevindt. Daarna gaat X met 50% kans naar toestand x_1 of x_2; stel dat het toestand x_2 was. Vervolgens blijft X met 70% kans in toestand x_2, of gaat met 30% kans terug naar toestand x_0; stel dat X in toestand x_2 blijft. Dit proces kunnen we herhalen zo lang we willen. De toestand waarin X zich op tijdstip t bevindt, is bijgevolg een random variable, die we als $X^{(t)}$ noteren.

Dit leidt ons tot een kansdistributie, genoteerd als $P^{(t)}(X^{(t)})$, die voor iedere toestand x beschrijft wat de kans is dat X zich daarin bevindt op tijdstip t. Gegeven de kansdistributie voor het huidige tijdstip t, kunnen we de kansdistributie voor het volgende tijdstip t+1 berekenen met de formule:

$$P^{(t+1)}(X^{(t+1)}{=}x') = \sum_{x \in \mathcal{X}} P^{(t)}(X^{(t)}{=}x) \cdot \mathcal{T}(x \to x') \tag{9}$$

We verduidelijken de intuïtie hierachter a.d.h.v. het voorbeeld in figuur 9. Stel dat de initiële kansdistributie de uniforme verdeling is: $P^{(0)}(X) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Om $P^{(1)}(X^{(1)}{=}x_0)$ te bepalen, kijken we vanuit welke toestanden we x_0 kunnen bereiken; vanuit x_1 en x_2, maar niet vanuit x_0. Vervolgens bepalen we de

[9] Voor paren toestanden waartussen geen pijl getekend is, is de kans op een transitie 0.

Tijdstip	Kansdistributie		
	x_0	x_1	x_2
0	0.3333	0.3333	0.3333
1	0.3500	0.2500	0.4000
2	0.3075	0.2375	0.4550
3	0.3146	0.2131	0.4723
4	0.3015	0.2106	0.4879
5	0.3043	0.2034	0.4923
6	0.3002	0.2030	0.4968
7	0.3013	0.2009	0.4978
8	0.3000	0.2009	0.4991
9	0.3004	0.2002	0.4994
10	0.3000	0.2002	0.4998
11	0.3001	0.2001	0.4998
12	0.3000	0.2001	0.4999
13	0.3000	0.2000	0.5000
14	0.3000	0.2000	0.5000
15	0.3000	0.2000	0.5000

Figuur 10: Voorbeeld van een Markov Chain.

kans dat we ons nu in toestand x_1 bevinden en de transitie naar x_0 doen: $\frac{1}{3} \cdot 0.75 = 0.25$. Daarnaast bepalen we de kans dat we ons nu in toestand x_2 bevinden en de transitie naar x_0 doen: $\frac{1}{3} \cdot 0.30 = 0.10$. Deze twee kansen dienen we uiteraard te sommeren: $P^{(1)}(X^{(1)}{=}x_0) = 0.25 + 0.10 = 0.35$. Om tot $P^{(1)}(X^{(1)})$ te komen, herhalen we dit proces simpelweg voor $x' = x_1$ en $x' = x_2$.

Laat ons nu die kansdistributie voor een hoge waarde van t opstellen. Figuur 10 toont een fragment van dit berekeningsproces. We zien dat de kansdistributie niet meer wijzigt vanaf t=13. We zeggen dat de kansdistributie 'geconvergeerd' is, en noemen deze distributie de 'stationaire kansdistributie'. Meer algemeen geldt: een kansdistributie $\pi(\mathcal{X})$ is een stationaire kansdistributie voor een Markov Chain $\mathcal{T}$ op voorwaarde dat:

$$\pi(\mathcal{X}{=}x') = \sum_{x \in Dom(\mathcal{X})} \pi(\mathcal{X}{=}x) \cdot \mathcal{T}(x \to x') \tag{10}$$

MCMC Sampling genereert een opeenvolging samples door een 'random walk' te doen, zoals in een voorgaande alinea werd uitgelegd. Op ieder tijdstip t vormt $X^{(t)}$ dus het huidige sample, afkomstig uit de distributie $P^{(t)}(X^{(t)})$. Door de uitleg rond stationaire distributies zien we dat, naarmate we meer en meer samples genereren, deze afkomstig zijn uit een distributie die alsmaar korter bij de stationaire kansdistributie ligt. Dus, om samples te genereren uit een willekeurige kansdistributie $P(\mathcal{X})$, zouden we een Markov Chain kunnen opstellen waarvan de stationaire distributie overeenkomt met $P(\mathcal{X})$ en vervolgens MCMC Sampling toepassen op deze Markov Chain. Gibbs Sampling is een concreet algoritme, gebaseerd op dit idee.

Vooraleer we Gibbs Sampling kunnen toelichten, bespreken we hoe we een scenario met meerdere variabelen aanpakken; het voorbeeld in figuur 10 gebruikte slechts één variabele, X. Wanneer we met meerdere variabelen werken, definiëren we simpelweg een toestand voor iedere combinatie van waarden die

de variabelen kunnen aannemen. Het transitiemodel zouden we perfect op basis van zulke toestanden kunnen definiëren. In de praktijk deelt men het transitiemodel echter op: men definieert één transitiemodel $\mathcal{T}_i$ per variabele, omdat dat eenvoudiger is om mee te werken. Transitiemodel $\mathcal{T}_i$ beschrijft transities tussen toestanden waarbij alleen de waarde van x_i wijzigt: $\mathcal{T}_i((x_{-i}, x_i) \rightarrow (x_{-i}, x'_i))$, waarbij $X_{-i} = \mathcal{X} - \{X_i\}$ en x_{-i} een toekenning aan X_{-i} is. Wanneer de Markov Chain een transitie maakt, houdt dit in dat we ieder individueel transitiemodel moeten toepassen; om een volgend sample te bekomen, passen we ieder transitiemodel toe, en de toestand waarin we ons dan uiteindelijk bevinden, stelt het nieuwe sample voor.

2.5.4 Gibbs Sampling

Gibbs Sampling is een concreet sampling-algoritme, gebaseerd op MCMC Sampling, dat toelaat om op efficiënte wijze samples te genereren uit eender welke kansdistributie. Het is bovendien zowel op Bayesian Networks als op Markov Networks toepasbaar. In dit deel bestuderen we het eigenlijke algoritme, en in deel 2.5.5 zien we hoe Gibbs Sampling neerkomt op MCMC Sampling.

Beschouw een Markov Network geparametriseerd volgens een verzameling factoren Φ, en noem de bijhorende kansdistributie $P(\mathcal{Z})$. Indien we naar een marginale kans $P(\mathcal{Y}{=}y)$ op zoek zijn, met $\mathcal{Y} \subset \mathcal{Z}$, gebruiken we Gibbs Sampling om te samplen uit de kansdistributie $P(\mathcal{Z})$. Indien we naar een conditionele kans $P(\mathcal{Y}{=}y \mid \mathcal{E}{=}e)$ op zoek zijn, gebruiken we Gibbs Sampling om rechtstreeks uit de a-posteriori kansdistributie $P((\mathcal{Z}{-}\mathcal{E}) \mid \mathcal{E}{=}e)$ te samplen – herinner dat $\mathcal{E} \subset \mathcal{Z}$ en $\mathcal{Y} \subseteq (\mathcal{Z} - \mathcal{E})$. Voor de algemeenheid zullen we samplen uit de kansdistributie $P'(\mathcal{X}) = P(\mathcal{X}|\mathcal{E}{=}e)$ met $\mathcal{X} \subseteq \mathcal{Z}$, wat beide scenario's omvat.[10] Stel bovendien dat we T samples willen genereren. Het Gibbs Sampling algoritme doorloopt dan volgende stappen:

1. Genereer een initieel sample $x^{(0)} = \langle x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)} \rangle$, met $n = |\mathcal{X}|$, gebruikmakend van Forward Sampling.
2. Nu moeten we nog de overige $T - 1$ samples genereren. Het t-de sample genereren we door:
   - 2.1 Beschouw het vorige sample $x^{(t-1)}$ als het (voorlopige) huidige sample $x^{(t)}$.
   - 2.2 Doorloop iedere variabele X_i.
   - 2.3 Vervang $x_i^{(t)}$, de waarde voor variabele X_i in het huidige sample, door een nieuwe waarde. Deze nieuwe waarde samplen we uit $P'(X_i|x_{-i})$.
   - 2.4 Wanneer we alle X_i's doorlopen hebben, is het nieuwe sample klaar. Het maakt nu dus deel uit van de output van het algoritme.

Om in stap 2.3 te kunnen samplen uit $P'(X_i|x_{-i})$, moeten we deze kansdistributie eerst opstellen als volgt. Herinner dat iedere factor ϕ_j gedefinieerd is over een eigen verzameling variabelen $\mathcal{S}_j$. We introduceren de notatie $(x_j)_{-i}$ om te verwijzen naar een toekenning van een waarde aan de variabelen in $\mathcal{S}_j - \{X_i\}$. We stellen de kansdistributie $P'(X_i|x_{-i})$ op door voor iedere $x_i \in Dom(X_i)$ de

[10] Indien $\mathcal{X} = \mathcal{Z}$ en bijgevolg $\mathcal{E} = \emptyset$, stelt $P'(\mathcal{X})$ de (gewone) kansdistributie $P(\mathcal{Z})$ voor; indien $\mathcal{X} \subset \mathcal{Z}$ en bijgevolg $\mathcal{E} \neq \emptyset$, stelt $P'(\mathcal{X})$ een a-posteriori kansdistributie voor.

kans $P'(x_i|x_{-i})$ te berekenen, wat als volgt kan. Herinner dat we een conditionele kans kunnen berekenen door $P(y|e) = P(y,e)/P(e)$, waardoor geldt:

$$P'(x_i|x_{-i}) = \frac{P'(x_i, x_{-i})}{P'(x_{-i})} = \frac{P'(x_i, x_{-i})}{\sum_{x'_i \in Dom(X_i)} P'(x'_i, x_{-i})} = \frac{P'(x_i, x_{-i})}{\sum_{x'_i} P'(x'_i, x_{-i})}$$

In de tweede gelijkheid passen we de definitie van de marginale kans toe. De laatste gelijkheid stelt simpelweg de verkorte notatie voor. Iedere kans is gedefinieerd als het product van de factoren (zie vergelijking 3). We delen deze factorisatie nu op in de factoren die X_i gebruiken en de factoren die X_i niet gebruiken:

$$\begin{aligned} P'(x_i|x_{-i}) &= \frac{P'(x_i, x_{-i})}{\sum_{x'_i} P'(x'_i, x_{-i})} \\ &= \frac{\frac{1}{Z} \cdot \prod_{j:X_i \in S_j} \phi_j(x_i, (x_j)_{-i}) \cdot \prod_{j:X_i \notin S_j} \phi_j(x_j)}{\frac{1}{Z} \cdot \sum_{x'_i} \prod_{j:X_i \in S_j} \phi_j(x'_i, (x_j)_{-i}) \cdot \prod_{j:X_i \notin S_j} \phi_j(x_j)} \\ &= \frac{\prod_{j:X_i \in S_j} \phi_j(x_i, (x_j)_{-i})}{\sum_{x'_i} \prod_{j:X_i \in S_j} \phi_j(x'_i, (x_j)_{-i})} \end{aligned}$$

De laatste gelijkheid verklaren we als volgt: aangezien het tweede product in de noemer geen x'_i bevat, neemt het niet deel aan de sommatie, waardoor we het buiten de sommatie kunnen plaatsen en bijgevolg kunnen schrappen.

Het is interessant na te gaan welke variabelen we precies nodig hebben om een kans $P'(x_i|x_{-i})$ te berekenen. Herinner dat voor iedere factor $\phi_j(S_j)$ de variabelen/knopen S_j een clique vormen in de graaf. Alle variabelen/knopen in een clique zijn buren van elkaar. Aangezien we voor het berekenen van $P'(x_i|x_{-i})$ alleen factoren dienen te beschouwen die X_i gebruiken, is iedere benodigde variabele een buur van X_i. De verzameling buren van een variabele/knoop X in een Markov Network, noemt men het 'Markov Blanket' van X. Om $P'(x_i|x_{-i})$ te berekenen, dienen we dus alleen te kijken naar de waarde die de variabelen in het Markov Blanket van X_i aannemen; niet noodzakelijk naar de waarde van alle X_{-i}.

2.5.5 Gibbs Sampling en MCMC Sampling

We zullen nu verduidelijken dat Gibbs Sampling eigenlijk neerkomt op MCMC Sampling, dus op een 'random walk' doen in een Markov Chain waarvan de stationaire distributie overeenkomt met $P'(\mathcal{X})$. De impliciet gedefinieerde Markov Chain – ook wel de Gibbs Chain genoemd – bevat één toestand per mogelijke combinatie van waarden van de variabelen; op het transitiemodel komen we zo dadelijk terug. Een random walk begint met het kiezen van een starttoestand, wat meteen het eerste sample vormt. Dit komt overeen met stap 1 van het algoritme.

De random walk gebruikt vervolgens het transitiemodel om naar een volgende toestand te gaan, die het volgende sample voorstelt. Indien het transitiemodel gedefinieerd is als de samenstelling van een individueel transitiemodel

per variabele, past de random walk dus iedere $\mathcal{T}_i$ toe om tot de nieuwe toestand (en dus het nieuwe sample) te komen. Dit komt overeen met stap 2 van het algoritme. Merk namelijk op dat het huidige sample overeenkomt met de huidige toestand waarin de random walk zich bevindt. In stap 2.3 updaten we een specifieke x_i van het sample; de overige elementen in het sample laten we ongewijzigd. Dit komt dus overeen met het toepassen van één $\mathcal{T}_i$. Zoals stap 2.2 laat zien, doorlopen we alle variabelen, en passen we dus alle $\mathcal{T}_i$'s achtereenvolgens toe.

We hebben dus duidelijk gemaakt dat Gibbs Sampling neerkomt op een random walk door een Markov Chain. We hebben er echter nog niet bij stilgestaan of deze Markov Chain de gewenste stationaire distributie heeft, zijnde $P'(\mathcal{X})$. Daartoe kijken we naar het (impliciet gedefinieerde) transitiemodel van de Markov Chain. In stap 2.3 samplen we de nieuwe x_i uit $P'(X_i|x_{-i})$. Bijgevolg definieerden we $\mathcal{T}_i$ impliciet als:

$$\mathcal{T}_i((x_{-i}, x_i) \rightarrow (x_{-i}, x'_i)) = P'(x'_i|x_{-i}) \tag{11}$$

Om te bewijzen dat de a-posteriori kansdistributie $P'(\mathcal{X})$ inderdaad de stationaire distributie van deze Markov Chain is, moeten we bewijzen dat $P'(\mathcal{X})$ de stationaire distributie is voor iedere individuele $\mathcal{T}_i$. Het bewijs voor een specifieke $\mathcal{T}_i$ verloopt als volgt. Wegens vergelijking 10 moeten we aantonen dat:

$$P'(\mathcal{X}{=}(x_{-i}, x'_i)) = \sum_{x_i \in Dom(\mathcal{X}_i)} P'(\mathcal{X}{=}(x_{-i}, x_i)) \cdot \mathcal{T}_i((x_{-i}, x_i) \rightarrow (x_{-i}, x'_i))$$

Wegens vergelijking 11 geldt:

$$P'(\mathcal{X}{=}(x_{-i}, x'_i)) = \sum_{x_i \in Dom(\mathcal{X}_i)} P'(\mathcal{X}{=}(x_{-i}, x_i)) \cdot P'(x'_i|x_{-i})$$

Merk op dat $P'(x'_i|x_{-i})$ eigenlijk niet deelneemt aan de sommatie; deze factor bevat x_i niet. Bijgevolg kunnen we deze buiten de sommatie brengen:

$$P'(\mathcal{X}{=}(x_{-i}, x'_i)) = P'(x'_i|x_{-i}) \cdot \sum_{x_i \in Dom(\mathcal{X}_i)} P'(\mathcal{X}{=}(x_{-i}, x_i))$$

Vermits de sommatie over alle waarden van X_i loopt, komt dit neer op een marginalisatie, waardoor we de marginale kans $P'(x_{-i})$ overhouden:

$$P'(\mathcal{X}{=}(x_{-i}, x'_i)) = P'(x'_i|x_{-i}) \cdot P'(x_{-i})$$

Wegens de kettingregel geldt deze gelijkheid inderdaad. We concluderen dat de a-posteriori kansdistributie $P'(\mathcal{X}) = P(\mathcal{X}|e)$ de stationaire distributie is van de impliciet gedefinieerde Markov Chain.

Dus, om een marginale kans $P(y)$ (resp. een conditionele kans $P(y|e)$) te benaderen, genereren we m.b.v. Gibbs Sampling samples uit de kansdistributie $P(\mathcal{Z})$ (resp. de a-posteriori kansdistributie $P((\mathcal{Z}{-}\mathcal{E}) \mid e)$). Vervolgens benaderen we de gevraagde kans door de fractie samples waarvoor geldt dat $\mathcal{Y}{=}y$. Voor de volledigheid vermelden we nog dat men de eerste 20 samples best negeert, omdat deze in het algemeen afkomstig zijn uit een kansdistributie die nog te ver

van de stationaire kansdistributie – zijnde de kansdistributie waaruit we willen samplen – afligt [16].

2.6 Bayesian Network Learning met Complete Data

In dit deel introduceren we eerst wat learning betekent in de context van Probabilistic Graphical Models. Vervolgens bekijken we een specifiek scenario: het leren van de parameters van een Bayesian Network op basis van complete training data, m.b.v. Maximum Likelihood Estimation.

2.6.1 Introductie

Toen we het over inference hadden (zie deel 2.4 en 2.5), gingen we ervan uit dat het Probabilistic Graphical Model (PGM) volledig gegeven was: enerzijds kenden we de structuur van de graaf, en anderzijds kenden we de parameters van de parametrisatie. Voor een Bayesian Network houdt dat laatste in dat alle conditionele kansdistributies reeds ingevuld waren; voor een Markov Network houdt dat in dat alle factoren reeds ingevuld waren. In de praktijk is het echter niet eenvoudig om de structuur en parameters van een PGM manueel te bepalen, waardoor men overschakelt op 'learning'.

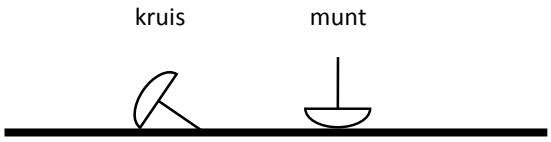
Veronderstel dat we een PGM willen opstellen waarvan we de onderliggende kansdistributie P noemen. Bij de learning-aanpak verzamelt men eerst een grote hoeveelheid data, zogenaamde training data. Dit is een verzameling samples waarvoor we ervan uitgaan dat P deze gegenereerd zou kunnen hebben. We gaan er dus van uit dat P een grote kans zou koppelen aan ieder van die samples. Met behulp van learning-technieken kunnen we de structuur en parameters van een PGM zo bepalen dat de onderliggende kansdistributie inderdaad een grote kans koppelt aan de training data. Meer bepaald hopen we een kansdistributie te leren die een goede benadering is van P. Deze benadering kunnen we uiteindelijk gebruiken om een kans te associëren aan samples die geen deel uitmaakten van de training data.

We zullen alleen bestuderen hoe de parameters van een PGM geleerd kunnen worden; het leren van de structuur van een PGM is niet relevant in de context van DeepDive en zou ons veel te ver leiden. Aangezien we uitgaan van een vaste structuur, ligt het vast uit welke variabelen het PGM bestaat. Daardoor kunnen we een onderscheid maken in de training data die we gebruiken. Enerzijds kunnen de samples simpelweg bestaan uit een toekenning aan iedere variabele, wat we 'complete data' of 'fully observed data' noemen. Anderzijds zijn in de praktijk dikwijls alleen samples voorhanden die bestaan uit een toekenning aan een deelverzameling van de variabelen, wat we 'incomplete data' of 'partially observed data' noemen.

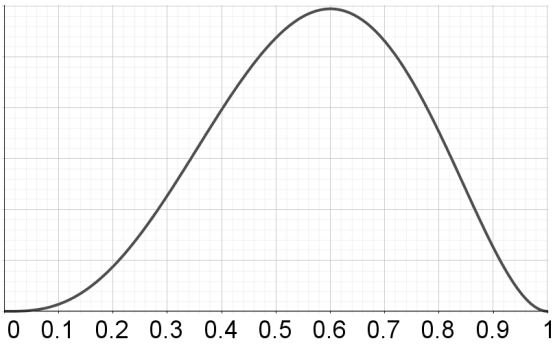
2.6.2 Maximum Likelihood Estimation principe

In dit deel focussen we op het leren van de parameters van een Bayesian Network met complete data. Hiervoor gebruiken we een methode genaamd Maximum Likelihood Estimation (MLE). Vooraleer we zien hoe MLE kan worden toegepast op Bayesian Networks, illustreren we het algemene principe.

Een uitstekend voorbeeld om de werking van MLE te illustreren, is het opgooien van een duimspijker [15]. Deze kan op kruis of munt landen, zoals



Figuur 11: Definitie van kruis en munt voor een duimspijker.



Figuur 12: Plot van $L(\theta : (K, M, M, K, K))$.

gedefinieerd in figuur 11. Beschouw een probabilistisch model dat de "werking" van de duimspijker beschrijft. Het bestaat uit één parameter, θ, zijnde de kans dat de duimspijker bij het opgooien op kruis landt; de kans dat de duimspijker op munt landt, is $1-\theta$. We benadrukken dat dit model geen PGM is, omdat we zonder graaf werken. Het is op voorhand niet duidelijk wat een geschikte waarde voor θ is, aangezien we een ander gedrag verwachten dan bij het opgooien van een (unbiased) muntstuk. Daarom zullen we hier een learning-techniek voor gebruiken, namelijk MLE.

Het MLE principe zegt dat we de parameters zo moeten kiezen dat de kans die het model aan de training data koppelt, maximaal is. Om een geschikte waarde voor θ te vinden, hebben we dus in de eerste plaats training data nodig. Deze kunnen we eenvoudig verzamelen door een duimspijker te nemen en deze een aantal keer op te gooien, en telkens te noteren of hij op kruis (K) of munt (M) landde. Iedere worp levert ons dus één sample. Een mogelijke training dataset[11] is $\mathcal{D} = (K, M, M, K, K)$.

De kans die het model, geparametriseerd door θ, aan deze training data koppelt, is: $P((K, M, M, K, K) : \theta) = P(K : \theta) \cdot P(M : \theta) \cdot P(M : \theta) \cdot P(K : \theta) \cdot P(K : \theta) = \theta^3 \cdot (1-\theta)^2$. Meer algemeen definieert men in deze context de likelihood-functie: $L(\theta : (K, M, M, K, K)) = P((K, M, M, K, K) : \theta)$. Vermits de likelihood afhangt van de keuze van θ, kunnen we hier een plot van maken, zoals getoond in figuur 12. We zien hoe de kans varieert in functie van onze keuze van θ. Het MLE principe stelt dat we θ zo moeten kiezen dat $L(\theta : (K, M, M, K, K))$ maximaal is. De figuur toont ons dat we daartoe $\theta = 0.6$ moeten kiezen, wat we intuïtief hadden kunnen verwachten, vermits we $3/5$ keer kruis gooiden.

Laat ons het MLE principe nu veralgemenen. Ten eerste noteren we de

[11] Men spreekt van een 'set' (verzameling) alhoewel eenzelfde sample meermaals kan voorkomen. Een correctere benaming zou 'ongesorteerde lijst' zijn.

training data als $\mathcal{D} = (d_1, \ldots, d_M)$, met M het aantal samples en ieder sample d_i een toekenning aan alle variabelen $\mathcal{X}$.

In het voorbeeld met de duimspijker gebruikten we impliciet een random variable X met twee mogelijke waarden: kruis en munt. Aangezien dit een binaire random variable is, volstond het om één parameter te gebruiken. Het is nuttig om in het algemeen per variabele evenveel parameters te gebruiken als de grootte van zijn domein, b.v. θ_{kruis} en θ_{munt}, met als restrictie dat hun som 1 moet zijn. Voor een model met één variabele die K mogelijke waarden heeft, is de verzameling parameters van het model dan gedefinieerd als $\Theta = \{\theta_1, \ldots, \theta_K\}$, waarvan de som 1 moet zijn en iedere $\theta_k \in [0; 1]$.

Verder definiëren we de likelihood-functie als:

$$L(\Theta : \mathcal{D}) = P(\mathcal{D} : \Theta) = \prod_{m=1}^{M} P(d_m : \Theta) \tag{12}$$

Vermits iedere d_m één van de K mogelijke waarden aanneemt, kunnen we dit product herschrijven als:

$$L(\Theta : \mathcal{D}) = \prod_{k} \theta_k^{M[k]} \tag{13}$$

met $M[k]$ het aantal samples dat de k-de waarde aannam. In het voorbeeld met de duimspijker schreven we de likelihood ook reeds zo: $\theta^3 \cdot (1-\theta)^2 = \theta^3_{kruis} \cdot \theta^2_{munt}$. In de praktijk gebruikt men vaak de log-likelihood, simpelweg gedefinieerd als $\mathcal{L}(\Theta : \mathcal{D}) = \log(L(\Theta : \mathcal{D})) = \sum_k M[k] \cdot \log(\theta_k)$.

Ten slotte veralgemenen we het bepalen van het maximum van de (log-)likelihood, door de afgeleide van de (log-)likelihood gelijk te stellen aan 0. Laat ons terug naar het voorbeeld met de duimspijker gaan. Stel dat we $M[0]$ keer kruis gooiden en $M[1]$ keer munt gooiden, dan geldt: $\mathcal{L}(\theta : \mathcal{D}) = M[0] \cdot \log(\theta) + M[1] \cdot \log(1 - \theta)$. Het maximum, waar de afgeleide 0 is, is dan:

$$\begin{aligned}
&\frac{M[0]}{\theta} - \frac{M[1]}{1-\theta} = 0 \Leftrightarrow \frac{M[0] \cdot (1-\theta) - M[1] \cdot \theta}{\theta \cdot (1-\theta)} = 0 \Leftrightarrow \\
&M[0] - M[0] \cdot \theta - M[1] \cdot \theta = 0 \Leftrightarrow \theta \cdot (M[0] + M[1]) = M[0] \Leftrightarrow \\
&\theta = \frac{M[0]}{M[0] + M[1]} = \frac{M[0]}{M}
\end{aligned}$$

We moeten θ dus kiezen als de fractie samples die kruis was. Met de concrete dataset $\mathcal{D} = (K, M, M, K, K)$ krijgen we inderdaad opnieuw dat $\theta = 0.6$. Ook meer algemeen geldt dat men het maximum van de (log-)likelihood bekomt door iedere θ_k te kiezen als de fractie samples die de k-de waarde aannam:

$$\theta_k = \frac{M[k]}{M} \tag{14}$$

2.6.3 MLE voor Bayesian Networks

We zullen nu zien we hoe we MLE kunnen toepassen op een Bayesian Network. De training data noemen we $\mathcal{D} = (d_1, \ldots, d_M)$. De parameters Θ van het Bayesian Network bestaan, zoals eerder vermeld, uit de invulling van alle conditionele kansdistributies. Informeel gesproken is er dus voor iedere conditi-

onele kansdistributie één parameter per cel van het tabelletje. We introduceren de notatie $d_m[X_i]$ om, in het m-de sample, de waarde overeenkomstig met variabele X_i aan te duiden. De notatie $d_m[\mathcal{Z}]$ heeft een analoge betekenis, met $\mathcal{Z}$ een deelverzameling van de variabelen.

Om de likelihood te bepalen, vertrekken we vanuit vergelijking 12:

$$\begin{aligned} L(\Theta : \mathcal{D}) &= \prod_{m=1}^{M} P(d_m : \Theta) \\ &= \prod_{m=1}^{M} \prod_{i=1}^{n} P(d_m[X_i] \mid d_m[Pa_{X_i}] : \Theta) \qquad \text{(vergelijking 1)} \\ &= \prod_{i=1}^{n} \left[\prod_{m=1}^{M} P(d_m[X_i] \mid d_m[Pa_{X_i}] : \Theta) \right] \quad \text{(product is commutatief)} \end{aligned}$$

Merk op dat een specifieke kans $P(d_m[X_i] \mid d_m[Pa_{X_i}] : \Theta)$ eigenlijk niet van alle parameters Θ afhangt, maar slechts van een deelverzameling die we als $\Theta_{X_i|Pa_{X_i}}$ noteren. Deze stellen de invulling van de conditionele kansdistributie van X_i voor. Verder kunnen we het gedeelte tussen de vierkante haakjes definiëren als de 'lokale likelihood' voor een variabele X_i, genoteerd als $L_i(\Theta_{X_i|Pa_{X_i}} : \mathcal{D})$, waardoor we krijgen:

$$L(\Theta : \mathcal{D}) = \prod_{i=1}^{n} L_i(\Theta_{X_i|Pa_{X_i}} : \mathcal{D}) \quad \text{met}$$

$$L_i(\Theta_{X_i|Pa_{X_i}} : \mathcal{D}) = \prod_{m=1}^{M} P(d_m[X_i] \mid d_m[Pa_{X_i}] : \Theta_{X_i|Pa_{X_i}}) \tag{15}$$

Vermits iedere conditionele kansdistributie andere parameters gebruikt, gebruikt iedere L_i een deelverzameling parameters die niet overlapt met de deelverzamelingen parameters die de andere L_i's gebruiken. Bijgevolg is $L(\Theta : \mathcal{D})$ een product van onafhankelijke factoren, waardoor we zeggen dat de likelihoodfunctie 'decomponeerbaar' is. Het cruciale gevolg hiervan is dat we $L(\Theta : \mathcal{D})$ kunnen maximaliseren door iedere L_i afzonderlijk van de rest te maximaliseren.

Nu rest nog de vraag hoe we een lokale likelihood L_i, zoals gedefinieerd volgens vergelijking 15, maximaliseren. Om de notatie overzichtelijk te houden, introduceren we $X = X_i$ en $U = Pa_{X_i}$. De conditionele kansdistributie (geassocieerd aan $X|U$) bestaat uit één parameter voor iedere combinatie van waarden die X en U kunnen aannemen, genoteerd als $\theta_{x|u}$. Iedere individuele kans in vergelijking 15 komt dus overeen met een $\theta_{x|u}$, afhankelijk van welke waarden X en U aannamen in het sample. Analoog aan vergelijking 13, kunnen we de lokale likelihood nu herschrijven als:

$$L_i(\Theta_{X|U} : \mathcal{D}) = \prod_{x \in Dom(X)} \prod_{u \in Dom(U)} \theta_{x|u}^{M[x,u]}$$

met $M[x, u]$ het aantal samples waarin $X{=}x$ en $U{=}u$. Om dit product te maximaliseren, moeten we iedere $\theta_{x|u}$ maximaliseren. Analoog aan vergelijking

14 kiezen we daartoe

$$\theta_{x|u} = \frac{M[x,u]}{M[u]} \tag{16}$$

met $M[u] = \sum_{x \in Dom(X)} M[x,u]$. Indien we met complete training data werken, is het leren van de parameters van een Bayesian Network m.b.v. MLE dus zeer eenvoudig: we optimaliseren iedere parameter afzonderlijk van de rest, door vergelijking 16 toe te passen.

2.7 Bayesian Network Learning met Incomplete Data

Wanneer we met 'incomplete' of 'partially observed' data te maken hebben, wordt MLE een stuk gecompliceerder.

2.7.1 Moeilijkheden

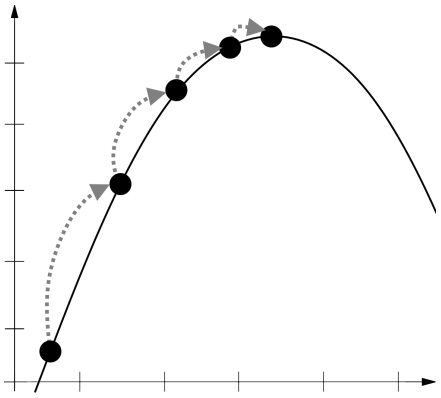
Beschouw een Bayesian Network waarvan we de onderliggende kansdistributie $P(\mathcal{X})$ noemen, met $\mathcal{X} = \{X_1, \ldots, X_n\}$ al de variabelen. Toen we in het vorige deel met complete training data werkten, bestond een sample van de training data uit een toekenning van een waarde aan iedere variabele X_i. Nu we met incomplete data werken, is ieder sample een toekenning aan $\mathcal{O} \subset \mathcal{X}$. De $\mathcal{O}$ staat voor de geobserveerde ('observed') variabelen. De overige variabelen, die we niet observeren, noemen we $\mathcal{H} = \mathcal{X} - \mathcal{O}$, de verborgen ('hidden') variabelen. Vermits ieder sample mogelijks een toekenning voorziet aan een andere deelverzameling variabelen, noteren we $\mathcal{O}[m]$ om naar de geobserveerde variabelen in het m-de sample uit de training data te verwijzen. Hun eigenlijke waarden noteren we als $o[m]$, wat dus op het sample zelf neerkomt. Analoog noteren we $\mathcal{H}[m]$ en $h[m]$.

We verduidelijken met een klein voorbeeldje. Beschouw een Bayesian Network $X \to Y \to Z$, met X, Y en Z allemaal binaire variabelen. Stel dat het eerste sample van de training data $o[1] = \langle x_0, z_1 \rangle$ is. Er geldt $\mathcal{O}[1] = \{X, Z\}$ en $\mathcal{H}[1] = \{Y\}$. In het vorige deel gebruikten we een compleet sample als $\langle x_0, y_0, z_0 \rangle$ om de kans $P(x_0, y_0, z_0)$ te bepalen. Analoog gebruiken we nu een incompleet sample als $\langle x_0, z_1 \rangle$ om de marginale kans $P(x_0, z_1)$ te bepalen. Herinner uit deel 2.2.1 dat we een marginale kans berekenen door over alle mogelijke waarden van de overige variabelen te sommeren. Hier geldt dus $P(x_0, z_1) = \sum_{y \in Y} P(x_0, y, z_1)$. Meer algemeen geldt: $P(o[m]) = \sum_{h[m] \in \mathcal{H}[m]} P(o[m], h[m])$, met $h[m]$ een mogelijke toekenning aan alle verborgen variabelen in het m-de sample.

De likelihood-functie in geval van incomplete data is dus gedefinieerd als:

$$L(\Theta : \mathcal{D}) = \prod_{m=1}^{M} P(o[m] : \Theta) = \prod_{m=1}^{M} \sum_{h[m]} P(o[m], h[m] : \Theta) \tag{17}$$

Doordat we met incomplete data werken, krijgen we met marginale kansen te maken, die sommaties introduceren in de likelihood-functie. Toen we met complete data werkten, was de likelihood decomponeerbaar als een product van lokale likelihood-functies, de L_i's. Daardoor konden we uiteindelijk iedere parameter afzonderlijk van de rest optimaliseren (volgens vgl. 16). Echter, nu we met incomplete data werken, kunnen we de likelihood niet meer zo schrijven; de likelihood is niet meer decomponeerbaar. Bijgevolg kunnen we niet langer



Figuur 13: Illustratie van de werking van gradient ascent.

iedere parameter afzonderlijk optimaliseren, en moeten we dus overschakelen op globale optimalisatietechnieken.

Een tweede moeilijkheid is de vorm van deze functie. In het geval van complete data heeft de functie slechts één maximum; er zijn geen lokale maxima, wat men een unimodale functie noemt. In het geval van incomplete data echter, zorgt de sommatie voor het sommeren van unimodale functies. Aangezien hun maxima niet noodzakelijk overeenkomen, zijn meerdere lokale maxima mogelijk, wat men een multimodale functie noemt. Dit bemoeilijkt het globale optimalisatieproces. We komen hier in deel 2.7.3 op terug.

2.7.2 Gradient Ascent

Gradient ascent is een veel gebruikt algoritme om het maximum van een functie te benaderen. Het berekenen van het maximum van een functie is een taak die in vele wetenschappelijke disciplines aan bod komt. Bijgevolg wordt gradient ascent niet louter binnen de context van Probabilistic Graphical Models gebruikt.

Gradient ascent bepaalt het maximum van een functie op iteratieve wijze. Het startpunt is een willekeurige toekenning van een waarde aan iedere parameter. Van daaruit neemt het algoritme bij iedere iteratie een stap in de richting van het maximum. Iedere iteratie worden de parameters van de functie dus zo aangepast dat ze korter bij het maximum komen te liggen. Figuur 13 illustreert dit proces (voor een functie die slechts één parameter gebruikt). De parameters worden aangepast op basis van de gradiënt van de functie in het huidige punt. De gradiënt van een functie is de vector bestaande uit de partiële afgeleide van die functie naar iedere variabele. De parameters worden aangepast op basis van de gradiënt omdat deze de richting van de sterkste stijging aanduidt. Om te weten wanneer het algoritme moet stoppen, wordt gekeken naar het verschil tussen de huidige waarden van de parameters en die in de vorige iteratie. Hiervoor kunnen we bijvoorbeeld de Euclidische afstand[12] gebruiken. Meer bepaald kijken we of dat verschil onder een bepaalde drempelwaarde ligt, waardoor we concluderen dat het algoritme geconvergeerd is (naar het maximum).

[12]De Euclidische afstand, ook wel L2-norm genoemd, is gedefinieerd als de vierkantswortel van de som van de kwadratische verschillen tussen de overeenkomstige parameters.

We zullen het gradient ascent algoritme nu formeel voorstellen. Beschouw een functie $f(\Theta)$, met $\Theta = \{\theta_1, \ldots, \theta_n\}$. De gradiënt noteren we als $\nabla f(\Theta) = (\frac{\partial f(\Theta)}{\partial \theta_1}, \ldots, \frac{\partial f(\Theta)}{\partial \theta_n})$. Het algoritme verloopt als volgt:

1. Ken een willekeurige waarde toe aan iedere parameter θ_i. De verzameling waarden noteren we als $\Theta^{(1)}$.
2. Noem $\Theta^{(t)}$ de huidige verzameling waarden van de parameters. Update deze door: $\Theta^{(t+1)} = \Theta^{(t)} + \alpha \cdot \nabla f(\Theta^{(t)})$.
3. Check of $\|\Theta^{(t+1)} - \Theta^{(t)}\| > \delta$, met δ de te kiezen drempelwaarde. Zo ja, ga terug naar stap 2. Zo nee, beëindig het algoritme; de functie bereikt haar maximum in $\Theta^{(t+1)}$.

De parameter α noemt men de 'learning rate'. Deze constante beïnvloedt hoe groot de aanpassingen aan Θ zijn; α beïnvloedt de grootte van de stap die we maken. Kiest men een te grote waarde voor α, dan riskeert men voorbij het maximum te stappen. Kiest men een te kleine waarde voor α, dan zullen er zeel veel iteraties nodig zijn vooraleer het maximum bereikt wordt. Het is dus niet eenvoudig om een geschikte waarde voor α te vinden. Men moet hier wat mee experimenteren.

Een andere vaak voorkomende taak in de praktijk, is het minimum van een functie berekenen. Hiervoor kan het algoritme 'gradient descent' gebruikt worden, wat analoog aan gradient ascent werkt. Het enige verschil qua implementatie is dat men bij het updaten van de parameters de gradiënt ervan aftrekt i.p.v. erbij optelt. In de praktijk worden vaak geavanceerdere algoritmes gebruikt om een functie te optimaliseren. Deze zijn echter fundamenteel gebaseerd op de klassieke gradient ascent/descent die we hier beschreven. Zulke geavanceerde algoritmes kijken niet alleen naar de eerste partiële afgeleiden, maar ook naar de tweede partiële afgeleiden, waardoor het algoritme sneller convergeert. Voorbeelden van zulke algoritmes zijn Conjugate Gradient Descent en BFGS; een uitstekende bron hierover is LeCun [17].

2.7.3 Gradient Ascent voor MLE

Laten we nu opnieuw kijken naar het toepassen van MLE op Bayesian Networks in de context van incomplete data. Ondertussen weten we dat we de likelihood-functie globaal moeten maximaliseren, waarvoor we gradient ascent kunnen gebruiken. Laat ons ervan uitgaan dat we met niet met de likelihood, maar wel met de log-likelihood $\mathcal{L}(\Theta : \mathcal{D})$ werken. Om gradient ascent toe te kunnen passen, moeten we de gradiënt van $\mathcal{L}$ kennen. De gradiënt bestaat uit de partiële afgeleide van $\mathcal{L}$ naar iedere parameter $\theta_{x|u}$, met x een concrete waarde voor een variabele X van de kansdistributie en u een concrete combinatie van waarden voor de parents van X.

We beginnen met de afgeleide van de kans op een bepaald incompleet sample o naar een parameter $\theta_{x|u}$. Dit lemma zal de uiteindelijke berekening van de partiële afgeleide van $\mathcal{L}$ naar $\theta_{x|u}$ vergemakkelijken. Herinner dat een $\theta_{x|u}$ overeenkomt met de entry $P(x|u)$ (in de conditionele kansdistributie aan X geassocieerd). Beide notaties kunnen dus door elkaar gebruikt worden, maar voor het bewijs is de tweede het handigst. Het lemma stelt:

$$\frac{\partial P(o)}{\partial P(x|u)} = \frac{1}{P(x|u)} \cdot P(x, u, o) \tag{18}$$

Het bewijs verloopt als volgt. Indien we een compleet sample s zouden gebruiken, is de kans $P(s)$ het product van de relevante entries in de conditionele kansdistributies (zie vgl. 1). We beschouwen de partiële afgeleide van $P(s)$ naar een $P(x|u)$. Indien $P(x|u)$ niet voorkomt in dit product, is de afgeleide gewoon gelijk aan 0. In het andere geval verdwijnt $P(x|u)$ als enige factor uit het product (doordat we ernaar afleiden), waardoor we de afgeleide kunnen schrijven als $P(s)/P(x|u)$. Samengevat geldt:

$$\frac{\partial P(s)}{\partial P(x|u)} = \begin{cases} \frac{1}{P(x|u)} \cdot P(s) & \text{als } s[X, Pa_X] = (x, u) \\ 0 & \text{anders} \end{cases}$$

Indien we met een incompleet sample o werken, moeten we, zoals we in deel 2.7.1 zagen, sommeren over alle mogelijke combinaties van waarden van de verborgen variabelen. Dit komt op hetzelfde neer als te kijken naar alle mogelijke complete samples waarvoor de waarden overeenkomstig met de geobserveerde variabelen O gelijk zijn aan o. Meer formeel:

$$P(o) = \sum_{s \,:\, s[O]=o} P(s)$$

Gegeven deze twee inzichten berekenen we:

$$\begin{aligned} \frac{\partial P(o)}{\partial P(x|u)} &= \sum_{s \,:\, s[O]=o} \frac{\partial P(s)}{\partial P(x|u)} \\ &= \sum_{s \,:\, s[O]=o,\ s[X,Pa_X]=(x,u)} \frac{1}{P(x|u)} \cdot P(s) \\ &= \frac{1}{P(x|u)} \cdot P(x, u, o) \end{aligned}$$

Bij de tweede gelijkheid voegen we de extra conditie $s[X, Pa_X] = (x, u)$ toe omdat de afgeleide van samples die hier niet aan voldoen toch 0 is. Om tot de laatste gelijkheid te komen, brengen we de constante $1/P(x|u)$ buiten de sommatie en zien we de overgebleven sommatie als de berekening van de marginale kans $P(x, u, o)$.

Met behulp van lemma 18 kunnen we nu bewijzen dat:

$$\frac{\partial \mathcal{L}(\Theta : \mathcal{D})}{\partial \theta_{x|u}} = \frac{1}{\theta_{x|u}} \cdot \sum_{m=1}^{M} P(x, u \mid o[m], \Theta) \tag{19}$$

Het bewijs verloopt als volgt. Merk op dat de notatie $P(x : \Theta)$ equivalent is met $P(x \mid \Theta)$; het komt er in beide gevallen op neer dat Θ gegeven is. De tweede notatie is echter het handigst voor het bewijs. Vertrekkende uit vergelijking 12 bekomen we de log-likelihood:

$$\mathcal{L}(\Theta : \mathcal{D}) = \sum_{m=1}^{M} \log(P(o[m] \mid \Theta))$$

Zoals we in het begin van deze paragraaf vermeldden, is $P(x|u) = \theta_{x|u}$; het was

slechts een kwestie van notatie. Er geldt :

$$\begin{aligned}
\frac{\partial \mathcal{L}(\Theta : \mathcal{D})}{\partial \theta_{x|u}} &= \frac{\partial \mathcal{L}(\Theta : \mathcal{D})}{\partial P(x|u)} \\
&= \sum_{m=1}^{M} \frac{1}{P(o[m] \mid \Theta)} \cdot \frac{\partial P(o[m] \mid \Theta)}{\partial P(x|u)} \\
&= \sum_{m=1}^{M} \frac{1}{P(o[m] \mid \Theta)} \cdot \frac{1}{P(x|u)} \cdot P(x, u, o[m] \mid \Theta) \quad \text{(lemma 18)} \\
&= \sum_{m=1}^{M} \frac{1}{P(x|u)} \cdot P(x, u \mid o[m], \Theta) \quad \text{(kettingregel)} \\
&= \frac{1}{\theta_{x|u}} \cdot \sum_{m=1}^{M} P(x, u \mid o[m], \Theta)
\end{aligned}$$

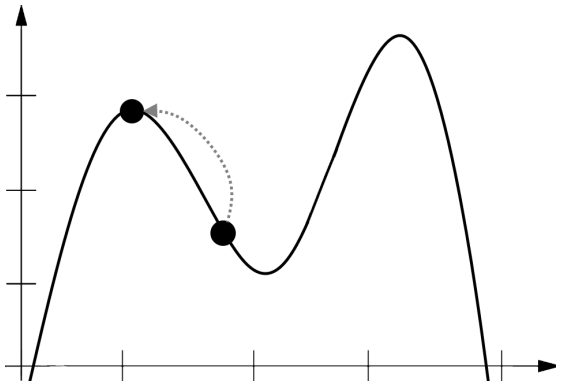
In de voorlaatste gelijkheid gebruikten we de kettingregel: $P(x, u, o[m] \mid \Theta) = P(o[m] \mid \Theta) \cdot P(x, u \mid o[m], \Theta)$.

Dankzij vergelijking 19 kunnen we nu gradient ascent toepassen (zie deel 2.7.2) om de log-likelihood te maximaliseren, en zodoende geschikte waarden te bekomen voor de parameters van het Bayesian Network, gebaseerd op incomplete training data. Om de gradiënt te berekenen, passen we vergelijking 19 toe voor iedere combinatie van x en u. Bijgevolg berekenen we voor het m-de sample de kans $P(x, u \mid o[m], \Theta)$ voor iedere combinatie van x en u, wat op de kansdistributie $P(X[m], U[m] \mid o[m], \Theta)$ neerkomt. Om die kansdistributie op te stellen, gebruiken we inference. We zien dus dat inference gebruikt wordt in iedere iteratie van gradient ascent, wat een grote computationele impact met zich meebrengt en bijgevolg het belang van approximate inference benadrukt. Wanneer we leren met incomplete data ligt de computationele kost dus aanzienlijk hoger dan wanneer we met complete data werken.

Herinner ten slotte dat de likelihood functie, alsook de log-likelihood functie, multimodaal is indien we met incomplete data werken. Bijgevolg is het perfect mogelijk dat gradient ascent in een lokaal maximum terecht komt i.p.v. in het globale maximum. Dit hangt van het startpunt af, zoals figuur 14 illustreert. Er bestaan verschillende technieken om met het probleem van locale optima om te gaan. Een eenvoudig voorbeeld hiervan is het gebruik van 'random restarts'. Gradient ascent wordt dan meermaals uitgevoerd, telkens vanuit een ander willekeurig startpunt, en het maximum met de hoogste waarde is het eindresultaat.

2.8 Markov Network Learning

Ten slotte bespreken we hoe we de parameters van een Markov Network kunnen leren. Opnieuw gebruiken we hiervoor Maximum Likelihood Estimation. Net als bij Bayesian Networks maken we het onderscheid tussen complete en incomplete training data.



