KU LEUVEN

FACULTEIT WETENSCHAPPEN

Beslissingsboom gebaseerde data-representatie voor semi-supervised learning met een toepassing in intensive care data mining

Ruben GUILLEMYN

Promotor: Prof. H. Blockeel
Affiliatie: *Dep. Computerwetenschappen*

Begeleider & Co-promotor: *Prof. C. Vens*
Affiliatie: *Dep. Maatschappelijke Gezondheidszorg en Eerstelijnszorg*

Co-promotor: *Fabian Guiza Grandas*
Affiliatie: *Lab voor Intensieve Geneeskunde*

Proefschrift ingediend tot het behalen van de graad van
Master of Science
Toegepaste informatica: Artificiële Intelligentie

Academiejaar 2017-2018

Inhoudsopgave

1 Proloog **1**

2 Inleiding **2**
2.1 Voorwoord . . . 2
2.2 Achtergrond . . . 2
2.3 Hypothese . . . 3
2.3.1 Doel van de scriptie . . . 3
2.4 Scope . . . 3
2.4.1 In Scope . . . 3
2.4.2 Out of Scope . . . 4
2.5 Doelstellingen . . . 4

3 Literatuurstudie **5**
3.1 Algemene concepten . . . 5
3.1.1 Inleiding . . . 5
3.1.2 Supervised learning . . . 5
3.1.3 Unsupervised learning . . . 5
3.2 Decision Trees . . . 6
3.3 Random forests . . . 7
3.4 kNN-algoritme . . . 8
3.5 Machine learning: Semi-supervised learning . . . 9
3.6 Self-learning . . . 9
3.7 Semi-supervised classification trees . . . 12
3.8 Support Vector Machines . . . 13
3.8.1 Lineaire SVM . . . 13
3.8.2 Niet-Lineaire SVM . . . 15
3.9 Conclusie . . . 16
3.9.1 Wat leert ons de literatuurstudie . . . 16
3.9.2 Welke elementen gaan we meenemen . . . 16

4 Voorgestelde methode **17**
4.1 Inleiding . . . 17
4.2 Model . . . 17

5 Implementatie **21**
5.1 Inleiding . . . 21
5.2 Plan van Aanpak . . . 21

5.2.1 Conversie naar ARFF . . . 21
5.2.2 Keuze van de test, labeled, unlabeled dataset . . . 24
5.2.3 Configuratie bestand . . . 25
5.2.4 Automatiseren van deeltaken via perl script . . . 26
5.2.5 Conversie in Java . . . 29
5.2.6 Toepassen van machine learning methodes . . . 29
5.3 Architectuur en Tools . . . 30

6 Evaluatie **32**
6.1 Datasets . . . 32
6.1.1 Invloed van SSL gedeelte . . . 33
6.2 Resultaten . . . 34
6.2.1 Parameters . . . 34
6.2.2 Resultaten kNN . . . 35
6.2.3 Resultaten SVM . . . 42
6.2.4 SVM's en self-learning . . . 44
6.3 Combinatie van Self-learning en de binaire transformatie . . . 47
6.4 Samenvatting . . . 49
6.4.1 Overzicht testen/uitbreidingen . . . 50

7 Inzetbaarheid van het Model **52**
7.1 Inleiding . . . 52
7.2 Toepassing in Ziekenhuisomgeving . . . 52
7.3 Conclusie . . . 56

8 Conclusie **57**
8.1 Conclusie . . . 57
8.2 Adviezen . . . 58
8.3 Verder onderzoek . . . 59
8.4 Subvragen . . . 59

9 Appendix **61**

Bibliografie **75**

Hoofdstuk 1

Proloog

Artifiële intelligentie en Geneeskunde beginnen meer en meer met elkaar te intraheren. Een eerste prille aanzet was alvast de supercomputer Watson die zich in zijn geboortejaren oriënteerde op het spelen van Jeopardy, wat in België overeenstemt met "Waagstuk". Nadat deze krachtige computer erin slaagde de beste spelers van dit spel te verslaan, was het duidelijk dat deze computer tot grootse dingen in staat was. Het duurde niet lang vooraleer hiermee patronen konden opgezocht worden in de wereld van de geneeskunde.[2]

Elektronica en andere hulpmiddelen die vroeger een dokter bijstonden, zijn op dit moment aan het evolueren in het assisteren van meer specifieke geavanceerde taken. Zo is het doel van deze thesis om methodes en manieren te identificeren, en om voorspellingen te brengen met data waarvan de classificatie niet gekend is. Dit kan resulteren in het voorstellen van behandelingen voor patiënten die langdurig in ICU (Intensive Care Unit) verblijven, met een onmiddellijk gevolg dat ze er geen permanente restanten of pijn aan overhouden.

De manier waarop de computer kan leren om met data van de patiënt behandelingen te suggereren, kan veelal verschillen. Binnen de AI bestaan er een reeks van processen om bepaalde concepten aan te leren. Dit kan variëren van supervised learning tot unsupervised learning. In deze scriptie staat de toepasbaarheid van semi-supervised learning centraal. Supervised learning is het leren van een functie of classificatie van data, en dit door een dataset die reeds een "target-value"heeft. Unsupervised is het groeperen van data in gelijkaardige verzamelingen. Door deze twee te combineren verkrijgen we semi-supervised learning.

Voor dit proefschrift wil ik vooral mijn begeleidster prof. Celine Vens bedanken. Zij heeft in vele opzichten bijgedragen in het mij oriënteren doorheen de diverse onderzoekspaden die veel bijbrengen in het uitwerken van de gestelde onderzoeksvraag.

Hoofdstuk 2

Inleiding

2.1 Voorwoord

Op basis van een persoonlijk interesse in het domein van de geneeskunde en mijn achtergrond vanuit mijn bacheloropleiding rond predictiemodellering, is dit een onderwerp dat mij uitermate aanspreekt. Het kunnen combineren van machine learning technieken en de mogelijke medische toepassingen, is voor mij een uitdaging die ik graag aanga. Zoals ook de recente gebeurtenissen van Alpha Go en het kunnen laten redeneren van een computer als mens, vormt de basis van mijn beslissing waarom ik voor de specialisatie Artificiële Intelligentie heb gekozen.

2.2 Achtergrond

Ter verduidelijking van een aantal termen en concepten in deze scriptie, heb ik bewust deze paragraaf willen toevoegen. De onderbouw van deze scriptie is finaal het maken van een predictiemodel. Het beschikken over datasets is hier ontegensprekelijk cruciaal. Deze datasets bevatten een reeks van voorbeelden, waarbij deze bestaan uit eigenschappen of "features" die geregistreerd zijn. Voordat een model aangemaakt kan worden, wordt typisch deze data opgesplitst in verschillende onderdelen.

- Trainingsdata: Gebruikt om het model op te stellen
- Testdata: Gebruikt bij het evalueren van het model

"Clus" is een tool gemaakt door onder meer Professor Celine Vens. Dit framework laat toe om aan clustering en supervised learning te doen. Clustering zelf is het zoeken van groepen, waarbij voorbeelden tot een groep behoren als ze dezelfde eigenschappen vertonen. Supervised learning probeert aan de hand van voorbeelden een model te maken zoals een beslissingsboom om een voorspelling te doen van een bepaalde eigenschap. Bij supervised learning bestaan de trainingsdata uit voorbeelden, waarbij de "target" of de eigenschap die we willen voorspellen, reeds gekend is.

2.3 Hypothese

De gestelde onderzoeksvraag is dat beslissingsbomen toepasbaar zijn in een semi-supervised model, waarbij de data eerst een binaire transformatie zal ondergaan en waarbij "missing data"kan voorkomen. Daarbovenop zullen deze modellen in een setting getest en geëvalueerd worden bij patiënten die een lange periode in de intensieve zorg verbleven hebben. Deze trends zijn specifiek aan te wenden in het opsporen van spierziektes en het identificeren van mogelijke problemen die optreden in het zenuwstelsel.[12] In deze thesis gaan we het onderzoek bevestigen of ontkrachten op basis van een empirische studie.

De data vanuit het Intensive Care Unit (ICU) zal als eerste indicatie gebruikt worden om een antwoord te formuleren op deze hypothese. De overkoepelende techniek die we hiervoor aanwenden is semi-supervised learning of kortweg SSL. Om de hypothese te kunnen beantwoorden is geopteerd om deze vraag op te splitsen in meerdere deelvragen. Dit zal toelaten om stapsgewijs tot een gegrond antwoord te komen. Op een eerste deelonderzoek zal aan de hand van een aantal bestaande SSL modelleringstechnieken gezocht worden naar een zo doeltreffend mogelijk model. In de tweede deelvraag onderzoeken we hoe effectief elke methode wel is. De combinatie hiervan zal tevens een duidelijk beeld geven welke de beste werkwijze is om via niet-gelabelde data de trainingsdata te verbeteren. Deze aanpak laat toe om eender welk machine learning algoritme toe te passen en te evalueren.

Centrale vragen:

1. Welke specifieke technieken bestaan er om predicties te maken, en zijn ze wel toegankelijk?
2. Hoe doeltreffend is een SSL techniek in het classificeren, en welke is de betere in de setting van beslissingsbomen?
3. Zijn er trends afleidbaar uit de data van patiënten die langdurig op ICU verblijven?

2.3.1 Doel van de scriptie

In de thesis gaan we op zoek naar een SSL techniek waarin zowel gelabelde als ongelabelde data gecombineerd wordt tot een "slimmere representatie", met de bedoeling predicties te maken op niet geziene voorbeelden. Dit is een vorm van data preprocessing waarna meerdere verschillende predictieve machine learning algoritmes kunnen toegepast worden.

2.4 Scope

2.4.1 In Scope

Om de stelling te bewijzen zullen we een semi-supervised setting opstellen. De essentie bestaat erin om extra features te crëeren of te vervangen voor de gelabelde data en dat aan de hand van Random Forests. SSL zelf probeert door middel van de ongelabelde data de gelabelde data te expanderen. Dit kan enerzijds leiden tot het optimaliseren van de classificatie van data, of met andere woorden, tot het vinden van een beter model. Om te

mogen stellen dat dit zo is, kan een vergelijking met een supervised en/of unsupervised methode aangewend worden.