Figuur 14: Gradient ascent belandt in een lokaal maximum.

2.8.1 Complete data

Beginnen doen we met het geval dat we complete training data gebruiken. Als parametrisatie van het Markov Network gebruiken we het Log-Linear Model uit deel 2.3.5, omdat dit de meest gebruikte parametrisatie is – zo ook bij DeepDive. Zoals we reeds in deel 2.3.5 aangaven, worden de features typisch door de gebruiker zelf gedefinieerd en moeten dus alleen nog de gewichten geleerd worden.

Wanneer men met een Markov Network in standaard parametrisatie (zie vergelijking 3) werkt, kan men het herleiden tot een equivalent Log-Linear Model, zoals reeds in deel 2.3.5 werd beschreven, en vervolgens het leerproces volgen dat we in dit deel beschrijven. Het is echter ook mogelijk om rechtstreeks de parameters van het Markov Network in standaard parametrisatie te leren. In dit geval worden geschikte waarden geleerd voor de compatibiliteiten van de factoren. Het eigenlijke leerproces verloopt volledig analoog aan onderstaande uitleg over het leerproces voor de Log-Linear Model parametrisatie.

Laat ons nu bepalen hoe we de gewichten van een Log-Linear Model kunnen leren, wat we als volgt noteren:

$$\begin{aligned} P(\mathcal{X} : \Theta) &= \frac{1}{Z(\Theta)} \exp\left(\sum_{i=1}^{m} \theta_i f_i(\mathcal{S}_i)\right), \text{ met} \\ Z(\Theta) &= \sum_{x \in Dom(\mathcal{X})} \exp\left(\sum_{i=1}^{m} \theta_i f_i(s_i)\right) \end{aligned} \tag{20}$$

De notatie $f_i(s_i)$ is de verkorte notatie voor $f_i(x[\mathcal{S}_i])$.

Beschouw de training data $\mathcal{D} = (d_1, \ldots, d_M)$. De log-likelihood is nu gedefinieerd als:

$$\begin{aligned} \mathcal{L}(\Theta : \mathcal{D}) &= \sum_m \log(P(d_m : \Theta)) \\ &= \sum_m \log\left(\frac{1}{Z(\Theta)}\right) + \sum_m \log\left(\exp\left(\sum_i \theta_i f_i(s_i)\right)\right) \end{aligned}$$

$$= \sum_m \sum_i \theta_i f_i(s_i) - M \cdot \log(Z(\Theta))$$
$$= \sum_i \theta_i \left(\sum_m f_i(s_i) \right) - M \cdot \log(Z(\Theta))$$

De notatie $f_i(s_i)$ is de verkorte notatie voor $f_i(d_m[\mathcal{S}_i])$. De vorm van de log-likelihood is erg onhandig. Daarom herleiden we deze verder naar de average log-likelihood, die, zoals de naam zegt, gedefinieerd is als het gemiddelde van de log-likelihoods van de samples. Vermits we in $\mathcal{L}(\Theta : \mathcal{D})$ de log-likelihoods van de samples reeds sommeren, rest alleen nog de deling door M. De average log-likelihood is dus gedefinieerd als:

$$\frac{1}{M} \cdot \mathcal{L}(\Theta : \mathcal{D}) = \sum_i \theta_i \cdot E_{\mathcal{D}}[f_i(\mathcal{S}_i)] - \log(Z(\Theta)) \tag{21}$$

met $E_{\mathcal{D}}[f_i(\mathcal{S}_i)]$ de gemiddelde waarde die $f_i(\mathcal{S}_i)$ aanneemt in de dataset $\mathcal{D}$, ook wel de empirische verwachte waarde (expected value) genoemd.

Doordat $Z(\Theta)$ een som bepaalt waar alle parameters in voor komen, is het niet mogelijk de (average log-)likelihood te schrijven als een product van de individuele parameters. De functie is dus niet decomponeerbaar, zelfs niet nu we met complete data werken. Bijgevolg zullen we gradient ascent gebruiken om de (average log-)likelihood te maximaliseren. Laat ons daarom de partiële afgeleide van de average log-likelihood naar een θ_i bepalen.

We beginnen met het uitrekenen van $\frac{\partial}{\partial \theta_i} \log(Z(\Theta))$:

$$\begin{aligned}
\frac{\partial \log(Z(\Theta))}{\partial \theta_i} &= \frac{1}{Z(\Theta)} \cdot \frac{\partial Z(\Theta)}{\partial \theta_i} \\
&= \frac{1}{Z(\Theta)} \cdot \sum_{x \in Dom(\mathcal{X})} \frac{\partial}{\partial \theta_i} \exp\left(\sum_{i=1}^{m} \theta_i f_i(s_i) \right) \\
&= \frac{1}{Z(\Theta)} \cdot \sum_{x \in Dom(\mathcal{X})} \exp\left(\sum_i \theta_i f_i(s_i) \right) \cdot f_i(s_i) \\
&= \sum_{x \in Dom(\mathcal{X})} \frac{1}{Z(\Theta)} \cdot \exp\left(\sum_i \theta_i f_i(s_i) \right) \cdot f_i(s_i) \\
&= \sum_{x \in Dom(\mathcal{X})} P(x : \Theta) \cdot f_i(s_i) \\
&= E_{\Theta}[f_i(\mathcal{S}_i)]
\end{aligned} \tag{22}$$

In de laatste gelijkheid passen we de definitie van de verwachte waarde (expected value) toe. De notatie $E_{\Theta}[f_i(\mathcal{S}_i)]$ is de verkorte notatie voor $E_{P(\mathcal{X}:\Theta)}[f_i(\mathcal{S}_i)]$. De partiële afgeleide van de average log-likelihood naar θ_i kan nu zeer eenvoudig berekend worden:

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \cdot \mathcal{L}(\Theta : \mathcal{D}) = E_{\mathcal{D}}[f_i(\mathcal{S}_i)] - E_{\Theta}[f_i(\mathcal{S}_i)] \tag{23}$$

Zoals we reeds zeiden, vereist het bepalen van $E_{\mathcal{D}}[f_i(\mathcal{S}_i)]$ dat we $f_i(s_i)$ berekenen

voor ieder sample. Anderzijds, om $E_\Theta[f_i(\mathcal{S}_i)]$ te bepalen, moeten we $P(x : \Theta)$ bepalen voor alle $x \in Dom(\mathcal{X})$. Hier gebruiken we inference voor. Aangezien we alle mogelijke kansen moeten bepalen (door inference), en dit in iedere iteratie van gradient ascent, zal men hier typisch approximate inference voor gebruiken.

Hoewel we zelfs voor complete data beroep moeten doen op gradient ascent, is er ook één positieve noot. De log-likelihood is namelijk unimodaal, waardoor random restarts niet nodig zijn. Gradient ascent bereikt dus gegarandeerd het globale maximum, ongeacht het startpunt.

2.8.2 Incomplete data

Door de sommatie die met $Z(\Theta)$ gepaard gaat, zijn we uiteraard ook in het geval van incomplete data verplicht om gradient ascent te gebruiken. Voor we de vorm van de average log-likelihood bepalen, merken we het volgende op. Definitie 20 kunnen we herschrijven als:

$$P(\mathcal{X} : \Theta) = \frac{1}{Z(\Theta)}\widetilde{P}(x) \text{ met } \widetilde{P}(x) = \exp\left(\sum_{i=1}^{m} \theta_i f_i(\mathcal{S}_i)\right)$$

Om de vorm van de average log-likelihood te bepalen, vertrekken we vanuit vergelijking 17:

$$\begin{aligned}
\frac{1}{M} \cdot \mathcal{L}(\Theta : \mathcal{D}) &= \frac{1}{M} \cdot \log\left(\prod_{m=1}^{M} \sum_{h[m]} P(o[m], h[m] : \Theta)\right) \\
&= \frac{1}{M} \cdot \sum_{m=1}^{M} \log\left(\sum_{h[m]} P(o[m], h[m] : \Theta)\right) \\
&= \frac{1}{M} \cdot \sum_{m=1}^{M} \log\left(\frac{1}{Z(\Theta)} \cdot \sum_{h[m]} \widetilde{P}(o[m], h[m] : \Theta)\right) \\
&= \frac{1}{M} \cdot \sum_{m=1}^{M} \log\left(\sum_{h[m]} \widetilde{P}(o[m], h[m] : \Theta)\right) - \log(Z(\Theta))
\end{aligned}$$

Laat ons nu de partiële afgeleide van de average log-likelihood bepalen. Merk op dat de vorm van de sommatie op de laatste regel, $\sum_{h[m]} \widetilde{P}(o[m], h[m] : \Theta)$, lijkt op de vorm van $Z(\Theta)$ uit definitie 20. Inderdaad, deze sommatie is niet de normalisatie-constante voor $P(\mathcal{X} : \Theta)$, maar wel voor de conditionele kansdistributie $P(\mathcal{H}[m] : o[m], \Theta)$. Bijgevolg verkrijgen we analoog aan vergelijking 22:

$$\frac{\partial}{\partial \theta_i} \log\left(\sum_{h[m]} \widetilde{P}(o[m], h[m] : \Theta)\right) = E_{P(\mathcal{H}[m]:o[m],\Theta)}[f_i(\mathcal{S}_i)]$$

Ten slotte bekomen we de partiële afgeleide van de average log-likelihood:

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \cdot \mathcal{L}(\Theta : \mathcal{D}) = \frac{1}{M} \cdot \sum_{m=1}^{M} E_{P(\mathcal{H}[m]:o[m],\Theta)}[f_i(\mathcal{S}_i)] - E_\Theta[f_i(\mathcal{S}_i)] \qquad (24)$$

De verwachte waardes in de eerste term bepalen we opnieuw m.b.v. approximate inference. Merk op dat we dit voor ieder sample apart moeten doen, wat computationeel enorm zwaar is. Dit maakt learning in geval van incomplete data dan een stuk zwaarder dan wanneer we complete data gebruiken (zie vergelijking 23).

Net als bij Bayesian Networks is de (average) log-likelihood multimodaal indien we met incomplete data werken. Opnieuw pakken we dit probleem aan door 'random restarts' te doen.

3 Markov Logic Networks

Markov Logic Networks vormen de tweede hoeksteen van DeepDive. In 2006 kwamen Matthew Richardson en Pedro Domingos [18] hier als eersten mee op de proppen. Een Markov Logic Network (MLN) combineert eerste-orde predicatenlogica met Markov Networks. DeepDive maakt gebruik van Markov Logic Networks om op gebruiksvriendelijke wijze een Markov Network te definiëren.

3.1 Eerste-orde predicatenlogica

Een eerste-orde predicatenlogica [19] is een logica waarin formules worden opgebouwd a.d.h.v. constanten, variabelen, functies, predicaten, booleaanse operatoren en kwantoren. De formules worden steeds tezamen met een vast, eindig domein van objecten gedefinieerd. Op die manier beschrijven de formules een formeel model waarin geredeneerd kan worden over de objecten in het domein.

Beschouw, bij wijze van voorbeeld, een domein bestaande uit twee personen, Anna en Bart genaamd. In principe zijn er meerdere personen op de wereld die Anna of Bart heten, maar we gaan nu uit van twee concrete/unieke personen. Stel dat we (op formele wijze) willen redeneren over de relaties tussen vriendschap, rookgedrag en kanker.

Meer formeel is een constante een symbool dat verwijst naar één specifiek object uit het domein, b.v. *Anna* verwijst naar de persoon genaamd Anna (het eerste element uit ons domein). Een variabele is een symbool dat naar meerdere objecten uit het domein kan verwijzen, b.v. x kan naar Anna of Bart verwijzen. Een functie krijgt een tupel objecten als input en mapt deze naar een object, b.v. $MoederVan(x)$. Een 'term' [19] is een logische expressie die naar een object verwijst. Een constante, een variabele, en een functie toegepast op een tupel termen, zijn alle drie termen.

Het basiselement voor een formule in eerste-orde predicatenlogica is een predicaat. Een predicaat is een symbool dat een eigenschap van een object beschrijft, of een relatie tussen meerdere objecten beschrijft. In de context van ons voorbeeldje hebben we $ZijnVrienden(x, y)$, $IsRoker(x)$ en $HeeftKanker(x)$. We korten deze voor de eenvoud af tot respectievelijk $Vr(x, y)$, $Ro(x)$ en $Ka(x)$. Meer formeel krijgt een predicaat als input een tupel termen. Een predicaat is steeds een bewering (over dat tupel termen), en kan dus waar of onwaar zijn.

De syntax van een formule is recursief gedefinieerd a.d.h.v. predicaten. Een predicaat toegepast op een tupel termen is een formule; dit is het basisgeval. De recursieve gevallen zijn de volgende. Ten eerste kunnen de booleaanse operatoren gebruikt worden om een nieuwe formule mee op te bouwen: de negatie ($\neg$) van een formule, de conjunctie ($\wedge$) van twee formules, de disjunctie ($\vee$) van twee formules, de implicatie ($\Rightarrow$) van twee formules en de equivalentie ($\Leftrightarrow$) van twee formules vormen allemaal een nieuwe formule. Ten tweede kunnen de kwantoren gebruikt worden: de existentiële kwantificatie ($\exists$) en universele kwantificatie ($\forall$) van een variabele toegepast op een formule, vormen allebei een nieuwe formule. Figuur 15 toont de definitie van een formule in eerste-order predicatenlogica a.d.h.v. een contextvrije grammatica. Een voorbeeldje van een formule: $\forall x Ro(x) \Rightarrow Ka(x)$, waarmee we zeggen "iedereen die rookt, heeft kanker".

Een formeel model zal typisch uit meerdere formules bestaan. Het formeel model wordt dan impliciet gedefinieerd als de conjunctie van de formu-

Formule	→	AtomischeFormule | ComplexeFormule
AtomischeFormule	→	Predicaat | Predicaat(Term,…)
ComplexeFormule	→	(Formule) | ¬Formule
	|	Formule ∧ Formule
	|	Formule ∨ Formule
	|	Formule ⇒ Formule
	|	Formule ⇔ Formule
	|	Kwantor Variabele,… Formule
Term	→	Constante
	|	Variabele
	|	Functie(Term,…)
Kwantor	→	∀ | ∃
Constante	→	Voorbeelden: *Anna*, *Bart*
Variabele	→	Voorbeeld: x
Functie	→	Voorbeeld: *MoederVan*
Predicaat	→	Voorbeelden: *Vr*, *Ro*, *Ka*

Figuur 15: Contextvrije grammatica voor een formule in eerste-order predicatenlogica, gebaseerd op Russell en Norvig [19].

les, zodat één grote formule overblijft. Een 'geground predicaat' is een predicaat waarvan de input vrij is van variabelen. De parameters bestaan dus louter uit constanten of functies (recursief) toegepast op constanten, zoals b.v. $Vr(Anna, MoederVan(Bart))$. Herinner dat een geground predicaat waar of onwaar is. Een 'mogelijke wereld' kent een truth-value toe aan ieder geground predicaat. Voor een gegeven mogelijke wereld kunnen we dus nagaan of de formules voldaan zijn, dat is, of ze allemaal naar 'waar' evalueren.

3.2 Definitie van een Markov Logic Network

Herinner dat een eerste-orde predicatenlogica tezamen met een vast, eindig domein van objecten gedefinieerd wordt. Een Markov Logic Network veronderstelt bovendien dat voor elk object in het domein een constante bestaat. Voor de eenvoud kunnen we het domein dus beschouwen als een verzameling constanten. Bijgevolg kunnen we, gegeven een verzameling constanten, alle mogelijke groundings van een predicaat genereren door iedere variabele in het predicaat te vervangen door alle constanten. Stel dat $\{Anna, Bart\}$ de verzameling constanten is en $Ro(x)$ het beschouwde predicaat. De twee mogelijke groundings zijn dan: $Ro(Anna)$ en $Ro(Bart)$. Ook van een formule kunnen we alle mogelijke groundings genereren. Daartoe vervangen we ieder predicaat in de formule door alle gegronde predicaten, er rekening mee houdend dat gelijknamige variabelen naar eenzelfde constante verwijzen. Voor de formule $\forall x Ro(x) \Rightarrow Ka(x)$ bekomen we dan: $Ro(Anna) \Rightarrow Ka(Anna)$ en $Ro(Bart) \Rightarrow Ka(Bart)$. Een ongeldig voorbeeld is $Ro(Anna) \Rightarrow Ka(Bart)$, omdat de waarde van x in het eerste predicaat niet overeenkomt met de waarde van x in het tweede predicaat.

Een Markov Logic Network L is gedefinieerd als een verzameling paren (F_i, w_i), waarbij F_i een formule in eerste-orde predicatenlogica is en w_i het geassocieerde gewicht (een reëel getal). Tezamen met een verzameling constanten C definieert L een Markov Network $M_{L,C}$, geparametriseerd als Log-Linear Model, als volgt [18]:

1. $M_{L,C}$ bevat een binaire random variable voor iedere grounding van ieder predicaat in L.
2. $M_{L,C}$ bevat een feature voor iedere mogelijke grounding van iedere formule F_i. De parameters van een feature zijn de gegrounde predicaten die in de beschouwde gegrounde formule voorkomen. De output van een feature is de truth-value van de beschouwde gegrounde formule (i.e. 1 als de formule waar is, 0 als ze onwaar is), waarbij de toekenning van een truth-value aan iedere parameter wordt gegeven door de beschouwde wereld. Het gewicht van een feature tenslotte, is de w_i die aan de formule geassocieerd is in de definitie.

Om de graafstructuur voor dit Markov Network te bepalen, gebruiken we vergelijking 6 uit paragraaf 2.3.5. Zo bekomen we namelijk de parametrisatie als product van factoren, waaruit we de Factor Graph of de ongerichte graaf opstellen. Stel dat we de Factor Graph willen opstellen. Het komt er dan op neer dat we een variabele-knoop toevoegen voor iedere grounding van ieder predicaat, en een factor-knoop toevoegen voor iedere mogelijke grounding van iedere formule F_i. Het gewicht dat we aan deze factor-knoop hangen is w_i. De factor-knoop is verbonden met alle gegrounde predicaten die in de beschouwde gegrounde formule voorkomen. Figuur 16 geeft een voorbeeld van een MLN en de bijhorende factor graph wanneer $\{Anna, Bart\}$ de verzameling constanten is. De eerste formule hebben we reeds besproken, en de tweede betekent dat als twee personen bevriend zijn hun rookgedrag overeenkomt (ze roken beiden wel of beiden niet).

3.3 Alternatieve voorstellingen

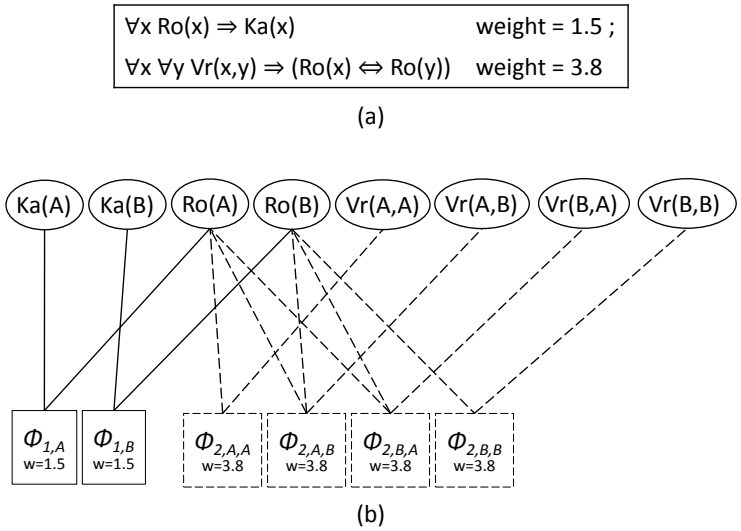
Wanneer we de definitie van een Markov Logic Network (als Markov Network) uit deel 3.2 rechtstreeks volgen, wordt de kans op een bepaalde wereld x beschreven volgens vergelijking 5 uit paragraaf 2.3.5. Gemakkelijkheidshalve herhalen we de voorstelling van de niet-genormaliseerde kans:

$$\widetilde{P}(x) = \exp\left(\sum_{i=1}^{m} w_i f_i(x)\right)$$

met m het aantal features. In dit deel beschouwen we twee alternatieve voorstellingen voor de kans op een wereld.

3.3.1 Gangbare voorstelling

Een veelvuldig gebruikte alternatieve voorstelling bekomen we als volgt. Beschouw even de features geassocieerd aan eenzelfde formule F_i; er is één feature per mogelijke grounding van de formule en al deze features hebben hetzelfde gewicht w_i. Telkens een grounding van de formule waar is, telt de sommatie er w_i bij. Bijgevolg kunnen we dit deel van de sommatie afkorten tot $w_i \cdot n_i(x)$,



Figuur 16: (a) Voorbeeld van een MLN. (b) Bijhorende Factor Graph. "A" en "B" stellen respectievelijk "Anna" en "Bart" voor. De stippellijnen dienen louter ter verduidelijking.

met $n_i(x)$ het aantal groundings dat waar is. De kans op een bepaalde wereld x kunnen we nu eenvoudiger schrijven als:

$$P(\mathcal{X}{=}x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{m} w_i n_i(x)\right) \tag{25}$$

met m het aantal formules en $\mathcal{X}$ de verzameling binaire variabelen (gegrounde predicaten dus). Aangezien deze voorstelling wat handiger is, zullen we steeds deze hanteren.

3.3.2 Kostgebaseerde voorstelling

Uit de voorgaande voorstellingen lijkt de kans op een mogelijke wereld x alleen gebaseerd te zijn op de groundings die waar worden gemaakt; het lijkt alsof er geen rekening wordt gehouden met groundings die onwaar zijn. In de volgende kostgebaseerde voorstelling is dat niet het geval [20].

De kost van een bepaalde wereld beschrijft – informeel gesproken – hoe "slecht" deze wereld matcht met de formules. Merk op dat de ideale wereld alle groundings van formules met een positief gewicht waar maakt en alle groundings van formules met een negatief gewicht onwaar maakt. Bijgevolg hangt de kost van een wereld af van de groundings die hier tegenin gaan, die als het ware een overtreding zijn: de groundings van formules met een positief gewicht die onwaar zijn, en de groundings van formules met een negatief gewicht die waar zijn. Meer formeel wordt de kost van een wereld x gedefinieerd als:

$$\text{kost}(x) = \sum_{f \in F^+} w_f \cdot n_f^O(x) + \sum_{f \in F^-} -w_f \cdot n_f^W(x) \tag{26}$$

met F^+ (resp. F^-) de verzameling formules met een positief (resp. negatief) gewicht, en $n_f^O(x)$ (resp. $n_f^W(x)$) het aantal groundings van formule f dat onwaar (resp. waar) is in x. Merk op dat voor iedere wereld x geldt dat $\text{kost}(x) \geq 0$ en de kost voor de ideale wereld steeds 0 bedraagt.

De relatie tussen de kost van een wereld x en zijn kans wordt gegeven door:

$$P(\mathcal{X}{=}x) = \frac{1}{Z} \exp\left(-\text{kost}(x)\right) \tag{27}$$

Hoewel deze voorstelling op het eerste zicht sterk verschilt van de gangbare voorstelling (zie vgl. 25), definiëren beide precies dezelfde kansdistributie. Hieronder volgt het bewijs.

We gebruiken de notaties $P_{dd}(x)$ en $P_{kost}(x)$ om te verwijzen naar, respectievelijk, de gangbare en kostgebaseerde voorstelling van de kans op wereld x. Analoog voor de niet-genormaliseerde kansen, $\widetilde{P}_{dd}(x)$ en $\widetilde{P}_{kost}(x)$, en voor de normalisatie-constantes, Z_{dd} en Z_{kost}. We moeten dus aantonen dat $P_{dd}(x) = P_{kost}(x)$. We noteren het totale aantal groundings van een formule f als: $n_f^T = n_f^W(x) + n_f^O(x)$. Merk op dat dit aantal onafhankelijk is van de beschouwde wereld.

Als een eerste tussenstap, tonen we aan dat:

$$\begin{aligned}
\widetilde{P}_{kost}(x) &= \exp\left[-\left(\sum_{f\in F^+} w_f \cdot n_f^O(x) + \sum_{f\in F^-} -w_f \cdot n_f^W(x)\right)\right] \\
&= \exp\left[-\sum_{f\in F^+} w_f \cdot (n_f^T - n_f^W(x)) - \sum_{f\in F^-} w_f \cdot n_f^W(x)\right] \\
&= \exp\left[-\sum_{f\in F^+} w_f \cdot n_f^T + \sum_{f\in F^+} w_f \cdot n_f^W(x) + \sum_{f\in F^-} w_f \cdot n_f^W(x)\right] \\
&= \exp\left[-\sum_{f\in F^+} w_f \cdot n_f^T + \sum_{f\in F} w_f \cdot n_f^W(x)\right]
\end{aligned}$$

met F de verzameling van alle formules. Om tot de laatste gelijkheid te komen, merken we op dat $F = F^+ \cup F^-$ waarbij $F^+ \cap F^- = \emptyset$.

Als een tweede tussenstap, tonen we aan dat:

$$\begin{aligned}
\log(Z_{kost}) &= \log\left[\sum_{x'} \exp\left(\sum_{f\in F} w_f \cdot n_f^W(x') - \sum_{f\in F^+} w_f \cdot n_f^T\right)\right] \\
&= \log\left[\sum_{x'}\left[\exp\left(\sum_{f\in F} w_f \cdot n_f^W(x')\right) \Big/ \exp\left(\sum_{f\in F^+} w_f \cdot n_f^T\right)\right]\right] \\
&= \log\left[\frac{1}{\exp\left(\sum_{f\in F^+} w_f \cdot n_f^T\right)} \cdot \sum_{x'} \exp\left(\sum_{f\in F} w_f \cdot n_f^W(x')\right)\right]
\end{aligned}$$

$$= - \sum_{f \in F^+} w_f \cdot n_f^T + \log \left[\sum_{x'} \exp \left(\sum_{f \in F} w_f \cdot n_f^W(x') \right) \right]$$

In de tweede gelijkheid merken we op dat de noemer geen gebruik maakt van x' en bijgevolg niet deelneemt aan de sommatie; we brengen deze constante buiten de sommatie.

Als derde tussenstap, schrijven we Z_{dd} volgens onze nieuwe notatie voor aantallen van groundings:

$$Z_{dd} = \sum_{x'} \exp \left(\sum_{f \in F} w_f \cdot n_f^W(x') \right)$$

Het aantonen van de eigenlijke gelijkheid is nu eenvoudig:

$$\begin{aligned}
&P_{dd}(x) = P_{kost}(x) \iff \frac{1}{Z_{dd}} \cdot \widetilde{P}_{dd}(x) = \frac{1}{Z_{kost}} \cdot \widetilde{P}_{kost}(x) \overset{\text{(log nemen)}}{\iff} \\
&\log \left(\widetilde{P}_{dd}(x) \right) - \log \left(Z_{dd} \right) = \log \left(\widetilde{P}_{kost}(x) \right) - \log \left(Z_{kost} \right) \overset{\text{(tussenstap 1)}}{\iff} \\
&\sum_{f \in F} w_f \cdot n_f^W(x) - \log \left(Z_{dd} \right) = - \sum_{f \in F^+} w_f \cdot n_f^T + \sum_{f \in F} w_f \cdot n_f^W(x) - \log \left(Z_{kost} \right) \\
&\iff \log \left(Z_{dd} \right) = \sum_{f \in F^+} w_f \cdot n_f^T + \log \left(Z_{kost} \right)
\end{aligned}$$

De laatste gelijkheid klopt inderdaad. Dit ziet men meteen in door de resultaten uit de tweede en derde tussenstap te gebruiken.

3.4 Motivatie achter MLNs

De motivatie achter Markov Logic Networks kunnen we vanuit twee perspectieven bekijken.

Ten eerste kunnen we MLNs zien als een uitbreiding van de eerste-orde predicatenlogica door het toelaten van onzekerheden. In de klassieke eerste-orde predicatenlogica is een mogelijke wereld pas waar wanneer álle formules waar zijn. Het is in de praktijk echter onmogelijk om een model op te stellen met formules die werkelijk altijd waar zijn. Het is bijvoorbeeld zeker niet altijd waar dat als je rookt je ook kanker hebt, maar we weten dat er wel degelijk een verband is. Een MLN laat ons toe deze kennis in het model op te nemen, en uit te drukken hoeveel vertrouwen we hebben in deze formule (d.m.v. een gewicht). De klassieke eerste-orde predicatenlogica maakt geen onderscheid tussen een wereld waarin alle formules onwaar zijn en een wereld waarin slechts één formule onwaar is. Zoals vergelijking 25 laat zien, houdt een MLN hier wel rekening mee. Een MLN zegt ons namelijk hoe waarschijnlijk een bepaalde wereld x is. Wanneer alle gegronde formules waar zijn, is $P(x)$ maximaal; wanneer geen enkele gegronde formule voldaan is $P(x)$ minimaal. Voor situaties die hier tussenin liggen, hangt de kans af van het aantal gegronde formules dat waar is en van de gewichten die aan de formules geassocieerd zijn. Hoe groter het gewicht van een formule, hoe groter de impact op de kans. Merk op dat wanneer een gewicht $+\infty$ is, we een formule uit de klassieke eerste-orde predicatenlogica

krijgen. Deze formule moet steeds waar zijn; zo niet, is $P(x) = 0$. Analoog beschrijft een negatief gewicht een formule waaraan de wereld liefst niet voldoet; als de formule waar is, vermindert de kans op deze wereld.

Ten tweede kunnen we een MLN zien als de compacte en gebruiksvriendelijke representatie van een groot Markov Network. Door vanuit de eerste-orde predicatenlogica te werken, kunnen we op een abstractere manier (hoger conceptueel niveau) redeneren over het probleem waar we een Markov Network voor willen opstellen. Een MLN is een template voor een Markov Network, omdat het precieze Markov Network afhangt van de verzameling constanten die gebruikt wordt.

3.5 Grounding in SQL

Voor het omvormen van een MLN naar het overeenkomstig Markov Network, moeten we alle gegronde formules genereren. Dit proces noemt men grounding. We hebben in voorgaande paragrafen reeds uitgelegd hoe we dit in het algemeen kunnen doen. DeepDive gebruikt SQL voor alles wat met data te maken heeft. Daarom bekijken we in dit deel hoe we grounding kunnen implementeren in SQL. Belangrijk is dat dit zeer efficiënt moet gebeuren, omdat grounding een aanzienlijke computationele kost met zich meebrengt voor grote MLNs.

3.5.1 Clausal form

De SQL-implementatie van grounding gaat ervan uit dat de formules in 'clausal form' staan, ook wel 'conjunctive normal form' genoemd [19]. Dat betekent dat iedere formule een conjunctie van 'clauses' is, waarbij een 'clause' een disjunctie van 'literals' is. Een 'literal' is simpelweg een predicaat of de negatie ervan. Een voorbeeldje maakt het snel duidelijk:

$$(Ro(x) \vee Ro(y)) \ \wedge \ (Vr(x,y) \vee Vr(y,x) \vee \neg Ka(y)) \ \wedge \ (\neg Ka(x))$$

Een formule in eerste-orde predicatenlogica kan steeds omgevormd worden tot een formule in clausal form. De formule in eerste-orde predicatenlogica is waar a.s.a. de formule in clausal form waar is, wat men 'equisatisfiability' noemt. De omvorming verloopt als volgt, met F_1, F_2 en F_3 formules in eerste-orde predicatenlogica [19]:

1. Elimineer equivalenties: $F_1 \Leftrightarrow F_2$ wordt $(F_1 \Rightarrow F_2) \wedge (F_2 \Rightarrow F_1)$.
2. Elimineer implicaties: $F_1 \Rightarrow F_2$ wordt $\neg F_1 \vee F_2$.
3. Schuif negaties naar binnen:
   (a) $\neg(\forall x \ F_1)$ wordt $\exists x \ \neg F_1$.
   (b) $\neg(\exists x \ F_1)$ wordt $\forall x \ \neg F_1$.
   (c) $\neg(F_1 \vee F_2)$ wordt $\neg F_1 \wedge \neg F_2$.
   (d) $\neg(F_1 \wedge F_2)$ wordt $\neg F_1 \vee \neg F_2$.
   (e) $\neg(\neg F_1)$ wordt F_1.
4. Geef iedere variabele geassocieerd aan een kwantor een unieke variabelenaam (wordt relevant in volgende stappen). Voorbeeld:
   $(\exists x \ Ro(x)) \ \wedge \ (\exists x \ Ka(x))$ wordt $(\exists x \ Ro(x)) \ \wedge \ (\exists y \ Ka(y))$.

∀x Ro(x) ⇒ Ka(x)	weight = 1.5 ;
∀x ∀y Vr(x,y) ⇒ (Ro(x) ⇔ Ro(y))	weight = 3.8

(a)

¬Ro(x) ∨ Ka(x)	weight = 1.5 ;
¬Vr(x,y) ∨ ¬Ro(x) ∨ Ro(y)	weight = 1.9 ;
¬Vr(x,y) ∨ Ro(x) ∨ ¬Ro(y)	weight = 1.9

(b)

Figuur 17: (a) Een MLN. (b) Overeenkomstig MLN in clausal form.

5. Skolemization: Verwijder iedere existentiële kwantor en vervang ieder voorkomen van de bijhorende variabele door een Skolem-functie. Een existentieel gekwantificeerde variabele komt steeds voor in dezelfde scope als een aantal universeel gekwantificeerde variabelen. De parameters van de Skolem-functie zijn die universeel gekwantificeerde variabelen[13]. Voorbeeld: $\forall x\ \exists y\ Vr(x, y)$ wordt $\forall x\ Vr(x, S(x))$, met $S(x)$ een Skolem-functie zo gedefinieerd dat beide formules equisatisfiable zijn. Gegeven een mogelijke wereld geldt dan concreet: als er inderdaad een invulling van y bestaat waarvoor $Vr(x, y)$ waar is, geeft $S(x)$ zo'n invulling terug; als er zo geen invulling bestaat, geeft $S(x)$ eender welke invulling van y terug.
6. Verwijder alle universele kwantoren. Voorbeeld: $\forall x\ Ro(x)$ wordt $Ro(x)$.
7. Gebruik de distributie-regel om van $F_1 \vee (F_2 \wedge F_3)$ naar $(F_1 \vee F_2) \wedge (F_1 \vee F_3)$ te gaan.

Beschouw een MLN $M_{L,C}$ waarin alle formules in eerste-orde predicatenlogica zijn uitgedrukt. Beschouw het MLN $M_{L',C}$ waarin diezelfde formules in clausal form zijn uitgedrukt. Voor een concrete wereld, moeten we aan $M_{L',C}$ de definities van de Skolem-functies toevoegen vooraleer we een formule kunnen evalueren, terwijl dat bij $M_{L,C}$ niet moet. Daarom zeggen we dat beide MLNs niet equivalent zijn, maar wel equisatisfiable. Het probleem echter met deze aanpak is dat het MLN een oneindig groot domein krijgt. Aangezien een Skolem-functie $S(x)$ een functie is, moeten ook $S(S(x))$, $S(S(S(x)))$, $S(S(S(S(x))))$, ... gedefinieerd zijn. Daarom gaat men er in de praktijk meestal van uit dat de formules vrij zijn van existentiële kwantoren.

Beschouw het MLN in figuur 17(a), bestaande uit twee formules. We passen voorgaand algoritme toe op beide formules. Doordat de tweede formule een equivalentie bevat, is zijn clausal form de conjunctie van twee gelijkaardige formules: $(\neg Vr(x, y) \vee \neg Ro(x) \vee Ro(y)) \wedge (\neg Vr(x, y) \vee Ro(x) \vee \neg Ro(y))$. Herinner dat een MLN eigenlijk gedefinieerd is als de conjunctie van al zijn formules. Bijgevolg kunnen we de clausal form van de tweede formule zien als twee deelformules, zoals figuur 17(b) toont. Het oorspronkelijke gewicht van de tweede formule, 3.8, hebben we verdeeld hebben over beide formules, 1.9 ieder. Dit is logisch: wanneer beide deelformules waar zijn, was de oorspronkelijke formule waar, en is het gewicht terecht 3.8; wanneer slechts één van beide deelformules waar is, houdt het steek om de helft van het oorspronkelijke gewicht toe te kennen.

[13]Wanneer de scope uit 0 parameters bestaat, spreekt men van een Skolem-constante.

Vr(x,y) ∨ Ro(x) ∨ ¬Ro(y)	weight = 2.6 ;
¬Ro(x) ∨ Ka(x)	weight = 1.7

(a)

Vr(Anna,Anna) ∨ Ro(Anna) ∨ ¬Ro(Anna)	weight = 2.6 ;
Vr(Anna, Bart) ∨ Ro(Anna) ∨ ¬Ro(Bart)	weight = 2.6 ;
Vr(Bart,Anna) ∨ Ro(Bart) ∨ ¬Ro(Anna)	weight = 2.6 ;
Vr(Bart, Bart) ∨ Ro(Bart) ∨ ¬Ro(Bart)	weight = 2.6 ;
¬Ro(Anna) ∨ Ka(Anna)	weight = 1.7 ;
¬Ro(Bart) ∨ Ka(Bart)	weight = 1.7

(b)

Figuur 18: (a) Een MLN. (b) Overeenkomstig geground MLN.

3.5.2 Geground MLN

Beschouw het MLN in figuur 18(a), bestaande uit twee formules (in clausal form). Als we opnieuw $\{Anna, Bart\}$ als de verzameling constanten beschouwen, kunnen we dit MLN ook schrijven als 'geground MLN', zoals figuur 18(b) toont. Dit gegrounde MLN maakt simpelweg het resultaat van de grounding expliciet; het is niks nieuws. Voor alle duidelijkheid bekomen we het gegrounde MLN door een formule toe te voegen voor iedere mogelijke grounding van iedere formule in het oorspronkelijke MLN. Het spreekt voor zich dat het oorspronkelijke MLN en het gegrounde MLN equivalent zijn in die zin dat ze exact hetzelfde Markov Network voorstellen.

Door over grounding te denken als het opstellen van het geground MLN, zullen we de implementatie van grounding in SQL makkelijker kunnen begrijpen. Merk tevens op dat het veel eenvoudiger is om vanuit het geground MLN tot de bijhorende Factor Graph te komen.

3.5.3 SQL algoritme

In deze paragraaf bespreken we het eigenlijke algoritme voor grounding in SQL. Herinner dat we grounding kunnen zien als het opstellen van een geground MLN. Het te grounden MLN bestaat uit een verzameling formules van de vorm $F = \vee_{i=1}^{n} l_i$, met n het aantal literals in de formule en l_i de i-de literal. Herinner dat een literal een predicaat is of de negatie van een predicaat. Dit noemen we respectievelijk een 'positieve' en een 'negatieve' literal.

Tot nog toe gingen we ervan uit dat we, gegeven een verzameling constanten, de groundings van een predicaat konden bekomen door iedere variabele in het predicaat te vervangen door alle constanten. Met $\{Anna, Bart\}$ als verzameling constanten, had het predicaat $Vr(x, y)$ dus vier mogelijke groundings. Echter, vanaf nu stellen we dat de groundings per predicaat gegeven moeten zijn. Zo zouden we bijvoorbeeld kunnen zeggen dat alleen $Vr(Anna, Bart)$ en $Vr(Bart, Anna)$ de groundings zijn; of dat dit zelfs de enige groundings zijn wanneer we $\{Anna, Bart, Charel\}$ als verzameling constanten gebruiken. Beschouw ter motivatie het predicaat $Geboortejaar(p, j)$, met p de naam van een persoon en j zijn/haar geboortejaar. De verzameling constanten bestaat dus uit een mix van persoonsnamen en jaartallen. Als we de groundings zouden ge-

nereren zoals voorheen, bekomen we ook groundings waarbij p door een jaartal werd vervangen en/of j door een persoonsnaam. Dit is uiteraard ongewenst en wordt vermeden door de nieuwe aanpak, i.e. door de mogelijke groundings van een predicaat manueel op te lijsten.

Het algoritme zal een PostgreSQL-query opstellen waarmee de groundings van een formule gegenereerd kunnen worden. De implementatie van het algoritme, i.e. de procedure voor het opstellen van Q, kan in eender welke programmeertaal die communicatie met een PostgreSQL database ondersteunt. Meer bepaald bestaat de input van het algoritme ten eerste uit de te grounden formule $F = \vee_{i=1}^{n} l_i$, zie figuur 19(a) als voorbeeld. Het precieze datatype dat gebruikt wordt om de formule voor te stellen is niet van belang; de implementatie van het algoritme moet gewoon kunnen opvragen uit welke literals de formule bestaat, of deze positief of negatief zijn en wat het gewicht van de formule is. Ten tweede bevat de input van het algoritme een database die per predicaat in het MLN een SQL-tabel T_i bevat, met daarin alle mogelijke groundings van dat predicaat. Zie figuur 19(b) als voorbeeld.

Het eigenlijke algoritme beschrijft de constructie van een PostgreSQL-query Q, die alle groundings van de formule F genereert. Figuur 19(c) toont alvast de opgestelde query Q; we bespreken het algoritme in de volgende alinea. Om uiteindelijk tot de groundings van de formule te komen, passen we de query Q toe op de database. Figuur 19(d) toont de output hiervan. Het spreekt voor zich hoe deze resultaten herleid kunnen worden tot de eigenlijke gegrounde formules, gegeven de oorspronkelijke formule en de output van de query Q. Figuur 19(e) toont de groundings van de formule.

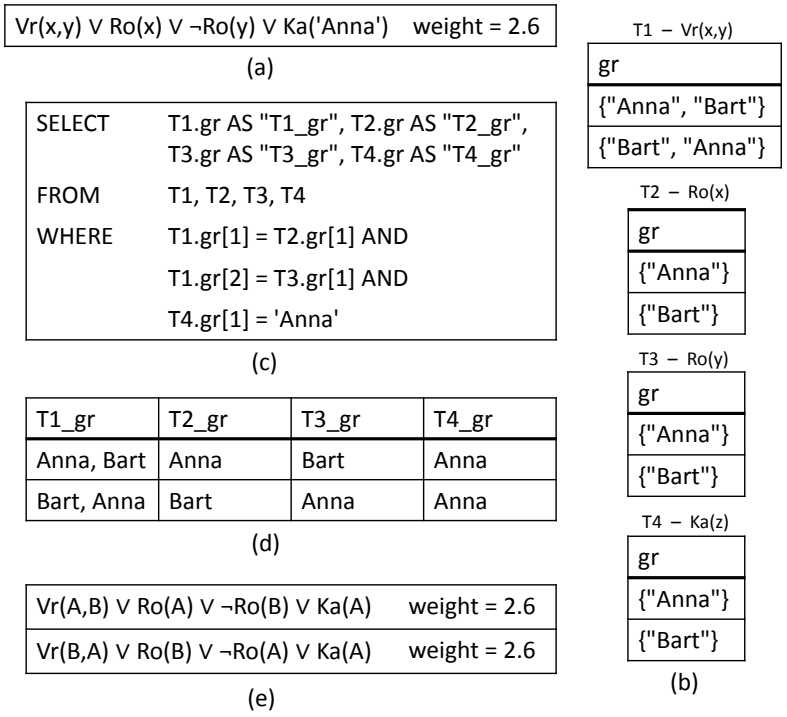
De PostgreSQL-query Q joint de tabellen overeenkomstig met de predicaten in de formule met elkaar, rekening houdend met eventuele constanten en er rekening mee houdend dat gelijknamige variabelen naar eenzelfde constante verwijzen. Het algoritme stelt de query Q op als volgt [20]:

1. De FROM clause van Q bevat 'T_i' voor iedere literal l_i.
2. De SELECT clause van Q bevat '$T_i.gr$' voor iedere literal l_i. Tot hier toe hebben we een gewone cross join opgesteld.
3. Voor iedere variabele x in F doen we het volgende. Beschouw alle l_i's in F die x als parameter hebben. Voeg een WHERE conditie toe die de overeenkomstige parameter in de bijhorende T_i's aan elkaar gelijkstelt.
4. Voor iedere constante c in iedere literal l_i, voegen we een WHERE conditie toe die de overeenkomstige parameter in T_i gelijkstelt aan c.
5. De WHERE clause van Q is de conjunctie van de WHERE condities die in de twee voorgaande stappen opgesteld werden.

Figuur 19(c) toont de query Q die het algoritme opstelt. We benadrukken dat het genereren van de groundings efficiënt zal gebeuren, doordat joins in SQL zeer efficiënt gebeuren.

3.5.4 Optimalisatie

In deze paragraaf beschrijven we een optimalisatie die we kunnen inbouwen in het grounding algoritme uit deel 3.5.3. Deze optimalisatie zorgt dat er minder groundings worden gegenereerd, zonder de kans op een mogelijke wereld te wijzigen. Bijgevolg kost het berekenen van de kans op een mogelijke wereld minder tijd (zie vgl. 25). We benadrukken dat de optimalisatie niet het grounding algoritme zelf versnelt.



Figuur 19: (a) Een formule uit een MLN. (b) De database bestaande uit de tabellen met de groundings van ieder predicaat in het MLN. (c) De opgestelde PostgreSQL-query – weet dat tabellen en arrays in PostgreSQL op index 1 beginnen. (d) De output van de query toegepast op de database. (e) De gereconstrueerde groundings van de formule ("Anna" werd afgekort tot "A" en "Bart" tot "B").

1	Vr(A,B) ∨ Ro(A) ∨ ¬Ka(B)	weight = 1.5
2	Vr(A,C) ∨ Ro(A) ∨ ¬Ka(C)	weight = 1.5
3	Vr(B,A) ∨ Ro(B) ∨ ¬Ka(A)	weight = 1.5
4	Vr(B,C) ∨ Ro(B) ∨ ¬Ka(C)	weight = 1.5
5	Vr(C,A) ∨ Ro(C) ∨ ¬Ka(A)	weight = 1.5
6	Vr(C,B) ∨ Ro(C) ∨ ¬Ka(B)	weight = 1.5

Figuur 20: Het geground MLN.

Deze optimalisatie is gebaseerd op het feit dat er typisch evidence voorhanden is. Herinner de betekenis van 'evidence' (uit deel 2.4): evidence omvat variabelen waarvan we de waarde reeds kennen. In de context van MLNs en Markov Networks betekent dit dat we van sommige gegrounde predicaten (binaire variabelen) reeds hun waarde (i.e. truth-value) kennen. DeepDive bekomt zulke evidence via 'distant supervision', wat in deel 4.4 zal worden toegelicht.

Om in te zien hoe de evidence deze optimalisatie mogelijk maakt, vertrekken we vanuit een voorbeeldje. Beschouw een MLN met, gemakkelijkheidshalve, slechts één formule (in clausal form): $Vr(x, y) \vee Ro(x) \vee \neg Ka(y))$ met een gewicht van 1.5. Veronderstel dat figuur 20 het gegrounde MLN toont, verkregen op basis van het grounding algoritme in SQL (zie deel 3.5.3). Veronderstel tenslotte dat we, als deel van de evidence, weten dat $Ro(B)$ waar is. Hierdoor evalueren de derde en vierde grounding sowieso naar *waar*, ongeacht de beschouwde wereld; in iedere mogelijke wereld zullen deze 2 gegrounde formules waar zijn. Bijgevolg leveren deze groundings in iedere mogelijke wereld een even grote bijdrage tot de niet-genormaliseerde kans van die wereld, waardoor deze groundings geen impact hebben op de eigenlijke kansen. Het is dus zinloos om deze groundings in acht te nemen bij het berekenen van de kans op een wereld; we kunnen ze gewoon negeren. Meer nog, we hadden ze in de eerste plaats zelfs niet moeten genereren. Naast de intuïtieve verklaring voor het feit dat we die groundings mogen weglaten, geven we hier in deel 4.5.3 een formeel bewijs voor.