Eenmaal er voldoende bewijskracht is, zal een applicatie ontwikkeld worden die deze methodiek implementeert. De tool zelf zal geprogrammeerd worden met Java 8. Indien we de hypothese niet met de eerste werkwijze kunnen evalueren, bestaat er de mogelijkheid nog een ander pad aan te snijden, met name, de SSL Clus methode. Deze neemt aan dat de keuze van opsplitsing in een boom zowel unsupervised als supervised kan gebeuren. Indien de uitkomst van de hypothese positief is, verifiëren we of neurale netwerken hier nog een bijkomende verbetering aan geeft.

Verder zullen we een representatie zoeken, waarin a.d.h.v. ongelabelde data deze representatie van de gelabelde data vervangen wordt en zo expressiever wordt. Met expressiever gaat naast de informatie van de features, gegevens rond clusters of groepen met gelijkaardige attributen vervangen worden.

2.4.2 Out of Scope

Hoewel kennis van supervised learning en unsupervised learning veel kan bijdragen tot het bevestigen of ontkrachten van de stelling, zullen deze methodieken niet centraal staan. Veel van deze werkwijzen hebben reeds aangetoond dat ze een predictie/classificatiemodel kunnen voorstellen. Voor deze hypothese is eerder gekozen om deze modellen te gaan optimaliseren. Er bestaan reeds een aantal Python programma's die een SSL setting toelaten. De vertaling van deze programma's naar Java valt ook buiten deze scriptie. De gebruikte binaire classificatie en de manier om features te genereren in een super-/unsupervised omgeving van Prof. Celine Vens [18], worden wel opgenomen in de literatuurstudie, omdat deze essentieel kunnen blijken voor de bevestiging van de stelling.

2.5 Doelstellingen

De doelstellingen van deze thesis zijn de volgende:

- Het uitvoeren van een empirische studie.
- Het vinden van een semi-supervised omgeving voor bomen.
- Het model toepasbaar maken op verschillende datasets.
- Het creëren van een applicatie die de stappen automatiseert bij het maken van SSL bomen.
- Het evalueren en vergelijken van verschillende bomen en modellen.
- Het trekken van conclusies of het gevonden model een verbetering is al dan niet.
- Het opmaken van een model dat begrijpbaar en verklaarbaar is voor een dokter.

Hoofdstuk 3

Literatuurstudie

3.1 Algemene concepten

3.1.1 Inleiding

Om tot semi-supervised learning te komen is het belangrijk om als eerste stap de twee elementaire machine learning technieken toe te lichten, met name, Supervised Learning en Unsupervised Learning. Het verduidelijkt tevens de aanpak die wordt gehanteerd om beide technieken in combinatie te gaan aanwenden.

3.1.2 Supervised learning

Supervised learning is het verkrijgen van een functie vanuit classificatie van data, en dit via een leerproces voor een dataset die reeds een "target-value" heeft. Elk data-voorbeeld kent dus reeds zijn toegekende "target"waarde of resultaat. Deze waarde kan zowel een binaire classificatie, maar evengoed een multi-classificatie hebben.[5]

Wanneer het leeralgoritme de trainingsdata verwerkt heeft in een mathematische functie, bestaat het doel eruit om voor nieuwe voorbeelden een predictie te maken van hun "target value"via de toepassing van deze functie. Om tot deze predicties te komen is het belangrijk dat elke voorspelling afkomstig is van een aantal assumpties, vaak de inductieve bias genoemd. Een algemeen principe die we bij deze assumpties in rekening nemen, is gekend als "Occam's Razor". Dit komt neer op "Keep it simple".

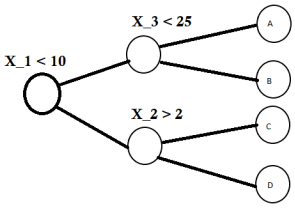
3.1.3 Unsupervised learning

De tegenhanger van het vorige concept is unsupervised learning. Het grote verschil bij deze methodiek is dat de target niet langer gekend is. De focus ligt dan ook niet meer in het vinden van de "target-value", maar eerder in het ontdekken van verbanden of verborgen structuren in de data. Belangrijk hierbij is het vinden van een reeks van gemeenschappelijke eigenschappen (clustering).

3.2 Decision Trees

Beslissingsbomen en hun accuraatheid vormen een belangrijk deel in het aantonen van de empirische studie. Deze bomen worden vooral gebruikt in een supervised omgeving. Voordat ze effectief gebruikt kunnen worden in SSL, zal een reeks van aanpassingen moeten uitgevoerd worden. Deze bomen proberen onder meer een classificatie toe te wijzen aan de voorbeelden. Hoe ze opgebouwd worden, wordt meestal beslist door de "information gain". Deze functie evalueert welke attributen van een dataset tot de beste onderverdeling leiden.

Iedere boom wordt opgebouwd aan de hand van knopen en bogen. De bovenste knoop wordt ook wel de wortel van de boom genoemd. De wortel van deze boom bevat alle gevallen van de dataset R. De vertakking van zo'n knoop leidt tot de onderverdeling van de voorbeelden, getest op een "feature" of attribuut van de data.



Figuur 3.1: Voorbeeld van een beslissingsboom

Unsupervised decision trees verbeteren, in vergelijking met supervised, niet de correctheid van de predictie. Het grote verschil bestaat er uit dat in plaats van één enkel attribuut nu rekening wordt gehouden met alle attributen van de dataset. Elke knoop binnen de boom stelt dan opnieuw een test op een attribuut voor, maar waarin het doel is veranderd. De clusters worden zo gekozen dat een minimalisatie de focus is van de variantie tussen de intra-clusters. Anderzijds wordt geprobeerd om de afstand tussen de clusters te maximaliseren. De Gini-coëfficient zelf is bepalend voor de variantie. Dit is een getal tussen het interval $[0, 1]$, die wanneer deze nadert tot 0, de examples binnen de cluster een grote gelijkheid gaan vertonen en tot 1 wanneer ze eerder veel verschillen.[17]

De selectie van de features is belangrijk in het bepalen van hoe de boom opgebouwd moet worden. In de meeste gevallen wordt de keuze bepaald door de "Kullback-Leibler divergentie" bij supervised decision trees. Dit is de functie die gekend is voor de informatie winst. De reden hiervoor is dat een attribuut met de hoogste winst zal leiden tot de snelste opsplitsing van de dataset. Toegepast bij bomen, wordt het als ID3 algoritme benoemd. Het minteken wordt weggewerkt, omdat p_i kleiner is dan 1 en in combinatie met de log wordt deze positief. Het herhalend gedeelte na het minteken, is de entropy van de partitie. De T slaat op een bepaalde test.

$$IG(T, R) = \sum_{i}^{c} -p_i \log_2 p_i - \sum_{i}^{c} -p_i \log_2 p_i [3]$$

Beslissingsbomen worden dus gemaakt d.m.v. de IG van de attributen. Deze manier van werken laat echter nog niet toe om ze te gebruiken in een SSL omgeving. Om ze toch toepasbaar te maken, worden Random Forests aanbevolen. De inductive bias bij het ID3 algoritme bestaat uit de volgende punten.

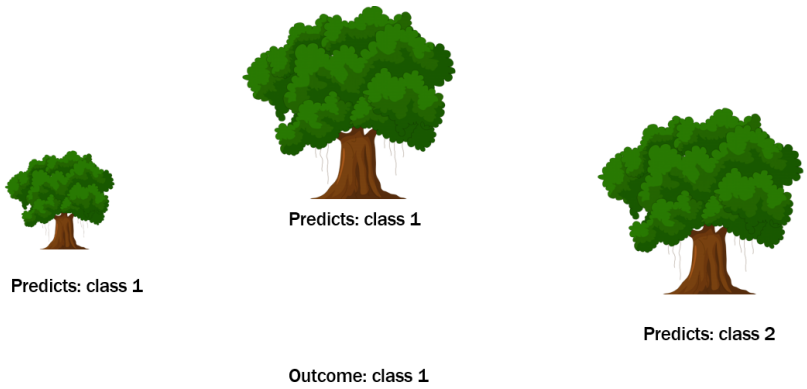
1. Elke knoop is gekozen, gebaseerd op de hoogste informatie winst
2. Voor elke knoop geldt dat er trainingsvoorbeelden zijn die tot een andere kind behoren. Zoniet zou deze knoop een blad zijn.

3.3 Random forests

Een groot nadeel van één grote beslissingsboom dat alle voorbeelden kan classificeren, is dat het model vrij complex wordt en dat er overfitting optreedt. Dit zijn zaken die best vermeden worden. Een manier om deze zaken weg te werken zijn random forests.[4] Deze methodiek neemt een gedeelte van de trainingsdata, waarbij meerdere keren hetzelfde trainingsvoorbeeld kan voorkomen. Op basis van deze onderverdeling, wordt de IG en/of variantie bepaald van een subset van de attributen en wordt een boom geconstrueerd. Dit proces wordt meerdere malen herhaald om zo verschillende "trees" te krijgen, waar ook de naam afkomstig van is. Om vervolgens te bepalen welke classificatie een voorbeeld krijgt, kan een gewogen gemiddelde genomen worden. Kort samengevat biedt Random Forests de volgende eigenschappen:

1. Ze kunnen door mensen eenvoudig geïnterpreteerd worden, omdat het pad afleidbaar is
2. Random Forests kunnen overweg met missing data
3. Foutmarges zijn beperkt door de verschillende bomen

Onderstaand voorbeeld leert ons dat wanneer elke boom een voorspelling heeft gedaan op een testsample, er via voting kan gewerkt worden om de classificatie te bepalen van een nieuw voorbeeld.



Figuur 3.2: Voorbeeld van Random Forests

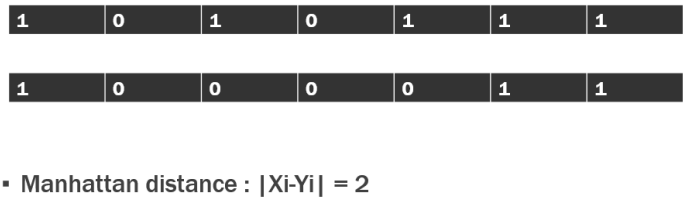
Een volgende belangrijke stap in deze scriptie, na het bepalen van deze reeks van bomen, is deze te converteren naar een binaire representatie. Deze mapping laat toe om verscheidene machine learning algoritmen erop toe te passen, waaronder het kNN algoritme. De aangehaalde methodieken zullen uitbundig beschreven worden in Hoofdstuk 5.