Het voorbeeld illustreert hoe men, door rekening te houden met de evidence, kan vermijden dat zulke "zinloze" groundings gegenereerd (en dus ook geëvalueerd) worden. De evidence is uiteraard deel van de input van het geoptimaliseerde algoritme. Meer bepaald voegen we een extra kolom *evidence* toe aan de tabellen die, per predicaat, alle mogelijke groundings van dat predicaat bijhouden; dit waren de tabellen die in figuur 19(b) getoond worden. Mogelijke waarden zijn 'true', 'false' of 'onbepaald', waarbij die laatste betekent dat er geen evidence voorhanden is voor dit gegrounde predicaat. De optimalisatie vertaalt zich naar een extra stap in het oorspronkelijke algoritme (uit deel 3.5.3), tussen de tweede en derde stap, namelijk:

* Voor iedere positieve (resp. negatieve) literal l_i, voegen we de WHERE conditie '$T_i.evidence <> true$' (resp. '$T_i.evidence <> false$') toe.

Dankzij deze optimalisatie worden groundings die sowieso naar *waar* zouden evalueren, niet meer gegenereerd. Hierdoor moeten minder groundings geëvalueerd worden bij het berekenen van de kans op een mogelijke wereld, wat uiteraard de rekentijd verlaagt.

3.6 Skolemization

Herinner dat we ons tot nog toe beperkten tot MLNs waarvan de formules vrij waren van existentiële kwantoren. In dit deel bekijken we een state-of-the-art aanpak voor Skolemization (i.e. het elimineren van existentiële kwantoren) [21, 22], en gaan we na of we er op die manier in slagen deze beperking op te heffen.

3.6.1 Motivatie

Herinner uit deel 3.5 dat, om grounding efficiënt te laten verlopen, de formules van een MLN in clausal form moeten staan. Eén van de stappen in het standaard algoritme voor het omvormen van een formule in eerste-orde predicatenlogica naar zijn clausal form, was Skolemization (stap 5). De aanpak tot Skolemization zoals daar werd toegelicht, noemen we 'textbook Skolemization'. Herinner echter het probleem met deze aanpak: het domein van het MLN wordt oneindig groot, waardoor deze aanpak in de praktijk niet geschikt is.

Een andere veelvuldig gebruikte aanpak is de volgende. We vervangen ieder predicaat waarin een existentieel gekwantificeerde variabele voorkomt door de disjunctie van alle mogelijke groundings van dat predicaat [18]. Een voorbeeldje, met $\{Anna, Bart\}$ als de verzameling constanten: $\forall x \, \exists y \; Vr(x, y)$ wordt $Vr(Anna, Anna) \vee Vr(Anna, Bart) \vee Vr(Bart, Anna) \vee Vr(Bart, Bart)$. Merk op dat deze aanpak, i.t.t. textbook Skolemization, een MLN produceert dat wel equivalent is met het oorspronkelijke. Echter, ook deze aanpak is in de praktijk niet geschikt. De vervanging door alle mogelijke groundings, zorgt er namelijk voor dat de formules enorm lang zullen worden, waardoor het MLN zijn compactheid verliest. Meer algemeen vervangt deze werkwijze een stuk eerste-orde predicatenlogica door een stuk propositielogica, wat uiteraard niet de bedoeling is van MLNs.

Beide technieken zijn dus niet geschikt in de praktijk. Idealiter bestaat er een Skolemization-techniek die het MLN aanpast (de existentiële kwantoren worden geëlimineerd), zonder hierbij functies te introduceren of omzettingen naar propositielogica te doen. In het verdere verloop van dit deel bespreken we een recent ontwikkelde aanpak voor Skolemization en gaan we na in welke mate deze geschikt is.

3.6.2 Weighted First-Order Model Counting

De state-of-the-art aanpak voor Skolemization die we zullen bespreken, is van toepassing op Weighted First-Order Model Counting (WFOMC). Om WFOMC toe te lichten, vertrekken we vanuit het 'Boolean satisfiability problem' voor eerste-orde predicatenlogica, typisch afgekort tot SAT. Gegeven een formule in eerste-orde predicatenlogica houdt het SAT probleem in dat men dient te bepalen of er een mogelijke wereld bestaat zodat de formule naar *waar* evalueert. Een veralgemening van het SAT probleem, is #SAT. Het #SAT probleem telt hoeveel mogelijke werelden er bestaan waarin de formule naar *waar* evalueert.

Het WFOMC probleem veralgemeent #SAT nog verder: iedere mogelijke wereld heeft een eigen gewicht. Het WFOMC probleem sommeert de gewichten van alle mogelijke werelden waarin de formule naar *waar* evalueert. Merk op dat, als iedere mogelijke wereld een gewicht van 1 heeft, WFOMC zich reduceert tot #SAT. Meer formeel is de WFOMC van een formule in eerste-orde

predicatenlogica Δ gedefinieerd als [21]:

$$\text{WFOMC}(\Delta, \mathcal{D}, w, \overline{w}) = \sum_{\omega \models_{\mathcal{D}} \Delta} \prod_{l \in \omega_0} \overline{w}(\text{pred}(l)) \prod_{l \in \omega_1} w(\text{pred}(l)), \tag{28}$$

met $\mathcal{D}$ de verzameling constanten (relevant voor de groundings), w en $\overline{w}$ functies die een gewicht toekennen aan ieder predicaat, en de functie $pred(l)$ die het predicaat geassocieerd aan literal l teruggeeft.

De sommatie gaat dus over alle mogelijke werelden ω die de formule Δ naar *waar* laten evalueren; zo'n ω noemt men ook wel een 'model' voor Δ. Wanneer we ω zien als een verzameling literals, bestaat deze uit twee disjuncte deelverzamelingen: ω_1 (resp. ω_0) bevat alle literals die naar waar (resp. onwaar) evalueren. Het gewicht van een mogelijke wereld is het product van de gewichten van de verzameling literals.

Merk op dat de WFOMC van een formule en de niet-genormaliseerde kans op een wereld enigszins op elkaar lijken: beide gebruiken formules in eerste-orde predicatenlogica, en beide gebruiken een notie van gewichten. Het is dan ook niet verwonderlijk dat beide technieken veelvuldig gebruikt worden in de context van probabilistische learning en inference [21]. Tot slot benadrukken we dat de WFOMC gedefinieerd is t.o.v. een formule, terwijl een niet-genormaliseerde kans betrekking heeft op een mogelijke wereld. Op het precieze verband tussen beide komen we in deel 3.6.4 terug.

3.6.3 Skolemization voor WFOMC

De Skolemization-procedure die we hier toelichten, krijgt als input een triple $(\Delta, w, \overline{w})$. De output is het triple $(\Delta', w', \overline{w}')$, dat dezelfde WFOMC heeft ($\text{WFOMC}(\Delta, \mathcal{D}, w, \overline{w}) = \text{WFOMC}(\Delta', \mathcal{D}', w', \overline{w}')$) en waarbij Δ' vrij is van existentiële kwantoren. Merk op dat de procedure losstaat van de gebruikte verzameling constanten $\mathcal{D}$. Het elimineren van één existentiële kwantor gebeurt als volgt [21].

Veronderstel dat Δ een deelexpressie van de vorm $\exists y, \phi(y, \mathbf{x})$ bevat, waarbij $\phi(y, \mathbf{x})$ het deel van de formule voorstelt waarin y en $\mathbf{x}$ als vrije variabelen voor komen. We noteren $\mathbf{x}$ vetgedrukt om aan te geven dat het een verzameling van variabelen voorstelt. Om de existentiële kwantor y uit Δ te verwijderen, voeren we ten eerste een Tseitin-predicaat $Z(\mathbf{x})$ en een Skolem-predicaat $S(\mathbf{x})$ in. Vervolgens vervangen we de deelexpressie $\exists y, \phi(y, \mathbf{x})$ in Δ door $Z(\mathbf{x})$. De uiteindelijke formule, Δ', bekomen we ten slotte door de conjunctie te nemen van de aangepaste Δ en:

$$\begin{aligned} \forall \mathbf{x}, \forall y,\ & Z(\mathbf{x}) \vee \neg\phi(y, \mathbf{x}) && \wedge \\ \forall \mathbf{x},\ & S(\mathbf{x}) \vee Z(\mathbf{x}) && \wedge \\ \forall \mathbf{x}, \forall y,\ & S(\mathbf{x}) \vee \neg\phi(y, \mathbf{x}) \end{aligned} \tag{29}$$

De gewicht-functies w' en $\overline{w}'$ zijn dezelfde als w en $\overline{w}$, maar worden nog uitgebreid met $w'(Z) = 1$, $\overline{w}'(Z) = 1$, $w'(S) = 1$ en $\overline{w}'(S) = -1$.

We illustreren de Skolemization-procedure a.d.h.v. een voorbeeldje [21]. Stel dat $\Delta = \forall x, \exists y, \mathit{WorksFor}(x, y) \vee \mathit{Boss}(x)$, wat zegt dat iedereen werk heeft (oftewel werk je voor iemand, ofwel ben je zelf de baas). We passen de Skolemization-

procedure hierop toe om y te elimineren, en bekomen:

$$\begin{aligned}
&\forall x,\ Z(x) && \wedge \\
\forall x,\forall y,\ &Z(x) \vee \neg\,[\mathit{WorksFor}(x,y) \vee \mathit{Boss}(x)] && \wedge \\
&\forall x,\ Z(x) \vee S(x) && \wedge \\
\forall x,\forall y,\ &S(x) \vee \neg\,[\mathit{WorksFor}(x,y) \vee \mathit{Boss}(x)]
\end{aligned}$$

De intuïtie achter deze Skolemization-procedure is de volgende. De extra formule die wordt toegevoegd (vgl. 29), zorgt impliciet voor het definiëren van volgende equivalentie:

$$\forall \mathbf{x}, Z(\mathbf{x}) \iff \exists y, \phi(y, \mathbf{x})$$

We zeggen "impliciet" aangezien we uiteraard geen existentiële kwantor kunnen gebruiken. Daarom wordt deze equivalentie afgezwakt, waardoor extra modellen geïntroduceerd worden in Δ' (die dus niet aanwezig waren in Δ). Maar voor ieder extra model met gewicht W, wordt er ook een extra model met gewicht $-W$ geïntroduceerd, waardoor de uiteindelijke WFOMC ongewijzigd blijft.

Om deze uitleg meer concreet te maken, zullen we voor het voorgaande voorbeeldje de WFOMC van de oorspronkelijke formule en die na Skolemization berekenen. Veronderstel dat $\mathcal{D} = \{\alpha, \beta\}$, en $\mathit{WorksFor}(x, y)$ en $\mathit{Boss}(x)$ eenvoudigheidshalve elk slechts één mogelijke grounding hebben, nl. $\mathit{WorksFor}(\alpha, \beta)$ en $\mathit{Boss}(\alpha)$. Figuur 21(a) toont dan de oorspronkelijke gegronde formule. Veronderstel ten slotte dat w en $\overline{w}$ telkens 1 teruggeven als gewicht. Figuur21(b) toont voor iedere mogelijke wereld zijn bijdrage tot de WFOMC; de totale WFOMC is dus 3. Laten we nu de formule na Skolemization beschouwen. Figuur 21(c) toont deze, en figuur 21(d) toont voor iedere mogelijke wereld zijn bijdrage tot de WFOMC. Per oorspronkelijke mogelijke wereld hebben we nu 4 mogelijke werelden, door het introduceren van Z en S. De som van deze 4 WFOMC-bijdrages komt steeds overeen met de WFOMC-bijdrage van de overeenkomstige mogelijke wereld in de oorspronkelijke formule. Beschouw i.h.b. de vierde (oorspronkelijke) wereld, wiens WFOMC-bijdrage 0 is: de formule na Skolemization introduceert hier twee modellen met tegengestelde gewichten, zodat de WFOMC-bijdrage opnieuw 0 is.

Om meerdere existentiële kwantoren uit een formule te elimineren, moeten we de Skolemization-procedure simpelweg herhalen voor iedere existentiële kwantor. De volgorde waarin de existentiële kwantoren doorlopen worden, is van belang: dit moet van binnen naar buiten toe gebeuren, m.a.w. van rechts naar links. Indien we van links naar rechts zouden werken, bevat de $\phi(y, \mathbf{x})$ in vgl. 29 namelijk zelf nog existentiële kwantoren, waardoor we er meer introduceren dan elimineren.

3.6.4 WFOMC-encoding van een MLN

We haalden reeds aan dat er een verband is tussen de WFOMC van een formule en de niet-genormaliseerde kans op een wereld. Nu zullen we hier verder op ingaan, om in te zien hoe de Skolemization-procedure voor WFOMC gebruikt kan worden in de context van MLNs. We beginnen met het definiëren van de WFOMC-encoding $(\Delta, w, \overline{w})$ van een MLN [21]. Voor iedere formule $\phi_i(\mathbf{x_i})$ met gewicht w_i in het MLN, voegen we aan Δ de formule $\forall \mathbf{x_i}, P_i(\mathbf{x_i}) \iff \phi_i(\mathbf{x_i})$ toe. De gewicht-functies w en $\overline{w}$ kennen aan ieder oorspronkelijk predicaat (in

WorksFor(α,β) ∨ Boss(α)

(a)

wereld	truth-values WorksFor(α,β)	Boss(α)	WFOMC-bijdrage
1	1	1	1*1 = 1
2	1	0	1*1 = 1
3	0	1	1*1 = 1
4	0	0	n.v.t. (0)

WFOMC = 3

(b)

Z(α) ∧
Z(α) ∨ ¬[WorksFor(α,β) ∨ Boss(α)] ∧
Z(α) ∨ S(α) ∧
S(α) ∨ ¬[WorksFor(α,β) ∨ Boss(α)]

(c)

wereld	truth-values WorksFor(α,β)	Boss(α)	Z(α)	S(α)	WFOMC-bijdrage
1a	1	1	1	1	1*1*1*1 = 1
1b	1	1	1	0	n.v.t. (0)
1c	1	1	0	1	n.v.t. (0)
1d	1	1	0	0	n.v.t. (0)
2a	1	0	1	1	1*1*1*1 = 1
2b	1	0	1	0	n.v.t. (0)
2c	1	0	0	1	n.v.t. (0)
2d	1	0	0	0	n.v.t. (0)
3a	0	1	1	1	1*1*1*1 = 1
3b	0	1	1	0	n.v.t. (0)
3c	0	1	0	1	n.v.t. (0)
3d	0	1	0	0	n.v.t. (0)
4a	0	0	1	1	1*1*1*1 = 1
4b	0	0	1	0	1*1*1*(-1) = -1
4c	0	0	0	1	n.v.t. (0)
4d	0	0	0	0	n.v.t. (0)

WFOMC = 3

(d)

Figuur 21: (a) De oorspronkelijke gegronde formule. (b) De WFOMC-bijdrage van alle mogelijke werelden voor de oorspronkelijke formule. (c) De gegronde formule na Skolemization. (d) De WFOMC-bijdrage van alle mogelijke werelden voor de formule na Skolemization.

$\phi_i(\mathbf{x_i})$ dus) een gewicht van 1 toe, en $w(P_i) = e^{w_i}$ en $\overline{w}(P_i) = 1$.

Het formele verband tussen inference in een MLN en WFOMC toegepast op de WFOMC-encoding $(\Delta, w, \overline{w})$ van datzelfde MLN, is:

$$P_{\mathcal{D}}(\phi) = \frac{\text{WFOMC}(\Delta \wedge \phi, \mathcal{D}, w, \overline{w})}{\text{WFOMC}(\Delta, \mathcal{D}, w, \overline{w})}, \tag{30}$$

met $\mathcal{D}$ de verzameling constanten en ϕ een conjunctie van literals. We verklaren de rol van ϕ nader a.d.h.v. een voorbeeld. Beschouw een MLN dat uit slechts één formule bestaat: $Fr(x,y) \wedge Sm(x) \Rightarrow Sm(y)$, met een gewicht van 2. Veronderstel dat $\mathcal{D} = \{\alpha, \beta\}$ en ieder predicaat, eenvoudigheidshalve, slechts één mogelijke grounding heeft, nl. $Fr(\alpha,\beta)$, $Sm(\alpha)$ en $Sm(\beta)$ respectievelijk. De WFOMC-encoding Δ van dit MLN is: $P(x,y) \iff Fr(x,y) \wedge Sm(x) \Rightarrow Sm(y)$, waarbij $w(P) = e^2$ en alle andere gewichten 1 zijn. Indien we bv. $P(Fr(\alpha,\beta)$, willen berekenen, een marginale kans dus, is $\phi = Fr(\alpha,\beta))$. Indien we de kans op een bepaalde wereld willen berekenen, is bv. $\phi = Fr(\alpha,\beta) \wedge \neg Sm(\alpha) \wedge Sm(\beta)$.

We zullen bovenstaand voorbeeld nu gebruiken om het verband in vgl. 30 beter te begrijpen. Figuur 22(a) toont de kansdistributie van alle mogelijke werelden voor het oorspronkelijke MLN, en figuur 22(b) toont de WFOMC-bijdrage van iedere mogelijke wereld voor de WFOMC-encoding. Merk op dat de normalisatie-constante Z gelijk is aan de WFOMC. Wanneer we m.b.v. vgl. 30 de kans op bv. wereld 4 bepalen, rest ons nog de teller van de breuk te berekenen. Aangezien de noemer van de breuk overeenkomt met de normalisatie-constante, is het niet te verwonderen dat de teller overeenkomt met de niet-genormaliseerde kans op een wereld. Figuur 22(b) bevat reeds alle nodige informatie om de

wereld	truth-values Fr(α,β)	Sm(α)	Sm(β)	$\tilde{P}(w_i)$	$P(w_i)$
1	1	1	1	e^(2*1)	0.14
2	1	1	0	e^(2*0)	0.02
3	1	0	1	e^(2*1)	0.14
4	1	0	0	e^(2*1)	0.14
5	0	1	1	e^(2*1)	0.14
6	0	1	0	e^(2*1)	0.14
7	0	0	1	e^(2*1)	0.14
8	0	0	0	e^(2*1)	0.14

Z ≈ 52.72

(a)

wereld	truth-values Fr(α,β)	Sm(α)	Sm(β)	P(α,β)	WFOMC-bijdrage
1a	1	1	1	1	1*1*1*e^2 = e^2
1b	1	1	1	0	n.v.t. (0)
2a	1	1	0	1	n.v.t. (0)
2b	1	1	0	0	1*1*1*1 = 1
3a	1	0	1	1	1*1*1*e^2 = e^2
3b	1	0	1	0	n.v.t. (0)
4a	1	0	0	1	1*1*1*e^2 = e^2
4b	1	0	0	0	n.v.t. (0)
5a	0	1	1	1	1*1*1*e^2 = e^2
5b	0	1	1	0	n.v.t. (0)
6a	0	1	0	1	1*1*1*e^2 = e^2
6b	0	1	0	0	n.v.t. (0)
7a	0	0	1	1	1*1*1*e^2 = e^2
7b	0	0	1	0	n.v.t. (0)
8a	0	0	0	1	1*1*1*e^2 = e^2
8b	0	0	0	0	n.v.t. (0)

WFOMC ≈ 52.72

(b)

Figuur 22: (a) De oorspronkelijke gegronde formule. (b) De WFOMC-bijdrage van alle mogelijke werelden voor de oorspronkelijke formule.

WFOMC in de teller te berekenen: werelden $4a$ en $4b$ beschrijven de WFOMC-bijdrages van alle mogelijke werelden voor de formule $\Delta \wedge wereld_4$. De WFOMC van deze formule bedraagt dus e^2, waardoor $P(w_4) = e^2/52.72 = 0.14$ bedraagt, wat inderdaad overeenkomt met de kans in figuur 22(a).

3.6.5 Conclusie

Wanneer we met MLN te maken krijgen waarin existentiële kwantoren aanwezig zijn, kunnen we dus als volgt te werk gaan. Ten eerste stellen we de WFOMC-encoding op (zoals in deel 3.6.4 beschreven). Vervolgens passen we hierop de Skolemization procedure toe (zie deel 3.6.3). Tenslotte kunnen we de gewenste kansen berekenen m.b.v. vgl. 30.

We benadrukken dat deze aanpak fundamenteel verschilt van de tot dus ver besproken werkwijze. De twee eerder besproken technieken voor Skolemization herleiden het MLN tot een ander MLN (vrij van existentiële kwantoren), waarna we het tot een Markov Network herleiden, en hierop inference en learning toepassen. Wanneer we echter bovenstaande aanpak hanteren, stellen we meteen de WFOMC-encoding op van het MLN en werken daarmee verder; we gaan nooit meer terug naar een MLN of Markov Network. In dit geval vormt WFOMC de basistaak van inference en learning. Daarom blijven we ons beperken tot MLNs die vrij zijn van existentiële kwantoren. De vraag of het überhaupt mogelijk is om een Skolemization-procedure te ontwikkelen die het MLN alleen wijzigt (zodat we Markov Networks kunnen blijven gebruiken) en dit zonder bijkomstige nadelen, is een onderzoeksvraag op zich; hier gaan we in deze masterproeftekst niet op in.

Tot slot komen we nog even terug op het berekenen van de WFOMC van een

formule. We hanteerden volgende aanpak: (1) stel de groundings van de formule op, en (2) bepaal de WFOMC-bijdrage van iedere mogelijke wereld. Deze aanpak noemt men ook wel 'propositional WFOMC', omdat de grounding-stap erop neerkomt dat we de eerste-orde predicatenlogica herleiden tot propositielogica. Een andere optie is 'lifted' WFOMC, waarbij de WFOMC rechtstreeks in eerste-orde predicatenlogica berekend wordt. Deze aanpak brengt uiteraard een veel lagere computationele kost met zich mee. Zo spreekt men ook van 'lifted inference', waarbij kansen rechtstreeks in eerste-orde predicatenlogica berekend worden i.p.v. eerst een omzetting naar een Markov Network te doen. Het spreekt voor zich dat lifted inference mogelijk is m.b.v. lifted WFOMC (volgens vgl. 30). Daarnaast bestaan er ook andere benaderingen, zoals bv. een lifted versie van het Variable Elimination algoritme uit deel 2.4.2. Ook voor lifted inference bestaan zowel exacte als benaderende algoritmes. Voor een uitgebreid overzicht, verwijzen we naar Van den Broeck [22].

4 Relation Extraction met DeepDive

Relation Extraction is het extraheren van een relatie uit een grote collectie ongestructureerde data. DeepDive [1, 23] is een state-of-the-art systeem om aan Relation Extraction te doen. Het steunt hiervoor op Markov Logic Networks, wat we in hoofdstuk 3 hebben toegelicht. In dit hoofdstuk lichten we de algemene werking van DeepDive toe; de specifieke details van DeepDive laten we buiten beschouwing.

4.1 Introductie

Om onze uitleg rond de werking van DeepDive concreter te maken, introduceren we het volgende voorbeeld. Stel dat we een verzameling nieuwsartikels hebben waaruit we willen extraheren wie met wie getrouwd is. Nieuwsartikels zijn uiteraard ongestructureerde data. De relatie die we willen extraheren, noemen we $Getrouwd(p1, p2)$. Aangezien we met onzekerheden te maken hebben, zoals bijvoorbeeld contradictorische informatie, werkt DeepDive met kansen. Concreet zal DeepDive een tabel $Getrouwd(p1, p2, kans)$ opstellen, die voor een paar personen $(p1, p2)$ aangeeft hoe zeker het is dat ze met elkaar getrouwd zijn. Wanneer een artikel bijvoorbeeld de zin *"Anna en haar man Bart ..."* bevat, willen we dus dat $(Anna, Bart)$ in de tabel *Getrouwd* zit en hier een zeer hoge kans aan gekoppeld is. Vervolgens kunnen we ervan uitgaan dat wanneer de kans b.v. 0.9 of meer bedraagt, het beschouwde koppel wel degelijk getrouwd is. Zulke koppels vormen dan de eigenlijke geëxtraheerde relatie $Getrouwd(p1, p2)$.

Vooraleer we de werking van DeepDive bespreken, introduceren we alvast het centrale idee achter het bepalen van de kans dat een paar personen met elkaar getrouwd is. Hierdoor zal de werking van DeepDive namelijk veel logischer lijken. De kans dat een paar personen getrouwd is, hangt af van de 'features' die aan dat paar personen geassocieerd zijn. In dit voorbeeld is een feature de tekst tussen de namen van de personen; in deel 4.3 behandelen we een algemenere definitie. De zin *"Anna en haar man Bart ..."* bepaalt dus dat *"en haar man"* één van de features van het koppel $(Anna, Bart)$ is. Aan alle features die in de nieuwsartikels voorkomen, wordt een gewicht gekoppeld, wat geleerd wordt a.d.h.v. training data. Zo'n gewicht zegt hoe waarschijnlijk het is dat de aanwezigheid van de bijhorende feature bepaalt dat het koppel inderdaad getrouwd is. De feature *"en haar man"* zal dus ongetwijfeld een hoog gewicht krijgen. De uiteindelijke kans dat een koppel getrouwd is, hangt af van de gewichten van alle features die aan het koppel geassocieerd zijn. Dus ieder koppel dat *"en haar man"* als feature heeft, zal met hoge kans getrouwd zijn.

DeepDive realiseert Information Extraction door volgende stappen te doorlopen:

1. Data preprocessing
2. Kandidaten en features bepalen
3. Distant supervision
4. Markov Network opstellen
5. Learning
6. Inference

In de volgende delen beschrijven we wat iedere stap precies inhoudt.

4.2 Data preprocessing

DeepDive gebruikt een relationeel databasesysteem (PostgreSQL) om de data bij te houden en te queryen. De eerste stap bestaat erin de collectie ongestructureerde data in te lezen in de database. Meer bepaald voegen we iedere zin van ieder document toe aan de tabel *Zinnen*. Vervolgens passen we op iedere zin een aantal Natural Language Processing (NLP) technieken toe om meer info uit iedere zin te halen.[14] Concreet betreft het volgende technieken [25]:

- Tokenization: een zin opdelen in tokens, waarbij een token een woord of leesteken is.
- Lemmatization: de grondvorm van een woord bepalen. Voorbeelden: *kastjes* wordt *kast*, *liep* en *loopt* worden beide *lopen*.
- Part-of-Speech (POS) tagging: de woordsoort (b.v. onderwerp, werkwoord of lijdend voorwerp) van ieder woord in een zin bepalen.
- Named Entity Recognition (NER): een woord (of woordgroep) van een zin toekennen aan één van de vooraf bepaalde categorieën zoals 'persoon', 'locatie' en 'datum'. Voorbeeld: voor de zin *"Anna gaat naar New York"* krijgen we dat "Anna" een persoon is en "New York" een locatie is; de overige woorden horen tot geen enkele categorie.

De NLP gerelateerde info wordt ook allemaal in de tabel *Zinnen* opgeslagen (als extra kolommen).

4.3 Kandidaten en features bepalen

Na het verwerken van de data stellen we de verzameling 'kandidaten' op. Dit zijn de tupels waaraan DeepDive een kans moet toekennen. In ons voorbeeld zijn dat dus paren personen, maar de vraag is welke precies. Het spreekt voor zich dat we, van de koppels die in de data beschreven worden, zo veel mogelijk koppels effectief willen extraheren. Het is dus niet erg dat we te veel koppels als kandidaat beschouwen, we willen er gewoon geen over het hoofd zien. Dit principe noemt men 'high-recall'. In ons voorbeeld beschouwen we ieder paar personen dat in eenzelfde zin staat als een kandidaat-tupel. Om te bepalen welke personen in een zin voorkomen, gebruiken we de Named Entity Recognition tags uit de vorige stap. Alle kandidaten worden in de tabel *Kandidaten* opgeslagen.

Voor iedere kandidaat houden we ook zijn verzameling features bij in de tabel *Kandidaten*. Een feature is eender welke info die relevant is om de kans die aan de kandidaat geassocieerd is te bepalen. Deze info kan dus zeer uiteenlopend zijn en is volledig afhankelijk van de concrete toepassing. Zoals in de introductie beschreven, is in ons voorbeeld de tekst tussen de twee personen een mogelijke feature. Iedere zin waar kandidaat-koppel $(Anna, Bart)$ in voorkomt, resulteert dus in een nieuwe feature voor dit koppel (tenzij die feature reeds in hun verzameling features zat). Figuur 23 geeft een voorbeeld van een mogelijke invulling van de tabel *Kandidaten*.

In de praktijk gebruikt DeepDive de tekst tussen de personen niet rechtstreeks als feature. Deze tekst vormt, tezamen met de NLP annotaties van de zin (uit stap 4.2), slechts het startpunt voor de eigenlijke geëxtraheerde features. DeepDive gebruikt namelijk verschillende generieke syntactische en semantische features [26]. Enkele voorbeelden zijn: de POS-tags en NER-tags van de tekst

[14] DeepDive gebruikt hier CoreNLP [24] voor, een veelvuldig gebruikte NLP toolkit.

tupel	features
(Anna,Bart)	{“en haar man”, “is getrouwd met”}
(Charlie,Dirk)	{“en haar man”}
(Emma,Frank)	{“en haar vader”}

Figuur 23: Een mogelijke invulling van de tabel *Kandidaten*.

tussen de personen, opeenvolgingen van 1 à 3 woorden in de tekst tussen de personen na lemmatization, de som van het aantal karakters waaruit beide persoonsnamen bestaan, ... Zoals te verwachten valt, zijn niet al deze generieke features even geschikt voor ons voorbeeld. Zo is ‘de som van het aantal karakters waaruit beide persoonsnamen bestaan’ geen relevante feature[15], maar zijn het al dan niet voorkomen van woorden als “husband” en “wife” dat wel[16]. In het algemeen is het gebruik van zulke generieke features dus een goed startpunt, maar gaat men best alsnog op zoek naar domein-specifieke features, net omdat deze typisch relevanter zijn voor de classificatie. De features hoeven zich zeker niet te beperken tot tekstuele kenmerken, ook visuele informatie kan relevant zijn. Indien men bijvoorbeeld met getabelleerde data werkt, zullen de relatieve posities van entiteiten in de tabel typisch iets zeggen over hun onderlinge relatie. Een concreet voorbeeld van een mogelijks zinvolle feature is ‘de entiteiten van het kandidaat-tupel staan in eenzelfde rij’.

4.4 Distant supervision

We definiëren het ‘label’ van een kandidaat-tupel als de waarde die aan het kandidaat-tupel wordt toegekend. In ons voorbeeld zijn er twee mogelijke labels voor een kandidaat-koppel: ‘true’ of ‘false’, om aan te geven dat het kandidaat-koppel respectievelijk wel of niet getrouwd is. Een kandidaat-koppel dat wel (resp. niet) getrouwd is, noemen we een positief (resp. negatief) voorbeeld.

Distant supervision, ook wel ‘data programming’ genoemd, houdt in dat we op heuristieke wijze een verzameling voorbeelden genereren. We gaan dus (op heuristieke wijze) het label bepalen van een deelverzameling van de kandidaat-tupels. Een eerste mogelijke aanpak is het gebruik van een reeds bestaande database. Dit kan een online database zijn (zoals Wikidata [27] of YAGO [28]), of een database die iemand al eerder manueel had opgesteld. Voor ieder tupel in *Kandidaten* kijken we dus of het in de reeds bestaande database zit, en zo ja, nemen we het label over. Dit label stockeren we in de kolom *label* van de tabel *Kandidaten*. Een tweede mogelijke aanpak is het toepassen van zelf opgestelde heuristische regels. Deze zijn volledig afhankelijk van het concrete probleem. In ons voorbeeld zouden we onder meer volgende regels kunnen hanteren:

1. Als één van de features van een kandidaat-koppel het woord “echtgenoot” of “echtgenote” bevat, labelen we het als ‘true’.
2. Als één van de features van een kandidaat-koppel het woord “broer”, “zus”, “vader”, “moeder” of dergelijke bevat, labelen we het als ‘false’.

Op ieder kandidaat-tupel zijn dus een aantal van deze heuristieken van toepassing (mogelijks geen enkele). Bijgevolg is het mogelijk dat eenzelfde

[15] DeepDive zal voor deze feature een gewicht leren met een relatief kleine absolute waarde.
[16] DeepDive zal voor deze features gewichten leren met relatief grote absolute waardes.

kandidaat-tupel verschillende labels heeft gekregen. Stel bijvoorbeeld dat volgens de reeds bestaande database $(Anna, Bart)$ wel een koppel is, maar dat één van hun features het woord "broer" bevat. Wanneer een kandidaat-koppel vaker 'true' dan 'false' als label kreeg, beschouwen we 'true' als het uiteindelijke label; analoog indien 'false' vaker voorkwam. Aan een kandidaat-tupel dat even vaak 'true' als 'false' gelabeld werd, of helemaal niet gelabeld werd, kennen we het bijzondere label 'onbepaald' toe. Het eindresultaat van deze stap is dus dat ieder kandidaat-tupel een label heeft ('true', 'false' of 'onbepaald').

We bespreken nu wat het nut van distant supervision eigenlijk is. Zoals we in de introductie (zie deel 4.1) hebben vermeld, is er een gewicht geassocieerd aan iedere feature, waarmee we uiteindelijk de kans voor een kandidaat-tupel zullen berekenen. Deze gewichten worden geleerd a.d.h.v. training data. Zulke training data worden gegenereerd op basis van de (positieve en negatieve) voorbeelden die distant supervision heeft bepaald. In deel 4.6 beschrijven we hoe dit precies in zijn werk gaat. Merk op dat we de voorbeelden manueel hadden kunnen bepalen; we hadden zelf de labels kunnen bepalen van een aantal kandidaat-tupels (mits opzoekingswerk). Zulke 'hand-labeling' is echter enorm tijdrovend wanneer men een groot aantal kandidaat-tupels wilt labelen. Distant supervision daarentegen laat ons toe om snel en eenvoudig een groot aantal labels te bepalen. Herinner dat distant supervision heuristieken hanteert, waardoor foutieve labels, en bijgevolg ook foutieve training data, mogelijk zijn. In deel 4.6 zullen we echter zien dat dit geen probleem is.

4.5 Markov Network opstellen

Om in de volgende stap (zie deel 4.6) de kans te kunnen bepalen die bij een kandidaat-tupel hoort, gebruikt DeepDive een Markov Network, afgeleid van een Markov Logic Network (MLN). In dit deel bespreken we eerst hoe het MLN wordt opgebouwd, en vervolgens hoe het herleid wordt naar het bijhorende Markov Network.

4.5.1 Markov Logic Network opstellen

Herinner dat een MLN gedefinieerd is als een verzameling paren (F_i, w_i), met F_i een formule in eerste-orde predicatenlogica en w_i het bijhorende gewicht. DeepDive stelt het MLN op als volgt. Voor iedere unieke feature f wordt het paar $(\forall p_1\ \forall p_2\ \mathit{Getrouwd}(p1, p2);\ w_f)$ toegevoegd, en de groundings van het predicaat $\mathit{Getrouwd}(p1, p2)$ in de formule met gewicht w_f zijn die kandidaat-tupels die feature f bezitten. We benadrukken dat iedere feature een eigen gewicht w_f heeft, dat voorlopig nog onbekend is en door het leren (zie deel 4.6) zal worden ingevuld. Beschouw bij wijze van voorbeeld de eerder gebruikte tabel *Kandidaten* zoals getoond in figuur 24(a) (de kolom *label* wordt niet getoond). Figuur 24(b) toont een mapping van iedere feature naar een gewicht. Figuur 24(c) toont het bijhorende MLN.

Om de overgang naar de bijhorende Factor Graph te vereenvoudigen, herleiden we het MLN tot een geground MLN. Hiervoor gebruiken we het algoritme dat we in deel 3.5 gedefinieerd hebben. De mogelijke groundings van predicaat $\mathit{Getrouwd}(p1, p2)$ zijn die kandidaat-tupels die feature f bezitten; we kijken hiervoor naar de tabel *Kandidaten*. Figuur 24(d) toont het gegrounde MLN.

tupel	features
(Anna,Bart)	{“en haar man”, “is getrouwd met”}
(Charlie,Dirk)	{“en haar man”}
(Emma,Frank)	{“en haar vader”}

(a)

feature	gewicht
“en haar man”	w_1
“is getrouwd met”	w_2
“en haar vader”	w_3

(b)

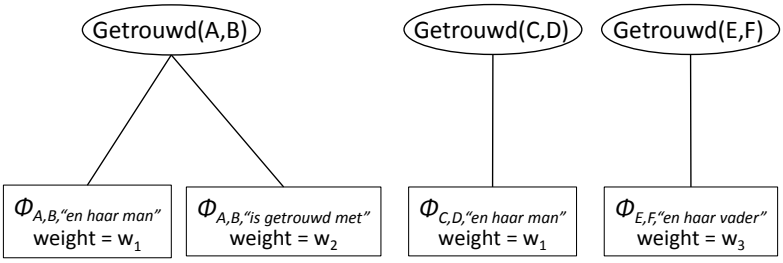
$\forall p_1 \forall p_2$ Getrouwd(p_1, p_2)	weight = w_1 ;
$\forall p_1 \forall p_2$ Getrouwd(p_1, p_2)	weight = w_2 ;
$\forall p_1 \forall p_2$ Getrouwd(p_1, p_2)	weight = w_3

(c)

Getrouwd(Anna,Bart)	weight = w_1 ;
Getrouwd(Charlie,Dirk)	weight = w_1 ;
Getrouwd(Anna,Bart)	weight = w_2 ;
Getrouwd(Emma,Frank)	weight = w_3

(d)

Figuur 24: (a) De tabel *Kandidaten*. (b) De verzameling features met hun geassocieerde gewichten. (c) Het bijhorende MLN. (d) Het gegrounde MLN.



Figuur 25: De Factor Graph die bij het MLN in figuur 24(c) hoort.

4.5.2 MLN herleiden tot Markov Network

In deel 3.2 beschreven we reeds hoe een MLN een Markov Network definieert. Ten eerste gebruikten we vergelijking 6 (uit deel 2.3.5) om de Log-Linear Model parametrisatie van het Markov Network te herleiden tot een product van factoren. Op basis daarvan stelden we de Factor Graph representatie van het Markov Network op, wat op het volgende neerkwam. We voegen een variabele-knoop toe voor iedere grounding van ieder predicaat, en een factor-knoop met gewicht w_i voor iedere mogelijke grounding van iedere formule F_i. Een factor-knoop is verbonden met alle gegrounde predicaten die in de beschouwde gegrounde formule voorkomen. Figuur 25 toont de Factor Graph die bij het MLN in figuur 24(c) hoort. We benadrukken dat het gewicht dat aan een factor geassocieerd is, afhangt van de feature.

Ten tweede kwamen we tot vergelijking 25 (uit deel 3.3.1) om de kans op een bepaalde wereld te berekenen. Herinner dat een bepaalde wereld bestaat uit de toekenning van een truth-value aan ieder geground predicaat – aan iedere binaire variabele dus. We herhalen de vergelijking hier gemakkelijkheidshalve:

$$P(\mathcal{X}{=}x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{m} w_i n_i(x)\right) \tag{31}$$

met m het aantal formules in het MLN, $\mathcal{X}$ de verzameling binaire variabelen, en $n_i(x)$ het aantal groundings van de i-de formule dat naar waar evalueert in de beschouwde wereld. Laat ons er bij wijze van voorbeeld van uitgaan dat de features *"en haar man"* en *"is getrouwd met"* er inderdaad op duiden dat het kandidaat-koppel getrouwd is, en dat de feature *"en haar vader"* er inderdaad op duidt dat het kandidaat-koppel niet getrouwd is. We gaan er dus van uit dat w_1 en w_2 grote positieve getallen zijn, en w_3 een groot negatief getal is. Uit bovenstaande vergelijking volgt dan, zoals verwacht, dat de wereld waarin $\mathit{Getrouwd}(A, B) = \mathit{true}$, $\mathit{Getrouwd}(C, D) = \mathit{true}$ en $\mathit{Getrouwd}(E, F) = \mathit{false}$ de hoogste kans zal hebben.

4.5.3 Formules met extra domeinkennis

Tot nog toe beschreven we hoe DeepDive automatisch de nodige formules (met te leren gewichten) aan het MLN toevoegt, gebaseerd op de features van de kandidaten. Daarnaast laat DeepDive toe dat de gebruiker zelf nog extra formules toevoegt aan het MLN. Op die manier kan hij extra domeinkennis aan het model toevoegen, en zo tot betere resultaten komen. Deze formules zijn uiteraard volledig afhankelijk van het concrete probleem. In ons voorbeeld zouden de volgende gepast zijn [29]:

1. $\mathit{Getrouwd}(p_1, p_2) \Rightarrow \mathit{Getrouwd}(p_2, p_1)$ met als gewicht b.v. 3.0. Deze formule drukt de symmetrie van getrouwd zijn uit.
2. $\mathit{Getrouwd}(p_1, p_2) \Rightarrow \mathit{Getrouwd}(p_1, p_3)$ met als gewicht b.v. -1.0. Deze formule drukt, door het negatieve gewicht, uit dat eenzelfde persoon niet met meerdere personen getrouwd kan zijn.

Dit wijzigt uiteraard niets aan de manier waarop DeepDive de Factor Graph opstelt en de kans op een bepaalde wereld berekent.

Hierboven kozen we ervoor om het teken van het gewicht te bepalen op basis van het geval dat de formule naar waar zou evalueren. Bijvoorbeeld: indien de symmetrie-relatie voldaan is, moet de kans verhogen, waardoor we voor een positief gewicht moeten kiezen. Het is echter ook mogelijk om vanuit de negatie van de formule te redeneren: indien de symmetrie-relatie niet voldaan is, moet de kans verlagen, waardoor we voor een negatief gewicht moeten kiezen. In het tweede geval verkrijgen we dus de formule: $\mathit{Getrouwd}(p_1, p_2) \wedge \neg \mathit{Getrouwd}(p_2, p_1)$ met -3.0 als gewicht. We kunnen ons nu terecht afvragen of die keuze een impact heeft op de kansdistributie van alle mogelijke werelden. Zo dadelijk zullen we aantonen dat dit niet het geval is; het maakt niet uit of je de gewone formule of zijn negatie gebruikt.

Vooraleer we hier verder op ingaan, tonen we het volgende aan. Beschouw de kansdistributie P over alle mogelijke werelden $x_1, x_2, \ldots, x_n$. Beschouw dezelfde kansdistributie P' waarbij we voor iedere mogelijke wereld eenzelfde constante c optellen bij de geëxponeerde waarde van zijn niet-genormaliseerde kans. Voor een wereld x_i krijgen we dus:

$$\widetilde{P}(x_i) = \exp\left(\sum_{i=1}^{m} w_i f_i(x)\right) \quad \text{en} \quad \widetilde{P'}(x_i) = \exp\left(c + \sum_{i=1}^{m} w_i f_i(x)\right)$$

In deel 3.5.4 verklaarden we reeds intuïtief waarom P en P' gelijk zijn aan elkaar, maar nu zullen we dit formeel doen. Ten eerste geldt $\widetilde{P'}(x_i) = \exp(c) \cdot \widetilde{P}(x_i)$,

wegens $\exp(a+b) = \exp(a) \cdot \exp(b)$. Laat ons de normalisatie-constante voor P' noteren als Z'; die voor P noteren we als Z. We weten per definitie dat $Z' = \widetilde{P'}(x_1) + \widetilde{P'}(x_2) + \ldots + \widetilde{P'}(x_n)$. Op basis van de voorgaande gelijkheid kunnen we Z' nu schrijven als: $Z' = \exp(c) \cdot (\widetilde{P}(x_1) + \widetilde{P}(x_2) + \ldots + \widetilde{P}(x_n))$, wat gelijk is aan $\exp(c) \cdot Z$. Wanneer we nu de definitie van $P'(x_i)$ samenvoegen met de voorgaande gelijkheden, bekomen we: $P'(x_i) = \widetilde{P'}(x_i)/Z' = (\exp(c) + \widetilde{P}(x_i))/(\exp(c) \cdot Z) = P(x_i)$. We hebben dus aangetoond dat $P'(x_i) = P(x_i)$ voor alle x_i.

Laat ons nu aantonen dat de kansdistributie van een MLN met een bepaalde formule F met gewicht w_f niet wijzigt wanneer we er de negatie van nemen; de negatie betreft enerzijds de logische negatie van F (i.e. $\neg F$) en anderzijds het tegengestelde gewicht (i.e. $-w_f$). We beperken ons tot een informeel bewijs. Beschouw een willekeurig MLN dat de formule F bevat met 1.0 als gewicht, en veronderstel dat er 100 mogelijke groundings zijn voor F. Beschouw een willekeurige wereld w_1, en noem x de geëxponeerde waarde van $\widetilde{P}(w_1)$ zonder de bijdrage van F te beschouwen. Veronderstel dat 60 (van de 100) groundings van F waar zijn in w_1, waardoor we $+60.0$ optellen bij x; de resulterende geëxponeerde waarde noemen we x^P, zodat $x^P = x + 60.0$. Beschouw nu de negatie: in w_1 zijn 40 (van de 100) groundings van $\neg F$ waar, met -1.0 als gewicht. Daardoor tellen we -40.0 op bij x; de resulterende geëxponeerde waarde noemen we x^N, zodat $x^N = x + (-40.0)$. Bijgevolg geldt dat $x^N = x^P + (-100.0)$; we hebben -100.0 opgeteld bij de oorspronkelijke geëxponeerde waarde van $\widetilde{P}(w_1)$.

Beschouw nog een tweede willekeurige wereld w_2, waarin 70 (van de 100) groundings van F waar zijn. Op volledig analoge wijze geldt ook hier dat $x^N = x^P + (-100.0)$. Meer algemeen geldt dit voor alle mogelijke werelden $x_1, x_2, \ldots, x_n$. Laat ons de kansdistributie van het MLN met de oorspronkelijke formule P noemen. De kansdistributie van het MLN met de negatie, P' genoemd, is dus dezelfde als P maar waarbij voor iedere mogelijke wereld -100.0 werd opgeteld bij de geëxponeerde waarde van zijn niet-genormaliseerde kans. Zoals hierboven reeds bewezen, wijzigt dit de kansdistributie niet; $P' = P$. Nu hebben we ook meteen aangetoond dat de keuze tussen de oorspronkelijke formule en zijn negatie geen impact heeft op de kansdistributie van de mogelijke werelden.

4.6 Learning en Inference

Nu het Markov Network is opgesteld, kunnen de gewichten van de parameters geleerd worden a.d.h.v. training data. De te leren parameters zijn voor alle duidelijkheid de w_i's in vergelijking 31. DeepDive gebruikt complete samples als training data; ieder sample bestaat uit de toekenning van een waarde aan iedere (binaire) variabele. De training data worden als volgt gegenereerd op basis van de positieve en negatieve voorbeelden die distant supervision (zie deel 4.4) ons bezorgde. Noem de verzamelingen positieve en negatieve voorbeelden respectievelijk P en N. We noemen een wereld 'consistent' met P en N als de wereld 'true' heeft toegekend aan de variabelen overeenkomstig met P en 'false' heeft toegekend aan de variabelen overeenkomstig met N; de waarden die de wereld heeft toegekend aan de variabelen met label 'onbepaald' zijn niet van belang. De training data bestaat uit alle mogelijke werelden die consistent zijn met P en N. In deel 2.8.1 beschreven we reeds hoe de parameters van een

Markov Network geleerd kunnen worden wanneer we complete training data gebruiken. Het resultaat van deze leerfase is dat ieder gewicht w_i nu aangeeft in welke mate de aanwezigheid van de geassocieerde feature f_i er op duidt dat een kandidaat-koppel wel of niet tot de relatie hoort.

Het uiteindelijke doel van DeepDive is om voor ieder kandidaat-tupel de kans te bepalen dat het tot de relatie hoort. Herinner dat ieder kandidaat-tupel overeenkomt met een binaire random variable in het Markov Network; deze noemen we v_i. DeepDive benadert de marginale kans $P(v_i)$ m.b.v. Gibbs Sampling, wat reeds in deel 2.5.4 werd beschreven. Merk op dat deze kansberekening geen gebruik maakt van de labels die door distant supervision werden voorzien. Het zou inderdaad niet goed zijn om deze als 'evidence' te zien, en dus de de conditionele kans $P(v_i|e)$ te berekenen, omdat er (door de heuristische aanpak) fouten in de training data kunnen zitten. Door het Markov Network te trainen met veel training data, kan het zulke fouten abstraheren en dus leren hoe relevant iedere feature wel degelijk is.

We kunnen de kans die aan iedere kandidaat-tupel geassocieerd is, zien als de output van DeepDive. Echter, in de context van het opstellen van een eigen knowledge base, zal men typisch alleen geïnteresseerd zijn in de "feitelijke" relaties. Daartoe houdt men alleen de kandidaat-tupels over waaraan een hoge kans (b.v. minstens 0.9) werd toegekend.

4.7 Relatie tot een klassieke aanpak

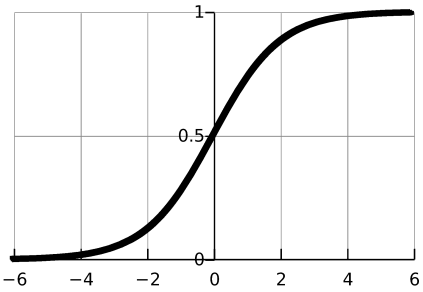
De centrale taak van Relation Extraction is bepalen of een kandidaat-tupel al dan niet tot de te extraheren relatie behoort, wat een binair classificatie-probleem is. De aanpak die DeepDive hanteert is op het eerste zicht echter enorm verschillend van een klassieke aanpak daarvoor. In dit deel gaan we na welke gelijkenissen en verschillen er zijn, en wat de motivatie hiervoor is.

4.7.1 Classificatie m.b.v. Logistic Regression

We beginnen met uit te leggen wat we precies verstaan onder een 'klassieke' aanpak voor classificatie m.b.v. Machine Learning[17] [19]. Het doel van classificatie is bepalen tot welke klasse een te classificeren object behoort. Het aantal klassen is steeds eindig; indien er twee mogelijke klassen zijn, spreekt men van binaire classificatie. Een voorbeeld van een binair classificatie-probleem is, zoals reeds gezegd, bepalen of een kandidaat-tupel wel of niet tot de te extraheren relatie behoort. Binaire classificatie vindt men ook terug in andere domeinen, bijvoorbeeld in Computer Vision: gegeven een afbeelding van een hond of kat, bepaal welk dier wordt afgebeeld (i.e. "hond" of "kat"). Dit zullen we als lopend voorbeeld hanteren in de rest van dit deel.

Een eerste kenmerk van een klassieke aanpak tot classificatie, is dat een te classificeren object steeds wordt voorgesteld door zijn feature-vector. De feature-vector beschrijft de waarden van een aantal zelf gekozen karakteristieken van het te classificeren object. Nuttige karakteristieken voor ons hond-vs-kat probleem zouden kunnen zijn: afstand tussen de oren, diameter van de ogen, verhouding van de lengte van de snorharen t.o.v. de breedte van het hoofd, en aanwezigheid van variatie in kleur. Een feature neemt steeds een numerieke waarde aan; zowel continue als discrete waarden zijn mogelijk. De i-de waarde

[17] Zoals uit de verdere uitleg zal blijken, bedoelen we eigenlijk Supervised Machine Learning.



Figuur 26: De sigmoid $\sigma(z) = 1/(1 + e^{-z})$, met een karakteristieke S-vorm.

van de feature-vector van een object beschrijft de waarde van de i-de karakteristiek van dat object. Een hond zou als feature-vector bv. $(6.5, 1.1, 0.4, 0)$ kunnen hebben, en een kat bv. $(4.2, 1.2, 0.7, 1)$.

Een Machine Learning model bepaalt de klasse waartoe een object hoort op basis van de feature-vector van dat object. Meer bepaald is de output de kans dat het object tot een bepaalde klasse hoort. De manier waarop die kans berekend wordt, is afhankelijk van het gebruikte Machine Learning model. Er bestaan vele mogelijke Machine Learning modellen, het ene complexer dan het andere. Enkele veel gebruikte zijn: een Support Vector Machine (SVM), een Neuraal Netwerk, en Logistic Regression. Laat ons Logistic Regression i.h.b. beschouwen, omdat dit model het sterkst lijkt op het Markov Logic Network. De klasse "hond" stellen we voor door het getal 1, en de klasse "kat" door het getal 0. Beschouw een willekeurige input (afbeelding met een hond of kat) met feature-vector v. Logistic Regression definieert de kans dat v tot klasse 1 ("hond") behoort, als:

$$P(v{=}1) = \sigma(w \bullet v + b) = \frac{1}{1 + \exp(-w \bullet v - b)} \tag{32}$$

Eerst wordt het dot product van de gewicht-vector en de feature-vector berekend. Vervolgens wordt hierop de 'sigmoid' toegepast, met zijn karakteristieke S-vorm, zoals in figuur 26 afgebeeld. Deze functie, genoteerd als $\sigma(z)$, plaatst een zachte treshold op zijn input, om deze naar $[0; 1]$ te mappen. De rol van b is om deze treshold te kunnen verplaatsen, weg van 0. De gewicht-vector w wordt gebruikt om een gewicht te koppelen aan iedere feature. Deze gewichten vormen, tezamen met b, de parameters van dit type Machine Learning model.

Een tweede kenmerk is dat de werking van eender welk Machine Learning model gekenmerkt wordt door een aantal parameters (de gewichten en b in het geval van Logistic Regression). Deze parameters worden geleerd uit voorbeelden (training data), zijnde een collectie tupels van de vorm $(\textit{feature vector}, \textit{klasse})$; objecten waarvan we de bijhorende klasse reeds kennen. In ons voorbeeld zouden dit foto's zijn waarvoor we reeds weten of het een hond of kat is die erop staat afgebeeld.

Een derde kenmerk is het gebruik van gradient descent (zie deel 2.7.2), toegepast op een kostfunctie, om geschikte waarden te leren voor de parameters. Een kostfunctie bepaalt hoe slecht de huidige waarden van de parameters zijn, gegeven de voorbeelden, en moet dus geminimaliseerd worden. Zodra deze pa-

rameters geleerd zijn, hebben we een getraind Machine Learning model, dat we kunnen gebruiken om objecten te classificeren waarvan we de klasse nog niet kenden. Stel dat de gewichten in ons voorbeeld $(5.2, 0.8, -3.1, -4.6)$ zijn en $b = -24.5$, dan zouden de eerdere vermelde feature-vectoren voor een hond en een kat correct geclassificeerd worden. Een niet eerder beschouwde afbeelding met als feature-vector $(4.6, 1.5, 0.8, 1)$ zou als kat geclassificeerd worden: de kans dat deze afbeelding een hond afbeeldt, bedraagt 0.16% (en 99.84% voor een kat).

4.7.2 Gelijkenis met Logistic Regression

Om de kandidaat-tupels te classificeren (als wel of niet tot de geëxtraheerde relatie behorend), steunt DeepDive fundamenteel op het MLN zoals in deel 4.5 beschreven. DeepDive baseert zich dus enerzijds op de features van de kandidaat-tupels, en anderzijds op extra domeinkennis. Wanneer we ons beperken tot de features, kunnen we de DeepDive methodologie min of meer zien als Logistic Regression, als volgt.

Herinner dat het MLN de formule $(\mathit{Getrouwd}(p1, p2); w_f)$ bevat voor iedere feature f, en dat de groundings van *Getrouwd* in de formule met w_f alle kandidaat-tupels zijn die feature f bezitten. Bijgevolg bevat het Markov Network, geparametriseerd als Log-Linear Model (LLM), een LLM-feature met gewicht w_f voor iedere feature f van ieder kandidaat-tupel. Een LLM-feature is steeds gedefinieerd als de truth-value van de gegrounde formule, i.e. de truth-value van het kandidaat-tupel in de beschouwde wereld. We kunnen de kans op een wereld x daarom ook algemener schrijven als:

$$\begin{aligned} P(x) &= \frac{1}{Z} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} w_i \cdot \mathit{hasFeature}(t,i) \cdot f_t(x)\right) \\ &= \frac{1}{Z} \exp\left(\sum_{t=1}^{n} f_t(x) \cdot \sum_{i=1}^{m} w_i \cdot \mathit{hasFeature}(t,i)\right) \\ &= \frac{1}{Z} \exp\left(\sum_{t=1}^{n} f_t(x) \cdot (w \bullet v)\right) \end{aligned} \tag{33}$$

De eerste sommatie gaat over alle tupels en de tweede over alle mogelijke features (van alle kandidaten tezamen), zodat iedere combinatie van een tupel en een mogelijke feature doorlopen wordt. De functie $\mathit{hasFeature}(t, i)$ geeft 1 terug indien het t-de tupel de i-de feature bezit, en anders 0. De $f_t(x)$ is de bijhorende LLM-feature, i.e. de truth-value van het t-de tupel in x.

Wanneer we nu één specifiek kandidaat-tupel beschouwen en ervan uitgaan dat het waar is in de beschouwde wereld x, is zijn bijdrage tot de $\widetilde{P}(x)$ gelijk aan: $\sum_{i=1}^{m} w_i \cdot \mathit{hasFeature}(t, i)$, oftewel $w \bullet v$ in het kort, met w de gewichtvector en v zijn (binaire) feature-vector. Logistic Regression classificeert één tupel, maar DeepDive classificeert eigenlijk alle tupels tegelijk (door in termen van een wereld te werken). Daartoe sommeert DeepDive over het dot product van de tupels die waar zijn in de wereld. Hieruit leiden we af dat het dot product van een kandidaat-tupel wel degelijk representatief is voor de classificatie van dat tupel, zoals dat ook bij Logistic Regression het geval is.

Om onze uitleg volledig te maken, vermelden we nog dat een kandidaat-tupel een bepaalde feature f meermaals kan bezitten, laat ons zeggen k keer. Het LLM

bevat een LLM-feature voor ieder voorkomen van f, wat neerkomt op k identieke LLM-features. In bovenstaande vergelijking zou de functie $\mathit{hasFeature}(t, i)$ daarom veralgemeend moeten worden naar $\mathit{featureCount}(t, i)$, die voor het t-de tupel en de i-de feature teruggeeft hoe vaak het tupel de feature bezit. De componenten van vector v zijn dan niet langer binaire waarden, maar natuurlijke getallen (inclusief 0) die aanduiden hoe vaak het tupel iedere mogelijke feature bezit. Deze representatie is vergelijkbaar met het Bag-of-Words model [19], wat vaak gebruikt wordt in de context van Natural Language Processing.

Laat ons dit verband nagaan op theoretisch vlak, door vanuit vgl. 33 te vertrekken. Beschouw het specifieke scenario waarin er slechts één kandidaat-tupel t bestaat, waardoor er slechts twee mogelijke werelden zijn: wereld x_1 waarin t waar is, en wereld x_0 waarin t onwaar is. Merk op dat $\widetilde{P}(x_1) = \exp(w \bullet v)$ en $\widetilde{P}(x_0) = \exp(0)$. Bijgevolg geldt dat:

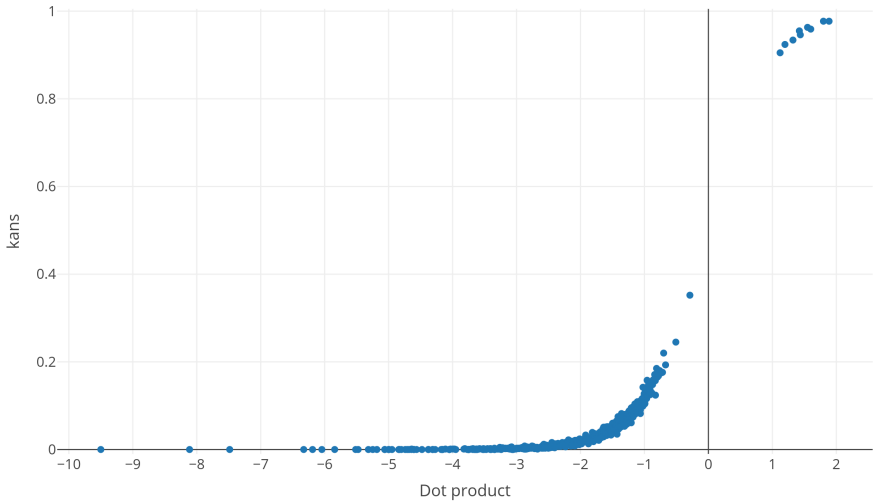
$$P(x_1) = \frac{\widetilde{P}(x_1)}{\widetilde{P}(x_1) + \widetilde{P}(x_0)} = \frac{\exp(w \bullet v)}{\exp(w \bullet v) + 1} = \frac{1}{1 + \exp(-w \bullet v)}$$

Dit is precies hoe we m.b.v. Logistic Regression (zie vgl. 32) de kans zouden berekenen dat t, met feature-vector v, tot de klasse van te extraheren kandidaat-tupels behoort (met als treshold $b = 0$).

We valideren deze uitspraak ook experimenteel, voor een algemener scenario. Op de website van DeepDive [29] is een volledig uitgewerkt voorbeeld terug te vinden waarmee de werking van het systeem geïllustreerd wordt. Men extraheert er de relatie $\mathit{Getrouwd}(p1, p2)$ uit nieuwsartikels, wat we als lopend voorbeeld gebruikten in deze masterproeftekst. De data (nieuwsartikels) en code voor het Relation Extraction proces zijn dus al beschikbaar, waardoor we bovenstaande uitspraak snel kunnen valideren; in deel 5 gaan we verder in op de praktische kant van DeepDive. We gebruiken 100 nieuwsartikels, wat neerkomt op een totaal van 633 kandidaat-koppels en 61097 unieke features. We verwijderen de eigen regels uit de definitie van het MLN, en laten DeepDive vervolgens de gewichten leren. Ten slotte stellen we een relatie op die per kandidaat-koppel de volgende informatie bijhoudt: de kans dat het kandidaat-koppel getrouwd is, en de som van de gewichten van de features die het kandidaat-koppel bezit (i.e. het dot product). Figuur 27 visualiseert de kans in functie van het dot product, van ieder kandidaat-koppel. Deze curve heeft, zoals verwacht, dezelfde vorm als de sigmoid. In de buurt van de treshold, waar het dot product ongeveer 0 is en de kansen tussen 0.2 en 0.8 schommelen, bevinden zich weinig datapunten. Dit wijst op een classificatie met een hoge nauwkeurigheid: het is zeer duidelijk tot welke klasse ieder kandidaat-tupel behoort.

Net zoals Logistic Regression, maakt ook de DeepDive methodologie gebruik van training data om de parameters van het Markov Network te leren. Beide passen daartoe een gradiënt-gebaseerde optimalisatie toe. Bij DeepDive wordt de likelihood gemaximaliseerd (gradient ascent), terwijl bij Logistic Regression de kostfunctie geminimaliseerd wordt (gradient descent). Aangezien de likelihood beschrijft hoe "goed" de keuze van de parameters is en de kost hoe "slecht", wordt de gradiënt op tegengestelde wijze gebruikt voor het updaten van de parameters.

We concluderen dat, indien we de DeepDive methodologie beperken tot het gebruik van features, deze aanpak volledig gelijk loopt met de klassieke aanpak



Figuur 27: De kans in functie van het dot product, voor 636 kandidaat-tupels.

tot classificatie. Samengevat geldt ook voor de DeepDive methodologie dat:

1. een te classificeren object (kandidaat-tupel) voorgesteld wordt door zijn feature-vector;
2. de classificatie mede afhangt van de parameters van het model (de gewichten), en een MLN i.h.b. gelijkt op Logistic Regression;
3. die parameters geleerd worden uit training data, m.b.v. gradiënt-gebaseerde optimalisatie.

4.7.3 Wat DeepDive uniek maakt

Het fundamentele verschil tussen de DeepDive methodologie en de klassieke aanpak tot classificatie, is dat de eerste in staat is om eventuele domeinkennis in het model op te nemen. Dankzij die domeinkennis heeft de DeepDive methodologie het potentieel om betere resultaten te behalen dan de klassieke aanpak. Het spreekt ook helemaal niet voor zich hoe we extra domeinkennis zouden kunnen opnemen in een klassiek Machine Learning model. De reden daarvoor is de volgende. In de klassieke aanpak wordt ieder te classificeren object afzonderlijk beschouwd; de features van het object zijn het enige relevante voor de classificatie van dat object. Domeinkennis, daarentegen, zegt iets over de relaties tussen de te classificeren objecten. De classificatie van het ene object hangt mede af van (of heeft een impact op) de classificatie van de andere objecten. Domeinkennis forceert ons dus om de te classificeren objecten in hun geheel te beschouwen. Daarom werkt DeepDive steeds met de notie van ‘een mogelijke wereld’: een mogelijke classificatie van alle te classificeren objecten tegelijkertijd.

Een Markov Logic Network is de ideale keuze voor een Machine Learning model dat zowel features als domeinkennis moet ondersteunen. We zagen inderdaad reeds dat domeinkennis zeer eenvoudig en bijzonder compact beschreven kan worden m.b.v. eerste-orde predicatenlogica. Uiteraard had men ook kunnen opteren om de gebruiker rechtstreeks een Markov Network te laten opstellen

(i.p.v. een MLN), maar op vlak van gebruiksvriendelijkheid zou dat duidelijk geen goede keuze zijn.

Naast het toelaten van domeinkennis, biedt DeepDive nog twee meerwaarden op praktisch vlak. Om deze toe te lichten, moeten we wat verder ingaan op de praktische kant van DeepDive. DeepDive voorziet namelijk een eigen declaratieve programmeertaal, 'DDlog' genaamd, waarin de gebruiker het volledige Relation Extraction proces kan beschrijven. Deze taal is gebaseerd op de Datalog-syntax, en werd uitgebreid met de mogelijkheid een MLN te definiëren (in eerste-orde predicatenlogica). Op de precieze syntax en semantiek komen we in deel 5.2 terug.

De eerste praktische meerwaarde die DDlog voorziet, heet 'weight tying'. Herinner uit deel 4.5.1 hoe we het MLN gingen opstellen: per unieke feature f wordt het paar $(\forall p_1\ \forall p_2\ \mathit{Getrouwd}(p1, p2);\ w_f)$ toegevoegd, en de groundings van het predicaat $\mathit{Getrouwd}(p1, p2)$ in de formule met gewicht w_f zijn die kandidaat-tupels die feature f bezitten. Het spreekt voor zich dat er typisch duizenden features geëxtraheerd worden, waardoor de gebruiker een MLN zou moeten definiëren dat uit duizenden regels bestaat – dit is uiteraard niet gebruiksvriendelijk. De weight tying in DDlog laat ons echter toe dit samen te vatten tot één regel:

```
@weight(feature) Getrouwd(p1,p2) :- Kandidaten(p1, p2, feature).
```

Dit genereert precies hetzelfde gegronde MLN als voorheen: per entry in *Kandidaten* wordt de gegronde formule $\mathit{Getrouwd}(p1, p2)$ gegenereerd, met het gewicht afhankelijk van *feature*. Met de notatie @*weight*(*feature*) koppelen/binden we dus het gewicht van de gegronde formule aan de beschouwde feature. We benadrukken dat weight tying louter voor een compactere notatie zorgt, het is zeker geen fundamentele uitbreiding.

Ten tweede laat DDlog toe om 'User-Defined Functions' (UDFs) te hanteren binnen het Relation Extraction proces. Dit zijn functies die de gebruiker implementeerde in een programmeertaal naar keuze, zoals Python. Meer bepaald laat DDlog toe een functie te definiëren wiens implementatie in een opgegeven script kan teruggevonden worden. Op die manier wordt het haalbaar om alle stappen van het Relation Extraction proces in één DDlog-bestand te implementeren. Het extraheren van de verzameling kandidaat-tupels uit de nieuwsartikels, bijvoorbeeld, kan m.b.v. een Python-script. De output van zo'n UDF zal typisch gebruikt worden om een relatie in de database in te vullen. De precieze syntax van UDFs leggen we pas in deel 5.2.2 uit. Ook UDFs bieden uiteraard geen fundamentele meerwaarde, ze bevorderen louter de gebruiksvriendelijkheid.

Herinner ten slotte dat DeepDive vereist dat alle kandidaat-tupels gekend zijn, vooraleer de classificatie ervan van start gaat. De reden daarvoor is dat er per kandidaat-tupel een variabele wordt geïntroduceerd in het Markov Network, waarvoor vervolgens de marginale kans berekend wordt. Het zou dus niet mogelijk zijn om het getrainde model te hergebruiken voor een andere collectie nieuwsartikels, met andere kandidaat-tupels in. Onze enige optie zou zijn om het volledige proces opnieuw te doorlopen[18]. In de klassieke aanpak daarentegen, is dat niet het geval, omdat alleen naar de feature vector van het te classificeren object wordt gekeken om de kans te bepalen. Het Machine Learning

[18]Ré [1] bespreekt technieken die een incrementele aanpak toelaten, zodat men strikt genomen niet volledig opnieuw moet beginnen.

tupel	feature-vector
(Anna,Bart)	(1,1,0)
(Charlie,Dirk)	(1,0,0)
(Emma,Frank)	(0,0,1)

(a)

tupel	feature-vector	kans
Fictief_1	(1,1,1)	0.82
(Anna,Bart)	(1,1,0)	0.99
Fictief_2	(1,0,1)	0.38
(Charlie,Dirk)	(1,0,0)	0.92
Fictief_3	(0,1,1)	0.27
Fictief_4	(0,1,0)	0.88
(Emma,Frank)	(0,0,1)	0.05
Fictief_5	(0,0,0)	0.50

(b)

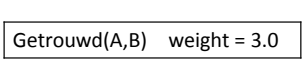
Figuur 28: (a) De echte tabel *Kandidaten*. (b) De uitgebreide tabel *Kandidaten* en hypothetische kansen.

model kan dus probleemloos een (zinvolle) kans koppelen aan objecten die niet eerder beschouwd werden. Dit verschil lijkt een beperking van de DeepDive methodologie aan het licht te brengen, maar we kunnen deze eenvoudig opheffen als volgt.

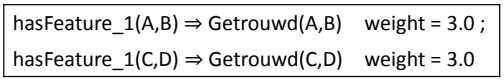
Beschouw de verzameling van alle 2^n mogelijke (binaire) feature-vectoren, met n het aantal unieke features. We berekenen het verschil van deze verzameling en de verzameling van alle feature-vectoren die aan een kandidaat-tupel geassocieerd zijn. Voor iedere feature-vector in de resulterende verzameling definiëren we een fictief kandidaat-tupel, dat zogezegd die feature-vector bezit. Op die manier bestaat er dus minstens één kandidaat-tupel voor iedere mogelijke feature-vector. Wanneer DeepDive het model nu getraind heeft, kunnen we een niet eerder beschouwd tupel t toch classificeren. Daartoe beschouwen we de feature-vector van t en bepalen het overeenkomstige fictieve kandidaat-tupel. De kans dat dit fictieve kandidaat-tupel tot te extraheren tupels behoort, komt overeen met die voor t. Figuur 28(a) geeft een voorbeeld van een mogelijke invulling van *Kandidaten*, en 28(b) illustreert hoe deze wordt uitgebreid volgens bovenstaande aanpak. Stel nu dat we een tupel $(Gert, Hanne)$, met feature-vector $(0, 1, 0)$, willen classificeren. We kijken simpelweg naar de kans bij het overeenkomstige fictieve kandidaat-tupel *Fictief_4*, zijnde 0.88.

Merk op dat de domeinkennis geen impact heeft op deze fictieve kandidaat-tupels. Formules die deel uitmaken van de domeinkennis bestaan steeds uit minstens twee predicaten, met minstens één gelijknamige variabele. Anders zouden deze formules namelijk geen relaties kunnen uitdrukken tussen de te classificeren objecten. Aangezien een fictief kandidaat-tupel geen toekenningen voorziet van waarden aan de variabelen in het predicaat, kan het dus geen deel uitmaken van de groundings van zulke formules.

De mogelijkheid om domeinkennis op te nemen in het model, aan weight tying te doen, en om UDFs te integreren, maken de DeepDive methodologie een zeer krachtige en gebruiksvriendelijke aanpak voor Relation Extraction.



(a)



(b)

	truth-values		impliciet		expliciet	
wereld	Getrouwd(A,B)	Getrouwd(C,D)	$\widetilde{P}(w_i)$	$P(w_i)$	$\widetilde{P}(w_i)$	$P(w_i)$
1	1	1	e^(3.0*1)	0.476	e^(3.0*2)	0.476
2	1	0	e^(3.0*1)	0.476	e^(3.0*2)	0.476
3	0	1	e^(3.0*0)	0.024	e^(3.0*1)	0.024
4	0	0	e^(3.0*0)	0.024	e^(3.0*1)	0.024

(c)

Figuur 29: (a) Geground MLN bij impliciete implicatie. (b) Geground MLN bij expliciete implicatie. (c) Kansdistributie van de mogelijke werelden voor beide benaderingen.

4.7.4 Expliciete implicaties

Laat ons opnieuw kijken naar de formules die DeepDive opstelt voor de features: per unieke feature f wordt het paar $(\forall p_1\ \forall p_2\ \mathit{Getrouwd}(p1, p2);\ w_f)$ toegevoegd, en de groundings van het predicaat $\mathit{Getrouwd}(p1, p2)$ in de formule met gewicht w_f zijn die kandidaat-tupels die feature f bezitten. Inmiddels weten we uit deel 4.7.2 dat DeepDive daardoor het dot product sommeert van alle tupels die waar zijn in de wereld. De gelijkenis met Logistic Regression vormt dus de motivatie voor het gebruik van die specifieke formule.

Wanneer we echter vanuit de eerste-orde predicatenlogica vertrekken, zou het natuurlijker aanvoelen om volgende formules op te stellen: per unieke feature f voegen we het paar $(\forall p_1\ \forall p_2\ \mathit{hasFeature\_f}(p1, p2) \Rightarrow \mathit{Getrouwd}(p1, p2);\ w_f)$ toe, waarbij de groundings van het predicaat $\mathit{Getrouwd}(p1, p2)$ nu alle mogelijke kandidaat-tupels zijn. Merk op dat $\mathit{hasFeature\_f}(p1, p2)$ gekend is voor ieder kandidaat-tupel (deze info halen we uit de tabel *Kandidaten*); er worden geen (binaire) variabelen gegenereerd voor deze gegronde predicaten.

Hoewel beide benaderingen op het eerste zicht verschillend lijken, zijn ze toch equivalent. We zeggen dat de eerste benadering gebruik maakt van 'impliciete' implicaties en de tweede van 'expliciete'. We werken een voorbeeldje uit om het verband tussen beide beter te begrijpen. Beschouw enerzijds een MLN met een impliciete implicatie $(\forall p_1\ \forall p_2\ \mathit{Getrouwd}(p1, p2);\ 3.0)$, en anderzijds hetzelfde MLN met een expliciete implicatie $(\forall p_1\ \forall p_2\ \mathit{hasFeature\_1}(p1, p2) \Rightarrow \mathit{Getrouwd}(p1, p2);\ 3.0)$. We gaan ervan uit dat er in totaal twee mogelijke groundings zijn, $\mathit{Getrouwd}(A, B)$ en $\mathit{Getrouwd}(C, D)$, en dat de eerste in het bezit is van *feature_1* maar de tweede niet. Figuren 29(a) en 29(b) tonen dan hun geground MLN, en figuur 29(c) toont hun kansdistributies over de mogelijke werelden.

Om in te zien waarom beide benaderingen equivalent zijn, vertrekken we vanuit het MLN met de expliciete implicatie. Merk op dat de tweede gegronde formule steeds naar waar zal evalueren, omdat het antecedent steeds false is. Dit is onafhankelijk van de beschouwde wereld: in iedere mogelijke wereld levert deze grounding eenzelfde bijdrage van $e^{3.0}$ tot de niet-genormaliseerde kans.

Daardoor heeft deze grounding geen impact op de uiteindelijke kansdistributie van de mogelijke werelden. We hebben dit reeds bewezen in deel 4.5.3. Bijgevolg zijn alleen implicaties waarvan het antecedent waar is van belang, wat betekent dat het kandidaat-tupel die feature bezit. In dat geval komt de truth-value van de gegrounde formule (implicatie) overeen met de truth-value van het kandidaat-tupel (het consequent). De benadering waarin het MLN met impliciete implicaties gebruikt wordt, past dit inzicht toe. Het enige essentiële verschil tussen beide benaderingen is dus de computationele zwaarte: de benadering met expliciete implicaties genereert veel meer groundings, waardoor de benadering met impliciete implicaties de voorkeur geniet.

Ten slotte namen we nog de proef op de som voor het eerder vernoemde voorbeeld vanop de website van DeepDive. We wilden dus nagaan of beide benaderingen hier dezelfde resultaten zouden opleveren. Eerst gingen we na welke features de grootste impact hebben op de kansen. We selecteerden de 3 features met het grootste positieve gewicht, en de 3 features met het grootste negatieve gewicht. Deze gebruikten we om beide MLNs te definiëren, telkens bestaande uit 6 formules met hardcoded gewichten. Door de grote computationele kost van de aanpak met expliciete implicaties, moesten we ons beperken tot slechts 25 artikels. Uiteindelijk lieten we DeepDive de kansen berekenen voor de kandidaat-tupels. Beide benaderingen gaven, zoals verwacht, vergelijkbare resultaten. Indien we DeepDive de gewichten van de features zelf lieten leren (i.p.v. hardcoded gewichten te gebruiken), gaf de aanpak met expliciete implicaties echter geen zinvolle resultaten. We zijn er niet in geslaagd hier een verklaring voor te vinden.

We zagen dus twee motivaties voor de manier waarop DeepDive het MLN opstelt: voor de gelijkenis met Logistic Regression, en uit computationele overwegingen.

5 Toepassing: baas-bedrijf relatie voor Limburg

In dit deel werken we een concrete toepassing uit volgens de DeepDive methodologie. Het doel is om, gegeven een verzameling nieuwsartikels over Limburgse bedrijven, er de relatie baas-bedrijf uit te extraheren. We verwijzen alvast naar figuur 39, waarin enkele tupels uit het eindresultaat getoond worden. We lichten de verschillende fases toe die we tijdens de ontwikkeling hebben doorlopen, en leggen vanuit die context ook de voornaamste implementatie-gerelateerde aspecten uit. We concluderen met de inzichten die we hebben verworven dankzij het realiseren van deze toepassing.

5.1 Motivering en relevantie

De keuze om op één provincie te focussen, is simpelweg een computationele overweging. Wanneer we immers op grotere schaal zouden werken (bv. alle Belgische bedrijven), zouden we met veel grotere hoeveelheden data te maken krijgen, zodanig groot dat een gemiddelde PC niet meer volstaat. De specifieke keuze voor de provincie Limburg was een persoonlijke overweging.

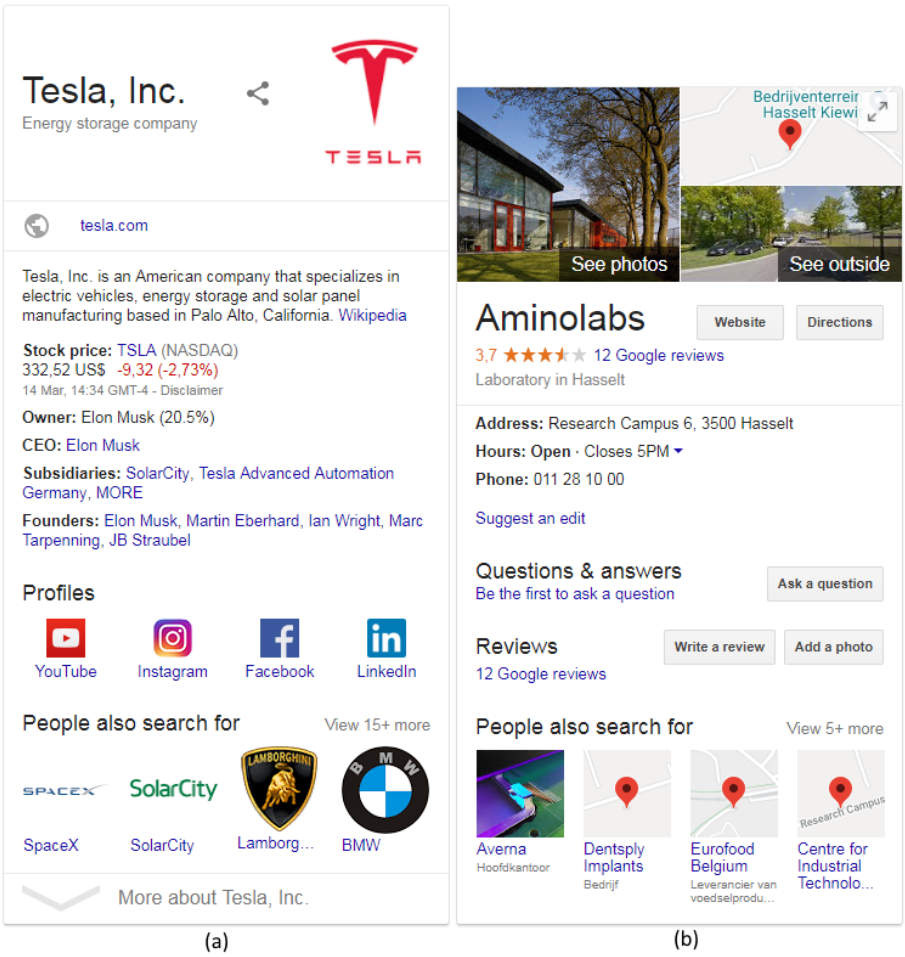
Het opstellen van de baas-bedrijf relatie voor Limburgse bedrijven is relevant om twee redenen. Ten eerste kan deze gestructureerde data een bijdrage leveren tot het Semantic Web. We kunnen deze relatie immers in het RDF data model[19] voorstellen, en vervolgens toevoegen aan een publieke knowledge base zoals Wikidata. Google steunt o.a. op zulke knowledge bases om zijn semantic search te verbeteren, wat zich het duidelijkst uit in Google's 'Knowledge Panel'; figuur 30 geeft twee voorbeelden. Voor een groot bedrijf als Tesla zien we dat de baas (CEO) gekend is, maar voor een relatief klein bedrijf als CLB Group ontbreekt deze informatie. Door het publiceren van onze vergaarde data zouden de Knowledge Panels voor Limburgse bedrijven deze informatie in de toekomst wel kunnen bevatten.

Ten tweede kan de baas-bedrijf relatie dienst doen als (deel van een) dataset voor het realiseren van inzichten, (Big) Data Analytics dus. Een eenvoudig voorbeeld is bepalen welke bedrijven deel uitmaken van een bedrijfsgroep, waarbij het feit dat meerdere bedrijven eenzelfde baas hebben een goede indicatie zou kunnen zijn. Veronderstel dat we ook weten op welk tijdstip iemand de baas werd van een bedrijf. Dan kunnen we kijken naar bedrijven waarvoor meerdere bazen bepaald werden, om zo de overnames die het bedrijf doorheen de tijd onderging in kaart te brengen. Tevens kunnen we dan het onderscheid maken tussen de oprichter en de huidige baas van het bedrijf.

5.2 Eerste fase: snelle ontwikkeling

Wanneer men de DeepDive methodologie hanteert, werkt men in verschillende iteraties. We beginnen met het realiseren van een initiële versie, en gaan deze vervolgens stap voor stap verbeteren om uiteindelijk tot kwalitatieve resultaten te komen. Om het proces dat we hebben doorlopen beknopter te kunnen beschrijven, delen we het op in enkele fases. Eén zo'n fase neemt een aanzienlijke hoeveelheid tijd in beslag, en kan uit meerdere aan elkaar gerelateerde iteraties bestaan; dit zal steeds duidelijk worden toegelicht.

[19] Herinner: dit zijn triples van de vorm *(subject, relation, object)*.



Figuur 30: Het Knowledge Panel dat verschijnt bij het opzoeken van (a) Tesla en (b) CLB Group.

In dit deel lichten we de eerste fase toe. Deze bestaat uit slechts één iteratie, en heeft als doel om zo snel mogelijk een eerste end-to-end implementatie van onze toepassing te realiseren. Er ligt nu nog geen focus op de kwaliteit van de resultaten. De redenering hierachter is dat we alleen zeker kunnen zijn van het effect van een wijziging door naar de outputs te kijken; we willen vermijden dat we ons louter op speculaties baseren en daardoor foutieve beslissingen zouden nemen. We zullen nu de verschillende onderdelen bespreken van de eerste iteratie van onze toepassing.

5.2.1 Nieuwsartikels verzamelen

De hele toepassing staat of valt met het verzamelen van kwalitatieve data. We kunnen zo veel data verzamelen als we willen, maar als deze geen aanwijzingen bevat waaruit het Machine Learning model verbanden kan afleiden tussen een baas en bedrijf, zullen er nooit kwalitatieve resultaten uit voortkomen. Onze keuze valt op de website van 'Made in Limburg' [30], waarop alleen maar nieuwsartikels staan over Limburgse bedrijven en ondernemers. Net omdat deze website zich focust op alles wat met ondernemen te maken heeft, vermoeden we dat hun nieuwsartikels waardevol zijn voor ons Relation Extraction probleem. Het spreekt voor zich dat we dit ook kort manueel hebben gevalideerd.

Made in Limburg biedt een lijst aan met alle (Limburgse) bedrijven waarvoor nieuwsartikels beschikbaar zijn. We schrijven een Python-script om deze lijst op te vragen, en per bedrijf alle beschikbare nieuwsartikels te downloaden. Deze data wordt opgeslagen als een TSV-bestand[20] met twee kolommen: de eerste is het ID van het artikel[21], en de tweede is de eigenlijke tekstuele inhoud van het artikel. We geven het bestand de naam "articles.tsv". Er zijn 4704 bedrijven, goed voor in totaal 10 460 nieuwsartikels. Om de uitvoeringstijd te beperken en het ontwikkelingsproces te versnellen, beschouwen we in deze eerste iteratie alleen de artikels uit de eerste 100 bedrijven, wat 501 nieuwsartikels zijn in totaal.[22]

Herinner dat DeepDive als doel heeft de gebruiker zo sterk mogelijk te ondersteunen in zijn Relation Extraction taak. Daartoe automatiseert DeepDive het werk van de gebruiker zo veel mogelijk. Bijgevolg moeten we ons strikt houden aan de structuur/indeling die DeepDive aan een applicatie oplegt. Tabel 1 geeft een overzicht van de meest relevante delen. Aangezien het bestand "articles.tsv" deel uitmaakt van de input, dienen we het in de folder "input" te plaatsen.

De eerste stap in onze DeepDive toepassing is het opstellen van de relatie *articles*. Deze willen we invullen met de data in het gelijknamige bestand "articles.tsv". Het enige wat we daartoe moeten coderen in "app.ddlog" is:

```
articles(
    doc_id  text,
    content text
).
```

[20] TSV (Tab-Separated Values): waarden van opeenvolgende kolommen worden gescheiden door een tab.

[21] Het ID van een artikel definiëren we als: *<bedrijfsnaam>__<titel van artikel>*.

[22] De totale uitvoeringstijd bedraagt 31 minuten. De NLP preprocessing duurt 20 minuten, features bepalen duurt 3 minuten, kandidaten bepalen en distant supervision duren beide minder dan 1 minuut, en grounding, learning en inference duren tezamen 7 minuten.

Bestand/folder	Beschrijving
app.ddlog	De gebruiker definieert het volledige Relation Extraction proces in dit bestand. Dit gebeurt in de programmeertaal 'DDlog'. DeepDive voert dit bestand uit.
deepdive.conf	In dit bestand gebeurt de configuratie van algemene parameters van DeepDive. Dit omvat enerzijds de parameters voor de Gibbs Sampling (zie deel 5.2.7), en anderzijds voor de evaluatie (zie deel 5.2.8).
input	In deze folder plaatsen we alle data die als input fungeert. Dit zijn uiteraard de nieuwsartikels, maar ook eventuele reeds bestaande databases zouden hieronder komen te staan. Per input dient men een TSV-bestand te voorzien.
udf	In deze folder komen alle User-Defined Functions (UDFs). Dit zijn scriptjes (Shell of Python) waarmee de data in een relatie verwerkt wordt. In deel 5.2.2 geven we hier een eerste voorbeeld van waardoor dit meteen duidelijk wordt.

Tabel 1: Structuur/indeling van een DeepDive applicatie.

Op deze manier definiëren we de relatie *articles*, bestaande uit de kolommen *doc_id* en *content* (met als datatype *text*). Hoewel we als gebruiker geen code schrijven om deze relatie in te vullen met de data in "articles.tsv", doet DeepDive dit voor ons. Bij de uitvoering van "app.ddlog" gaat DeepDive namelijk na of er voor iedere relatie die op deze manier gedefinieerd werd, een gelijknamig bestand aanwezig is in de folder "input" en vult de relatie hiermee in. Indien er geen bestand aanwezig is, genereert DeepDive een foutmelding.

5.2.2 NLP preprocessing

Nu we de nieuwsartikels hebben ingelezen, gaan we aan ieder woord een aantal NLP tags toevoegen. Concreet doen we aan tokenization, lemmatization, Part-of-Speech (POS) tagging, en Named Entity Recognition (NER); deze technieken werden reeds toegelicht in deel 4.2. Dit is louter een voorbereidende stap om het bepalen van de kandidaten en hun features te vergemakkelijken.

Aan de universiteit van Stanford werd een uitstekende NLP toolkit ontwikkeld, CoreNLP [24] genaamd. DeepDive voorziet hier een naadloze integratie voor. Deze toolkit is echter niet geschikt voor onze toepassing, omdat we met Nederlandstalige artikels te maken hebben en CoreNLP deze taal niet ondersteunt. Daarom kozen we uiteindelijk voor Frog [31], een geavanceerde NLP toolkit voor Nederlands, ontwikkeld aan de universiteit van Tilburg en Antwerpen. Frog zelf draait als een server die we kunnen aanspreken via de Frog-client in de Python-library genaamd PyNLPl [32]. Concreet laat de Frog-client toe om eenvoudig een stuk te verwerken tekst naar de Frog-server te versturen; het antwoord bestaat uit het lemma, de POS-tag en de NER-tag van ieder woord

in de tekst.

Het is onze bedoeling om ieder nieuwsartikel op die manier te verwerken, en zodoende de relatie *annotated_articles* op te stellen. Allereerst definiëren we deze relatie in "app.ddlog", als volgt:

```
annotated_articles(
    doc_id          text,
    tokens          json,
    lemmas          json,
    pos_tags        json,
    ner_tags        json
).
```

Deze relatie zal één rij bevatten voor ieder artikel. Het veld *tokens* is de lijst met alle tokens (i.e. woorden en leestekens) in het artikel. De velden *lemmas*, *pos_tags* en *ner_tags* zijn lijsten die voor ieder token het overeenkomstige lemma, POS-tag en NER-tag bevatten.

Om deze relatie op gebruiksvriendelijke wijze in te kunnen vullen, voorziet DDlog de ondersteuning van User-Defined Funcions (UDFs). Een UDF is een Python- of shell-script met als input één rij van een bepaalde relatie, en als output eender welk aantal rijen van een andere relatie. We implementeren een UDF genaamd "nlp_markup" in het bestand "nlp_markup.py"; dit bestand dienen we in de foler "udf" te plaatsen. De input is een rij uit de relatie *articles*, en de output is één rij uit de relatie *annotated_articles*. In dit Python-script gebruiken we de Frog-client om de NLP tags van het artikel te bepalen. Om deze UDF beschikbaar te maken als een functie in "app.ddlog", voegen we volgende voor zich sprekende code toe[23]:

```
function nlp_markup over (
    doc_id  text,
    content text
) returns rows like annotated_articles
implementation "udf/nlp_markup.py" handles tsj lines.
```

Nu zijn we in staat om de relatie *annotated_articles* daadwerkelijk in te vullen. Daartoe voegen we volgende code toe aan "app.ddlog":

```
annotated_articles +=
    nlp_markup(doc_id, content) :- articles(doc_id, content).
```

Voor iedere rij in *articles* wordt dus de UDF "nlp_markup" opgeroepen, en de output wordt toegevoegd aan de relatie *annotated_articles*.

5.2.3 Kandidaten bepalen

Een kandidaat-paar voor de relatie baas-bedrijf bestaat uit een persoon (voorgesteld door zijn naam) en een bedrijf (voorgesteld door de naam). Dankzij de NER-tags voor ieder woord in ieder artikel, kunnen we deze vrij eenvoudig genereren. Frog voorziet namelijk de tag "PER" voor een persoonsnaam en

[23] De code `handles tsj lines` is een implementatiedetail; dit beschrijft dat de database en het Python-script rijen van de input- en output-relatie aan elkaar doorgeven in TSJ-formaat (tab-separated JSON).

"ORG" voor een bedrijfsnaam. Merk op dat een persoons- of bedrijfsnaam uit meerdere opeenvolgende woorden kan bestaan.

Ten eerste stellen we de relatie *person_mentions* op, waarin we alle persoonsnamen zullen plaatsen die in de artikels voorkomen. We voegen volgende code toe aan "app.ddlog" om de relatie te definiëren:

```
person_mentions(
    doc_id          text,
    mention         text,
    begin_index     int,
    end_index       int
).
```

Voor iedere persoonsnaam slaan we op in welk artikel deze voorkwam, wat de eigenlijke persoonsnaam is, en de index van het eerste en laatste woord waaruit de persoonsnaam bestaat.

Aangezien we ons op de NER-tags van woorden baseren om persoonsnamen te identificeren, implementeren we een UDF genaamd "map_person_mentions" in een gelijknamig Python-script. De input is een rij uit *annotated_articles*, en de output zijn nul of meer rijen uit *person_mentions*. Iedere opeenvolging van woorden met "PER" als NER-tag beschouwen we als een persoonsnaam. De definitie van de UDF in "app.ddlog" en de eigenlijke invulling van de relatie op basis van deze UDF, verlopen volledig analoog aan die in deel 5.2.2. Er zijn 2147 vermeldingen van persoonsnamen in onze 501 nieuwsartikels, waarvan 1016 unieke.

Ten tweede gaan we op analoge wijze te werk om de bedrijfsnamen in de artikels te identificeren. Zo definiëren we de relatie *company_mentions*, die we invullen op basis van de UDF genaamd "map_company_mentions". Iedere opeenvolging van woorden met "ORG" of "MISC"[24] als NER-tag, beschouwen we als een bedrijfsnaam. Er zijn 2245 vermeldingen van bedrijfsnamen in onze nieuwsartikels, waarvan 1078 unieke.

Ten slotte bepalen we de kandidaten voor de baas-bedrijf relatie: dit zijn alle unieke combinaties van een persoonsnaam en een bedrijfsnaam die in eenzelfde artikel voorkomen. Om de relatie te definiëren, voegen we volgende code toe aan "app.ddlog":

```
candidates(
    person       text,
    company      text
).
```

Om de relatie vervolgens in te vullen, voegen we het volgende toe:

```
candidates(p_mention, c_mention) *:-
    person_mentions(doc_id, p_mention, _, _),
    company_mentions(doc_id, c_mention, _, _).
```

Met de asterisk duiden we in DDlog aan dat het resultaat geen duplicaten mag bevatten (het equivalent van een "SELECT DISTINCT" in SQL). De underscore wordt gebruikt als placeholder om geen variabele naam te moeten koppelen aan

[24] In de praktijk bleken woorden met "MISC" als NER-tag vaak op een bedrijfsnaam te duiden.

een kolom die toch niet gebruikt wordt om het resultaat in te vullen. Zo bekomen we 6124 kandidaten.

Merk op dat we de relatie *candidates* ook hadden kunnen opstellen door rechtstreeks een UDF toe te passen op *annotated_articles*, dus zonder eerst de relaties *person_mentions* en *company_mentions* op te stellen. In deel 5.2.4 zal echter blijken dat we deze twee relaties sowieso nodig hebben om de features van een kandidaat-paar te bepalen.

5.2.4 Features bepalen

Herinner dat DeepDive zich enerzijds baseert op de features van een kandidaat-tupel en anderzijds op zelf opgestelde formules met domeinkennis om te bepalen of een kandidaat-tupel al dan niet tot de te extraheren relatie behoort. We zullen nu de features bepalen van ieder kandidaat-paar voor onze baas-bedrijf relatie. Deze houden we bij in de relatie *features*, gedefinieerd als:

```
features(
    person      text,
    company     text,
    feature     text
).
```

Een rij in deze relatie beschrijft één feature van een bepaald kandidaat-paar.