3.4 kNN-algoritme

Een belangrijk algoritme om te bepalen of een test voorbeeld tot een bepaalde klasse behoort is het kNN. Dit algoritme gaat voor elke feature de afstand bepalen tussen alle voorbeelden uit de trainingsdata. De totale som van deze afstanden wordt dan gebruikt als afstand tussen de twee examples. Daarna is het van belang om voor elk test voorbeeld zijn k-dichtste buren te kennen. Als predictie voor het nieuwe voorbeeld is de majority van deze k-examples bepalend.[10] Het eenvoudigste is wanneer elke feature numeriek is en er sprake is van totale ordening. Dit vormt het meest aangewende algoritme binnen de proefopstelling, waar het zal vergelijken worden met varianten, zoals self-learning en andere methodieken. De 2 volgende werkwijzen zijn hiervoor toepasbaar:

- Manhattan Distance : $d(X,Y) = \sum_{i=1}^{nr.OfFeatures} |X_i - Y_i|$
- Euclidean Distance : $d(X,Y) = \sqrt{\sum_{i=1}^{nr.ofFeatures} (X_i - Y_i)^2}$

In deze scriptie is geopteerd om de Manhattan distance te gebruiken, vooral omdat deze eenvoudiger te implementeren en te gebruiken is. Ook zijn er niet direct voordelen van de ene ten opzichte van de andere in deze omgeving. De keuze is dan ook eigenlijk vrij te kiezen wanneer er geen sprake is van numerieke ordening, maar van nominale waardes. Dan kan geopteerd worden om voor elke nominale waarde waar er een verschil is, een afstand te nemen van 1, of zoals verder ontdekt werd, deze te transformeren naar een vector.



Figuur 3.3: Voorbeeld Manhattan Distance

3.5 Machine learning: Semi-supervised learning

De thesis focust zich vooral op Semi-supervised Learning. Deze techniek gaat de twee methoden van Supervised en Unsupervised combineren. In sommige gevallen is gebleken dat dit tot betere voorspellingen leidt. Deze manier wordt vaak aangewend wanneer de data bestaat uit een klein aantal gevallen waarvan een classificatie gekend is. Daarenboven is er een grote beschikbaarheid van ongelabelde data. De voorbeelden afkomstig van de "intensive care unit" voldoen aan deze twee voorwaarden. [20]

Naast het voorspellen van de ongelabelde data, zal ook, eenmaal het model ontwikkeld is, geëvalueerd worden of op de reeds gekende gelabelde data een verbetering merkbaar is ten opzichte van de originele supervised manier. De reden hiertoe is om te controleren of het gevonden model de juiste predictie doet. Wanneer echter een ander resultaat bekomen wordt, kan dit leiden tot het evalueren van een optimalisatie.

3.6 Self-learning

Self-learning is één van de meest gebruikte SSL technieken om een model te creëren. Deze wijze van werken probeert telkens niet-gelabelde data toe te voegen bij de gelabelde door een vooropgestelde threshold op te stellen. Deze threshold is gericht op een zekerheid dat een voorbeeld die classificatie zal ontvangen en ze vervolgens aan de training toe te voegen.[5] Dit proces kan iteratief toegepast worden totdat de threshold niet meer geldt. Het model bestaat dan uiteindelijk uit de combinatie van de deze twee sets waardoor ze kunnen toegepast worden op de test voorbeelden. Het algoritme bestaat dus uit twee stappen die als volgt gaan:

1. Toevoegen van ongelabelde data (die met een zekerheid kunnen geclassificeerd worden);
2. Opbouwen van het model
3. Predicties op de te evalueren data.

De eerste stap wordt N-maal herhaald, met name, vanaf het moment één voorbeeld kandidaat is om in de training te worden opgenomen, kan de eerste stap hernomen worden. Het is hier belangrijk dat er voldoende gelabelde data beschikbaar is. Zo niet zou als voorbeeld bij 1 trainingsvoorbeeld telkens dezelfde target bijgehouden worden. Finaal zou het model uiteindelijk slecht scoren. Stel dat de data bestaat uit de training voorbeelden zoals in onderstaande tabel is opgenomen. Daarnaast zijn er de aannames voor

elk attribuut dat het domein $\mathbb{N} = \{1, 2, 3, 4\}$ en de target $\mathbb{N} = \{0, 1\}$. De threshold bij dit voorbeeld wordt gezet op 80%.