Net als in deel 4.3 gebruiken we generieke syntactische en semantische features, gebaseerd op de tekst tussen en rond het kandidaat-paar en de bijhorende NLP tags. DeepDive voorziet hier reeds een Python-library voor, DDlib genaamd [26], die ons van volgende features voorziet:

1. De opeenvolging van woorden tussen het kandidaat-paar.
2. De opeenvolging van lemma's bij de woorden tussen het kandidaat-paar.
3. De opeenvolging van POS tags bij de woorden tussen het kandidaat-paar.
4. De opeenvolging van NER tags bij de woorden tussen het kandidaat-paar.
5. Een tweedelige feature, die (1) de lengte van de persoonsnaam omvat en (2) die van de bedrijfsnaam.
6. Een tweedelige feature, die (1) aangeeft of de persoonsnaam met een hoofdletter begint en (2) hetzelfde voor de bedrijfsnaam.
7. Alle opeenvolgingen van 1 à 3 woorden tussen het kandidaat-paar.
8. Indien de persoonsnaam vóór de bedrijfsnaam staat in de tekst, beschouwen we alle 9 combinaties van 1 à 3 woorden vóór de persoonsnaam en 1 à 3 woorden na de bedrijfsnaam. Analoog indien de bedrijfsnaam vóór de persoonsnaam staat. Voor iedere combinatie zijn er twee features: de opeenvolging van lemma's die bij deze woorden hoort, en hetzelfde voor de bijhorende NER tags.

DDlib genereert standaard ook een aantal features op basis van woorden in een zelf te definiëren woordenboek, en op basis van de grammaticale afhankelijkheden tussen de persoons- en bedrijfsnaam in de tekst (zgn. 'dependency parsing' [25]). We maken hier echter geen gebruik van in onze toepassing, omdat uit het eerder vernoemde voorbeeld rond de relatie $Getrouwd(p1, p2)$ bleek dat deze

niet zinvol waren en we op die manier het aantal features wat beperkter kunnen houden[25].

Om alle features van een kandidaat-paar te bepalen, moeten we alle voorkomens van dat kandidaat-paar in de artikels doorlopen. Vervolgens spreken we DDlib aan om, gegeven het voorkomen en het artikel (met zijn NLP tags), de bijhorende features te genereren. Op die manier vullen we dus de relatie *features* in. Het spreekt voor zich dat we dit via een UDF realiseren; we noemen deze "extract_features". Aangezien de aanroep van deze UDF enigszins verschilt van die in voorgaande delen, is deze wel noemenswaardig:

```
features += extract_features(
    person, p_begin, p_end,
    company, c_begin, c_end,
    tokens, lemmas, pos_tags, ner_tags
) :- candidates(person, company),
     person_mentions(doc_id, person, p_begin, p_end),
     company_mentions(doc_id, company, c_begin, c_end),
     annotated_articles(doc_id, tokens, lemmas, pos_tags, ner_tags).
```

De input van deze UDF is dus een rij uit de equi-join van vier relaties. Alle kandidaten tezamen bezitten 4 187 911 features en er zijn 299 779 unieke features.

5.2.5 Distant supervision

Om een gewicht te kunnen leren voor iedere feature heeft DeepDive nood aan voorbeelden. Dit zijn kandidaat-paren waarvan we weten of ze al dan niet tot de baas-bedrijf relatie behoren. Op basis van die voorbeelden zal DeepDive automatisch de eigenlijke training data (mogelijke werelden) genereren. Om de voorbeelden te verzamelen, passen we distant supervision toe (zie deel 4.4).

We beginnen met het definiëren van de relatie *labels* om de voorbeelden in op te slaan:

```
labels(
    person     text,
    company    text,
    label      int,
    reason     text
).
```

Het veld *label* beschrijft of het kandidaat-paar al dan niet tot de relatie behoort; we spreken van een positief $(+1)$ of negatief (-1) label. In het veld *reason* houden we bij waarom we een bepaald label toekenden aan het kandidaat-paar. Op die manier kunnen we eenvoudiger evalueren in welke mate de verschillende labeling-technieken geschikt zijn, wat relevant is naar volgende iteraties toe. Merk op dat eenzelfde kandidaat-paar meerdere labels kan toegewezen krijgen.

Ten eerste maken we gebruik van een reeds bestaande database, namelijk de "Top 500" van VKW Limburg [33]. Deze bevat de 500 grootste ondernemingen in Limburg, tezamen met één van hun directie- of bestuursleden. Uiteraard

[25] Herinner dat het Markov Network dat de DeepDive methodologie hanteert, lijkt op een Logistic Regression classifier. Een "good practise" binnen Machine Learning bestaat erin het aantal features beperkt te houden wanneer men met relatief eenvoudige modellen werkt; Logistic Regression valt onder die categorie.

hebben we geen rechtstreekse toegang tot deze private database, maar de website toont wel alle resultaten in een HTML-pagina, in een gestructureerde vorm. Met een Python-script kunnen we eenvoudig de nodige informatie extraheren en verwerken, en deze uiteindelijk in het TSV-bestand "top500.tsv" opslaan. We plaatsen dit bestand in de folder "input" en in "app.ddlog" definiëren we de gelijknamige relatie:

```
top500(
    person   text,
    company  text
).
```

Voor ieder kandidaat-paar dat voorkomt in *top500* voegen we een positief label toe, als volgt:

```
labels(person, company, +1, "top 500 vkw limburg") :-
    candidates(person, company),
    top500(person, company).
```

We kunnen de Top 500 ook gebruiken om negatieve voorbeelden te genereren. Indien de persoonsnaam van een kandidaat-paar niet overeenkomt met de persoonsnaam die de Top 500 specificeert voor datzelfde bedrijf, voegen een negatief label toe:

```
labels(person, company, -1, "top 500 vkw limburg") :-
    candidates(person, company),
    top500(boss, company),
    [boss != person].
```

De Top 500 laat ons toe om 5 positieve labels te genereren, en 174 negatieve.

Ten tweede stellen we een aantal heuristische regels op waarmee we labels kunnen toekennen aan een deel van de kandidaten. Dankzij het doornemen van enkele nieuwsartikels en wat algemene domeinkennis, komen we al snel tot de volgende heuristische regels:

1. Als de tekst tussen het kandidaat-paar een positief keyword bevat en de persoons- en bedrijfsnaam kort bij elkaar staan, kennen we een positief label toe. De verzameling positieve keywords bestaat uit: ceo, directeur, zaakvoerder, oprichter, stichter en bestuursvoorzitter.
2. Als de tekst tussen het kandidaat-paar een negatief keyword bevat en de persoons- en bedrijfsnaam kort bij elkaar staan, kennen we een negatief label toe. De verzameling negatieve keywords: gedeputeerde, voorzitter, collega, zoon, dochter, echtgenoot en echtgenote.
3. Als er een persoons- of bedrijfsnaam staat in de tekst tussen het kandidaat-paar, kennen we een negatief label toe.
4. Als de persoons- en bedrijfsnaam ver van elkaar staan, kennen we een negatief label toe.

We implementeren deze heuristische regels in de UDF genaamd "supervise", in het gelijknamig Python-bestand. We passen deze functie toe op ieder voorkomen van ieder kandidaat-paar. De definitie en aanroep van de UDF in "app.ddlog" verlopen analoog aan die in het vorige deel. De heuristische regels

leveren ons respectievelijk 36, 14, 9621 en 4689 labels, wat een totaal van 14 360 labels is. Tezamen labelen ze 5490 kandidaten, wat betekent dat op eenzelfde kandidaat gemiddeld 2 à 3 heuristische regels toegepast konden worden.

Tot slot bepalen we voor ieder kandidaat-paar zijn finale label. Herinner dat aan eenzelfde kandidaat meerdere labels toegewezen kunnen zijn, of zelfs geen enkel label. Aan een kandidaat zonder labels of met even veel positieve als negatieve labels, geven we 'onbepaald' als finale label, met 0 als numerieke waarde. Aan een kandidaat met meer positieve labels dan negatieve, kennen we 'positief' toe als finale label, met een positief getal als numerieke waarde; analoog voor 'negatief'.

We kunnen het finale label van een kandidaat-paar eenvoudig bepalen door de som te nemen over al zijn labels. Om te zorgen dat we ook de kandidaten zonder labels in acht nemen, voegen we eerst alle kandidaten toe aan *labels* met label 0 en zonder *reason*:

```
labels(person, company, 0, NULL) :- candidates(person, company).
```

Vervolgens nemen we per kandidaat de som over zijn labels en stellen zodoende de relatie *resolved_labels* op:

```
resolved_labels(person, company, SUM(label)) :-
    labels(person, company, label, _).
```

We benadrukken dat *resolved_labels* alle kandidaten bevat, telkens met hun label voorgesteld door een numerieke waarde. Maar liefst 5509 van de 6124 kandidaten hebben een finaal label verschillend van 0, en kunnen dus als training data gebruikt worden.

5.2.6 MLN definiëren

Vooraleer we het eigenlijke MLN gaan definiëren, moeten we expliciet beschrijven dat alle kandidaten deel uitmaken van het MLN. Daartoe definiëren we alvast de 'variabele relatie' genaamd *person_company*:

```
person_company?(
    person   text,
    company  text
).
```

Om het verschil tussen een gewone relatie en een variabele relatie aan te duiden, moeten we in DDlog een vraagteken toevoegen achter de naam van de relatie; dit geldt alleen voor het definiëren van de relatie. Iedere rij in een variabele relatie komt overeen met een (booleaanse) variabele in het bijhorende Markov Network. Dankzij deze relatie weet DeepDive dus voor welke variabelen/kandidaten berekend moet worden wat de (marginale) kans is dat ze tot de baas-bedrijf relatie horen.

Daarnaast moeten we ook de training data koppelen aan deze relatie, op basis van de relatie *resolved_labels* (uit de vorige stap). Aan de kandidaten met een positief finaal label dienen we de (booleaanse) waarde *true* te geven, *false* aan die met een negatief finaal label, en *null* aan die met 'onbepaald' als finaal label. Om alle kandidaten in *person_company* op te nemen, telkens met hun gepaste booleaanse waarde, doen we het volgende:

```
person_company(person, company) = if l > 0 then TRUE
                                  else if l < 0 then FALSE
                                  else NULL end :-
    resolved_labels(person, company, l).
```

Dankzij deze eerste stap in het opstellen van het MLN, weet DeepDive (1) welke variabelen deel uitmaken van het Markov Network en (2) welke training data voorhanden is.

Nu kunnen we het eigenlijke MLN definiëren, bestaande uit een verzameling formules met gewichten. Herinner uit deel 4.7.3 dat het MLN in principe één formule bevat per unieke feature, maar dat DDlog toelaat aan weight tying te doen om dit samen te vatten tot één formule. In ons geval komt dit neer op:

```
@weight(f)
person_company(person, company) :- features(person, company, f).
```

Naast de formules op basis van de features, kunnen we ook nog zelf opgestelde formules toevoegen, die extra domeinkennis omvatten. Met deze formules beschrijven we relaties tussen de variabelen/kandidaten onderling. In het geval van onze baas-bedrijf relatie zijn de volgende waarschijnlijk zinvol:

1. Een bedrijf heeft slechts één baas:

   ```
   @weight(-1.0)
   person_company(p1, c) => person_company(p2, c) :- TRUE.
   ```

2. Een persoon is de baas van slechts één bedrijf:

   ```
   @weight(-1.0)
   person_company(p, c1) => person_company(p, c2) :- TRUE.
   ```

De specifieke waarden voor de gewichten van deze formules hebben we intuïtief gekozen. Vermits de formules vaak, maar niet altijd, waar zullen zijn, kozen we relatief kleine gewichten. In de volgende fase zullen we op zoek gaan naar meer geschikte waarden.

5.2.7 Learning en inference

DeepDive zal, gegeven het MLN en de training data, de learning en inference volledig automatisch realiseren. We hoeven zelf alleen nog maar te zorgen voor de configuratie van een aantal parameters die hier verband mee houden. Dit gebeurt in het bestand "deepdive.conf". We plaatsen er de volgende regel:

```
deepdive.sampler.sampler_args:
    "-l 1000 -i 1000 -a 0.01 -d 1.00 --sample_evidence"
```

De optie `-l 1000` duidt aan dat gradient ascent tijdens het leren van de gewichten van de features, 1000 iteraties zal doorlopen; `-i 1000` dat we ons bij de Gibbs Sampling baseren op 1000 samples om aan inference te doen; `-a 0.01` dat de learning rate α, die gebruikt wordt bij het updaten van de gewichten, op 0.01 is ingesteld; `-d 1.00` dat de learning rate bij iedere iteratie ongewijzigd blijft (deze parameter heet de 'diminish rate'); en `--sample_evidence` dat DeepDive ook voor de reeds gelabelde variabelen/kandidaten (i.e. die met de waarde true of false) de marginale kans moet berekenen. De L2-regularisatie

parameter λ die gebruikt wordt bij het berekenen van de likelihood, laten we op zijn default waarde van 0.01 staan; voor de betekenis en motivatie achter L2-regularisatie verwijzen we naar Russell en Norvig [19]. Voor een overzicht van alle configureerbare parameters verwijzen we naar de website van DeepDive [29].

Om DeepDive het volledige relation extraction proces te laten doorlopen, voeren we het commando `deepdive run` uit in de installatie-omgeving. Naar volgende iteraties toe is het nuttig om in grote lijnen te begrijpen wat DeepDive dan precies doet. Zo wordt onze toepassing allereerst gecompileerd, waarbij allerlei syntactische en semantische checks gebeuren. Ook wordt een 'execution plan' opgesteld. Dit beschrijft de verschillende te doorlopen stappen, rekening houdend met onderlinge afhankelijkheden (bv. de relatie *annotated_articles* kan pas opgesteld worden van zodra *articles* werd ingevuld). De volgorde van de verschillende stappen is grofweg [29]:

1. Het invullen van alle zelf gedefinieerde relaties. Er is één stap per relatie, met als naam simpelweg de naam van de betreffende relatie.
2. Het MLN herleiden tot het overeenkomstige Markov Network. Deze stap heeft de naam "grounding".
3. De learning stap, "learning" genaamd. DeepDive definieert de relatie *dd_inference_result_weights_mapping* en plaatst er de features met hun geleerde gewicht in.
4. De inference stap, "inference" genaamd. DeepDive definieert de relatie *person_company_inference* en plaatst er de kandidaten met hun marginale kans in.
5. Het genereren van een plot voor de evaluatie (zie deel 5.2.8). Deze stap heet "calibration".

Voor een volledige beschrijving van het opstellen van het execution plan, verwijzen we naar de website van DeepDive [29].

De compilatie resulteert in een Shell-script voor iedere stap, dat al het nodige werk van die stap realiseert (bv. het definiëren van een relatie en deze invullen op basis van een TSV-bestand). Tot slot voert DeepDive al deze Shell-scripts uit, in de gepaste volgorde, zodat alle resultaten voorhanden zijn. De evaluatie van de resultaten bespreken we in het volgende deel.

5.2.8 Evaluatie

Algemeen:
Gegeven de marginale kans p dat een kandidaat-paar tot de baas-bedrijf relatie behoort, moeten we beslissen of het er nu wel of niet toe behoort. We kunnen simpelweg $p = 0.5$ als drempelwaarde kiezen: als $p \geq 0.5$ behoort het kandidaat-paar wel tot de relatie, anders niet. Aan het einde van deel 4.6 argumenteerden we reeds kort waarom de drempelwaarde beter hoger zou zijn (bv. 0.9); hier komen we zo dadelijk op terug.

Wanneer men een Machine Learning gebaseerde aanpak hanteert voor een bepaald probleem, is het cruciaal om het geleerde model te evalueren volgens een onafhankelijke test. Hierbij wordt het getrainde model geëvalueerd op basis van niet eerder geziene data. Een veelvoorkomend probleem is namelijk 'overfitting', waarbij het Machine Learning model de training data uitstekend weet

te classificeren maar het niet in staat is om niet eerder geziene data correct te classificeren. In dit geval heeft het model eigenlijk niets geleerd; het heeft gewoon gememoriseerd hoe het de training data moet classificeren.[26] Om een onafhankelijke test mogelijk te maken, deelt men de collectie voorbeelden op in twee disjuncte delen: de 'training data' en 'test data', typisch bestaande uit respectievelijk 75% en 25% van alle voorbeelden. De training data omvat voorbeelden die gebruikt worden tijdens de leerfase (learning). De test data wordt niet gebruikt tijdens de leerfase, waardoor deze uit niet eerder geziene voorbeelden bestaat en bijgevolg geschikt is om de onafhankelijke test mee uit te voeren.

Ook DeepDive ondersteunt het onderscheid tussen training data en test data. Concreet moeten we configureren hoeveel procent van de voorbeelden wordt "weggenomen", om als test data dienst te doen; DeepDive spreekt van 'holdout data'. Om 25% van de voorbeelden als test data te gebruiken, voegen we de volgende regel toe in "deepdive.conf":

```
deepdive.calibration.holdout_fraction: 0.25
```

Herinner hoe we in deel 5.2.6 de waarde *true* of *false* toekenden aan de gepaste kandidaten (op basis van hun label in *resolved_labels*). We zeiden toen dat *alle* gelabelde kandidaten gebruikt zouden worden om de training mee te realiseren. Door het toevoegen van bovenstaande regel aan "deepdive.conf" wijzigen we dit enigszins. DeepDive selecteert namelijk op willekeurige wijze ongeveer 25% van de gelabelde kandidaten, om de collectie test data mee op te stellen.[27]. Tijdens de leerfase negeert DeepDive de waarde van die kandidaten, waardoor alleen de overige 75% van de gelabelde kandidaten gebruikt wordt.

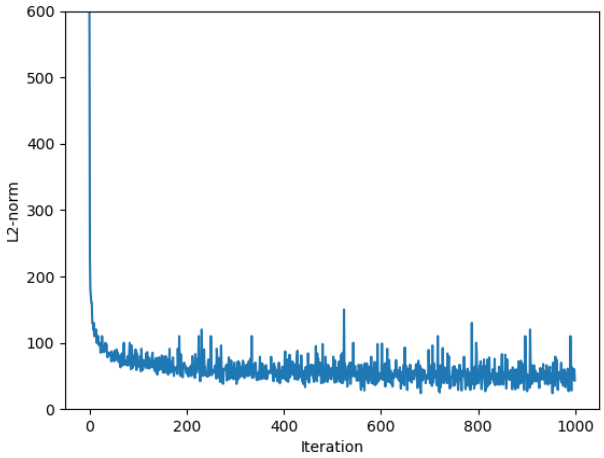
Met deze informatie in het achterhoofd, zullen we nu een aantal mogelijkheden bespreken om onze toepassing te evalueren. Deze gebruiken we na iedere iteratie om te beslissen of een volgende iteratie aangewezen is.

Leercurve:
Een eerste manier om de kwaliteit van onze toepassing in te schatten, is het visualiseren van de 'leercurve'. Deze beschrijft de log-likelihood van de training data i.f.v. iedere iteratie die het gradient ascent algoritme doorloopt. Hoewel dit een frequent gebruikt hulpmiddel is in de Machine Learning wereld, biedt DeepDive hier geen directe ondersteuning voor. Wel is het mogelijk om per iteratie op te vragen hoe sterk de gewichten aangepast werden t.o.v. de voorgaande iteratie. De grootte van de wijziging wordt beschreven door de L2-norm van het verschil tussen de gewichten in de huidige en de voorgaande iteratie.

Uiteraard is er een verband tussen de log-likelihood en de grootte van de wijziging van de gewichten: hoe groter de likelihood wordt, hoe kleiner de aanpassing van de gewichten zal zijn. De gradiënt van de log-likelihood beschrijft het formele verband tussen beide. Idealiter zien we de grootte van de wijziging afnemen naarmate meer iteraties doorlopen worden, om uiteindelijk te convergeren naar een zeer kleine wijziging. In dat geval is het gradient ascent algoritme immers geconvergeerd naar een lokaal (of hopelijk het globale) maximum. Figuur 31 toont de leercurve voor onze eerste versie. De grootte van de wijziging

[26] Voor meer informatie omtrent overfitting en technieken om dit te beperken, verwijzen we naar Russell en Norvig [19].

[27] DeepDive genereert automatisch de relatie *dd_graph_variables_holdout* om de test data in te stockeren.



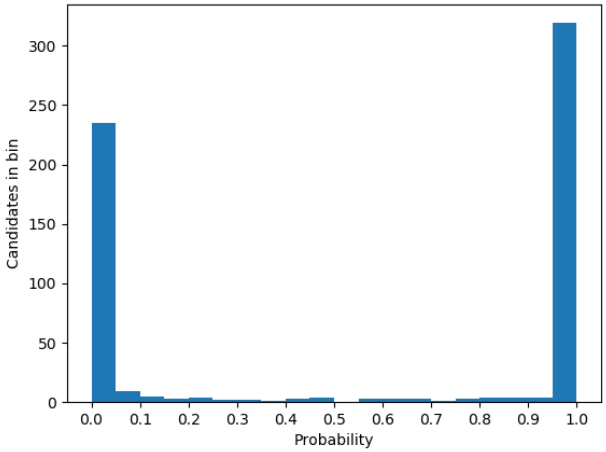
Figuur 31: De leercurve voor de eerste versie van onze toepassing.

van de gewichten neemt inderdaad af, vooral in de eerste iteraties, en convergeert uiteindelijk. Hierdoor vermoeden we dat het model de training data goed zal weten te classificeren.

Om deze plot te genereren, hebben we een Python-script geschreven dat gebruikmaakt van Psycopg [34] voor de communicatie met de PostgreSQL database en PyPlot [35] voor de eigenlijke plot.

Kansverdeling:
Ons tweede hulpmiddel bestaat erin de marginale kansen van de kandidaten voor te stellen in een histogram. Dit doen we voor de training data, de test data, en alle data (inclusief niet-gelabelde kandidaten). Ieder histogram bestaat uit 20 bins, overeenkomstig met opeenvolgende kansgebieden van 0.05 breed, en één bin bevat alle kandidaten met een marginale kans in dit kansgebied. Figuur 32 geeft een hypothetisch voorbeeld: de eerste bin toont dat er 235 kandidaten zijn met een marginale kans tussen 0.00 (inclusief) en 0.05 (exclusief), de tweede bin toont dat er 9 kandidaten zijn met een marginale kans tussen 0.05 (inclusief) en 0.10 (exclusief), enzovoort.

Uit de vorm van het histogram kunnen we afleiden hoe (on)zeker DeepDive is over zijn classificaties. Uiteraard zouden we willen dat DeepDive steeds zeer zeker is, wat zich vertaalt in het associëren van een zeer lage kans (bv. < 0.05) of een zeer hoge (bv. ≥ 0.95) aan alle kandidaten. Voor zulke kandidaten kunnen we dus met grote zekerheid zeggen of ze al dan niet tot de baas-bedrijf relatie behoren. Omgekeerd duidt een kans tussen pakweg 0.25 en 0.75 op een grote onzekerheid; we zijn niet zeker of onze classificatie wel degelijk correct is. Wanneer we deze redenering vertalen naar het histogram van een kansdistributie, willen we veel kandidaten zien in de eerste en laatste bins en zeer weinig in de tussenliggende bins. Idealiter heeft het histogram dus een hoekige U-vorm. In ons hypothetisch voorbeeld (figuur 32) was dat het geval. Figuur 33 toont het histogram voor (a) de training data, (b) de test data en (c) alle data (inclusief niet-gelabelde kandidaten) voor de eerste versie van onze toepassing. Doordat



Figuur 32: Visualisatie van een kansverdeling d.m.v. een histogram.

we bij de distant supervision zeer weinig positieve voorbeelden hebben gegenereerd (zie deel 5.2.5), is het niet verwonderlijk dat de laatste bin van figuur 33(a) amper kandidaten bevat; het zijn er 9 om precies te zijn. In de volgende iteratie zullen we het aantal positieve en negatieve voorbeelden beter moeten balanceren. Desondanks er slechts 36 positieve labels waren over alle voorbeelden, bevat de laatste bin van figuur 33(b) maar liefst 93 kandidaten. Bijgevolg zullen er zeer veel false positives zijn, wat ons laat vermoeden dat de precision zeer laag zal zijn.

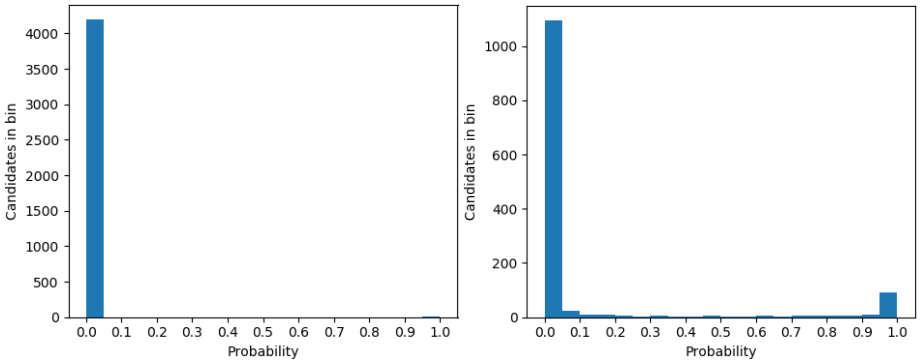
Ook om deze plots te genereren, hebben we een Python-script geschreven dat gebruikmaakt van Psycopg [34] en PyPlot [35].[28]

Precision en recall:
Op basis van de drempelwaarde hebben we voor ieder kandidaat-paar een *voorspelling* over het feit of het wel of niet tot de baas-bedrijf relatie behoort. Verder weten we van alle gelabelde kandidaten (i.e. de training en test data) of ze *werkelijk* wel of niet tot de relatie behoren. De voornaamste manier om de kwaliteit van onze toepassing te evalueren, bestaat erin de voorspelde klasse van ieder kandidaat-paar te vergelijken met zijn werkelijke klasse. We maken hierbij steeds het onderscheid tussen de training en test data.

Twee veelvuldig gebruikte metrieken voor het evalueren van een binaire classificatie-taak zijn 'precision' en 'recall' [19]. Beschouw alvast tabel 2 met de definitie van een 'true positive', 'false positive', 'true negative' of 'false negative' kandidaat-paar. Het aantal kandidaat-paren in iedere categorie noteren we respectievelijk als TP, FP, TN en FN. De precision en recall zijn vervolgens gedefinieerd als:

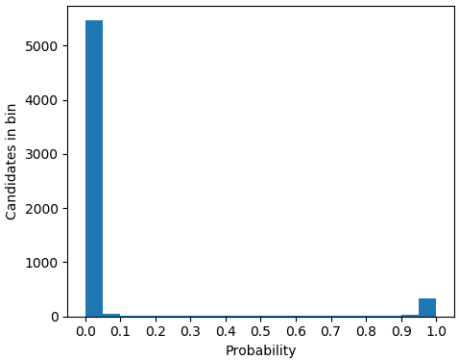
$$\textit{precision} = \frac{TP}{TP + FP} \text{ en } \textit{recall} = \frac{TP}{TP + FN}$$

[28] DeepDive genereert automatisch de relatie *person_company_inference* om de marginale kans voor ieder kandidaat-paar bij te houden.



(a)

(b)



(c)

Figuur 33: Histogram van de kansverdeling van (a) de training data, (b) de test data en (c) alle data, voor de eerste versie van onze toepassing.

	Voorspelling: wel	Voorspelling: niet
Werkelijk: wel	true positive	false negative
Werkelijk: niet	false positive	true negative

Tabel 2: Definitie van 'true positive', 'false positive', 'true negative' en 'false negative'.

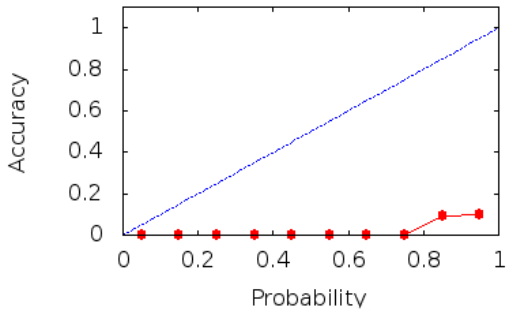
De precision beschrijft hoeveel procent van de kandidaat-paren waarvoor we voorspeld hebben dat ze tot de relatie behoren, er ook werkelijk tot behoort. De recall beschrijft voor hoeveel procent van de kandidaat-paren die werkelijk tot de relatie behoren, we ook voorspeld hebben dat ze tot de relatie behoren. Het spreekt voor zich dat een verhoging van de drempelwaarde typisch overeenkomt met een verhoogde precision, maar een verlaagde recall; en omgekeerd voor een verlaging van de drempelwaarde.

Herinner de motivatie achter het opstellen van de baas-bedrijf relatie voor Limburgse bedrijven: we zouden deze data kunnen publiceren in de context van het Semantic Web of van Data Analytics. Bijgevolg is het belangrijk om alleen kandidaten in de relatie te plaatsen die er wel degelijk in thuis horen (hoge precision). We hechten dus meer belang aan de precision dan aan de recall. Aan het einde van deel 4.6 stelden we daarom voor om alleen kandidaten wiens kans minstens 0.9 bedraagt, aan het uiteindelijke resultaat toe te voegen.

Om een geschikte drempelwaarde te bepalen, kunnen we ons baseren op het eerder opgestelde histogram van de kansverdeling. Indien we zien dat DeepDive zeer zeker is over de classificaties, kunnen we de drempelwaarde best plaatsen net vóór de sterke toename in het aantal kandidaten per bin (bv. bij 0.85, 0.90 of 0.95). Op die manier hebben we een hoge precision en houden we tegelijkertijd ook de recall hoog. Indien DeepDive echter een vrij grote onzekerheid vertoont, zijn we genoodzaakt de drempelwaarde extra hoog te plaatsen (bv. minstens 0.95) om alsnog een hoge precision te garanderen; dit zal een lagere recall met zich meebrengen. In ons geval (zie figuur 33(a)) lijkt een drempelwaarde van 0.95 geschikt. De precision en recall zijn dan respectievelijk 0.02 en 0.67 voor de test data. Dit zijn duidelijk extreem slechte resultaten die erop wijzen dat DeepDive helemaal niets geleerd heeft. De precision en recall voor de training data zijn respectievelijk 1.0 en 1.0, wat bevestigt dat DeepDive alleen gememoriseerd heeft hoe het de training data moet classificeren.

Voor het berekenen van de precision en recall hebben we opnieuw een Python-script geschreven dat gebruikmaakt van Psycopg [34].

DeepDive's ondersteuning:
Tenslotte bespreken we in welke (beperkte) mate DeepDive ondersteuning biedt voor de evaluatie van een toepassing [29] – dit was tevens de motivatie voor het realiseren van de hierboven genoemde Python-scripts. Ten eerste genereert DeepDive automatisch een histogram van de kansverdeling voor de training data, en voor alle data. Zoals ondertussen duidelijk is, hebben we dit idee overgenomen omdat we het een zinvol hulpmiddel vinden. De implementatie echter, hebben we niet overgenomen. DeepDive's implementatie laat namelijk niet eenvoudig toe de breedte van de bins aan te passen; deze staat standaard op 0.1 terwijl 0.05 ons meer geschikt lijkt. Daarenboven bevat het histogram voor de



Figuur 34: DeepDive's accuracy plot voor de eerste versie van onze toepassing.

training data eigenlijk niet alleen de training data maar ook de test data, wat volgens de documentatie niet de bedoeling is en dus een foutje is in de implementatie. Daardoor ontbreekt zowel het histogram voor de training data als dat voor de test data. Bijgevolg waren we genoodzaakt zelf een implementatie te voorzien.

Ten tweede stelt DeepDive automatisch een 'accuracy' plot op, gebaseerd op de indeling in bins. De accuracy van een bin is gedefinieerd als de fractie kandidaat-paren in die bin die *true* heeft als werkelijk label. Figuur 34 toont die voor onze toepassing. DeepDive stelt dat de accuracy plot dicht bij de "ideale lijn" moet liggen; dit is de rechte door de punten $(0.0, 0.0)$ en $(1.0, 1.0)$. Aangezien deze uitspraak, naar onze mening, niet volledig correct is, kozen we ervoor om geen gebruik te maken van de accuracy plot. Zoals reeds geargumenteerd, is het wenselijk dat er zeer weinig kandidaat-paren zitten in de bins waar DeepDive onzeker over is (bv. die geassocieerd aan de kansgebieden tussen 0.25 en 0.75). Stel dat er in één zo'n bin slechts één kandidaat-paar aanwezig is, dan is de accuracy voor die bin gelijk aan 0.0 of 1.0. Bijgevolg vertoont de accuracy plot daar een piek of dal, wat een grote afwijking is t.o.v. de ideale lijn. Dus, hoewel er slechts één kandidaat-paar in zo'n bin aanwezig is, wat zeer goed is, lijkt het toch alsof de toepassing het slecht doet. Merk op dat indien er vele kandidaat-paren in zulke bins zitten, hun accuracies inderdaad liefst de ideale lijn volgen. Bijgevolg faalt de accuracy plot ook in de omgekeerde richting: indien er vele kandidaat-paren in zulke bins zitten, wat slecht is, maar hun accuracies de ideale lijn volgen, lijkt het toch alsof onze toepassing het goed doet. De precision daarentegen heeft nooit last van een te klein aantal kandidaat-paren, omdat de recall deze steeds aanvult: kiezen we de drempelwaarde zodanig groot dat er zich slechts één kandidaat-paar boven de treshold bevindt, dan zal de bijzonder lage recall ons dit duidelijk maken. De uitspraak rond de ideale lijn achten we alleen correct voor de eerste en laatste bins, omdat deze typisch wel vele kandidaat-paren bevatten. Dus, we gebruiken de precision en recall i.p.v. de accuracy plot, en aangezien DeepDive deze metrieken niet ondersteunt, hebben we ze zelf geïmplementeerd.

5.3 Tweede fase: verbeteringen

We hebben vastgesteld dat de eerste versie van onze toepassing geen kwalitatieve resultaten behaalt. De focus lag dan ook louter op een snelle ontwikkeling. Nu we een eerste end-to-end versie ter beschikking hebben, kunnen we overgaan

tot het analyseren en verbeteren van ieder onderdeel. De focus verschuift dus naar de kwaliteit van de resultaten. Er zijn meerdere iteraties nodig om de vele verbeteringen te realiseren en volledig op punt te stellen, zelfs voor eenzelfde onderdeel. Om de bespreking beknopter te kunnen houden, bundelen we al deze iteraties en spreken we van 'de tweede fase' en 'de tweede versie' van onze toepassing. We zullen de uiteindelijke doorgevoerde verbeteringen bespreken per onderdeel van de toepassing, en we eindigen met de evaluatie van deze tweede versie.

5.3.1 Distant supervision

Opdat een Machine Learning model goede resultaten kan behalen voor een bepaald classificatie-probleem, moet het kwalitatieve features ter beschikking hebben (en hier uiteindelijk zinvolle gewichten voor leren). Zoals de precision en recall voor de test data illustreerden (resp. 0.02 en 0.67), zijn we hier in de eerste versie duidelijk niet in geslaagd. Om de resultaten van onze toepassing drastisch te verbeteren, moeten we dus voor kwalitatieve features zorgen. Daarom gaan we alleen met nieuwsartikels werken waarvoor we zeker weten dat ze kwalitatieve features bevatten.

Om kwalitatieve nieuwsartikels te identificeren, steunen we op de heuristische regels die distant supervision gebruikt. Veronderstel namelijk dat we erin slagen om zeer kwalitatieve heuristische regels te formuleren, die ons toelaten om met vrij grote zekerheid een label toe te kennen aan een kandidaat-paar (high-precision en low-recall). Het lijkt ons logisch dat nieuwsartikels waarop zo'n heuristische regel kon worden toegepast, gegarandeerd kwalitatieve features zullen bevatten. Deze nieuwsartikels beschouwen we dus als zijnde 'kwalitatief', en de features hierin zijn de enige die gebruikt zullen worden bij de training (zie deel 5.3.4).

Herinner de heuristische regels die we in de eerste versie van onze toepassing gebruikten (zie deel 5.2.5). Om na te gaan of een bepaalde regel wel degelijk kwalitatief is, bekijken we enkele artikels over kandidaten waarop de regel toegepast werd. We vergelijken het toegekende label met het gepaste label. Zoals verwacht blijken de eerste twee regels wel degelijk kwalitatief.

De laatste regel blijkt helemaal niet kwalitatief. Er zijn vele nieuwsartikels waar een bepaalde baas en bedrijfsnaam vermeld worden in één van de eerste alinea's en in een latere alinea. Beschouw het voorkomen bestaande uit de persoonsnaam in de eerste alinea en de bedrijfsnaam in de latere alinea. Doordat ze ver van elkaar staan, kennen we er ten onrechte een negatief label aan toe. Aangezien dit een frequent voorkomend scenario is, besluiten we de regel volledig te elimineren.

Doordat de derde regel geen rekening houdt met het aantal woorden tussen de persoons- en bedrijfsnaam, is ook deze regel niet kwalitatief. Wanneer ze namelijk ver van elkaar staan en er een derde persoons- of bedrijfsnaam tussen hen staat, is het mogelijk dat deze bij een vierde persoons- of bedrijfsnaam hoort. In dit geval bevat de tekst tussen beide geen kwalitatieve features. Daarom passen we de regel aan, zodat deze alleen toegepast wordt indien de persoons- en bedrijfsnaam dicht bij elkaar staan. Wanneer we nu opnieuw enkele artikels bekijken over kandidaten waarop deze regel werd toegepast, achten we de regel nog steeds niet kwalitatief genoeg. Het negatief label wordt namelijk regelmatig onterecht toegekend, om diverse redenen: er werd ten onrechte een derde

persoons- of bedrijfsnaam geïdentificeerd door Frog, de derde naam verschilt van de persoon of het bedrijf maar verwijst er toch naar (bv. de voornaam wordt weggelaten), de persoon is de baas van meerdere bedrijven, ... Daarom besluiten we om deze regel alsnog te elimineren.

Om te markeren welke nieuwsartikels kwalitatief zijn, houden we voor iedere labeling bij op welk nieuwsartikel we ons baseerden. Daartoe voegen we aan de relatie *labels* het attribuut 'doc_id' toe; dit vereist enkele voor zich sprekende aanpassingen in "app.ddlog". De aanpassingen aan de heuristische regels voeren we door in de UDF "supervise" (in "supervise.py"). De twee overgebleven heuristische regels genereren uiteraard nog steeds respectievelijk 36 en 14 labels, en labelen in totaal 44 verschillende kandidaten.

Laat ons even verder ingaan op het gebruik van de Top 500 om positieve en negatieve labels te genereren. Het is aannemelijk te veronderstellen dat de artikels waarin zo'n kandidaat-paar voorkomt, enkele kwalitatieve features bevatten. We zouden er dus voor kunnen kiezen om alle nieuwsartikels waarin zo'n kandidaat-paar voorkomt als kwalitatief te beschouwen, en de features hierin te gebruiken bij de training. Maar i.t.t. de artikels waarop één van de eerste twee regels konden toegepast worden, zijn we niet helemaal zeker dat er kwalitatieve features aanwezig zijn. Bovendien komt een kandidaat-paar typisch in vele nieuwsartikels voor, waardoor we zeer veel features zouden genereren, waarvan slechts een beperkt aantal wel degelijk kwalitatief is. Aangezien het centrale idee in deze fase erin bestaat alleen kwalitatieve features te gebruiken, besluiten we de Top 500 niet te gebruiken; dit vereist enkele triviale aanpassingen aan "app.ddlog". Er zijn nu dus slechts 44 kandidaten gelabeld van de 6124, i.p.v. 5509 tijdens de eerste iteratie, maar ze zijn veel kwalitatiever.

5.3.2 Kandidaten bepalen

Ook een kwalitatieve selectie van kandidaten is van belang. Indien we namelijk kandidaat-paren identificeren die eigenlijk geen kandidaat-paar horen te zijn, zullen hun features zeker niet kwalitatief zijn, wat een negatieve impact zal hebben op de kwaliteit van onze toepassing.

Om te bepalen hoe we ongeldige kandidaat-paren kunnen elimineren, beginnen we met enkele willekeurige rijen in de relatie *person_mentions* te bekijken. Zo merken we dat er vele namen zijn die uit slechts één woord bestaan. We besluiten deze buiten beschouwing te laten omdat ze onvoldoende specifiek zijn; we vinden louter een voor- of achternaam onvoldoende. Daardoor elimineren we 373 van de 1078 unieke voorkomens van persoonsnamen. Tevens merken we enkele zeer lange persoonsnamen op. We besluiten om persoonsnamen bestaande uit meer dan vier woorden, te negeren. Op die manier elimineren we nog eens 5 unieke persoonsnamen die inderdaad allemaal onterecht als persoonsnaam geïdentificeerd werden door Frog.

Voor de relatie *company_mentions* voeren we een gelijkaardige verbetering in. We merken namelijk dat alle bedrijfsnamen bestaande uit meer dan vier woorden, onterecht als bedrijfsnaam geïdentificeerd werden. Op die manier elimineren we 15 van de 1078 unieke voorkomens.

Na beide UDF's te hebben aangepast, zijn er nu 3668 kandidaten i.p.v. 6124. Herinner dat we iedere unieke combinatie van een persoons- en bedrijfsnaam in eenzelfde artikel als kandidaat beschouwen. Door de nieuwsartikels te bekijken van enkele kandidaten, merken we dat onze definitie niet streng genoeg is; deze

laat zeer veel ongeldige kandidaat-paren toe. Een persoonsnaam aan het begin van een artikel kan namelijk gekoppeld worden aan een bedrijfsnaam aan het eind van het artikel. Dit is typisch geen geldig kandidaat-paar. En indien het wel een geldig kandidaat-paar betreft, staat er zodanig veel tekst tussen de persoons- en bedrijfsnaam dat deze relatief weinig kwalitatieve features bevat. Precies om die reden hadden we ook de vierde heuristische regel bij distant supervision geëlimineerd. We besluiten uiteindelijk om het voorkomen van een persoons- en bedrijfsnaam in eenzelfde document pas als kandidaat-paar te beschouwen op voorwaarde dat er maximaal tien woorden tussen hen staan. In "app.ddlog" definiëren we de relatie *candidates* daarom als:

```
candidates(p_mention, c_mention) *:-
    person_mentions(doc_id, p_mention, p_begin, p_end),
    company_mentions(doc_id, c_mention, c_begin, c_end),
    [[p_begin < c_begin, c_begin-p_end <= 10];
     [c_begin < p_begin, p_begin-c_end <= 10]].
```

We houden nu nog 608 kandidaten over, wat betekent dat we nog eens 3060 ongeldige of niet-kwalitatieve kandidaat-paren geëlimineerd hebben.

5.3.3 Features bepalen

Herinner dat we DDlib gebruiken om features te extraheren voor ieder voorkomen van ieder kandidaat-paar. Zoals we in deel 5.3.1 gemotiveerd hebben, willen we alleen de features uit kwalitatieve nieuwsartikels gebruiken tijdens de training. Daarom voegen we aan de relatie *features* het attribuut 'doc_id' toe, om bij te houden in welk nieuwsartikel deze feature voorkomt. Dit vereist enkele triviale aanpassingen in "app.ddlog" en de implementatie van de UDF "extract_features".

Daarnaast kijken we terug naar de features die DDlib genereert op basis een bepaald voorkomen van een kandidaat-paar (zie deel 5.2.4). De features gebaseerd op de eerste 4 puntjes blijken niet echt zinvol. Aangezien de volledige opeenvolging van de woorden tussen het kandidaat-paar gebruikt wordt, is het zeer onwaarschijnlijk om precies diezelfde opeenvolging terug te zien bij een ander kandidaat-paar. Doordat deze features zodanig specifiek zijn, zijn ze eigenlijk waardeloos. We passen de implementatie van DDlib aan om te zorgen dat deze features niet langer gegenereerd worden.

Om dezelfde reden passen we ook puntje 7 aan. We zorgen namelijk dat DDlib alleen opeenvolgingen van 1 à 2 woorden (i.p.v. 1 à 3) als feature genereert. Het is hier minder aannemelijk dat deze aanpassing inderdaad een verbetering met zich meebrengt, maar de reductie in het aantal features zal hier ongetwijfeld een rol in spelen. Analoog passen we puntje 8 aan, waarbij nu alle combinaties van 1 à 2 woorden rond het kandidaat-paar (i.p.v. 1 à 3) als feature geëxtraheerd worden.

Alle kandidaten tezamen bezitten 316 871 features en er zijn 40 415 unieke features.

5.3.4 MLN definiëren

We benadrukken nogmaals dat we alleen features zullen gebruiken uit de kwalitatieve nieuwsartikels, geïdentificeerd door het feit dat er een heuristische regel

op kon worden toegepast. Tevens beperken we de verzameling kandidaten tot die die gelabeld zijn. De niet-gelabelde kandidaten bezitten namelijk slechts enkele van de kwalitatieve features. Hun aanwezigheid zou het voor DeepDive bijgevolg moeilijker maken om geschikte gewichten te leren voor de features. Merk op dat alle kandidaat-paren die we daadwerkelijk willen beschouwen in de relatie *labels* zitten. Om te vermijden dat ook de andere kandidaat-paren mee worden opgenomen in het Markov Network, verwijderen we volgende regel uit "app.ddlog":

```
labels(person, company, 0, NULL) :- candidates(person, company).
```

De relatie *resolved_labels* behouden we voor alle duidelijkheid wel. De kandidaat-paren met 0 als finale label – deze hebben exact even veel positieve als negatieve labels – negeren we ook. Daartoe herdefiniëren we de variabele relatie *person_company* als:

```
person_company(person, company) = if l > 0 then TRUE
                                  else FALSE end :-
    resolved_labels(person, company, l), [l != 0].
```

We beschouwen nu de eigenlijke definitie van het MLN. Om te zorgen dat alleen de features uit kwalitatieve nieuwsartikels gebruikt worden, passen we de formule voor de features aan:

```
person_company(person, company) :-
    labels(person, company, _, _, doc_id),
    features(person, company, f, doc_id).
```

De twee zelf opgestelde formules (met extra domeinkennis), behouden we. Wel gaan we na of we hun gewichten goed hadden gekozen. We kijken naar de grootte van de gewichten van de voornaamste features. Dit zijn de features met de grootste positieve en negatieve gewichten; we raadplegen hier de relatie *dd_inference_result_weights_mapping* voor. Deze gewichten bevinden zich in de orde van $+0.50$ en -0.45. In deel 5.3.6 geven we enkele voorbeelden van de voornaamste features. Aangezien onze formules zeker niet voor alle kandidaten waar zijn, houdt het steek om ze als één van de voorname features te beschouwen. Daarom lijkt het aangewezen om de huidige gewichten van onze regels, zijnde -1.0 voor beide, minder groot te maken. We experimenteren met een aantal kleine aanpassingen en bekijken telkens hun impact op de precision en recall. Uiteindelijk besluiten we om beide gewichten op 0.5 in te stellen.

5.3.5 De nieuwsartikels

In de eerste fase gebruikten we alleen de nieuwsartikels van de eerste 100 bedrijven, wat neerkwam op 501 artikels. Slechts een beperkt aantal zal kwalitatief zijn en dus wel degelijk gebruikt worden; het zijn er 100 om precies te zijn. Het spreekt voor zich dat we zo veel mogelijk kwalitatieve features willen genereren. Om tot meer kwalitatieve features te komen, beschouwen we nu alle nieuwsartikels (van alle bedrijven dus). Daarom plaatsen we in het bestand "articles.tsv" alle 10 460 nieuwsartikels. We hoeven niets te wijzigen aan de definitie van de relatie *articles* in "app.ddlog". Ook aan de NLP preprocessing stap (zie deel

5.2.2) verandert er niets. Doordat we nu alle nieuwsartikels beschouwen, neemt de uitvoeringstijd significant toe.[29]

Door nu alle nieuwsartikels te beschouwen, zijn er 770 kwalitatieve nieuwsartikels (i.e. artikels waarop een heuristische regel kon toegepast worden). De heuristische regels labelen 456 van de 7689 kandidaten, waarvan 113 negatief, 341 positief en 2 onbepaald. Die 454 kandidaten met een positief of negatief label maken deel uit van het Markov Network. Zoals in deel 5.2.8 vermeld, was het wenselijk om het aantal positief en negatief gelabelde kandidaten meer in balans te hebben, wat nu het geval is. De 454 kandidaten bezitten tezamen 47 915 unieke features die in kwalitatieve nieuwsartikels voorkomen.

5.3.6 Evaluatie

Algemeen:
Om het effect van aanpassingen sneller te kunnen evalueren, laat DeepDive toe om een individuele stap (of enkele stappen) van het execution plan te doorlopen. Indien we verplicht zouden zijn om voor iedere iteratie het volledige relation extraction proces opnieuw te doorlopen, zou de ontwikkeling veel meer tijd kosten. Om bijvoorbeeld na te gaan of de aanpassingen rond voorkomens van personen het gewenste effect geven, willen we alleen de relatie *person_mentions* opnieuw invullen. Daartoe voeren we het volgende commando uit in de installatieomgeving:

```
deepdive redo person_mentions
```

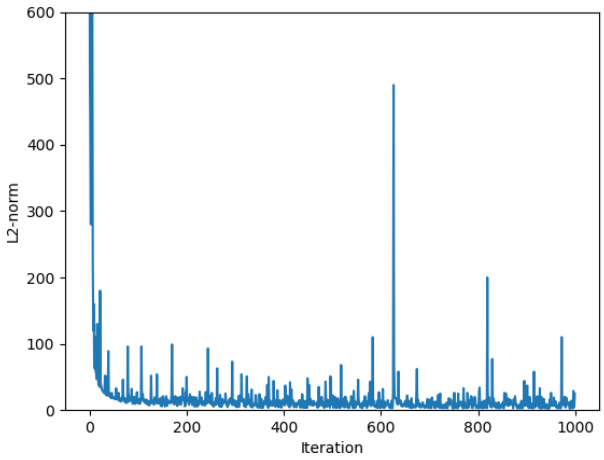
DeepDive hercompileert onze toepassing en voert vervolgens alleen het nodige uit om deze relatie opnieuw in te vullen. Indien DeepDive merkt dat één van de relaties waarop *person_mentions* steunt, ook geherdefinieerd werd, zal DeepDive deze eerst opnieuw invullen.

We benadrukken dat we in deze tweede versie meer kwalitatieve resultaten nastreefden. Daartoe hebben we getracht om meer kwalitatieve kandidaten, features, en heuristische regels te bepalen. Bovendien beschouwden we alleen de kandidaten en features die in kwalitatieve nieuwsartikels voorkwamen. Om na te gaan of de tweede versie van onze toepassing inderdaad betere resultaten genereert, gebruiken we dezelfde evaluatie-technieken als in de eerste fase.

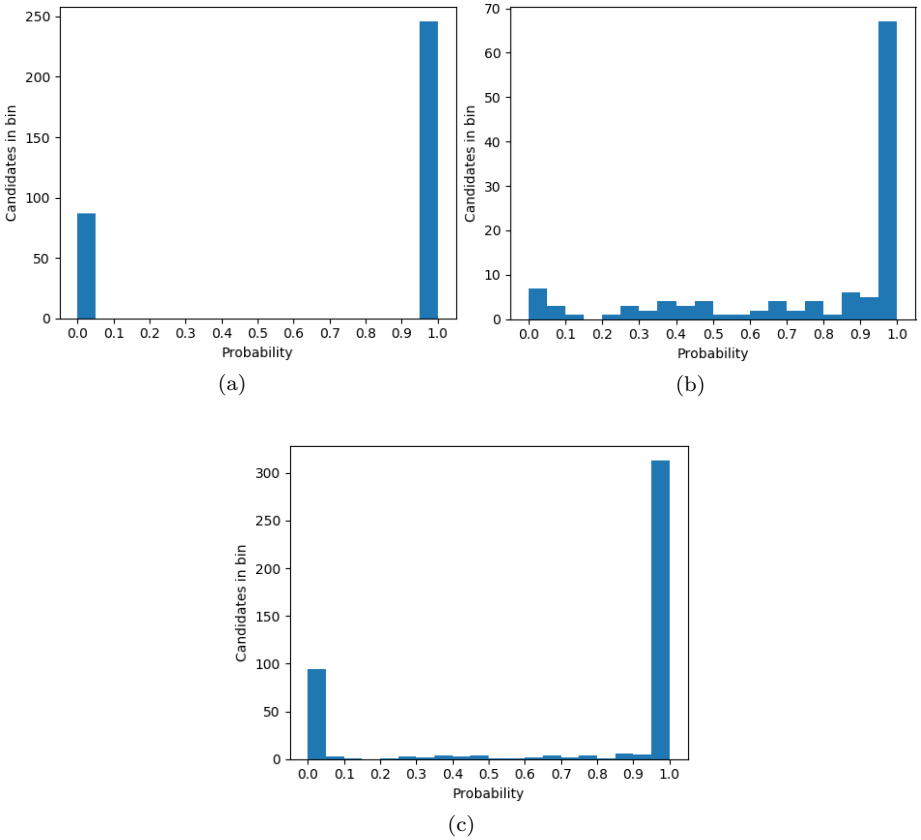
Leercurve, kansverdelingen, precision en recall:
Figuur 35 toont de leercurve. De grootte van de wijziging van de gewichten neemt zeer sterk af in de eerste iteraties, en convergeert al snel naar een zeer kleine aanpassing. Net als voor de eerste versie verwachten we hierdoor dat DeepDive de training data goed zal weten te classificeren.

Figuur 36 toont de histogrammen voor de tweede versie. Het histogram van de kansverdeling van alle data (i.e. de training en test data in dit geval) laat ons vermoeden dat DeepDive er ditmaal wel is in geslaagd om te leren hoe het de kandidaten moet classificeren. We zien namelijk een hoekige U-vorm, maar nu met realistische aantallen kandidaten in de eerste en laatste bin.

[29] De totale uitvoeringstijd bedraagt 3 uur en 32 minuten. De NLP preprocessing duurt 3 uur en 10 minuten, kandidaten bepalen duurt minder dan 1 minuut, features bepalen duurt 13 minuten, distant supervision duurt 5 minuten, en grounding, learning en inference duren tezamen 3 minuten.



Figuur 35: De leercurve voor de tweede versie van onze toepassing.



Figuur 36: Histogram van de kansverdeling van (a) de training data, (b) de test data en (c) alle data, voor de tweede versie van onze toepassing.

Het histogram van de test data heeft een minder hoekige U-vorm dan dat van de training data, wat er op duidt dat DeepDive minder zeker is. De kandidaat-paren zijn meer gelijkmatig verdeeld over de kansgebieden tussen 0.00 en 0.95. Wel is een drempelwaarde van 0.95 duidelijk aangewezen, net zoals bij de eerste versie. De precision en recall voor de training data bedragen respectievelijk 1.0 en 1.0. We zouden hierdoor kunnen vermoeden dat DeepDive opnieuw louter gememoriseerd heeft, maar de precision en recall van de test data spreken dit gelukkig tegen: deze zijn respectievelijk 0.94 en 0.66. We concluderen dat onze tweede versie wel degelijk kwalitatieve resultaten behaalt.

Voornaamste features:
Aangezien DeepDive zeer goede resultaten behaalt, verwachten we dat de voornaamste features die DeepDive geleerd heeft ook intuïtief relevant zijn. De features met de grootste positieve gewichten bevatten woorden zoals "directeur", "stichter", "zaakvoerder" en "ceo". Zoals verwacht zijn de positieve termen uit de heuristische regels inderdaad kwalitatieve features. Daarnaast zijn ook features met woorden als "weten", "zeggen", "aldus" en een komma (bv. ", aldus" of ", die") relevant. Deze zijn minder voor de hand liggend, maar houden intuïtief wel steek. Tenslotte bleken ook de woorden "Mathieu", "investeringsmaatschappij" en "aandelenpakket" relevant. Deze houden uiteraard geen steek; DeepDive vond deze woorden belangrijk omdat ze wel vaak voorkomen, maar eigenlijk aan zeer weinig unieke kandidaat-paren gelinkt zijn.

Analoog bevatten de features met de grootste negatieve gewichten woorden die deel uitmaken van de negatieve termen in de heuristische regels (woorden als "voorzitter", "dochter/zoon", "gedeputeerde", "collega" en "echtgenote"). Enkele intuïtief relevante features bevatten woorden zoals "Prof.", "volgen ... op" (van "opvolgen") en "machtsoverdracht". Opnieuw zijn er ook een aantal schijnbaar relevante woorden, hoofdzakelijk persoonsnamen, die we toeschrijven aan overfitting op de training data. Vreemd genoeg blijken ook enkele features met de woorden "zeggen" en "aldus" relevant, die we reeds bij de features met de grootste positieve gewichten zagen. Hoewel hun gewichten hier beduidend kleiner zijn, laat het ons vermoeden dat deze woorden toch minder relevant zijn dan verwacht.

We weten nu dat de tweede versie van onze toepassing zeer goede resultaten behaalt. Bovendien is DeepDive er ook in geslaagd relevante features te leren die geen deel uitmaken van de positieve of negatieve termen, wat noodzakelijk zal zijn om ook niet-gelabelde kandidaten te classificeren. Wel moeten we onze resultaten enigszins relativeren: we hadden ons beperkt tot het gebruik van features uit kwalitatieve nieuwsartikels en de kandidaten die erin voorkomen, waardoor alle kandidaten reeds gelabeld waren. Ons ultieme doel is eigenlijk om ook voor niet-gelabelde kandidaten te weten te komen of ze al dan niet tot de baas-bedrijf relatie behoren, en dit met een hoge precision. Hier zullen we ons in de derde fase op toespitsen.

5.4 Derde fase: niet-gelabelde kandidaten

Dankzij de tweede versie weten we dat er potentieel zit in onze toepassing. Op basis van kwalitatieve features en extra domeinkennis kon DeepDive zeer goed bepalen of een kandidaat-tupel tot de baas-bedrijf relatie behoort of niet. In deze fase zullen we proberen om ook niet-gelabelde kandidaten correct te

classificeren, om zo onze baas-bedrijf relatie uit te breiden. Deze fase noemen we de 'derde fase', en we spreken ook van de 'derde versie' van onze toepassing. Ook de derde fase bestaat uit meerdere iteraties die we samenvoegen om de bespreking beknopter te maken.

5.4.1 Voorbereidingen

Vooraleer we de niet-gelabelde kandidaten gaan introduceren, voeren we nog twee andere aanpassingen door. Herinner dat we ons in de tweede fase beperkten tot de features die in kwalitatieve nieuwsartikels voorkomen. De niet-gelabelde kandidaten bezitten in het algemeen weinig van die features, net omdat ze niet in die artikels voorkomen. Daarom beschouwen we vanaf nu alle features van een kandidaat-paar, in eender welk nieuwsartikel. De feature-gebaseerde formule in het MLN wordt dus terug die uit de eerste versie, namelijk:

```
@weight(f)
person_company(person, company) :- features(person, company, f).
```

Aangezien kandidaat-paren niet langer gelabeld hoeven te zijn, passen we de invulling van *person_company* aan naar:

```
person_company(person, company) = if l > 0 then TRUE
                                  else if l < 0 then FALSE
                                  else NULL end :-
    resolved_labels(person, company, l).
```

Dit is opnieuw de definitie uit de eerste versie (zie deel 5.2.6).

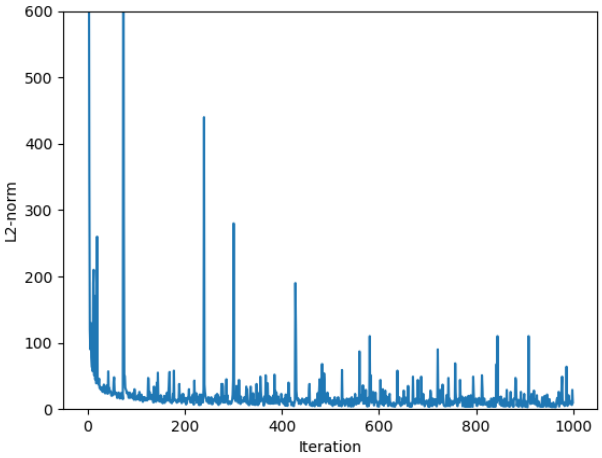
5.4.2 Niet-gelabelde kandidaten

Van de in totaal 7689 kandidaten zijn er slechts 456 gelabeld. Zoals reeds gezegd, is het doel van deze fase om ook de 7233 niet-gelabelde kandidaten te classificeren. We gaan ons echter beperken tot een willekeurige selectie van (ongeveer) 200 daarvan, omdat we vermoeden dat dit de kwaliteit van de toepassing ten goede zal komen. Wanneer relatief veel kandidaten gelabeld zijn, zal het voor DeepDive namelijk makkelijker zijn om te leren welke features relevant zijn voor de classificatie. Immers, als er vele niet-gelabelde kandidaten zijn met features die de gelabelde kandidaten niet bezitten, kan DeepDive door het ontbreken van het label niet leren of het een relevante feature is of niet. De computationele kost wijzigt amper t.o.v. de tweede versie.

Om de niet-gelabelde kandidaat-paren bij te houden, definiëren we de relatie *some_unlabeled_candidates* als volgt:

```
some_unlabeled_candidates(
    person   text,
    company  text
).
```

Vervolgens stellen we de UDF genaamd "random_select" op. Deze roepen we op voor ieder van de 7233 niet-gelabelde kandidaat-paren. De willekeurige selectie van (ongeveer) 200 van die kandidaat-paren realiseren we door ze met een kans van 200 over 7233 te returnen, en ze ten slotte aan de relatie *some_unlabeled_candidates* toe te voegen. De oproep van de UDF en de invulling van de relatie gebeuren als volgt:



Figuur 37: De leercurve voor de derde versie van onze toepassing.

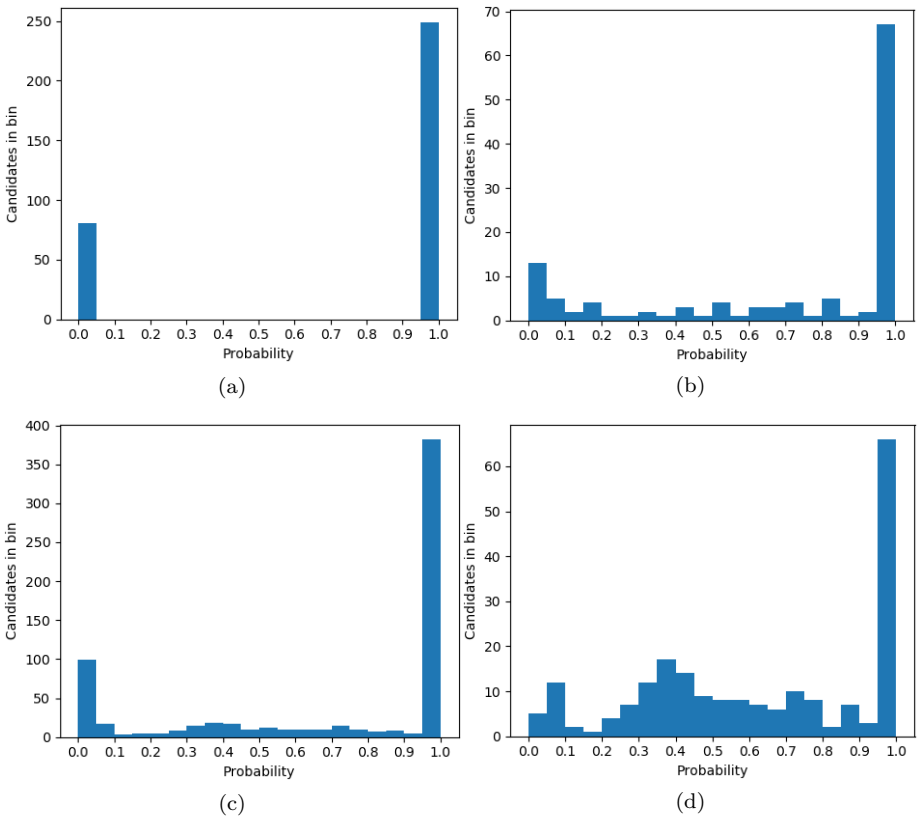
```
some_unlabeled_candidates += random_select(person, company) :-
    candidates(person, company),
    !EXISTS[resolved_labels(person, company, _)].
```

Bij de uitvoering die we gebruiken voor de evaluatie, werden 206 niet-gelabelde kandidaten geselecteerd.

5.4.3 Evaluatie

Leercurve:
Net als in de twee voorgaande fases gebruiken we de leercurve, kansverdelingen, precision en recall om de kwaliteit van de derde versie van onze toepassing te bepalen. We benadrukken dat het doel van de derde versie is om ook niet-gelabelde kandidaten te classificeren met een hoge precision. Figuur 37 toont de leercurve. In tegenstelling tot de twee voorgaande versies, zijn er aanzienlijk grotere fluctuaties in de grootte van de wijziging van de gewichten, wat ons laat vermoeden dat DeepDive het moeilijker had om geschikte gewichten te leren. Aangezien de grootte van de wijziging wel degelijk convergeert, vermoeden we dat DeepDive alsnog een hoge precision zal behalen voor de training data.

Kansverdelingen:
Figuur 38 toont de histogrammen van de kansverdeling van (a) de training data, (b) de test data, en (c) alle data. Merk op dat de kandidaat-paren in de relatie *some_unlabeled_candidates* onder "alle data" vallen. Opnieuw hebben deze drie histogrammen een vrij hoekige U-vorm, waardoor we vermoeden dat DeepDive de kandidaten (inclusief de niet-gelabelde) goed kan classificeren. Het is echter zinvol om ook het histogram met alleen de niet-gelabelde kandidaten te visualiseren; figuur 38(d) toont dit. Dit histogram heeft een veel minder hoekige U-vorm, wat erop duidt dat DeepDive minder zeker is. Wel lijkt een treshold van 0.95 nog steeds aangewezen om een hoge precision te garanderen.



Figuur 38: Histogram van de kansverdeling van (a) de training data, (b) de test data, (c) alle data en (d) de niet-gelabelde data, voor de derde versie van onze toepassing.

Precision en recall:
Zoals op basis van de leercurve te verwachten viel, zijn de precision en recall voor de training data opnieuw allebei 1.0. Die voor de test data bedragen respectievelijk 0.94 en 0.68. De precision en recall wijzigen dus niet noemenswaardig t.o.v. de tweede versie – daar bedroegen ze resp. 0.94 en 0.66. Ondanks het toevoegen van 206 niet-gelabelde kandidaten, lijkt het voor DeepDive niet moeilijker te zijn geworden om de test data goed te classificeren.

Om te evalueren hoe kwalitatief de classificatie van de niet-gelabelde kandidaten is, is een manuele evaluatie de enige optie. Dit houdt in dat we zelf de nodige nieuwsartikels doornemen om te bepalen of de persoon wel/niet de baas is van het bedrijf. Het is uiteraard niet de bedoeling om *alle* kandidaat-paren manueel te valideren – anders hadden we DeepDive niet hoeven gebruiken in de eerste plaats – maar we zullen er enkele evalueren om een ruwe schatting te bekomen. Voor 66 van de 206 niet-gelabelde kandidaten voorspellen we dat ze wel tot de baas-bedrijf relatie behoren (i.e. hun marginale kans bedraagt minstens 0.95). Het lijkt ons gepast om 20 van de 206 niet-gelabelde kandidaten manueel te evalueren, omdat dit een aanzienlijk deel is (ongeveer 1/10) en praktisch haalbaar blijft. Aangezien we meer belang hechten aan de precision, besluiten we om willekeurig 10 positieve en 10 negatieve voorspellingen te selecteren i.p.v. er willekeurig 20 te selecteren uit beide groepen tezamen. Voor deze 20 manuele evaluaties zijn $TP = 6$, $FP = 4$, $TN = 6$ en $FN = 4$. Onze schattingen voor de precision en recall bedragen bijgevolg respectievelijk 0.60 en 0.60. Hoewel dit slechts ruwe schattingen zijn, lijkt het ons veilig te concluderen dat de recall praktisch niet wijzigt t.o.v. van de test data, maar de precision wel significant afneemt. Het is enigszins logisch dat de precision afneemt, omdat deze kandidaat-paren de zeer kwalitatieve features (zoals "CEO" en "zaakvoerder") niet bezitten. Desondanks hadden we toch een hogere precision verwacht, eerder in de buurt van 0.80. Zo dadelijk formuleren we een mogelijke verklaring hiervoor.

Van de 456 gelabelde kandidaten, zijn er 316 waar DeepDive een marginale kans van minstens 0.95 aan koppelt; deze beschouwen we als wel tot de baas-bedrijf relatie behorend. Zoals reeds gezegd voorspellen we voor 66 van de 206 niet-gelabelde kandidaten dat ze tot de baas-bedrijf relatie behoren, waardoor we in totaal $316 + 66 = 382$ paren hebben weten extraheren. Figuur 39 toont er 30 willekeurige hieruit, waarbij paren met een ongeldige bedrijfsnaam werden genegeerd; hieronder bespreken we het probleem van ongeldige bedrijfsnamen. De volledige baas-bedrijf relatie, met alle 382 paren, hebben we gepubliceerd op `michielvm.synology.me/thesis/table`.

Voornaamste features:
Gezien de goede resultaten voor de gelabelde kandidaten en de minder goede resultaten voor de niet-gelabelde kandidaten, zijn we erg benieuwd wat de voornaamste features zijn die DeepDive geleerd heeft. Net als in de tweede versie zien we dat de features met de grootste positieve gewichten termen bevatten als "CEO", "directeur", "oprichter", "zaakvoerder" en "bestuursvoorzitter". Opnieuw vindt DeepDive features met woorden als "weten", "zeggen" en een komma belangrijk, en nu ook "besluiten", "bevestigen" en "topman", wat zeker steek houdt. Anderzijds zijn er opnieuw features met een onterecht hoog gewicht, bijvoorbeeld met de woorden "vandaag", "schadevergoeding" of "Mathieu". Deze woorden zijn zeer specifiek, in de zin dat ze vaak voorkomen in de

#	baas	bedrijf
1	Roger Anné	Aminolabs
2	Pieter Rober	Pure Air
3	Hugo Knevels	Sleurs Industries
4	Dominic Stas	Mediahuis Limburg bij Concentra
5	Filip Biesmans	Tectum Group
6	Stijn Bijnens	LRM
7	Ulrich Penzkofer	NRB Groep
8	Ronny Ruyters	ACA IT-Solutions
9	Dirk Trippaers	ME Construct
10	Anke Coenen	Wearable Stories
11	Jos Vandael	Imes Dexis
12	Hans Wilmots	BDO België
13	Ann Boffé	Mercuri Urval
14	Peter Geurden	Different Hotels
15	Joep Kempen	Team Cyclis
16	Paul Smeets	Xtendit Solutions
17	Jan Vansant	ACP-groep
18	Yves Smolders	Versuz
19	Paul Hiel	Punch Metals
20	Koen Lambrechts	TVL
21	Wouter Uten	UgenTec
22	Axel Verberckmoes	Levenstond Seafood
23	Cor van Otterloo	Punch Powertrain
24	Clark Taverniers	Vio Interim
25	Bart Hiemeleers	Genker Plantencentrum
26	Ben Lambrechts	Hogeschool PXL
27	Ilja Theuwissen	Pure IT
28	Peter Guelinckx	Groep Delorge
29	Tom Heeren	DHZ
30	Antoon Jean Vanherk	Kempische Metaalwerken

Figuur 39: Dertig willekeurige rijen uit onze baas-bedrijf relatie.

artikels, maar slechts in de context van een klein aantal verschillende kandidaat-paren. Daarom verwachten we dat deze features geen grote impact hebben op de precision, en ze dus niet de oorzaak kunnen zijn voor de verlaagde precision bij de niet-gelabelde kandidaten. De 100 features met de grootste positieve gewichten hebben we gepubliceerd op `michielvm.synology.me/thesis/features-pos`.

De features met de grootste negatieve gewichten bevatten opnieuw de negatieve termen, zoals "voorzitter", "gedeputeerde", "collega" en familiale relaties. Ook zien we opnieuw features die intuïtief steek houden, met woorden zoals "Prof.", "volgen ... op" en "machtsoverdracht". Daarnaast zijn er een aantal concrete persoons- en bedrijfsnamen die als voorname feature dienst doen, meer bepaald als ze deel uitmaken van de tekst tussen het kandidaat-paar. Hoewel deze geen steek houden (omdat ze te specifiek zijn), is het wel logisch ze hier te zien. Herinner namelijk hoe we in deel 5.3.1 twijfelden over het invoeren van de derde heuristische regel ("als de persoons- en bedrijfsnaam dicht bij elkaar en er een derde persoons- of bedrijfsnaam tussen hen staat, genereren we een negatief label"), en uiteindelijk besloten om deze niet in te voeren omdat hij te veel ruis veroorzaakt. Merk nu het verband op tussen beide: de features bestaande uit een concrete persoons- of bedrijfsnaam tussen een kandidaat-paar, zijn eigenlijk concrete invullingen van de derde persoons- of bedrijfsnaam in de heuristische regel. De 100 features met de grootste negatieve gewichten hebben we gepubliceerd op `michielvm.synology.me/thesis/features-neg`.

We concluderen dat DeepDive opnieuw erg zinvolle features geleerd heeft. Opnieuw beperken deze zich niet louter tot de features die meteen volgen uit de positieve en negatieve termen van de heuristische regels. Dat is precies wat de classificatie van de niet-gelabelde kandidaten mogelijk maakt.

Mogelijke verklaringen voor de verlaagde precision:
Het spreekt voor zich dat we een verklaring trachten te vinden voor de verlaagde precision voor de niet-gelabelde kandidaten; herinner dat deze 0.60 bedroeg tegenover 0.94 voor de test data.

Ten eerste merkten we tijdens de manuele evaluatie van de niet-gelabelde kandidaten op dat een significant aantal kandidaat-paren uit een ongeldige bedrijfsnaam bestaat; enkele voorbeelden zijn "CFO", "Klas" en "Truienaar". Deze hebben we voor alle duidelijkheid niet opgenomen in de bovengenoemde schattingen voor de precision en recall. Om vast te stellen hoe significant het probleem van ongeldige bedrijfsnamen is, besluiten we om 100 unieke kandidaat-paren, willekeurig geselecteerd, manueel te valideren. Slechts 66 van de 100 bleken wel degelijk uit een geldige bedrijfsnaam te bestaan. Aangezien deze kandidaat-paren ongeldig zijn, bezitten ze geen zinvolle features, waardoor DeepDive er een klein gewicht voor zal leren. Deze niet-kwalitatieve features zijn geen probleem voor de gelabelde kandidaten (de training en test data) omdat zulke kandidaten sowieso ook features met een hoog gewicht bezitten (de features gebaseerd op de positieve en negatieve termen). De gewichten van de niet-kwalitatieve features zijn in verhouding zodanig klein dat ze amper een impact hebben op de classificatie van gelabelde kandidaten. De niet-gelabelde kandidaten daarentegen, zijn minder vaak in het bezit van features met een hoog gewicht omdat ze al sowieso geen features bezitten die gebaseerd zijn op de positieve en negatieve termen. Bijgevolg hebben features met kleinere gewichten, waaronder die afkomstig van kandidaten met een ongeldige bedrijfsnaam, een grotere impact op de classificatie. Dit vertaalt zich in een grotere onzekerheid

voor de niet-gelabelde kandidaten, wat de vorm van de kansverdeling in figuur 38(d) verklaart. Ondanks de kleine gewichten die aan deze features gekoppeld worden, vermoeden we dat de impact op de precision en recall voor de niet-gelabelde kandidaten significant is, namelijk omdat het zo veel features betreft. Dit probleem is zeker een verbeterpunt naar een toekomstige iteratie toe.

Om verder te achterhalen wat de oorzaken zijn van de afname, kijken we naar de (niet-gelabelde) kandidaten die tijdens de manuele evaluatie foutief geclassificeerd werden (i.e. de false positives en false negatives). Meer bepaald kijken we naar de nieuwsartikels waarin ze verschijnen en de features die ze bezitten. Met het volgende frequent voorkomend patroon gaat DeepDive vaak de mist in: "P1 (B1), P2 (B2)", met P1 en P2 verschillende persoonsnamen en B1 en B2 verschillende bedrijfsnamen. Als lezer kunnen we dit patroon correct interpreteren, zijnde dat P1 de baas is van B1 en P2 van B2. Voor DeepDive lukt dit niet; het is een significante bron voor zowel false positives (bv. P1 is de baas van B2) als false negatives (bv. P1 is niet de baas van B1). DeepDive heeft inderdaad kleine gewichten geleerd voor de features gebaseerd op dit patroon. Hoewel het aantal features vrij beperkt is, zijn er vrij veel kandidaten die zo'n features bezitten, waardoor de impact op de precision en recall voor de niet-gelabelde kandidaten niet te onderschatten valt. Om de ambiguïteit rond dit patroon weg te werken, zouden we twee heuristische regels kunnen toevoegen: indien (P1,B1) het kandidaat-paar is, genereren we een positief label; indien (P1,B2) het kandidaat-paar is, genereren we een negatief label.

Bij de false negatives zagen we features (en artikels) met de woorden "president" en "ondernemer". Hoewel "president" duidelijk een positieve connotatie heeft, koppelt DeepDive er geen groot positief gewicht aan, omdat er enerzijds weinig features zijn met dit woord en anderzijds weinig kandidaten met zulke features. Dit zouden we kunnen verhelpen door het als een positieve term te beschouwen bij de heuristische regels. De impact op de precision en recall voor de niet-gelabelde kandidaten zal hier echter weinig door verbeteren. Voor het woord "ondernemer" zijn er wel vele features en ook vele kandidaten met zulke features. Aan enkele van die features koppelt DeepDive een vrij hoog positief gewicht, en aan enkele andere een vrij hoog negatief gewicht. Hoewel dit op het eerste zicht zeer vreemd is, blijkt dit terecht: voor het ene kandidaat-paar duidt dit op een positief verband (bv. in de context van een overname), maar voor het andere op een negatief verband (bv. in de context van een investering). DeepDive heeft dus terecht geen groot positief (of negatief) gewicht geleerd voor features op basis van het woord "ondernemer".

De algemene oorzaak voor de verlaagde precision voor de niet-gelabelde kandidaten is dus het ontbreken van features met grote gewichten, ook voor de false positives. De vele features met kleine gewichten zorgen tezamen voor een onterecht lage of zelfs hoge marginale kans. Voor het ontbreken van features met grote gewichten is geen ene specifieke reden, maar wel een samenloop van verschillende redenen (hierboven toegelicht).

Impact van formules met domeinkennis:
Ten slotte gaan we na hoe groot de impact is van de formules met extra domeinkennis. Deze hadden we reeds in deel 5.2.6 geïntroduceerd. In deel 5.3.4 besloten we het gewicht van beide formules in te stellen op -0.5, wat overeenkomt met één van de meer voorname features (met een vrij groot negatief gewicht dus). In afwezigheid van deze formules blijven de precision en recall voor de training data

gelijk (zijnde beide 1.0). Voor de test data daalt de precision van 0.94 naar 0.90 en de recall van 0.68 naar 0.67. De formules met extra domeinkennis hebben dus alleen een noemenswaardige impact op de precision, niet op de recall. Herinner de gelijkenis tussen Logistic Regression en de DeepDive methodologie voor een MLN zonder formules met extra domeinkennis (zie deel 4.7.2). Aangezien de formules met domeinkennis de kwaliteit van onze toepassing wel degelijk ten goede komen, biedt de DeepDive methodologie (i.e. het gebruik van een MLN) een meerwaarde tegenover het gebruik van Logistic Regression.

5.5 Conclusies en toekomstig werk

We formuleren een aantal conclusies omtrent onze toepassing en geven een aantal suggesties naar mogelijke volgende stappen. Tevens formuleren we enkele conclusies over onze opgedane ervaring en DeepDive als systeem. Onze toepassing had als doel om paren van Limburgse bedrijven en hun baas te extraheren uit nieuwsartikels, en zodoende de baas-bedrijf relatie op te bouwen. Vooral de correctheid van de geëxtraheerde paren was van belang (i.e. high-precision).

DeepDive heeft het resulterende model (Markov Network) getraind op basis van 454 gelabelde kandidaat-paren (341 positieve en 113 negatieve) en 206 niet-gelabelde kandidaat-paren, door rekening te houden met hun features en extra domeinkennis. De voornaamste features (i.e. de features met een groot positief of negatief gewicht) bleken bijzonder kwalitatief en beperkten zich niet tot features die rechtstreeks volgen uit de heuristische regels. Om een zo hoog mogelijke precision te garanderen, stopten we alleen kandidaat-paren met een marginale kans van minstens 0.95 in onze baas-bedrijf relatie, ten koste van een lagere recall. Zo behaalden we een precision van 0.94 en recall van 0.66 voor de test data[30], waar we erg tevreden mee zijn. Van de 454 gelabelde kandidaten hebben we er 316 in de baas-bedrijf relatie geplaatst (hun marginale kans bedroeg dus minstens 0.95). Om een ruwe schatting te bekomen van de precision en recall voor de niet-gelabelde kandidaten, hebben we een korte manuele evaluatie gedaan (met 10 positieve en 10 negatieve voorspellingen). De precision en recall bedroegen beide 0.60, waardoor de precision lager bleek te zijn dan verwacht/gehoopt. We hebben 66 van de 206 niet-gelabelde kandidaten aan de baas-bedrijf relatie toegevoegd, wat het totaal op 382 brengt.

Zoals in deel 5.4.3 beschreven, hebben we geanalyseerd waarom de precision voor de niet-gelabelde kandidaten significant lager ligt dan die voor de gelabelde kandidaten. Ten eerste zagen we dat slechts 66% van de geëxtraheerde bedrijfsnamen wel degelijk een bedrijfsnaam is. De Frog toolkit schiet hier duidelijk tekort, waardoor we aanraden om naar een alternatief op zoek te gaan. Dit probleem veroorzaakt vele zinloze features die relatief weinig voorkomen; DeepDive leert er (terecht) een klein gewicht voor. Ten tweede bleek DeepDive geen grote gewichten te leren voor features op basis van het patroon "P1 (B1), P2 (B2)", wat vaak voorkomt in de nieuwsartikels. We raden aan hier een positieve en negatieve heuristische regel voor te formuleren. Door dit probleem koppelt DeepDive (onterecht) een klein gewicht aan features die, hoewel het er weinig zijn, wel vaak voorkomen. Beide problemen zorgen ervoor dat de niet-gelabelde kandidaten vele features met kleine gewichten bezitten. In afwezigheid van andere kwalitatieve features beginnen zulke features zwaarder door

[30] We deelden de gelabelde data op in 75% training data en 25% test data.

te wegen, waardoor de classificatie faalt en de precision dus afneemt.

Daarnaast is het waarschijnlijk zinvol om aan 'coreference resolution' te doen. Coreferences zijn woorden (of woordgroepen) die naar eenzelfde woord verwijzen; bv. "het bedrijf" kan verwijzen naar een concrete bedrijfsnaam uit de voorgaande zin. Door hier rekening mee te houden, zal DeepDive waarschijnlijk makkelijker en meer relevante features kunnen identificeren. Aangezien Frog dit niet ondersteunt, is het opnieuw aangeraden naar een alternatieve NLP toolkit op zoek te gaan.

Door de vrij lage precision voor niet-gelabelde kandidaten, is onze toepassing momenteel nog niet geschikt voor bv. de invulling van Google's Knowledge Panels voor Limburgse bedrijven (zie deel 5.1). Ter verbetering zijn er nog een aantal extra iteraties nodig, waarin als eerste stappen de bovengenoemde suggesties uitgeprobeerd kunnen worden. De huidige staat van onze toepassing is uiteraard al een grote stap in de goede richting.

Alhoewel we ons beperkt hebben tot ongeveer 200 willekeurige niet-gelabelde kandidaten, kan men onze aanpak eenvoudig verderzetten om uiteindelijk alle 7233 niet-gelabelde kandidaten te classificeren. Men kan de niet-gelabelde kandidaten systematisch doorlopen, in delen van 200 kandidaat-paren; deze stopt men in de relatie *some_unlabeled_candidates* en vervolgens laat men DeepDive het relation extraction proces doorlopen[31]. Zo heeft DeepDive steeds voldoende training data voorhanden om de niet-gelabelde kandidaten te classificeren. Via deze aanpak kan men de baas-bedrijf relatie dus verder uitbreiden.

We hebben aan den lijve kunnen ondervinden dat relation extraction geen eenvoudige taak is. Het heeft ons vele iteraties gekost om tot onze uiteindelijke resultaten te komen, en zoals reeds gezegd zijn nog extra iteraties nodig. In het bijzonder merkten we hoe de kwaliteit van onze toepassing afhangt van vele factoren: de kwaliteit van de gebruikte data en NLP-toolkit (in het geval van tekst), de kwaliteit van de geëxtraheerde kandidaat-tupels en hun features, de kwaliteit van de training data (heuristische regels i.h.b.), en de geschiktheid van de regels met eigen domeinkennis en hun gewichten. Elk van deze aspecten moet on-point zijn om tot een kwalitatieve toepassing te komen. We zouden eerder van relation extraction "engineering" spreken omdat de methodologie en het gebruikte systeem hier geen impact op hebben.

Persoonlijk vinden we dat DeepDive een vrij hoge leercurve heeft. Het kost aanzienlijk veel tijd om vertrouwd te raken met het systeem, ondanks de uitgebreide documentatie. Het systeem is zodanig uitgebreid dat een snelle start niet echt mogelijk is. Desondanks vinden we dat DeepDive wel degelijk tegemoet komt aan de verwachtingen van een state-of-the-art systeem. Eens men het onder de knie krijgt, ziet men namelijk in dat DeepDive een snelle ontwikkeling stimuleert door veel automatisatie in te bouwen, het ondersteunen van een vrij high-level ontwikkeling (DDlog en UDF's), en doordat het mogelijk is om alleen de nodige stappen van het execution plan uit te voeren. Wel vinden we de mogelijkheden voor evaluatie ontoereikend, zoals in deel 5.2.8 geargumenteerd.

[31]Het is niet nodig om telkens het volledige proces te doorlopen: alleen de relatie *person_company* moet opnieuw worden ingevuld, en de learning en inference dienen uiteraard opnieuw te gebeuren. We hadden namelijk reeds alle kandidaten met hun features geëxtraheerd.

6 DeepDive methodologie toegepast op gestructureerde data

Bij het toelichten van de DeepDive methodologie (zie deel 4), gingen we er eigenlijk steeds vanuit dat het de bedoeling was om een relatie te extraheren uit *ongestructureerde* data (i.h.b. uit geschreven tekst). Hetzelfde gold voor onze toepassing, waar we de baas-bedrijf relatie extraheerden uit nieuwsartikels. In dit deel gaan we na in welke mate de DeepDive methodologie ook op *gestructureerde* data kan worden toegepast, om daaruit dus een relatie te extraheren die nog niet expliciet aanwezig was. Meer bepaald werken we een Proof of Concept uit om (1) te illustreren hoe de DeepDive methodologie zich vertaalt naar een scenario met gestructureerde data, (2) aan te tonen dat deze aanpak tot kwalitatieve resultaten kan leiden, en (3) het verband met ander wetenschappelijk werk duidelijk te maken.

6.1 Introductie en motivatie

Typisch heeft Relation Extraction als doel om gestructureerde informatie te extraheren uit een collectie ongestructureerde data. Het Web is een gigantische bron van ongestructureerde data. Dankzij systemen als DeepDive en vergelijkbare initiatieven, werden hier reeds grote hoeveelheden gestructureerde data uit geëxtraheerd. Soms wordt deze data als een publiek toegankelijke collectie triples[32] gepubliceerd, waardoor men van 'Linked Open Data' (LOD) spreekt. Op de website `http://lod-cloud.net` vindt men een overzicht terug van de vele LOD datasets en hun onderlinge connecties – men spreekt van de 'LOD Cloud'.

De LOD Cloud zal alsmaar groeien door nieuwe extracties uit ongestructureerde data. Echter, het lijkt ons haalbaar om ook nieuwe relationele informatie te extraheren uit de reeds beschikbare gestructureerde data, en zodoende de LOD Cloud nog verder uit te breiden. Beschouw als voorbeeld een hypothetische dataset die de stamboom van een familie beschrijft, in eender welk gestructureerd data-formaat (bv. als collectie triples of een SQL database). Veronderstel voorlopig dat alleen de relatie $heeftKind(o, k)$, met o de ouder en k het kind, gebruikt wordt om de stamboom te definiëren. Ons doel is om deze gestructureerde dataset verder aan te vullen met de relatie $zijnGetrouwd(p1, p2)$, met $p1$ en $p2$ personen die met elkaar getrouwd zijn. In onze hypothetische dataset kunnen we ervan uitgaan dat als personen $p1$ en $p2$ eenzelfde kind hebben, ze dan ook met elkaar getrouwd zijn. Deze relatie zouden we dus perfect kunnen extraheren m.b.v. volgende deductie-regel: $heeftKind(p1, k) \wedge heeftKind(p2, k) \implies zijnGetrouwd(p1, p2)$.[33]

Wanneer we de relatie $zijnGetrouwd(p1, p2)$ in een meer realistisch scenario zouden extraheren, maken we gebruik van reeds bestaande LOD datasets. Zulke datasets zijn veel complexer dan onze hypothetische dataset: ze zijn veel groter (zowel in aantal tupels als in aantal verschillende relaties), bevatten onvolledigheden, en kunnen tegenstrijdigheden bevatten (niet alle getrouwde persoonsparen voldoen aan bovenstaande deductie-regel). Door die probabilistische aspecten, lijkt de DeepDive methodologie ons een geschikte en beloftevolle aanpak.

[32] Herinner: dit is het formaat dat het RDF data model gebruikt.

[33] Voor de volledigheid zouden we ook moeten eisen dat $p1 \neq p2$, en zouden we tevens $zijnGetrouwd(p2, p1)$ kunnen deduceren.

Het Markov Logic Network dat DeepDive leert, zou bv. volgende voorname features kunnen beschouwen: het persoonspaar heeft een gemeenschappelijk kind, woont op hetzelfde adres, heeft ongeveer dezelfde leeftijd, ... Andere relationele informatie zou een bron voor negatieve features kunnen zijn; bv. als het persoonspaar reeds in de relatie *heeftBroer* voorkomt, zijn ze zeker niet getrouwd met elkaar. We geloven dat de combinatie van zulke features en extra domeinkennis een kwalitatief probabilistisch model (MLN) vormt voor het probleem.

In het volgende deel werken we een Proof of Concept uit, en daarna gaan we in op het verband met aanverwant wetenschappelijk onderzoek.

6.2 Proof of Concept: familiale relaties

In dit deel lichten we toe hoe de DeepDive methodologie gebruikt kan worden om een relatie te extraheren uit gestructureerde data. Onze aanpak is als het ware een vertaling van de DeepDive methodologie (voor ongestructureerde data) naar een scenario met gestructureerde data. Om onze uitleg te verduidelijken, werken we met een lopend voorbeeld: we gaan de relatie $brother(X, Y)$ extraheren uit andere gestructureerde informatie omtrent familiale relaties.

6.2.1 Gestructureerde data verzamelen

Als collectie gestructureerde data gebruiken we de 'family' dataset, opgesteld door Richards en Mooney [36]. Deze beschrijft de stamboom van een denkbeeldige familie, bestaande uit 86 personen verdeeld over 5 generaties. De relaties die gebruikt worden om de stamboom te definiëren, tonen we in tabel 3. De eerste drie relaties, *gender*, *parent* en *married*, zijn de 'primitieve relaties'; de bijhorende voorkomens werden manueel opgesteld door de auteurs van de dataset. De overige twaalf relaties zijn 'gededuceerde relaties'; deze voorkomens werden automatisch gegenereerd m.b.v. deductie-regels zoals $gender(X, \text{“}female\text{”}) \wedge married(X, Y) \implies wife(X, Y)$.

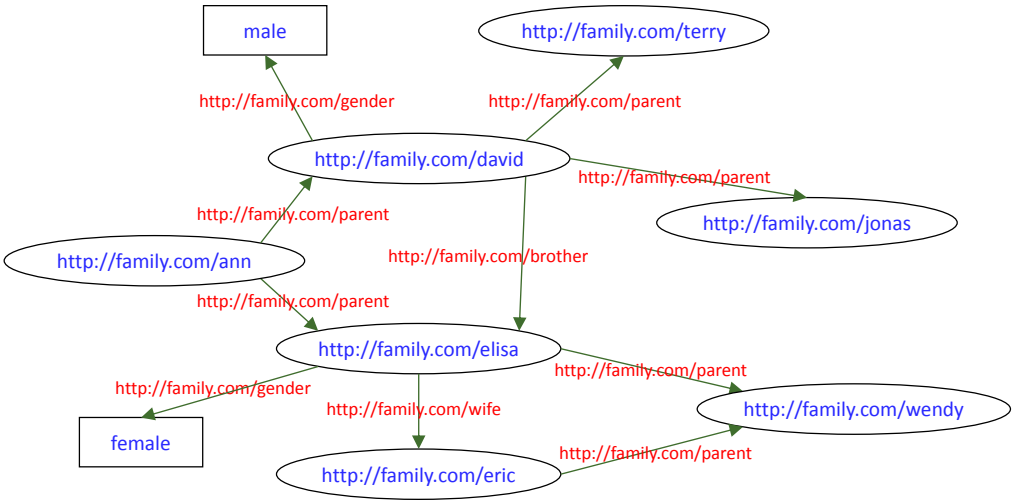
We kozen specifiek voor de family dataset omdat we verwachten dat die hoge slagingskansen biedt aan onze Proof of Concept (PoC). Het lijkt ons namelijk een vrij "gemakkelijke" dataset:

1. De grootte van de dataset is ideaal: er is voldoende data om aan supervised machine learning te doen, en niet zodanig veel dat het het proces zou bemoeilijken (door een enorm aantal features bv.).
2. De dataset is volledig: alle geldige voorkomens van de relaties uit tabel 3 zijn in de stamboom opgenomen.
3. Er staan geen fouten of tegenstrijdigheden in de data. Bijvoorbeeld dat er gescheiden personen zijn, of dat tussen een paar personen zowel de relatie *brother* als *wife* geldt.

Ons doel is om de relatie $brother(X, Y)$ te extraheren op basis van de relaties $gender(X, g)$, $parent(X, Y)$, $wife(X, Y)$ en $brother(X, Y)$ zelf. Figuur 40 visualiseert een klein deel van de dataset als graaf; het persoonspaar (*david*, *elisa*) behoort wel tot de relatie *brother* en (*elisa*, *eric*) niet. Dit lijkt ons een eenvoudig probleem en we verwachten dan ook dat onze aanpak kwalitatieve resultaten zal opleveren. Tevens verwachten we dat DeepDive o.m. zal leren dat "het geslacht van X moet *male* zijn", "X en Y moeten dezelfde ouders hebben", en "X en Y

Relatie	**Betekenis**	**Aantal voorkomens**
$gender(X, g)$	Het geslacht van persoon X is g; dit is de constante "male" of "female".	86
$parent(X, Y)$	Persoon X is een ouder van persoon Y.	120
$married(X, Y)$	Personen X en Y zijn met elkaar getrouwd. Voor een concreet getrouwd koppel (x, y) bevat de dataset alleen $married(x, y)$ of $married(y, x)$, niet beide.	50
$wife(X, Y)$	Persoon X is de echtgenote van persoon Y.	25
$husband(X, Y)$	Persoon X is de echtgenoot van persoon Y.	25
$mother(X, Y)$	Persoon X is de moeder van persoon Y.	60
$father(X, Y)$	Persoon X is de vader van persoon Y.	60
$daughter(X, Y)$	Persoon X is een dochter van persoon Y.	54
$son(X, Y)$	Persoon X is een zoon van persoon Y.	66
$sister(X, Y)$	Persoon X is een zus van persoon Y.	47
$brother(X, Y)$	Persoon X is een broer van persoon Y.	59
$aunt(X, Y)$	Persoon X is een tante van persoon Y.	82
$uncle(X, Y)$	Persoon X is een nonkel van persoon Y.	92
$niece(X, Y)$	Persoon X is een nicht van persoon Y.	92
$nephew(X, Y)$	Persoon X is een neef van persoon Y.	82

Tabel 3: Informatie over relaties in de 'family' dataset.