Examples	Attr. 1	Attr. 2	Attr. 3	Attr. 4	Target
1	2	2	3	1	0
2	1	2	3	4	1
3	3	2	2	1	0
4	3	2	1	1	0
5	1	2	3	1	0

Tabel 3.1: Voorbeeld self-learning

Vanuit de tabel wordt vervolgens kNN gebruikt op de niet-gelabelde data. Stel nu dat men beschikt over het volgend example:

Examples	Attr. 1	Attr. 2	Attr. 3	Attr. 4
1	2	2	3	1

Tabel 3.2: Voorbeeld self-learning ongelabelde data

Indien we nu 3-NN of de drie dichtste buren gebruiken en de afstand tussen elk element berekenen, dan zijn de respectievelijke distances gelijk aan $\{0, 4, 2, 3, 1\}$. Deze examples dragen bij tot de zekerheid dat iets geclassificeerd wordt zoals in het beschouwde voorbeeld 0 of 1 is. De drie beste buren stemmen voor classificatie 0. Omdat hier een zekerheid is van 100% bij de 3-NN, dan verkrijgt het beschouwend voorbeeld het target 0. In dit geval wordt vervolgens het voorbeeld opgenomen in de trainingsdata, omdat deze de threshold overstijgt.

Examples	Attr. 1	Attr. 2	Attr. 3	Attr. 4	Target
1	2	2	3	1	0
2	1	2	3	4	1
3	3	2	2	1	0
4	3	2	1	1	0
5	1	2	3	1	0
6	1	2	2	3	0

Tabel 3.3: Voorbeeld self-learning: niet opgenomen

Het niet-gelabeld voorbeeld die nu beschouwd wordt:

Examples	Attr. 1	Attr. 2	Attr. 3	Attr. 4
1	2	2	3	4

Tabel 3.4: Voorbeeld self-learning ongelabelde data: niet opgenomen

De respectievelijke afstanden afgeleid uit dit voorbeeld zijn: $\{3, 1, 5, 6, 4, 3\}$. In twee van de 3 gevallen wordt een 0 voorspeld en in één geval wordt als target 1 voorspeld, omdat er dus een zekerheid van 66% is en in het geval van onze threshold: $66\% < 80\%$ zal het example niet opgenomen worden. In de laatste stap zal zoals vermeld in punt 2 het model gebruikt worden om voorspellingen te doen op de testdata.

Een nadeel in deze methode dat tijdens het onderzoek is opgemerkt, is dat de afstand tot elke trainingsexample moet berekend worden. Dit kan zorgen bij de berekening met de talrijke trainvoorbeelden tot memory overflow. Anderzijds werd in de eerste versie gekozen om majority voting te doen op alle voorbeelden uit de trainingsdata en dit met een bepaald gewicht overeenstemmend met hun afstand, om dit te optimaliseren kiezen we om enkel de vijf dichtste voorbeelden te nemen en daar vervolgens de threshold op te berekenen Het resultaat hiervan is dat dit een flinke boost gaf in de uitvoeringstijd van het algoritme, daar waar vroeger alle voorbeelden in rekening werden genomen.

3.7 Semi-supervised classification trees

Tijdens het schrijven van deze thesis, werd door onderzoekers in Slovenië een semi-supervised methode ontwikkeld in Clus. Deze werkwijze was een belangrijk element dat niet kon ontbreken. Het werd alvast opgenomen in de vergelijking met de hierin voorgestelde SSL-methodiek. Hierdoor kan een duidelijker onderscheid worden gemaakt welke van de twee een beter resultaat verschaft en welke methode in sommige gevallen aan te raden is, gegeven de dataset/omgeving.

Deze methode maakt ook gebruik van de assumptie dat elementen die tot eenzelfde cluster behoren, hoogst waarschijnlijk ook dezelfde classificatie zullen ontvangen. Het grootste verschil tussen deze aanpak van werken en de zelf voorgestelde, is dat er bij hen geen gebruik gemaakt wordt van binaire transformatie van de data. Wel wordt er in beide gevallen voor de unsupervised data de PC (Predictive Clustering) aangewend.

De werkwijze bij PCT die zij voorstellen, start vanaf de Gini impurity. Deze meeteenheid duidt erop hoe vaak een een element een verkeerd label zou ontvangen, wanneer deze arbitrair gekozen zou worden en dit met relatie tot de labeling distributie.

$$Impurity(E) = Gini(E, Y)$$
$$Gini(E, Y) = 1 - \sum_{i=1}^{C} p_i^2$$

Bij de laatste formule duidt C op het domein dat de target kan aannemen. Voor het gebruik van deze methodes breiden ze de formule verder uit, waarbij ze naast informatie over ongelabelde data, ook de informatie over gelabelde data opnemen. In de uitgebreide formule wordt een gewogen som van de impurities toegepast samen met de classificatie.

$$Impurity_{SSL}(E) = w \cdot Impurity(E_l, Y) + \frac{1-w}{D} \cdot \sum_{i=1}^{D} Impurity(E, X_i)$$

De D in de formules, slaat op het aantal descriptieve of beschrijvende attributen. X_i, staat voor het i-de beschrijvende attribuut. Belangrijk is dat er onderscheid moet gemaakt worden tussen de nominale en de numerieke attributen. De Impurity wordt dan respectievelijk bepaald door de distributie bij nominale attributen en de variantie bij numerieke waardes. W, of het gewicht in de formule, laat toe om het supervised gedeelte zwaarder te laten doorwegen dan het unsupervised gedeelte. Deze parameter heeft een domein tussen $[0, 1]$. Waarbij 1 voor volledig supervised staat. Ik verwijs graag door voor de verdere werking van hun methode naar [13].

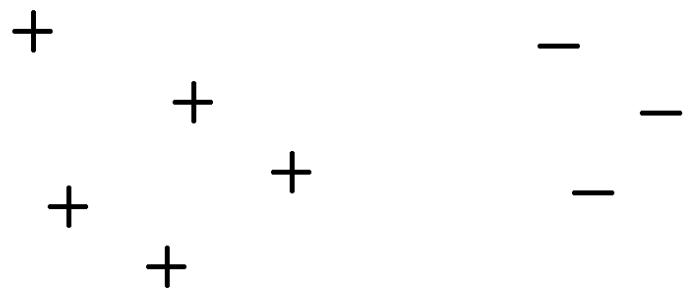
Samengevat komt deze methodiek neer op het uitbreiden in het kiezen van de beste test, namelijk door naast de labeled ook de niet-gelabelde data in evaluatie te nemen. Vervolgens gebruiken ze deze keuze van testen en random forests om bomen te generen en hun model te evalueren op de test data. Daar er hier ook opnieuw bomen gegenereerd worden, zou de uitbreiding via hun manier van werken, bestaan uit de binaire transformatie toe te passen en deze te vergelijken met hun originele.

3.8 Support Vector Machines

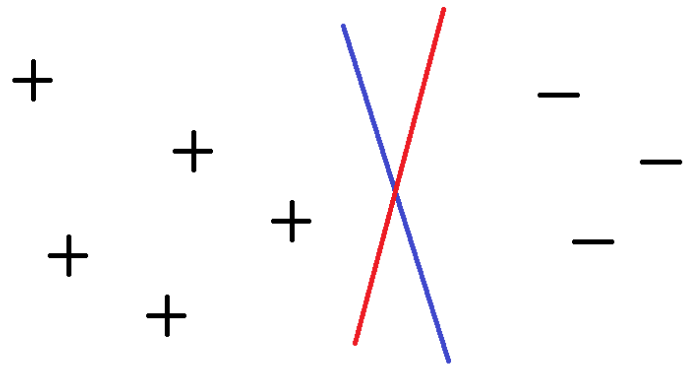
3.8.1 Lineaire SVM

Een totaal ander alternatief dan kNN zijn SVM's of Support Vector Machines. Deze techniek werd ontwikkeld in 1995 door Vapnik en Cortes.[15] Hun methode heeft reeds aangetoond dat ze zorgt voor goede predicties. Dit is eveneens een stuk die we niet mogen uitsluiten en onderwerp moet maken van deze studie.
SVM's werden in eerste instantie gebruikt voor classificatie, wat in deze scriptie ook wel de focus is. In de volgende stap bestaat het objectief eruit om vanuit een set van voorbeelden een hypervlak te vinden zodat er een duidelijke splitsing is tussen de positieve en de negatieve voorbeelden voorgesteld in een bepaalde dimensie.

In het 2-dimensionale vlak komt dit simpelweg neer in het vinden van een rechte. Zoals in figuur 3.5 te zien is, gaat het om het bepalen welke rechte de meest optimale is, cfr. gegeven het voorbeeld.



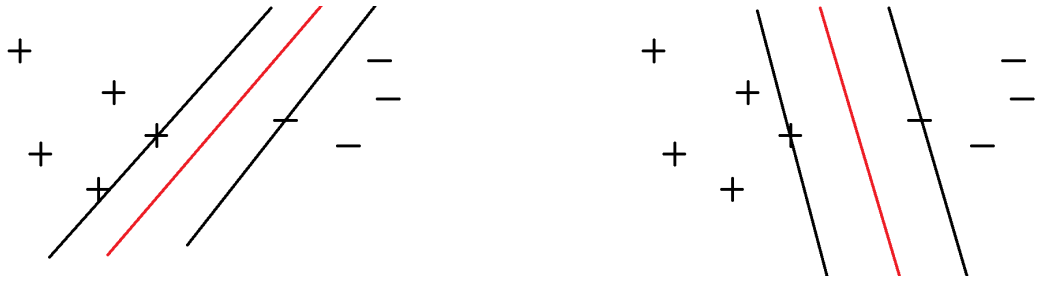
Figuur 3.4: Afbeelding voorbeeld SVM



Figuur 3.5: Afbeelding voorbeeld SVM

Om te bepalen welke van de lijn voor de beste separatie zorgt, moeten de dichtste positieve

en negatieve examples beschouwd worden. Tussen deze twee berekenen we vervolgens de afstand. De ruimte hiertussen wordt ook wel de margin genoemd, waarin de hyperplane middenin komt te staan. De beste hyperplane vinden, bestaat dan in het vinden van de margin waarin die maximaal is.[3] Onderstaande afbeelding toont dit ook aan dat de keuze van de scheidingslijnen ervoor zorgen dat de margin varieert.