Figuur 40: Enkele triples uit de family dataset.

mogen niet in de relatie *wife* voorkomen". Indien onze aanpak zou slagen voor dit (eenvoudig) probleem, weten we dat er potentieel in zit.

In onze aanpak willen we absoluut met het RDF data model werken; de motivatie hiervoor lichten we in deel 6.2.3 toe. De dataset is dan in essentie een collectie triples, waardoor men van een 'RDF triple store' spreekt. De oorspronkelijke family dataset staat min of meer in CSV-formaat, waarbij de eigenlijke relationele informatie in predicaat-vorm wordt voorgesteld. Om een predicaat "*brother*(*david*, *elisa*)" om te vormen naar het RDF data model, maken we hiervan:

```
<http://family.com/david> <http://family.com/brother>
                                        <http://family.com/elisa> .
```

Er zijn meerdere mogelijke formaten die het RDF data model volgen; dat in het bovenstaand voorbeeld heet 'N-Triples'. In dit formaat wordt een triple (*subject*, *predicate*, *object*) voorgesteld op één regel[34], bestaande uit respectievelijk het *subject*, *predicate* en *object*, en eindigend met een ".". De vier onderdelen worden van elkaar gescheiden door een spatie. Het *subject* en *predicate* moeten URI's zijn; het *object* mag een URI of literal (bv. "male") zijn. Tenslotte moeten de URI's omgeven worden door "<" en ">". Voor het omzetten van de oorspronkelijke dataset naar N-Triples formaat, implementeerden we een klein Python-script.

Om in de volgende stappen onze RDF triple store te kunnen queryen, moeten we deze via een database toegankelijk maken. Hier gebruiken we een Python-library voor, RDFLib [37] genaamd. Meer bepaald laat deze library ons toe om een Web server op te starten die onze triple store aanbiedt via een SPARQL-endpoint[35]. Hoe dit SPARQL-endpoint precies wordt aangesproken (om queries uit te voeren), bespreken we zo dadelijk. Merk op dat we, in tegenstelling tot het geval waarin we met geschreven tekst werken, nu geen NLP preprocessing doen.

6.2.2 Kandidaten bepalen

Het bepalen van de kandidaten is erg afhankelijk van de te extraheren relatie. Aangezien wij de relatie $brother(X, Y)$ willen extraheren, kiezen we ervoor om alle persoonsparen in de database te beschouwen. Zulke algemene aanpak leidt typisch tot een enorm aantal kandidaat-paren, 7310 in ons voorbeeld. Net zoals in de toepassing rond de baas-bedrijf relatie, zullen we het aantal kandidaten in het MLN uiteindelijk nog beperken (om efficiëntie-redenen). De definitie van de relatie *candidates* in "app.ddlog" spreekt voor zich.

Laat ons nu bespreken hoe we de relatie *candidates* gaan invullen. Bij de baas-bedrijf relatie vormden de nieuwsartikels onze input en gingen we daaruit de kandidaten genereren. Nu echter, bestaat de input uit een RDF triple store die de family dataset bevat. Om de kandidaten te genereren, voeren we volgende SPARQL-query uit op de RDF triple store[36]:

```
PREFIX fam: <http://family.com/>
```

[34] Ons voorbeeldje neemt twee regels in beslag wegens plaatsgebrek.

[35] SPARQL is de voornaamste query taal die binnen het RDF gebruikt wordt.

[36] Voor wie vertrouwd is met SQL, spreekt de syntax en semantiek van SPARQL-queries voor zich; voor meer info verwijzen we naar de officiële W3C Recommendation [38].

```
SELECT ?p1 ?p2
WHERE {
    ?p1 fam:gender ?g1.
    ?p2 fam:gender ?g2.
    FILTER ( ?p1 != ?p2 )
}
```

Daartoe hebben we een client nodig die deze query verstuurt naar het SPARQL-endpoint van de server. Hiervoor gebruiken we de Python-library genaamd SPARQLWrapper, die deel uitmaakt van RDFLib. Het resultaat van deze query is een collectie paren $(p1, p2)$; dit zijn de kandidaat-paren. We noemen dit Python-script "generate_candidates.py".

Om in DDlog een relatie in te vullen m.b.v. een Python-script, werken we met een UDF. Hier stuiten we echter op een praktisch probleem: een UDF is een script dat wordt opgeroepen op iedere rij van een relatie. Maar hier hebben we geen relatie als input; we willen het script "zo maar" oproepen, zonder inputs dus. DeepDive ondersteunt dit niet, maar we kunnen ons daar een weg omheen hacken als volgt. In "app.ddlog" definiëren we de relatie *run_once* als volgt:

```
run_once(
    niks   text
).
```

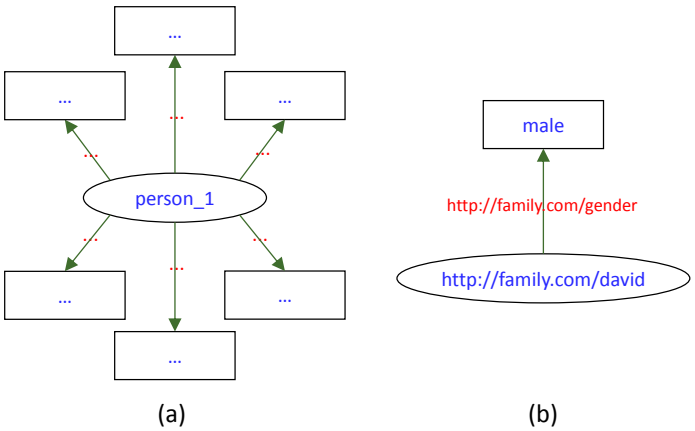
In de folder "input" plaatsen we een gelijknamig TSV-bestand met slechts één regel tekst. De relatie *run_once* bevat dus één rij. Vervolgens definiëren we de UDF "generate_candidates", met als implementatie het bovengenoemde Python-script "generate_candidates.py", en gebruiken deze om de relatie *candidates* in te vullen, als volgt:

```
function generate_candidates over (
    niks   text
) returns rows like candidates
implementation "udf/generate_candidates.py" handles tsj lines.
candidates += generate_candidates(niks) :- run_once(niks).
```

Op deze manier wordt onze UDF precies één keer opgeroepen. De UDF negeert zijn input uiteraard, en de output van de UDF is de collectie query-resultaten.

6.2.3 Features bepalen

De volgende stap bestaat erin de features van ieder kandidaat-paar te extraheren en zodoende de relatie *features* op te bouwen. De definitie van deze relatie in "app.ddlog" spreekt voor zich. In het geval van geschreven tekst baseerden we ons op de NLP tags van de tekst tussen en rond het kandidaat-paar om tot features te komen. Nu beschouwen we de RDF triple store als een graaf en gaan we op zoek naar een aantal algemene patronen hierin. De collectie features van een kandidaat-paar bestaat dan uit alle concrete invullingen van zulke patronen. Nu kunnen we duidelijk maken wat onze motivatie is om met het RDF data model te werken: dit model laat toe om de data als een graaf te zien en er met SPARQL-queries eenvoudig de gewenste patronen in te detecteren.



Figuur 41: (a) Graaf-structuur voor paden van lengte nul. (b) Voorbeeld.

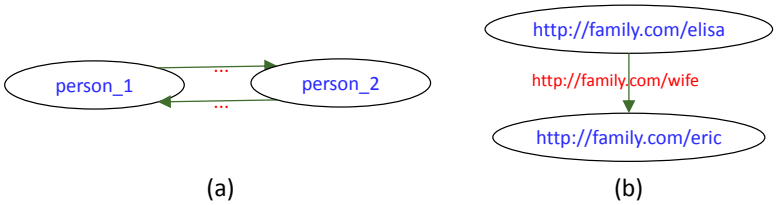
Paden van lengte nul:
Beschouw een kandidaat-paar $(person\_1, person\_2)$ waarvan we de features willen verzamelen. Het eerste patroon waar we naar op zoek gaan in de graafvoorstelling, zien we als de "eigenschappen" of "paden van lengte nul" van beide personen afzonderlijk. Figuur 41(a) toont het patroon (i.e. een spiraalstructuur) voor *person_1* en figuur 41(b) geeft een concreet voorbeeldje op basis van de family dataset; een rechthoek representeert een literal en een ovaal een URI. We detecteren dit patroon met de volgende SPARQL-query:

```
SELECT ?p ?v
WHERE {
    <person_1> ?p ?v .
    FILTER isLiteral(?v)
}
```

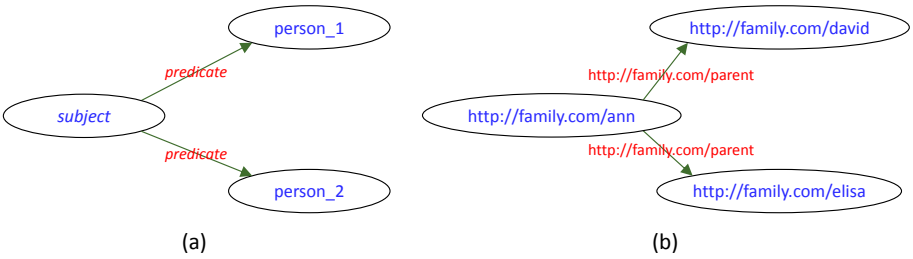
Deze SPARQL-query zal typisch meerdere resultaten opleveren. Ieder individueel resultaat (p, v) herleiden we tot een feature `0_1_PRED_<p>_VAL_<v>`. Met de "0" vooraan identificeren we het specifieke patroon waaruit deze feature afkomstig is, i.e. een pad van lengte nul. De "1" daarna verwijst naar het feit dat deze feature aan de eerste persoon in het paar toebehoort; voor *person_2* staat hier "2". Het vervolg spreekt voor zich. Dankzij dit patroon bekomen we bijvoorbeeld de feature *0_1_PRED_http://family.com/gender_VAL_male*, die ongetwijfeld kwalitatief is. Analoog genereren we op deze manier ook features voor *person_2*.

Paden van lengte één:
Het tweede patroon bestaat uit de rechtstreekse verbindingen tussen het persoonspaar, i.e. paden van lengte één tussen beiden. Figuur 42(a) toont het eigenlijke patroon en figuur 42(b) geeft een concreet voorbeeldje op basis van de family dataset. We detecteren dit patroon door:

```
SELECT ?p
WHERE {
    <person_1> ?p <person_2> .
}
```



Figuur 42: (a) Graaf-structuur voor paden van lengte één. (b) Voorbeeld.



Figuur 43: (a) Graaf-structuur voor paden van lengte twee. (b) Voorbeeld.

Bovenstaande query detecteert de predicaten die zorgen voor een rechtstreekse verbinding van *person_1* naar *person_2*. Ieder individueel resultaat (p) herleiden we tot een feature `1_1-2_PRED_<p>`. Met de "`1`" vooraan identificeren we opnieuw het specifieke patroon waaruit deze feature afkomstig, i.e. een pad van lengte één. De "`1-2`" verwijst naar de richting van de verbinding. Hiermee detecteren we o.a. de feature *1_1-2_PRED_http://family.com/wife*, die ongetwijfeld kwalitatief is. Analoog genereren we ook features op basis van verbindingen in de tegengestelde richting, i.e. features die beginnen met "`1_2-1`".

Paden van lengte twee:
Het derde en laatste patroon dat we gebruiken om features te genereren, zijn paden van lengte twee tussen het kandidaat-paar. Beschouw voorlopig alleen de triples *(subject, predicate, object)* waarin de personen van een kandidaat-paar de rol van *object* op zich nemen. Figuur 43(a) toont het patroon dat we beschouwen: de *objects* zijn met elkaar verbonden door eenzelfde *predicate* en eenzelfde *subject*. Figuur 43(b) geeft een concreet voorbeeldje. Om dit patroon de detecteren, gebruiken we volgende query:

```
SELECT ?p ?s
WHERE {
    ?s ?p <person_1> .
    ?s ?p <person_2> .
}
```

Voor ieder resultaat (p, s) genereren we twee features: `2_SP-SS_PRED_<p>` en `2_SP-SS_PRED_<p>_VAL_<s>`. Met de "`2`" vooraan identificeren we opnieuw de lengte van het pad. De "`SP-SS`" staat voor "Same Predicate - Same Subject". Een voorbeeld van de eerste feature is *2_SP-SS_PRED_http://family.com/parent*, die ongetwijfeld kwalitatief is. In andere scenario's zou de tweede feature van belang kunnen zijn. Analoog genereren we ook features voor het geval dat de

personen het *object* zijn binnen triples. Zulke features beginnen met "2_SP-SO", wat voor "Same Predicate - Same Object" staat.

Voor de invulling van de relatie *features* definiëren we de UDF genaamd "extract_features". In het gelijknamige Python-script implementeren we de hierboven beschreven SPARQL-queries. De implementatiedetails zijn analoog aan die in eerder beschreven delen. Net zoals wanneer we met ongestructureerde data werken, kunnen we een kandidaat-paar voorstellen door een binaire feature-vector die voor alle mogelijke features aanduidt of het kandidaat-paar de desbetreffende feature bezit.

6.2.4 Distant supervision

Net zoals bij ongestructureerde data, passen we distant supervision toe om kandidaat-paren te labelen als wel of niet tot de relatie $brother(X, Y)$ behorend. Opnieuw kunnen we hier reeds bestaande databases en heuristische regels voor gebruiken.

Reeds bestaande databases:
Reeds bestaande databases zijn uiteraard zelf al gestructureerde data, wat er dus op neerkomt dat we reeds bestaande RDF triple stores gaan gebruiken. De precieze triple stores die in aanmerking komen, zijn afhankelijk van het concrete probleem; men dient hierbij zeker de LOD Cloud in het achterhoofd te houden. In onze PoC hebben we alleen de family dataset ter beschikking.

Aangezien de family dataset de te extraheren relatie $brother(X, Y)$ bevat en volledig is, kunnen we eenvoudig positieve voorbeelden verzamelen via de volgende SPARQL-query:

```
PREFIX fam: <http://family.com/>
SELECT ?p1 ?p2
WHERE {
    ?p1 fam:brother ?p2 .
}
```

Dit levert ons 59 positieve voorbeelden, zoals tabel 3 reeds aangaf. We definiëren de UDF "supervise_pos", waarmee we deze query eenmalig uitvoeren (gebruikmakend van onze eerder beschreven hack). De resultaten stoppen we in de relatie *labels_pos*.

We streven ernaar om (ongeveer) evenveel negatieve voorbeelden te verzamelen, 59 dus. Aangezien de family dataset volledig is, kunnen we deze ook gebruiken om negatieve voorbeelden te verzamelen: als voor een kandidaat-paar $(p1, p2)$ het triple $(p1, fam{:}brother, p2)$ niet aanwezig is, is het een negatief voorbeeld. Zo'n kandidaat-paar kan (1) deel uitmaken van de relatie $wife(X, Y)$, die we ook beschouwen, (2) deel uitmaken van een andere relatie in tabel 3, of (3) volgens een niet-beschouwde relatie gerelateerd zijn. Aangezien we de relatie $wife(X, Y)$ mee beschouwen, willen we gegarandeerd een aantal negatieve voorbeelden die in deze relatie zitten. We besluiten om 20% van de 59 negatieve voorbeelden uit deze relatie te halen, wat neerkomt op 12 voorkomens. Deze selecteren we m.b.v. volgende query:

```
PREFIX fam: <http://family.com/>
```

```
SELECT ?p1 ?p2
WHERE {
    ?p1 fam:wife ?p2 .
}
ORDER BY RAND()
LIMIT 12
```

Analoog aan de positieve voorbeelden definiëren we de UDF "supervise_neg_wife" om deze query eenmalig uit te voeren; de resultaten stoppen we in de relatie *labels_neg_wife*.

Tenslotte selecteren we willekeurig nog 47 van de overige negatieve voorbeelden, met volgende query:

```
PREFIX fam: <http://family.com/>
SELECT ?p1 ?p2
WHERE {
    ?p1 fam:gender ?g1.
    ?p2 fam:gender ?g2.
    FILTER NOT EXISTS { ?p1 fam:brother ?p2 . } .
    FILTER NOT EXISTS { ?p1 fam:wife ?p2 . } .
    FILTER ( ?p1 != ?p2 )
}
ORDER BY RAND()
LIMIT 47
```

Opnieuw definiëren we een UDF en vullen zo de relatie *labels_neg_random* in.

Heuristische regels:
Naast reeds bestaande triple stores, kunnen we ook heuristische regels gebruiken om gelabelde voorbeelden te verzamelen. Dit is zinvol wanneer de dataset niet volledig is of we een relatie willen beschouwen die niet in de dataset aanwezig is. De heuristische regels zullen typisch deductie-regels zijn. Stel bijvoorbeeld dat we de relatie $grandparent(X, Y)$ willen beschouwen om negatieve voorbeelden uit te genereren. Deze relatie kunnen we deduceren met volgende regel: $parent(X, Z) \wedge parent(Z, Y) \implies grandparent(X, Y)$.[37] Deze kunnen we eenvoudig in een SPARQL-query omzetten, en die resultaten slaan we op in een relatie *labels_neg_grandparent* bijvoorbeeld. In ons voorbeeld maken we geen gebruik van heuristische regels.

In het algemeen lijkt het ons niet zinvol om in de heuristische regels rekening te houden met de geëxtraheerde features. Herinner dat we dit bij ongestructureerde data wel deden. Onze aanpak laat dit uiteraard wel toe.

Finale labels bepalen:
We definiëren de relatie *labels* om de labels in bij te houden, als volgt:

```
labels(
    person_1   text,
    person_2   text,
    label      int
).
```

[37] We moeten niet expliciet vereisen dat $X \neq Y$ omdat alle kandidaat-paren reeds uit verschillende personen bestaan.

Voor het invullen van deze relatie, aggregeren we de tot dusver opgestelde labels:

```
labels(pers_1, pers_2,  1) :- labels_pos(pers_1, pers_2).
labels(pers_1, pers_2, -1) :- labels_neg_wife(pers_1, pers_2).
labels(pers_1, pers_2, -1) :- labels_neg_random(pers_1, pers_2).
```

Op basis van de reeds bestaande triple stores en heuristische regels, heeft men typisch een groot aantal labels weten te genereren. Net zoals wanneer men ongestructureerde data gebruikt, kunnen er tegenstrijdige labels aanwezig zijn. (Aangezien de family dataset geen tegenstrijdigheden bevat, zal dat in ons voorbeeld niet het geval zijn.) Het finale label van een kandidaat-paar bepalen we door de som te nemen van al zijn labels:

```
resolved_labels(person_1, person_2, SUM(label)) :-
    labels(person_1, person_2, label).
```

6.2.5 MLN definiëren, learning en inference

Om het MLN te definiëren, stellen we eerst de definitie van de variabele relatie *brother* op, als volgt:

```
brother?(
    person_1   text,
    person_2   text
).
```

Herinner dat DeepDive een binaire variabele koppelt aan iedere rij van een variabele relatie. Tijdens de inference zal DeepDive bepalen wat de marginale kans is dat die variabele tot de relatie *brother* behoort.

We plaatsen alle kandidaat-paren in deze relatie en specificeren de training data, als volgt:

```
brother(person_1, person_2) = if l > 0 then TRUE
                              else if l < 0 then FALSE
                              else NULL end :-
    resolved_labels(person_1, person_2, l).
```

Aangezien in onze PoC het finale label van ieder kandidaat-paar verschilt van nul, zal aan iedere variabele de waarde *true* of *false* worden toegekend; nooit *null*.

Voor de eigenlijke definitie van het MLN beperken we ons voorlopig tot het gebruik van features. In deel 6.2.7 komen we terug op het gebruik van regels met extra domeinkennis. De huidige definitie is:

```
@weight(f)
brother(person_1, person_2) :- features(person_1, person_2, f).
```

Nu we het MLN volledig gedefinieerd hebben, kunnen we overgaan tot de learning en inference. In het bestand “deepdive.conf” voorzien we de nodige configuratie:

```
deepdive.sampler.sampler_args:
    "-l 200 -i 200 -a 0.01 -d 1.00 --sample_evidence"
```

We kiezen nu bewust kleinere waarden voor het aantal iteraties dat gradient ascent doorloopt (optie `-l`) en het aantal samples dat bij de Gibbs Sampling gebruikt wordt (optie `-i`), omdat we nu met veel minder kandidaat-paren werken.

Naar de evaluatie toe (zie deel 6.2.7) configureren we alvast dat 75% van de gelabelde data als training data gebruikt moet worden en de overige 25% als test data. Daartoe voegen we aan "deepdive.conf" het volgende toe:

```
deepdive.calibration.holdout_fraction: 0.25
```

Ten slotte voeren we in de installatie-omgeving het commando `deepdive run` uit. Hiermee geven we DeepDive de opdracht om het volledige relation extraction proces te doorlopen, inclusief de learning en inference. In deel 6.2.7 bespreken we de resultaten. De totale uitvoeringstijd bedraagt 9 minuten, waarvan het bepalen van de features 8 minuten in beslag neemt.

6.2.6 Optimalisatie van aanpak

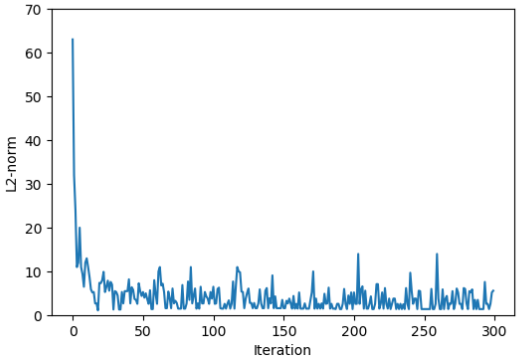
In de voorgaande delen hebben we besproken hoe de DeepDive methodologie rechtstreeks gemapt kan worden naar een scenario met gestructureerde data. Merk echter op dat we in onze PoC tijdens de distant supervision geen gebruik maakten van de reeds geëxtraheerde features. Bijgevolg kunnen we de distant supervision toepassen vóór het bepalen van de features; de volgorde van de stappen wordt nu: kandidaten bepalen, distant supervision, features bepalen, ... Herinner dat we tijdens de distant supervision slechts 118 van de 7310 kandidaat-paren labelen, 59 positieve en 59 negatieve. Aangezien we alleen deze in het MLN opnemen, is het niet nodig om ook de features van de overige kandidaat-paren te bepalen. Door dat te vermijden, verlagen we de computationele kost significant; we specificeren zo dadelijk hoe veel.

In onze PoC genereert de distant supervision 118 gelabelde kandidaat-paren, telkens met precies één label per kandidaat-paar. Aangezien alleen deze in het MLN worden opgenomen, is het niet nodig om eerst alle 7310 kandidaat-paren op te stellen. We kunnen m.a.w. beide stappen samenvoegen, wat de computationele kost verder verlaagt. Concreet voegen we aan de relatie *candidates* het veld *label* toe. Vervolgens vervangen we de UDF "generate_candidates" door een nieuwe: "candidates_and_supervise". In het gelijknamige Python-script voeren we de SPARQL-queries uit deel 6.2.4 uit. De output van deze UDF bestaat uit alle query-resultaten, waarmee we de relatie *candidates* opvullen. Om dit script eenmalig uit te voeren, gebruik we onze eerder beschreven hack. De relaties *labels_pos*, *labels_neg_wife*, *labels_neg_random* en *resolved_labels* zijn nu overbodig.

Dankzij deze twee optimalisaties bedraagt de totale uitvoeringstijd slechts 30 seconden i.p.v. 9 minuten.

6.2.7 Evaluatie

Leercurve, kansverdelingen, precision en recall:
Net zoals bij de baas-bedrijf relatie kijken we naar de leercurve, kansverdelingen, precision en recall om onze PoC te evalueren. Figuur 44 toont de leercurve; deze beschrijft de grootte van de wijziging van de gewichten per iteratie van het gradient ascent algoritme. We zien dat de grootte van de wijziging in de eerste iteraties sterk vermindert en uiteindelijk convergeert naar een kleine wijziging.



Figuur 44: De leercurve voor onze Proof of Concept.

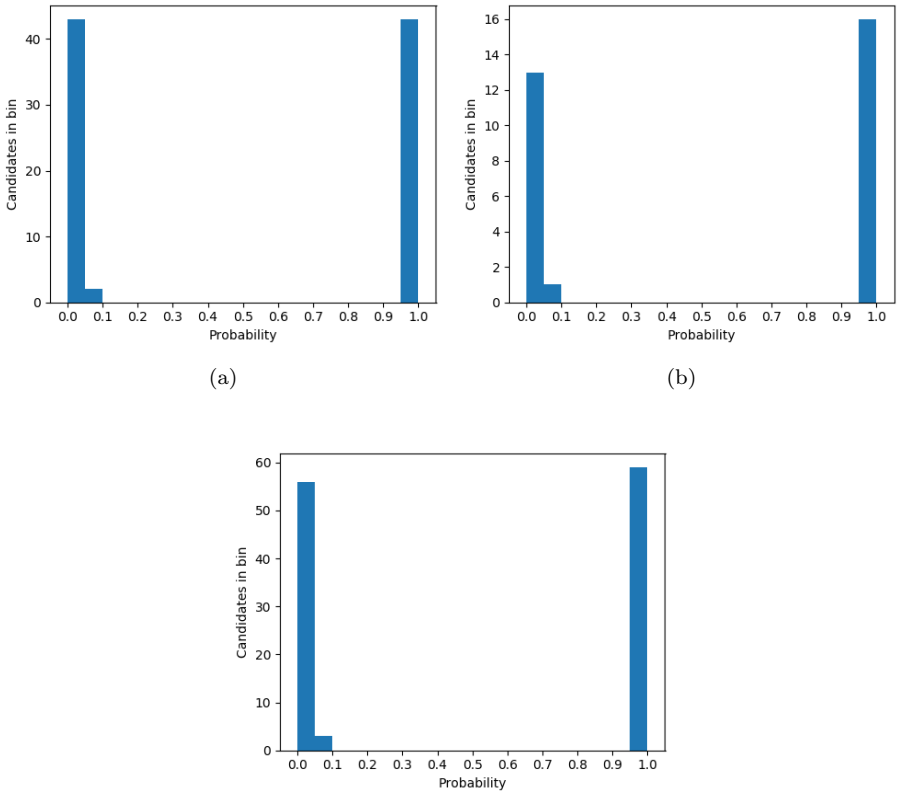
Hierdoor vermoeden we dat DeepDive goed geleerd heeft hoe het de training data moet classificeren.

Figuur 45 toont de kansverdelingen voor de training data, test data en alle data. Bij alle drie zien we een zeer hoekige U-vorm, wat erop duidt dat DeepDive zeer zeker is over zijn classificaties. Ieder kandidaat-paar heeft oftewel een zeer kleine marginale kans oftewel een zeer grote. Het lijkt aangewezen om de treshold op 0.95 in te stellen, zodat alleen kandidaat-paren met een marginale kans van minstens 0.95 geclassificeerd worden als tot de *brother* relatie behorend.

Ten slotte beschouwen we de precision en recall. Voor de training data bedragen deze beide 1.0, en voor test data eveneens! Dit is uiteraard een perfect resultaat, waardoor we concluderen dat onze aanpak zeker beloftevol is.

Voornaamste features:
Laten we nu naar de voornaamste features kijken die DeepDive geleerd heeft. Figuur 46 toont deze; voorlopig moet men de kolom 'gewicht*' negeren. Voor de eenvoud delen we de features in volgens de lengte van het pad. We beginnen met die gebaseerd op paden van lengte nul; figuur 46(a) toont deze. Beschouw voorlopig alleen de features die met het geslacht van de eerste persoon te maken hebben, i.e. de eerste en vierde feature. Het gewicht van de eerste is intuïtief correct. Voor de vierde hadden we een groot positief gewicht verwacht (i.p.v. een klein negatief gewicht), maar dat is wel degelijk logisch: de 47 willekeurige negatieve voorbeelden selecteren ook kandidaat-paren waarin de eerste persoon een man is. Aangezien een persoon oftewel een man oftewel een vrouw is, bezit ieder kandidaat-paar steeds precies één van beide features. Door het grote verschil in de grootte van beide gewichten, krijgen we wel degelijk het gewenste effect, i.e. dat de eerste persoon een man moet zijn om tot de relatie *brother* te behoren. Het geslacht van de tweede persoon (i.e. de derde en vierde feature) is irrelevant voor de classificatie; beide geslachten zijn geldig. Daarom is het logisch dat hun gewichten ongeveer gelijk zijn. Daarnaast is ook de grootte van hun gewichten logisch: deze liggen tussen die van de eerste en vierde feature, die respectievelijk een ongewenste en gewenste feature voorstellen. Hier tussenin ligt inderdaad een "neutrale" zone.

Figuur 46(b) toont de features gebaseerd op paden van lengte één. Zoals



Figuur 45: Histogram van de kansverdeling van (a) de training data, (b) de test data en (c) alle data, voor onze Proof of Concept.

#	feature	gewicht	gewicht*
1	0_1_PRED_http://family.com/gender_VAL_female	-1.289970	-1.069480
2	0_2_PRED_http://family.com/gender_VAL_male	-0.913184	-0.770283
3	0_2_PRED_http://family.com/gender_VAL_female	-0.890973	-0.893511
4	0_1_PRED_http://family.com/gender_VAL_male	-0.662900	-0.653152

(a)

#	feature	gewicht	gewicht*
1	1_1-2_PRED_http://family.com/brother	1.075170	/
2	1_2-1_PRED_http://family.com/brother	0.497255	/
3	1_1-2_PRED_http://family.com/wife	-0.216224	-0.221043
4	1_2-1_PRED_http://family.com/parent	-0.195871	-0.139132

(b)

#	feature	gewicht	gewicht*
1	2_SP-SS_PRED_http://family.com/parent	1.210130	2.0083500
2	2_SP-SO_PRED_http://family.com/gender	-0.387608	-0.2526050
3	2_SP-SS_PRED_http://family.com/brother	0.386308	/
4	2_SP-SO_PRED_http://family.com/gender_VAL_female	-0.365889	-0.2634640
5	2_SP-SO_PRED_http://family.com/brother	0.345099	/
6	2_SP-SO_PRED_http://family.com/parent	-0.335442	-0.3861620
7	2_SP-SS_PRED_http://family.com/parent_VAL_http://family.com/helen	0.210734	/
8	2_SP-SS_PRED_http://family.com/parent_VAL_http://family.com/harry	0.210734	/
9	2_SP-SO_PRED_http://family.com/gender_VAL_male	-0.205668	-0.0453379

(c)

Figuur 46: Alle features gebaseerd op paden van lengte nul (a) en één (b). De negen voornaamste features gebaseerd op paden van lengte twee (c). De kolommen 'gewicht' en 'gewicht*' zijn de gewichten van de features wanneer de *brother* triples respectievelijk wel en niet deel uitmaken van de RDF triple store.

verwacht heeft de eerste feature een groot positief gewicht. Voor de derde feature hadden we een groter negatief gewicht verwacht, maar aangezien slechts 12 van de 118 kandidaat-paren deze feature bezitten (terwijl andere features veel vaker voorkomen), is dit wel logisch. Analoog komt de vierde feature alleen voor bij enkele van de 47 willekeurige negatieve voorbeelden, waardoor deze terecht een klein negatief gewicht krijgt. De tweede feature kan voorkomen bij een positief kandidaat-paar, i.e. indien de eerste persoon een man is, alsook bij een negatief kandidaat-paar, i.e. indien de eerste persoon een vrouw is. Daarom is een vrij neutraal gewicht inderdaad logisch.

Figuur 46(c) ten slotte toont de negen voornaamste features gebaseerd op paden van lengte twee; er zijn in totaal 104 zulke features. De feature die aanduidt dat het kandidaat-paar een gemeenschappelijke ouder heeft (i.e. de eerste feature), bezit een groot positief gewicht, zoals verwacht. De zesde feature duidt aan dat het kandidaat-paar een gemeenschappelijk kind heeft, waardoor de *brother* relatie zeker niet geldt, en een negatief gewicht dus gepast is. De derde en vijfde feature duiden beide op het bestaan van een gemeenschappelijke broer, waardoor ze terecht een positief gewicht krijgen. Indien beide personen vrouwen zijn, kan de *brother* relatie onmogelijk gelden, waardoor een groot negatief gewicht gepast is voor de vierde feature. Indien beide personen echter mannen zijn, zijn het soms broers (maar niet altijd), waardoor een klein negatief gewicht inderdaad logisch is voor de negende feature. De tweede feature is de veralgemening van de vierde en de negende. Hoewel deze feature ook bij positieve voorbeelden aanwezig kan zijn, krijgt deze door zijn groter aantal voorkomens een gewicht dat overeenkomt met dat van de vierde feature. De zevende en achtste feature zijn, net als de overige 95 features, niet relevant omdat ze specifiek zijn voor deze dataset.

We hebben dus vele zinvolle features gedetecteerd en DeepDive heeft er de gepaste gewichten voor geleerd.

Evaluatie zonder de *brother* triples:
Men kan onze aanpak tot evaluatie bekritiseren door te zeggen dat deze niet echt realistisch is. Het doel van test data is om een onafhankelijke test te hebben, waarmee we kunnen inschatten hoe goed het geleerde model in staat zal zijn om andere persoonsparen te classificeren. Maar, ieder positief voorbeeld bezit de feature *1_1-2_PRED_http://family.com/brother* en geen enkel negatief voorbeeld bezit deze. Bijgevolg verraadt de aan- of afwezigheid van deze feature het antwoord al. We zouden dus een veel eenvoudigere classifier kunnen definiëren: "als het kandidaat-paar die feature bezit, dan is het antwoord *positief* en anders *negatief*"; deze zou dezelfde precision en recall behalen. In een meer realistisch scenario bevat de data zo geen features, waardoor we een lagere precision en recall zouden verwachten en onze evaluatie dus niet representatief is.

Om een representatieve test uit te voeren, verwijderen we deze feature uit de test data. Er zijn eigenlijk meerdere features aanwezig die op de *brother* triples gebaseerd zijn, bv. ook *2_SP-SO_PRED_http://family.com/brother*. Om al deze features te elimineren, moeten we simpelweg alle *brother* triples uit de test data verwijderen en pas daarna hun features bepalen. Zoals de twee onderstaande scenario's echter zullen duidelijk maken, is het beter om de *brother* triples ook uit de training data te verwijderen.

Indien DeepDive zich voor de classificatie inderdaad alleen zou baseren op de feature *1_1-2_PRED_http://family.com/brother*, zou dit zich vertalen in een

zeer groot gewicht. Zoals figuur 46 laat zien, is dat niet het geval voor ons voorbeeld; er zijn nog andere features met ongeveer even grote gewichten (en zelfs grotere). In dit scenario is de feature *1_1-2_PRED_http://family.com/brother* dus niet dominerend voor de classificatie. Door de *brother* triples te verwijderen uit de test data, bezit geen enkel persoonspaar uit de test data deze feature, ongeacht het correcte label. Bijgevolg levert deze feature geen meerwaarde voor de classificatie en kunnen we hem dus even goed volledig weglaten uit het model, door deze ook bij de training te negeren.

Indien deze feature wel een zeer groot gewicht zou hebben, domineert deze de classificatie. De afwezigheid van deze feature zorgt dat DeepDive sowieso het negatieve label voorspelt voor het persoonspaar. Bijgevolg worden alle test data als negatieve voorbeelden bestempeld, waardoor we zeer slechte resultaten bekomen. Om dit ongewenste effect te vermijden, nemen we deze feature ook tijdens het trainen niet op in het model.

In beide scenario's komen we tot de conclusie dat we de feature beter volledig weglaten uit het model, niet alleen uit de test data dus. Wanneer we het relation extraction proces nu opnieuw doorlopen, gaan we als volgt te werk. Eerst gebruiken we de oorspronkelijke dataset (mét de *brother* triples) om de gelabelde kandidaten te genereren. Vervolgens verwijderen we alle *brother* triples en bepalen we de features van de kandidaten. Aan de definitie van het MLN, het leren en de inference wijzigt uiteraard niets. De precision en recall bedragen nog steeds 1.0, zowel voor de training als de test data. De nieuwe gewichten van de features worden getoond in de kolom 'gewicht*' van figuur 46. We concluderen dat onze aanpak ook in dit meer realistisch scenario tot kwalitatieve resultaten leidt.

Impact van ruis en extra domeinkennis:
Ondertussen zijn we zeker dat onze aanpak beloftevol is. Maar zoals in deel 6.2.1 gezegd, kozen we bewust voor dit specifieke probleem omdat het ons vrij eenvoudig leek en onze aanpak op die manier meer kans op slagen had. Bijgevolg moeten we onze uitstekende resultaten enigszins nuanceren: in een complexer scenario zal de kwaliteit waarschijnlijk afnemen, wat niet wegneemt dat onze aanpak sowieso al geslaagd is in zijn opzet.

Om in te schatten in welke mate onze aanpak bestand is tegen onvolledigheden en/of tegenstrijdigheden in de dataset, besluiten we om een beetje ruis in te voeren. Concreet verwijderen we 10% van alle rijen in de relatie *features*, om voor onvolledigheden te zorgen. Tevens wijzigen we het label van 10% van de kandidaten, wat overeenkomt met 12 kandidaten. Aan 6 positieve kandidaten geven we een negatief label en vice versa, wat voor tegenstrijdigheden zorgt. Zonder ruis bedroegen de precision en recall voor de training data beide 1.0; door het introduceren van ruis blijft de precision ongewijzigd maar zakt de recall naar 0.76, wat significant is. Voor de test data bedroegen de precision en recall ook beide 1.0 in afwezigheid van ruis; nu bedragen deze respectievelijk 0.86 en 0.54. Zoals verwacht maken onvolledigheden en tegenstrijdigheden in de dataset het leerproces een stuk gecompliceerder. Dit vertaalt zich niet alleen naar de precision en recall, maar ook naar de features die DeepDive het belangrijkst vindt: deze omvatten nu meer irrelevante features.

Ten slotte verifiëren we of het introduceren van formules met extra domeinkennis een positieve impact kan hebben op de kwaliteit van de toepassing. Bij onze toepassing rond de baas-bedrijf relatie zagen we een kleine verbetering. De volgende regel lijkt ons zinvol:

De transitiviteit van de *brother* relatie:

```
@weight(1.0)
brother(person_1, person_2), brother(person_2, person_3)
=> brother(person_1, person_3) :- [person_1 != person_3] .
```

We kozen voor een gewicht van 1.0 omdat dat overeenkomt met de gewichten van de voornaamste features. Om de impact van deze regel met extra domeinkennis te evalueren, beschouwen we opnieuw het hierboven beschreven scenario, i.e. waarbij er ruis in de data aanwezig is.[38] Zowel voor de training data als voor de test data is er geen noemenswaardige impact. Mogelijks zijn andere regels of een ander gewicht meer geschikt, maar hier gaan we niet verder op in. Het belangrijkste is dat we ook dit aspect van de DeepDive methodologie voor ongestructureerde data hebben kunnen mappen naar een scenario met gestructureerde data.

6.3 Verband met MLN Structure Learning

Onze aanpak voor het mappen van de DeepDive methodologie naar een scenario met gestructureerde data is verwant aan 'MLN Structure Learning'. In dit deel leggen we uit wat MLN Structure Learning is, en bespreken we de gelijkenissen en verschillen met onze aanpak. Ten slotte tonen we via een standaard benchmark aan dat onze aanpak een succesvol alternatief kan zijn om aan MLN Structure Learning te doen.

6.3.1 Inductive Logic Programming

Vooraleer we toelichten wat MLN Structure Learning is, focussen we op een eenvoudiger aanverwant probleem, 'Inductive Logic Programming' (ILP) genaamd [39]. ILP heeft als doel om, gegeven een knowledge base (i.e. een collectie gestructureerde data) in predicaat-vorm, een logic program op te stellen dat een relatie naar keuze zo goed mogelijk beschrijft. De te beschrijven relatie noemt men de 'target' relatie. In zijn meest eenvoudige vorm genereert het ILP systeem een logic program dat uit slechts één formule bestaat. De formule die wordt opgesteld is een implicatie in eerste-orde predicatenlogica waarvan het consequent de target relatie is. Meer formeel bestaat de input uit een aantal positieve en negatieve voorbeelden voor de target relatie, tezamen met achtergrondkennis. We kunnen de oorspronkelijke family dataset als voorbeeld gebruiken. Veronderstel dat $\mathit{wife}(X, Y)$ onze target relatie is. De positieve (resp. negatieve) voorbeelden zijn groundings van het predicaat *wife* die wel (resp. niet) in de dataset voorkomen. De achtergrondkennis bestaat uit alle groundings van alle andere predicaten, tezamen met het feit of ze al dan niet in de dataset voorkomen. Op basis van die input zou een ILP systeem ons bijvoorbeeld de volgende formule kunnen leveren: $\mathit{gender}(X, \text{“}female\text{”}) \wedge \mathit{married}(X, Y) \implies \mathit{wife}(X, Y)$.

Naar het volgende deel toe is het zinvol om de algemene aanpak van een ILP systeem toe te lichten. De twee voornaamste aspecten zijn het zoeken naar kandidaat-formules en het evalueren van de kwaliteit van een kandidaat-formule

[38] Aangezien de precision en recall voor de test data in afwezigheid van ruis beide 1.0 bedroegen, zouden we de impact van de extra domeinkennis moeilijk kunnen vaststellen in dit scenario.

[39]. Qua antecedent van de kandidaat-formules (implicaties) beperken we ons tot conjuncties van predicaten. Veronderstel dat we ons tevens beperken in de lengte van de formules; bv. dat het antecedent een conjunctie van maximaal vijf predicaten mag zijn. Zelfs onder deze restricties zijn er zodanig veel mogelijke formules dat het computationeel niet haalbaar is om ze allemaal te evalueren. We moeten deze ruimte van mogelijke formules – ook wel 'search space' genoemd – op een efficiëntere (heuristische) wijze doorzoeken. In het algemeen pakt men dit als volgt aan. De initiële verzameling kandidaat-formules bestaat simpelweg uit alle mogelijke predicaten; ieder antecedent bestaat uit slechts één predicaat. Om de search space in te perken, gaan we alleen verder met de meest kwalitatieve kandidaat-formules, bv. de beste tien. De kwaliteit van een kandidaat-formule bepalen we op basis van het aantal positieve en negatieve voorbeelden dat het antecedent deduceert; idealiter worden alle positieve voorbeelden gededuceerd en geen enkel negatief voorbeeld. Zo zou bijvoorbeeld het antecedent $married(X, Y)$ veel beter geschikt zijn dan $father(X, Y)$. Vervolgens breiden we ieder van de tien beste kandidaat-formules uit met ieder van de predicaten[39], zodat de antecedenten van de nieuwe verzameling kandidaat-formules conjuncties van twee literals zijn. Opnieuw houden we alleen de 10 meest kwalitatieve over. We blijven dit proces herhalen zo lang de uitgebreide formules kwalitatiever zijn dan de die in de vorige iteratie (of de maximale lengte overschreden wordt). Uit de tien overblijvende kandidaat-formules kiezen we uiteindelijk de meest kwalitatieve als eindresultaat.

We benadrukken dat bovenstaande uitleg slechts de algemene aanpak beschrijft. De precieze invulling hangt af van het specifieke ILP systeem; ze onderscheiden zich hoofdzakelijk van elkaar door de manier waarop de search space doorzocht wordt. Daarnaast bestaan er ook vele ILP systemen die zich niet beperken tot implicaties; ze kunnen eender welke formule in eerste-orde predicatenlogica produceren als resultaat, waardoor de search space nog groter wordt. Een populair voorbeeld van zo'n systeem is CLAUDIEN [40].

6.3.2 MLN Structure Learning

Het doel van 'MLN Structure Learning' is om, gegeven een knowledge base, het MLN te bepalen dat hier zo goed mogelijk bij aansluit [41]. Men gaat dus op zoek naar die combinatie van formules (in eerste-orde predicatenlogica) en gewichten die de kans op de data in de knowledge base maximaliseert. Een eerste verschil met ILP is dat we nu een collectie formules gaan opstellen i.p.v. een logic program bestaande uit slechts één formule. Een tweede, meer fundamenteel verschil, is de manier waarop kandidaten geëvalueerd worden: bij ILP keken we naar het aantal correcte en incorrecte deducties, terwijl we nu naar de kans op de data kijken (omdat MLNs een probabilistisch aspect introduceren door gewichten te koppelen aan de formules).

We zullen nu meer formeel bespreken wat we bedoelen met de mate waarin het opgestelde MLN aansluit bij de knowledge base. Herinner dat een MLN herleid wordt tot een Markov Network met een binaire variabele voor iedere

[39] Eenvoudigheidshalve gaan we niet in op de mogelijke invullingen van de variabelen van predicaten. Voor de conjunctie van de binaire predicaten *mother* en *father* zijn er vele mogelijkheden, bv. $mother(X, Y) \wedge father(X, Y)$ of $mother(X, Y) \wedge father(Y, X)$. Daarenboven moet het ook mogelijk zijn om een nieuwe variabele te introduceren, bv. $mother(X, Z) \wedge father(Z, Y)$. Hierdoor is de search space dus nog groter dan tot dus ver beschreven.

mogelijke grounding van ieder predicaat. De knowledge base beschrijft een specifieke mogelijke wereld: variabelen die overeenkomen met aanwezige gegronde predicaten hebben de waarde *true* en de rest *false*. Het precieze doel is om dat MLN te bepalen waarvoor de kans op de knowledge base (i.e. die specifieke wereld) maximaal is. In de praktijk onderscheidt men twee vormen van MLN Structure Learning: 'generative' en 'discriminative' [42, 43]. Onze uitleg tot dus ver betreft het eerste geval. Bij discriminative MLN Structure Learning ligt de focus echter op één predicaat naar keuze. Er wordt alleen rekening gehouden met de variabelen die overeenkomen met groundings van dat ene predicaat; de overige variabelen stellen evidence voor. Het doel is dan om een MLN te bekomen dat de kans op dat deel van de knowledge base maximaliseert.

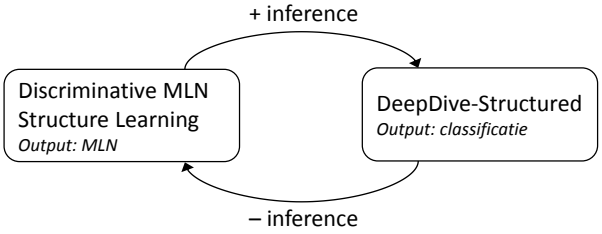
Om een geschikt MLN te vinden, moet de MLN Structure Learner (1) op zoek gaan naar de gepaste collectie formules, wat vergelijkbaar is met het doorzoeken van de search space bij ILP, en (2) een gepast gewicht bepalen voor iedere formule, waar men gradient ascent voor gebruikt (of een variant hierop). Toen Richardson en Domingos [18] in 2006 de notie van Markov Logic Networks voorstelden, beschreven ze tevens een zeer eenvoudige aanpak voor (generative) MLN Structure Learning. Ze pakten beide punten apart aan: eerst gebruikten ze een bestaand ILP systeem (meer bepaald CLAUDIEN [40]) om een collectie formules te bekomen, en vervolgens bepaalden ze de gepaste gewichten voor deze formules m.b.v. gradient ascent.

In later wetenschappelijk werk realiseerde men zich dat deze aanpak niet optimaal is [41]. Herinner namelijk het eerder vermelde fundamentele verschil tussen MLN Structure Learning en ILP: bij MLN Structure Learning is het de bedoeling om de kans op de knowledge base te maximaliseren, maar bij ILP wordt het aantal correcte/incorrecte deducties gemaximaliseerd. De recentere algoritmes focussen wel op die kans, en hanteren daarbij in het algemeen volgende aanpak [41, 42, 43]. In iedere iteratie stelt men een aantal kandidaat-MLNs op. Zo'n kandidaat-MLN wordt opgebouwd door eerst de gepaste formules te verzamelen en er vervolgens gepaste gewichten voor te leren. Door meteen de gepaste gewichten te leren, wordt het MLN geëvalueerd op basis van zijn kans op de data, zoals gewenst. Net als bij ILP worden alleen de meest kwalitatieve kandidaten behouden, bv. de tien beste kandidaat-MLNs. Iedere iteratie worden de kandidaat-MLNs uit de vorige iteratie verbeterd, zodat we uiteindelijk een geschikt MLN bekomen.

6.3.3 Gelijkenissen en verschillen

Gemakkelijkheidshalve korten we "onze aanpak voor het toepassen van de DeepDive methodologie op gestructureerde data", zoals in deel 6.2 besproken, af tot "DeepDive-Structured".

Nu we de nodige achtergrond over MLN Structure Learning hebben toegelicht, kunnen we het verband met DeepDive-Structured bespreken. We beginnen met het vergelijken van de input van beide. Bij MLN Structure Learning is dit een knowledge base in predicaat-vorm, en bij DeepDive-Structured is het een collectie triples. Beide krijgen dus een collectie gestructureerde data als input, alleen het data-formaat verschilt, maar dat is geen fundamenteel verschil. In onze PoC bv. vormden we de oorspronkelijke family dataset, die in predicaat-vorm staat, om tot een collectie triples (zie deel 6.2.1). We concluderen dat de input van MLN Structure Learning en DeepDive-Structured identiek is, name-

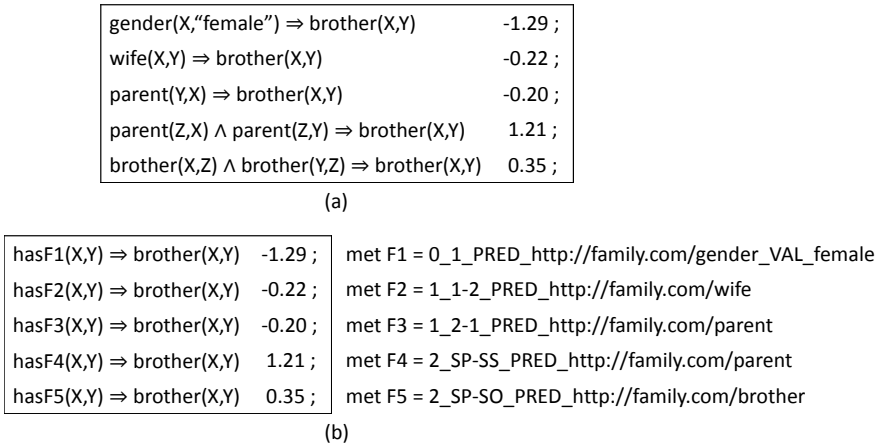


Figuur 47: Het verband tussen discriminative MLN Structure Learning en DeepDive-Structured. Deze zijn equivalent mits het toevoegen of weglaten van de inference-stap.

lijk een collectie gestructureerde data.

Vervolgens vergelijken we de outputs van beide. Bij MLN Structure Learning is dit een MLN dat nauw bij de data aansluit. Het stelt een compact, high-level model van de data voor. In het voorgaande deel hebben we niet besproken wat het nut daar eigenlijk van is. Herinner dat een MLN herleid kan worden naar een Markov Network met een binaire variabele voor iedere grounding van ieder predicaat. Via inference kunnen we de marginale kans berekenen dat zo'n variabele de waarde *true* aanneemt, en het overeenkomstig geground predicaat p dus geldt. Bijgevolg kennen we de kans dat de relationele info die p uitdrukt, klopt volgens het MLN. Op die manier kunnen we de bestaande collectie gestructureerde data verder aanvullen en eventuele fouten corrigeren. Indien bijvoorbeeld aan $p = getrouwd(Barack, Michelle)$ een hoge marginale kans gekoppeld wordt maar dit geground predicaat niet in de data aanwezig is, kunnen we besluiten om dit toe te voegen. En omgekeerd kunnen we, bij een kleine marginale kans, besluiten om een geground predicaat te verwijderen. Merk op dat we het MLN nu eigenlijk aan het gebruiken zijn voor classificatie: afhankelijk van de kans voor een geground predicaat zeggen we dat het desbetreffende tupel al dan niet deel uitmaakt van de desbetreffende relatie. Indien we aan discriminative MLN Structure Learning doen, en daarbij bv. op het predicaat *getrouwd* focussen, gebruiken we de marginale kansen om te bepalen welke persoonsparen tot de relatie *getrouwd* behoren. Dus, als we eerst een MLN leren m.b.v. discriminative MLN Structure Learning en er vervolgens inference op toepassen, doen we aan classificatie. Ook bij DeepDive-Structured stellen we een MLN op en doen we aan inference om uiteindelijk de tupels te classificeren als wel of niet tot een bepaalde relatie behorend. In dat opzicht hebben we nu twee verschillende benaderingen gezien om aan classificatie te doen in de context van gestructureerde data. De bovenste helft van figuur 47 visualiseert dit verband tussen beide. In de toekomst kan men onderzoeken hoe effectief de aanpak met MLN Structure Learning werkelijk is, en i.h.b. of deze betere resultaten levert dan DeepDive-Structured.

We kunnen bovenstaande redenering ook omkeren, als volgt. Zoals reeds gezegd levert discriminative MLN Structure Learning een MLN als output. Indien we in DeepDive-Structured de inference-stap zouden weglaten, is het eindresultaat ook hier een MLN. We hebben dus twee verschillende benaderingen gezien voor het leren van een MLN in de context van gestructureerde data. De onderste helft van figuur 47 visualiseert dit verband. Indien DeepDive-Structured zonder inference een kwalitatief MLN oplevert, zou dit een waardevol alternatief



Figuur 48: (a) Enkele mogelijke formules uit het MLN bekomen door discriminative MLN Structure Learning met de focus op de relatie $brother(X, Y)$. (b) Enkele formules uit het MLN dat we in deel 6.2 hebben bekomen.

kunnen zijn voor discriminative MLN Structure Learning. Op deze vraag gaan we nu verder in.

Om de MLNs die we volgens beide benaderingen verkrijgen met elkaar te vergelijken, gebruiken we opnieuw onze PoC uit deel 6.2. Indien we discriminative MLN Structure Learning zouden toepassen op de family dataset met de focus op $brother(X, Y)$, zou het resulterende MLN o.m. de formules in figuur 48(a) kunnen bevatten. Vervolgens beschouwen we het MLN dat DeepDive-Structured bekomt. Voor iedere unieke feature f voegt DeepDive automatisch een formule toe aan het MLN, namelijk: $(\forall X\ \forall Y\ brother(X, Y);\ w_f)$. De groundings van het predicaat $brother(X, Y)$ zijn alle persoonsparen die feature f bezitten. Zoals in deel 4.7.4 aangetoond, mogen we deze formule vervangen door: $(\forall X\ \forall Y\ hasFeature\_f(X, Y) \Rightarrow brother(X, Y);\ w_f)$, waarbij de groundings van $brother(X, Y)$ nu over alle persoonsparen gaan. We gaan uit van die laatste vorm omdat we het verband tussen beide MLNs dan eenvoudiger kunnen bespreken. Figuur 46 toont de voornaamste features die gedetecteerd werden. Het MLN bevat dus o.m. de formules die figuur 48(b) toont.

Uiteraard zijn de gewichten van de formules in het eerste MLN niet toevallig gelijk aan die in het tweede MLN. We hebben deze bewust zo gekozen omdat we willen aantonen dat de formules van beide MLNs eigenlijk dezelfde zijn. Beschouw de eerste formule van het MLN in figuur 48(a); wanneer we deze in woorden omzetten, verkrijgen we: "*als bij een persoonspaar* (X, Y) *persoon* X *een vrouw is, dan is* X *de broer van* Y". De eerste formule van het MLN in figuur 48(b) lezen we als: "*als een persoonspaar* (X, Y) *feature F1 bezit, dan is* X *de broer van* Y". Feature *F1* drukt op zijn beurt uit: "*de eerste persoon van het persoonspaar is een vrouw*", waardoor we inzien dat beide formules dezelfde betekenis hebben. Meer formeel stellen we vast dat we *F1* in predicaat-vorm kunnen schrijven als $gender(X, "female")$, voor een persoonspaar (X, Y). Voor alle duidelijkheid geven we nog een tweede voorbeeld, met betrekking tot de laatste formule uit beide MLNs. Feature *F5* betekent "*beide personen van het persoonspaar zijn de broer van een derde persoon*", wat in predicaat-vorm

neerkomt op $brother(X, Z) \wedge brother(Y, Z)$ voor een persoonspaar (X, Y). De laatste formule van het MLN in figuur 48(a) komt hier inderdaad mee overeen. De graafpatronen die we detecteren bij DeepDive-Structured laten zich dus vertalen naar patronen over formules in eerste-orde predicatenlogica. Zo komt bv. een feature `2_SP-SO_PRED_<p>` overeen met $<p>(X, Z) \wedge <p>(Y, Z)$ voor een predicaat $<p>$; analoog voor de andere features.

Met dit voorbeeld toonden we aan dat het MLN opgesteld door DeepDive-Structured in principe even kwalitatief kan zijn als het MLN opgesteld door discriminative MLN Structure Learning. DeepDive-Structured lijkt bijgevolg een zinvolle alternatieve aanpak. We zullen nu bespreken hoe beide benaderingen van elkaar verschillen. Zoals reeds gezegd, wordt bij discriminative MLN Structure Learning de search space van mogelijke formules op heuristische wijze doorzocht. Er worden meerdere kandidaat-MLNs opgesteld en hun formules worden iteratief verbeterd. Bij DeepDive-Structured daarentegen ligt het vast hoe we tot de collectie formules komen: we gaan in de data op zoek naar specifieke patronen (bv. $<p>(X, Z) \wedge <p>(Y, Z)$), en voegen iedere aanwezige invulling toe aan het MLN. Bij DeepDive-Structured is er dus geen sprake van het doorzoeken van de search space, waardoor de computationele kost lager ligt. Aangezien deze aanpak geen rekening houdt met hoe goed de formules bij de data aansluiten, verwachten we dat het resulterende MLN in het algemeen minder geschikt zal zijn. De huidige patronen om formules te selecteren, beschouwen alleen zeer korte formules (implicaties), waarbij het antecedent uit één predicaat bestaat of de conjunctie van twee predicaten. Discriminative MLN Structure Learning daarentegen laat langere formules toe, die complexere verbanden uitdrukken. Vanuit een optimistisch standpunt is onze aanpak dus 'eenvoudiger' dan die van discriminative MLN Structure Learning; vanuit een pessimistisch standpunt kan men van 'simplistischer' spreken. Het volgende deel zal hier meer duidelijkheid rond scheppen door onze aanpak op een standaard benchmark uit te testen.

6.3.4 De UW-CSE *AdvisedBy* benchmark

In het voorgaande deel hebben we aangetoond dat DeepDive-Structured zonder inference een beloftevolle alternatieve aanpak is voor discriminative MLN Structure Learning: het geleerde MLN hield intuïtief steek en sloot goed aan bij de data. In dit deel onderwerpen we onze aanpak aan een standaard benchmark binnen MLN Structure Learning. Op die manier zijn we zeker dat we een relevante dataset hanteren, en kunnen we onze resultaten vergelijken met die van de huidige state-of-the-art algoritmes om het praktisch nut van onze aanpak te beoordelen.

Eén van de standaard benchmarks voor het evalueren van een MLN Structure Learning algoritme is de UW-CSE *AdvisedBy* benchmark [18, 41, 42, 43]. De 'UW-CSE' dataset beschrijft het Computer Science and Engineering departement van de University of Washington (UW-CSE). Tabel 4 toont de relaties in deze dataset.[40] De bedoeling is om een MLN te leren dat deze dataset beschrijft, met de focus op de relatie *advisedBy*. De UW-CSE dataset is erg onvolledig volgens de auteurs [18], waardoor dit probleem onze aanpak meer op de proef zal

[40] In principe zijn er nog enkele andere predicaten, maar deze worden niet gebruikt door de eigenlijke benchmark en vermelden we daarom niet.

Relatie	**Betekenis**	**Aantal voorkomens**
$professor(X)$	Persoon X is een professor.	62
$position(X, Y)$	Professor X heeft positie Y (gewoon, emeritus, gastdocent, ...).	52
$student(X)$	Persoon X is een student.	216
$phase(X, Y)$	Student X zit in fase Y van zijn opleiding (pre_Quals, post_Quals of post_Generals).	140
$yearsInProgram(X, Y)$	Student X is al Y jaar bezig aan zijn opleiding.	140
$advisedBy(X, Y)$	Persoon X heeft persoon Y als promotor voor zijn doctoraat.	113
$tempAdvisedBy(X, Y)$	Persoon X heeft persoon Y als tijdelijke promotor voor zijn doctoraat.	37
$publication(P, X)$	Persoon X is een auteur van paper/publicatie P.	734
$taughtBy(C, P, Q)$	Vak C wordt gedoceerd door persoon P (professor) in kwartiel Q.	286
$TA(C, P, Q)$	Vak C wordt geassisteerd (Teaching Assistent) door persoon P (student) in kwartiel Q.	195
$courseLevel(C, L)$	Vak C is van niveau L (intro, undergraduate, advanced undergraduate, ...)	132

Tabel 4: Informatie over relaties in de 'UW-CSE' dataset.

stellen dan de family dataset.

Implementatie:
Aangezien de implementatie analoog is aan die in deel 6.2, beperken we ons tot de voornaamste aspecten. De dataset staat in predicaat-vorm, waardoor we deze nog moeten omvormen naar een RDF triple store (analoog aan deel 6.2.1). In tegenstelling tot de family dataset zijn er niet alleen binaire predicaten. De unaire predicaten $professor(X)$ en $student(X)$ vormen we om tot respectievelijk $role(X, \text{“}professor\text{”})$ en $role(X, \text{“}student\text{”})$. Het ternaire predicaat $taughtBy(C, P, Q)$ delen we op in $taughtBy(C, P)$ en $taughtIn(C, Q)$; analoog voor $TA(C, P, Q)$. Door een ternair predicaat op te splitsen, passen we de data eigenlijk aan, maar we vermoeden dat de impact op de kwaliteit van het MLN beperkt zal zijn.

Vooraleer we kunnen toelichten hoe we de gelabelde kandidaten bepalen, moeten we vooruitblikken op de evaluatie. De data in de UW-CSE dataset is opgedeeld in vijf domeinen: 'AI', 'graphics', 'language', 'systems' en 'theory'. De data uit vier van de vijf domeinen vormt de training data en de data uit het resterende domein vormt de test data; dit noemt men 'leave-one-out' testing [18]. Aangezien de vijf verschillende opdelingen van de data geëvalueerd zullen worden, spreekt men ook van '5-fold cross-validation' [43]. Om de evaluatie te vergemakkelijken, definiëren we per domein een specifieke relatie voor de kandidaten, bv. *candidates_ai*. Net zoals in deel 6.2 gebruiken we alle voorkomens in

de target relatie (*advisedBy*) als positieve voorbeelden, en selecteren we evenveel andere persoonsparen als negatieve voorbeelden. De vijf deelrelaties bevatten respectievelijk 70, 40, 18, 66 en 32 kandidaten. Tenslotte stellen we de relatie *candidates* op, bestaande uit alle kandidaten, door de vijf deelrelaties samen te voegen.

Herinner dat we bij de evaluatie van de *brother* relatie (zie deel 6.2.7) de *brother* triples negeerden, om een meer representatieve test te bekomen. Tevens motiveerden we waarom deze triples niet alleen bij de test data maar ook bij de training data genegeerd moeten worden.[41] Om dezelfde redenen verwijderen we alle *advisedBy* triples uit de dataset vooraleer we de features bepalen. De eigenlijke extractie van de features blijft ongewijzigd omdat de graafpatronen toepassing-onafhankelijk zijn (zie deel 6.2.3).

In de definitie van het MLN gebruiken we alleen de features; we gebruiken geen extra domeinkennis omdat het net de bedoeling is om volledig automatisch een MLN op te stellen. Afgezien van enkele praktische aanpassingen m.b.t. de evaluatie, verliep onze implementatie dus identiek aan die uit deel 6.2.

Evaluatie:
Zoals reeds gezegd evalueert men deze benchmark volgens een 5-fold cross-validation. Voor ieder van de vijf opdelingen moeten we dus de learning en inference herhalen. We beschrijven de nodige stappen voor het geval dat 'AI' de test data voorstelt en de training data uit de overige data bestaat; dit noemen we de 'AI-fold'. In "deepdive.conf" configureren we welke data DeepDive moet gebruiken als test data. Tot nog toe selecteerden we hier steeds 25% van alle gelabelde data voor, door een `holdout_fraction` van 0.25 in te stellen. Nu willen we echter dat de data in *candidates_ai* gebruikt wordt als test data. DeepDive laat de gebruiker toe om zelf een query te specificeren die de test data selecteert, als volgt:

```
deepdive.calibration.holdout_query: """
    INSERT INTO dd_graph_variables_holdout(variable_id)
    SELECT dd_id
    FROM candidates_ai NATURAL JOIN dd_variables_advised_by
"""
```

In de kolom *dd_id* van de relatie *dd_variables_advised_by* houdt DeepDive een uniek ID bij voor iedere variabele in het Markov Network (voor ieder kandidaattupel dus). Via de natural join komen we de ID's te weten van de variabelen die overeenkomen met de persoonsparen in *candidates_ai*. In de relatie *dd_graph_variables_holdout* stockeert DeepDive de ID's van die variabelen die als test data dienst doen.[42]

Zodra de learning en inference voltooid zijn, gaan we over tot de eigenlijke evaluatie van de AI-fold. We beperken ons tot de voornaamste evaluatiemetrieken. Figuur 50(a) toont de kansverdeling van de training data. DeepDive lijkt minder zeker over zijn classificaties dan in onze voorgaande toepassingen; een treshold van 0.80 lijkt aangewezen. De precision en recall voor de training

[41] In deze benchmark hebben we experimenteel vastgesteld dat de classificatie inderdaad gedomineerd zou worden door de feature `1_1-2_PRED_http://uw-cse.com/advisedBy`. Deze had een gewicht van 2.96 terwijl de gewichten van de overige features maximaal 1.08 bedroegen.

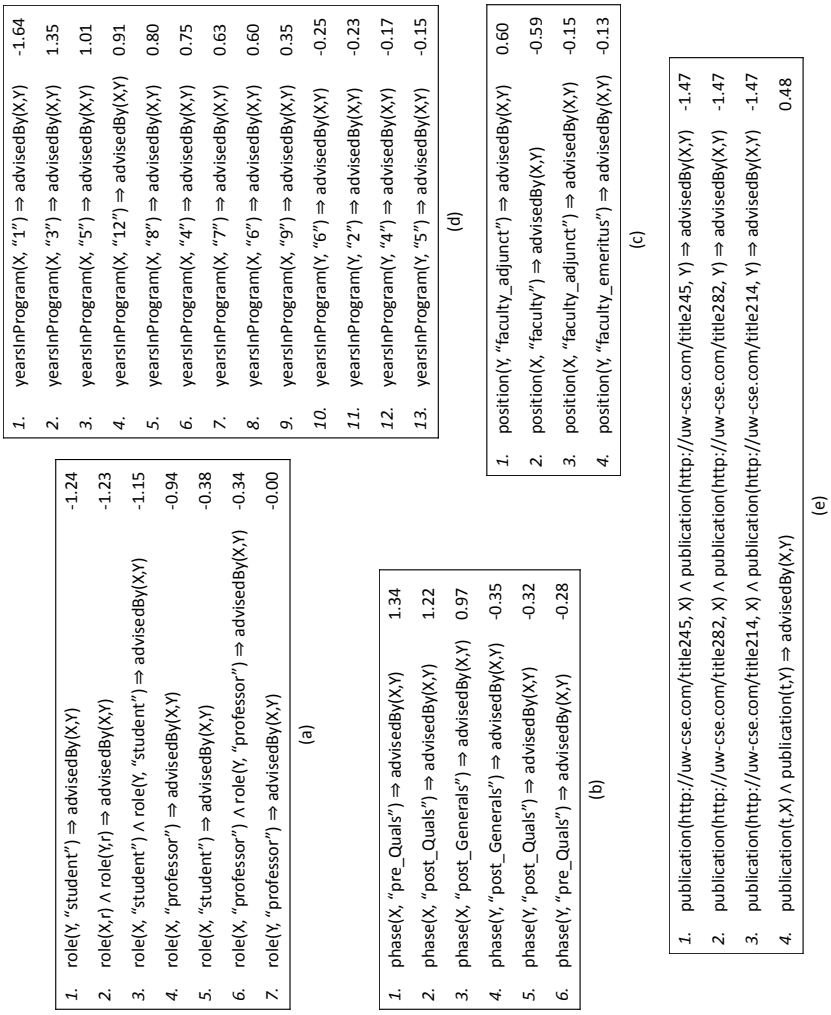
[42] Ook wanneer we met een `holdout_fraction` werken, vult DeepDive deze relatie in met de ID's van de willekeurig gekozen variabelen.

data bedragen dan respectievelijk 0.96 en 1.00, en voor de test data bedragen ze 0.85 en 0.94. Dit lijken ons zeer kwalitatieve resultaten; we zullen ze zo dadelijk vergelijken met die van de huidige state-of-the-art.

In het geleerde MLN verwachten we formules terug te zien die uitdrukken dat de eerste persoon een student moet zijn en de tweede een professor. Tevens verwachten we dat ze samen een paper gepubliceerd moeten hebben, opdat de *AdvisedBy* relatie kan gelden. Figuur 49 toont de voornaamste geleerde formules en hun gewichten, opgedeeld per predicaat. De formules gebaseerd op het predicaat *role* (zie figuur 49(a)) beschrijven inderdaad dat de eerste persoon een student moet zijn en de tweede een professor. Figuur 49(b) toont de formules gebaseerd op het predicaat *phase*: deze beschrijven dat de eerste persoon in één van de drie fases van zijn opleiding moet zitten, wat inderdaad impliceert dat het een student betreft; en de tweede persoon mag in geen enkele fase zitten, wat inderdaad impliceert dat het een professor betreft. Deze formules houden dus intuïtief steek. De formules gebaseerd op het predicaat *position* (zie figuur 49(c)) beschrijven een vergelijkbaar verband: aangezien alleen professoren een positie hebben aan de universiteit, mag alleen de tweede persoon er één bezitten. De vierde formule lijkt een foutief gewicht te hebben, maar een emeritus professor kan inderdaad geen promotor (meer) zijn voor een doctoraat. Analoog is het predicaat *yearsInProgram* alleen van toepassing op studenten. De formules in figuur 49(d) beschrijven inderdaad dat de eerste persoon een aantal jaar bezig mag zijn aan zijn opleiding en de tweede persoon niet; de eerste formule is hier een uitzondering op, waar we geen verklaring voor hebben. Volgens de vierde formule in figuur 49(e) moet het persoonspaar samen een paper gepubliceerd hebben opdat de eerste persoon de promotor kan zijn van de tweede, wat we inderdaad verwacht hadden. Merk op dat ook twee professoren samen een paper kunnen publiceren, wat kan verklaren waarom het gewicht wat lager ligt dan we intuïtief zouden verwachten. De overige formules op basis van het predicaat *publication* hebben betrekking tot specifieke publicaties en zijn dus niet relevant, ook al hebben deze soms hoge gewichten. Er bestaan geen formules gebaseerd op de predicaten *tempAdvisedBy*, *taughtIn*, *TAdIn* en *courseLevel*. De formules gebaseerd op de predicaten *taughtBy* en *TAdBy* hebben geen grote gewichten en zijn dus geen voorname features.

Tot dus ver hebben we alleen nog maar de AI-fold geëvalueerd. We herhalen het proces nu voor ieder van de vier resterende folds. Figuur 50 toont hun kansverdelingen voor de training data. We zien geen significante verschillen tussen de verschillende folds onderling, en globaal gezien lijkt een treshold van 0.80 gepast om een hoge precision te garanderen. Tabel 5 geeft een overzicht van de precision en recall per fold; het gerapporteerde gemiddelde is gewogen volgens het aantal kandidaten per domein. Onze aanpak behaalt een precision en recall van ongeveer 0.90 voor de test data, wat uitstekende resultaten zijn. De voorname formules van iedere fold zijn vergelijkbaar met die van de AI-fold (zie figuur 49) en tonen we daarom niet opnieuw.

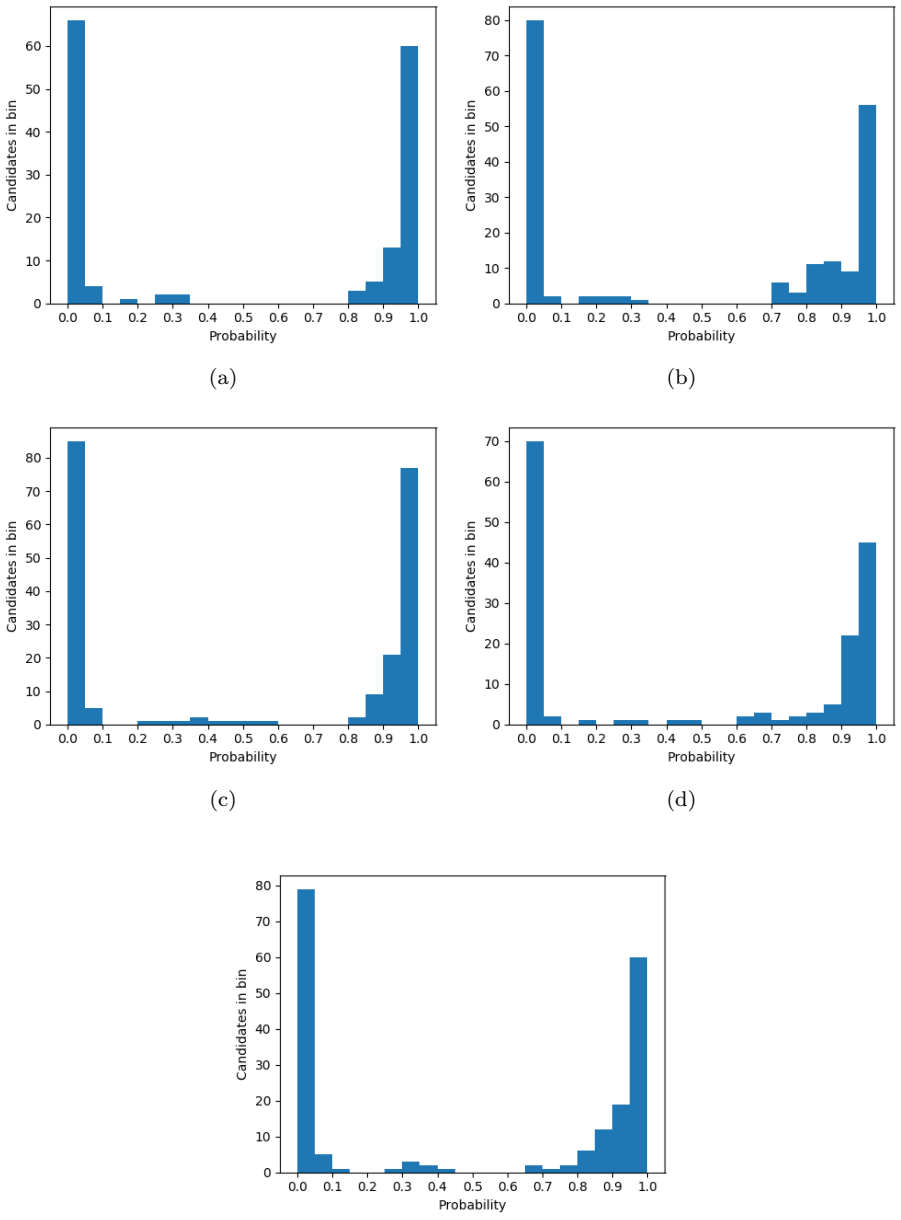
Onze aanpak heeft een MLN opgesteld dat intuïtief betekenisvolle formules bevat met gepaste gewichten. Tevens tonen de precision en recall aan dat het opgestelde MLN nauw bij de data aansluit. We concluderen dan onze aanpak een kwalitatief alternatief is voor de bestaande discriminative MLN Structure Learning algoritmes.



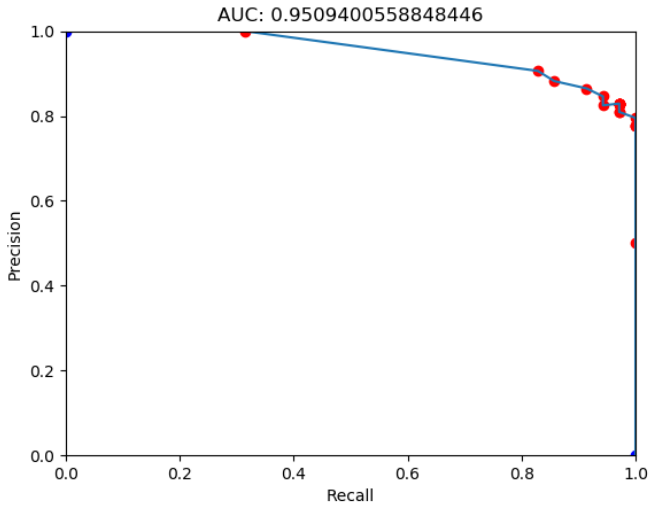
Figuur 49: De voornaamste formules die geleerd werden voor de *advisedBy* benchmark. De formules gebaseerd op het predicaat (a) *role*, (b) *phase*, (c) *position*, (d) *yearsInProgram* en (e) *publication*.

Fold	Training		Test	
	Precision	Recall	Precision	Recall
AI	0.96	1.00	0.85	0.94
graphics	0.97	0.91	0.95	1.00
language	0.94	0.99	1.00	1.00
systems	0.97	0.91	0.90	0.82
theory	0.95	0.95	1.00	0.75
Gemiddelde	0.96	0.95	0.92	0.89

Tabel 5: Overzicht van de precision en recall voor de training en test data.



Figuur 50: De kansverdelingen van de training data bij de (a) AI-fold, (b) graphics-fold, (c) language-fold, (d) systems-fold en (e) theory-fold.



Figuur 51: De precision-recall curve en de AUC voor de AI-fold.

Evaluatie volgens de AUC:
Om de performantie van onze aanpak te kunnen vergelijken met die van de state-of-the-art algoritmes, zijn we genoodzaakt een andere evaluatie-metriek te gebruiken. Standaard wordt namelijk de 'Area Under the precision-recall Curve' (AUC) gebruikt om de *advisedBy* benchmark te evalueren [43]. De AUC is een veelvuldig gebruikte evaluatie-metriek, en berekent men als volgt. Ten eerste stelt men de 'precision-recall curve' op; de x-as komt overeen met de recall en de y-as met de precision. Men laat de treshold voor de classificatie variëren tussen 0.0 en 1.0, en berekent telkens de bijhorende precision en recall. Op die manier bekomt men een collectie $(recall, precision)$ paren, die punten op de curve specificeren. Door vervolgens de opeenvolgende punten met elkaar te verbinden, verkrijgt men de precision-recall curve. Figuur 51 geeft een voorbeeld van een typisch verloop. Naarmate we de treshold verhogen, neemt de precision typisch toe maar verlaagt de recall; in de limiet bekomen we hierdoor het punt $(0.0, 1.0)$. Analoog bekomen we voor een lage treshold in de limiet typisch het punt $(1.0, 0.0)$. In het ideale scenario zorgen tresholds in de buurt van 0.5 voor een precision en recall die beide bijna 1.0 bedragen, waardoor de curve in de buurt van de rechterbovenhoek komt. De AUC is ten slotte simpelweg gedefinieerd als de oppervlakte onder de precision-recall curve; idealiter bedraagt deze 1.0.

Voor zover we weten dateert het huidige state-of-the-art algoritme voor discriminative MLN Structure Learning uit 2010. Dinh [43] heeft toen als deel van zijn doctoraat het algoritme genaamd DMSP (Discriminative MLN Structure learning based on Propositionalization) ontwikkeld. Voor de UW-CSE *AdvisedBy* benchmark behaalde hij een AUC van 0.264. Gebruikmakend van onze aanpak bekomen we een gemiddelde AUC van 0.938, gewogen volgens het aantal kandidaten per domein. Tabel 6 geeft een overzicht van de AUC's voor de test data per fold. De eerder vermelde figuur 51 visualiseert in feite de precision-recall curve voor de AI-fold; die voor de andere folds zijn vergelijkbaar. Op

Fold	AUC
AI	0.95
graphics	0.97
language	1.00
systems	0.86
theory	1.00
Gemiddelde	0.94

Tabel 6: Overzicht van de AUC voor de test data.

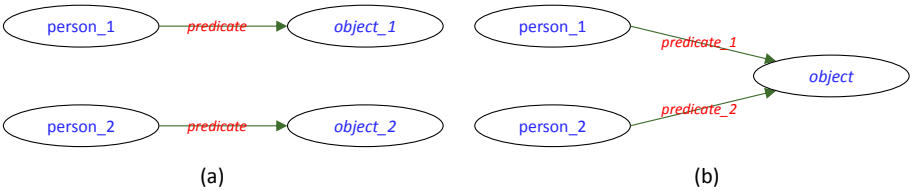
basis van de AUC's zouden we concluderen dat onze aanpak extreem veel beter presteert dan de huidige state-of-the-art. Maar zoals in deel 6.3.3 toegelicht, is onze aanpak om de search space te doorzoeken eenvoudiger dan die van de typische aanpak, waardoor we net minder kwalitatieve resultaten verwachten dan bij de state-of-the-art algoritmes. Het extreme verschil in AUC's is dus onrealistisch, waardoor we vermoeden dat onze evaluatie toch ergens verschilt van die van Dinh. Desalniettemin blijft onze eerdere conclusie geldig: onze aanpak heeft een gepast MLN geleerd voor deze benchmark en is dus geschikt om aan discriminative MLN Structure Learning te doen.

6.4 Conclusies en toekomstig werk

We zijn er als eerste in geslaagd om de DeepDive methodologie rechtstreeks te mappen op een scenario waarin met gestructureerde data gewerkt wordt i.p.v. met ongestructureerde data. Voor iedere stap hebben we beschreven hoe deze vertaald kan worden, en dit telkens verduidelijkt a.d.h.v. ons lopend voorbeeld omtrent familiale relaties. Het spreekt niet voor zich hoe men features kan extraheren uit gestructureerde data, maar onze suggestie om de data als graaf te zien en een aantal algemene (toepassing-onafhankelijke) graafpatronen te detecteren, leidt tot kwalitatieve features. Via onze geslaagde Proof of Concept toonden we aan dat er veel potentieel in onze aanpak zit.

Daarnaast stelden we vast dat deze aanpak ook gebruikt kan worden om aan discriminative MLN Structure Learning te doen, door de feature-gebaseerde formules om te vormen tot hun overeenkomstige predicaat-gebaseerde formules. Hoewel onze aanpak relatief eenvoudig is t.o.v. de reeds bestaande algoritmes, behaalden we uitstekende resultaten voor een standaard benchmark (nl. de UW-CSE *AdvisedBy* benchmark): het opgestelde MLN bestond uit formules die intuïtief relevant zijn en sloot zeer nauw aan bij de dataset. Onze aanpak blijkt dus een bruikbaar alternatief te zijn.

Naar toekomstig werk toe suggereren we om verder onderzoek te voeren naar graafpatronen om features mee te genereren. In onze aanpak beperkten we ons tot paden van lengte nul, één of twee, maar misschien blijken langere paden eveneens zinvol. Daarnaast zijn er nog vele varianten van deze paden mogelijk, die we ook niet uitgeprobeerd hebben. Beschouw in de context van onze PoC een kandidaat-paar waarbij de personen de rol van *subject* op zich nemen in de triples. We zouden dan een aantal features kunnen bedenken op basis van "`SP-DO`" (Same Predicate - Different Object); figuur 52(a) toont dit algemene graafpatroon. Hiermee bekomt men bv. de feature dat twee personen een ver-



Figuur 52: Suggesties voor andere algemene graafpatronen: (a) het "SP-DO" patroon en (b) het "DP-SO" patroon

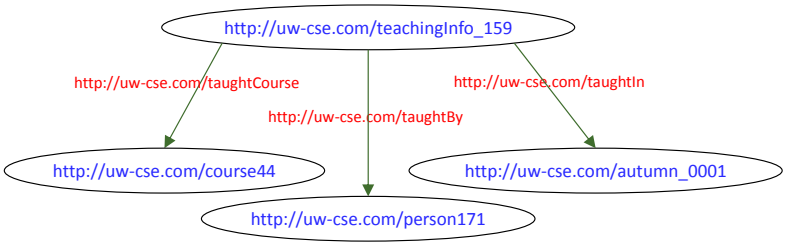
schillend geslacht hebben, wat nuttig kan zijn bij het extraheren van de relatie $married(X, Y)$. Figuur 52(b) toont een tweede suggestie; dit graafpatroon noemen we "DP-SO" (Different Predicate - Same Object). Wanneer een predicaat in meerdere talen aanwezig is in de dataset, bv. "parent" en "ouder", kan men op basis van dit patroon features opstellen om alsnog te weten te komen dat het persoonspaar een gemeenschappelijk kind heeft. Er zijn dus nog enorm veel graafpatronen waarvan het nut onderzocht kan worden.

Een tweede suggestie naar toekomstig werk is een toepassing te realiseren waarbij de data afkomstig is uit de LOD Cloud. Op die manier werkt men met een grotere dataset, waarin onvolledigheden en tegenstrijdigheden aanwezig zijn. In zo'n toepassing zullen de heuristische regels een belangrijkere rol krijgen en zal men ongetwijfeld meerdere iteraties nodig hebben om tot kwalitatieve resultaten te komen. Ook zullen eventuele tekortkomingen in onze aanpak aan het licht komen. Vervolgens kan men onze aanpak gebruiken om de LOD Cloud daadwerkelijk aan te vullen met nieuwe relaties, gebaseerd op de reeds aanwezige gestructureerde data.

We merkten dat onze aanpak tot feature extraction steeds voor een aantal features/formules zorgt die eigenlijk niet relevant zijn maar toch een hoog gewicht krijgen. Bij de *brother* PoC was dat bijvoorbeeld een concrete gemeenschappelijke ouder, en bij de *AdvisedBy* benchmark een concrete gemeenschappelijke publicatie. Om zulke ongewenste features/formules te vermijden, suggereren we om na de feature extraction alleen die features te behouden die relatief frequent voorkomen. Het spreekt echter niet voor zich hoe men tot de gewenste balans komt: indien men te strikt is, worden ook relevante features genegeerd en indien men niet strikt genoeg is, blijven nog te veel irrelevante features over. Hier kan dus verder onderzoek naar verricht worden.

Onze aanpak tot evaluatie van de *AdvisedBy* benchmark bleek niet overeen te komen met die van Dinh, waardoor we onze resultaten niet konden vergelijken met die van zijn state-of-the-art algoritme. We stellen voor om Dinh te contacteren, om te weten te komen hoe hij de evaluatie precies heeft aangepakt. Vervolgens kan men onze aanpak opnieuw evalueren en beoordelen hoe de prestaties van beide benaderingen zich verhouden t.o.v. elkaar.

Ten slotte kan men op zoek gaan naar een meer geschikte aanpak om predicaten met meer dan twee argumenten om te vormen naar triples. In de UW-CSE *AdvisedBy* benchmark splitsten we het ternair predicaat *taughtBy* op in twee triples, waardoor in principe enige betekenis verloren ging. Typisch vormt men een ternair predicaat om door één nieuw subject te definiëren en er de drie parameters aan te koppelen; figuur 53 illustreert dit. Het probleem met deze aanpak is echter dat onze huidige graafpatronen geen verband zouden zien tussen twee



Figuur 53: Standaard representatie voor een ternair predicaat.

personen. Een startpunt voor alternatieve representaties van zulke predicaten kan de W3C Note zijn omtrent N-ary relations [44].

7 Conclusies en toekomstig werk

In dit deel formuleren we de voornaamste conclusies omtrent het geleverde onderzoek en geven we een aantal suggesties naar toekomstig werk. Dit is hoofdzakelijk een bundeling van de eerder geformuleerde conclusies en suggesties bij de drie delen van deze masterproef (zie delen 4.7, 5.5 en 6.4).

In het eerste deel van deze masterproef gingen we na wat de DeepDive methodologie inhoudt en wat deze uniek maakt. We stelden vast dat DeepDive fundamenteel op een MLN steunt, om zo het gebruik van features te combineren met extra domeinkennis. Door het ondersteunen van extra domeinkennis onderscheidt DeepDive zich van een louter feature-gebaseerd model; dit maakt DeepDive uniek. We toonden aan dat de DeepDive methodologie in afwezigheid van extra domeinkennis sterk overeenkomt met Logistic Regression. Daarnaast biedt DeepDive ook twee praktische meerwaarden, als deel van zijn eigen programmeertaal. Dankzij weight tying kunnen we in één regel een MLN definiëren dat een formule bevat voor iedere unieke feature, en dankzij de ondersteuning van User-Defined Functions kunnen we Python-scripts integreren. We stelden vast dat het elimineren van existentiële kwantoren uit een MLN (zgn. Skolemization) niet voor zich spreekt en bestudeerden Weighted First-Order Model Counting (WFOMC), wat in een fundamenteel verschillende aanpak zou resulteren. In de toekomst kan men onderzoek voeren naar alternatieven die existentiële kwantoren elimineren door het MLN rechtstreeks aan te passen.

Het tweede deel van deze masterproef was een praktische toepassing, waarbij we de baas-bedrijf relatie extraheerden uit nieuwsartikels, gebruikmakend van DeepDive. We hebben ondervonden dat relation extraction een uitdagend probleem blijft, ondanks dat DeepDive reeds voor de geschikte methodologie en ondersteuning zorgt. In totaal hebben we 382 extracties gedaan. Voor de gelabelde kandidaten behaalden we een zeer hoge precision van 0.94. Voor de niet-gelabelde kandidaten bedroeg deze naar schatting slechts 0.66. De features die DeepDive geleerd heeft, hielden intuïtief steek. Vooraleer deze resultaten gepubliceerd kunnen worden, moet de classificatie van de niet-gelabelde kandidaten verder verbeterd worden. Daartoe suggereren we om een alternatieve NLP toolkit te gebruiken en heuristische regels toe te voegen op basis van het patroon “P1 (B1), P2 (B2)”. We stelden vast dat DeepDive als systeem nog onvoldoende ondersteuning biedt voor de evaluatie van de resultaten. Concreet stellen we voor om de leercurve visualiseerbaar te maken, de breedte van de bins van de kansverdelingen instelbaar te maken, en populaire metrieken zoals precision en recall te ondersteunen.

In het derde deel van deze masterproeftekst onderzochten we hoe de DeepDive methodologie aangepast kan worden om nieuwe relaties te extraheren uit reeds gestructureerde data. Onze graaf-gebaseerde aanpak voor het extraheren van features zorgde voor kwalitatieve features. Via een eenvoudige PoC toonden we aan dat onze aanpak beloftevol is. Tevens bleek onze aanpak tegelijkertijd een eenvoudig alternatief voor discriminative MLN Structure Learning. We hebben onze aanpak geëvalueerd op een standaard benchmark en concludeerden dat onze aanpak bruikbaar is in de praktijk. In de toekomst kan voor nog andere graafpatronen onderzocht worden of deze nuttig zijn voor het genereren van features. Ook kan men onze aanpak toepassen op data uit de LOD Cloud om deze daadwerkelijk aan te vullen. Ten slotte suggereren we om op zoek te gaan naar technieken die irrelevante features (en formules) uit het MLN elimineren.

8 Referenties

[1] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321, 2015.

[2] Wei Fan and Albert Bifet. Mining big data: current status, and forecast to the future. *SIGKDD Explorations*, 14(2):1–5, 2012.

[3] Netflix prize. `https://www.netflixprize.com/` [Geraadpleegd op 15 november 2017].

[4] Fabricio F. Costa. Big data in biomedicine. *Drug Discovery Today*, 19(4):433–440, 2014.

[5] DataCrops. Unstructured data extraction. `https://datacrops.com/blogs/7-steps-extract-insights-unstructured-data/` [Geraadpleegd op 12 december 2017].

[6] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. *IJCAI*, pages 2670–2676, 2007.

[7] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[8] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 601–610, 2014.

[9] Natalia Konstantinova. Review of relation extraction methods: What is new out there? *Communications in Computer and Information Science*, 436:15–28, 2014.

[10] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology*, pages 172–183, 1998.

[11] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 100–110, 2004.

[12] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676, 2007.

[13] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[14] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *The VLDB Journal*, 18(5):1065–1090, 2009.

[15] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[16] Charles Elkan. Log-linear models and conditional random fields. 2012.

[17] Yann LeCun et al. Efficient backprop. *Neural Networks: tricks of the trade, Springer*, pages 9–50, 1998.

[18] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[19] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[20] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, 4(6):373–384, 2011.

[21] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2014.

[22] Guy Van den Broeck. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, KU Leuven, 2013.

[23] Ce Zhang. *DeepDive: A Data Management System for Automatic Knowledge Base Construction*. PhD thesis, UW-Madison, 2015.

[24] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[25] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[26] Hazy Research Group. Deepdive generic features library. `http://deepdive.stanford.edu/gen_feats` [Geraadpleegd op 13 februari 2018].

[27] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[28] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *16th International Conference on the World Wide Web*, pages 697–706, 2007.

[29] Hazy Research Group. Deepdive tutorial. `http://deepdive.stanford.edu/example-spouse` [Geraadpleegd op 18 december 2017].

[30] Made in limburg. `http://www.madeinlimburg.be/bedrijven/` [Geraadpleegd op 8 maart 2018].

[31] A. Van den Bosch, G.J. Busser, W. Daelemans, and S. Canisius. An efficient memory-based morphosyntactic tagger and parser for dutch. In *Selected Papers of the 17th Computational Linguistics in the Netherlands Meeting*, pages 99–114, 2007.

[32] Maarten van Gompel. Pynlpl - python natural language processing library. `https://github.com/proycon/pynlpl` [Geraadpleegd 9 maart 2018].

[33] VKW Limburg. Top 500. `https://www.vkwlimburg.be/top-500web` [Geraadpleegd 8 maart 2018].

[34] Daniele Varrazzo. Psycopg. `http://initd.org/psycopg/` [Geraadpleegd 26 maart 2018].

[35] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[36] Bradley L. Richards and Raymond J. Mooney. Automated refinement of first-order horn-clause domain theories. *Machine Learning*, 19(2):95–131, 1995.

[37] Rdflib: a pure python package work working with rdf. `https://github.com/RDFLib` [Geraadpleegd op 28 april 2018].

[38] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Technical report, W3C, 2008.

[39] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[40] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[41] Stanley Kok and Pedro Domingos. Learning markov logic networks using structural motifs. In *Proceedings of the 27th International Conference on Machine Learning*, pages 551–558, 2010.

[42] Marenglen Biba, Stefano Ferilli, and Floriana Esposito. Discriminative structure learning of markov logic networks. In *Inductive Logic Programming, 18th International Conference, ILP 2008*, pages 59–76, 2008.

[43] Quang-Thang Dinh, Matthieu Exbrayat, and Christel Vrain. Discriminative markov logic network structure learning based on propositionalization and x2-test. In *Proceedings of the 6th International Conference on Advanced Data Mining and Applications: Part I*, ADMA'10, pages 24–35, 2010.

[44] Natasha Noy and Alan Rector. Defining n-ary relations on the semantic web. Technical report, W3C, 2006.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Studie en toepassing van relation extraction en structure learning volgens de DeepDive methodologie

Richting: **master in de informatica**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vanmunster, Michiel

Datum: **14/06/2018**