(a) Kleinere margin (b) Grotere margin

Figuur 3.6: Verschil margin

Het vinden van een lineaire functie in het twee dimensioneel vlak, is het vinden van de functie $y = ax + b$. Deze functie snijdt de x-as waar de functie $ax + b = 0$. Verder moeten nog voorwaarden opgelegd worden zodat de binaire classificatie gegarandeerd kan worden. Deze komen neer op het aanduiden dat alle positieve examples door de functie het beeld $+1$ krijgen en alle negatieve examples geëvalueerd worden naar -1. De condities voor de functie komen nu neer op:

- Als de classificatie $= 1$, dan moet de functie evalueren naar ≥ 1
- Als de classificatie $= -1$, dan moet de functie evalueren naar ≤ -1
- De functie definieert een maximale margin

Finaal geeft het bekomen van die functie een evaluatie manier weer, die aan alle voorgaande voorwaarden moet voldoen, en die toelaat om zowel nieuwe examples te toetsen als hun predictie klasse te associëren. Dit is echter niet altijd eenvoudig te vinden, meer bepaald omdat er vaak geen scheiding te vinden is in het 2-dimensioneel vlak. Wanneer dit het geval is, dan is een projectie naar een hogere dimensie noodzakelijk om na te gaan of daar een separatie mogelijk is, namelijk, dat wanneer in de 2D geen onderscheid is, we beter kijken of er in de derde dimensie geen functie bestaat die deze kan indelen. In dit geval worden geen lineaire SVM meer beschouwd maar niet-lineaire.[3] In het geval er geen onderscheid kan gemaakt worden, bestaat er een techniek om bepaalde voorbeelden te negeren en op deze onderverdeling dan een functie te vinden. Meestal wordt dit toegepast wanneer er sprake is van noisy data of data waarvan de classificatie waarschijnlijk niet klopt met de waargenomen features.

3.8.2 Niet-Lineaire SVM

Wanneer er geen lineaire separatie mogelijk is, dan controleren we eerst of deze niet bestaat in een andere dimensie. Een vereiste is wel dat de features van de voorbeelden eerst voorgesteld dienen te worden als vectoren. In de volgende stap projecteren we deze dan in een hogere dimensie. De transformatie wordt uniform uitgevoerd, namelijk dezelfde transformatie functie wordt aangewend voor alle vectoren. Indien een scheidingslijn gevonden kan worden in een hogere dimensie, dan transformeren we deze terug naar zijn originele dimensie, waardoor de functie niet langer lineair is, maar bijvoorbeeld kwadratisch kan zijn. Daarnaast vervalt de garantie dat de margin maximaal is. Typisch worden een aantal kernels gedefinieerd om aan te wenden. Voorbeelden hiervan zijn de Radial Basis Function of RBF, Sigmoid, Mahalanobis, ... In de meeste gevallen kiest men voor de RBF, die ook in deze scriptie opgenomen zijn om SVM's toe te passen.[14]

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0$$

Voor elke dataset is het belangrijk in het optimalisatieproces om de bijhorende parameters zoals γ te identificeren en te optimaliseren.

3.9 Conclusie

3.9.1 Wat leert ons de literatuurstudie

De literatuurstudie leert ons dat bij supervised learning het doel eruit bestaat om regressie te bepalen of de classificatie correctheid te maximaliseren. Het is tevens belangrijk dat de keuze van een node bij een boom gekozen wordt aan de hand van de beste classificatie. Bij unsupervised decision trees representeren zowel de wortel, knopen en bladen een cluster, waarin de keuze bepaald wordt om de variantie te minimaliseren binnen de intra-clusters. Daarenboven bieden Random Forests de mogelijkheid om overfitting te vermijden, en bomen te construeren waar telkens de fout op een andere plaats kan ontstaan, waardoor de fout geminimaliseerd kan worden en we een finale predictie kunnen maken.

3.9.2 Welke elementen gaan we meenemen

De combinatie van supervised- en unsupervised bomen zal ervoor zorgen om een eerste stap te zetten en ze aan te wenden in een semi-supervised omgeving. Dit vormt de kern van deze thesis.
Om beide zaken in verband te brengen moet eerst nog een transformatie gedaan zijn op de data. Deze transformatie bestaat eruit om voor elk voorbeeld zijn pad binnen de boom te reconstrueren en dit voor beide gevallen. Eénmaal dit lukt, kan beide representaties (super- en unsupervised) samengevoegd worden tot één lange voorstelling en kunnen we spreken van SSL beslissingsbomen. Finaal zal de voorgestelde methode van Levatić[13] ingezet worden om de vergelijking te maken zodoende te bepalen welke representatie het meest geschikt is.

Hoofdstuk 4

Voorgestelde methode

4.1 Inleiding

De voorgestelde methode is gebaseerd op de paper van Prof. Celine Vens [18], maar dan verder uitgewerkt op het unsupervised gedeelte. In dit hoofdstuk zullen we stapsgewijs een overzicht geven, hoe het voorgestelde model uiteindelijk moet opgesteld worden en hoe deze toepasbaar is op de verschillende datasets. Een afzonderlijk hoofdstuk hebben we hieraan gewijd, omdat deze de kern vormt in het begrijpen waarom dit model geldig is. Daarnaast wordt op zoek gegaan naar een methode waarin het mogelijk is om ongelabelde data op te nemen in de gelabelde data , met als doel de predictie te verbeteren. Elk voorbeeld wordt dan voorgesteld als zijn pad binnen de boom voor zowel de gelabelde dataset als de clustering informatie. Het voordeel om elk voorbeeld a.d.h.v. zijn pad voor te stellen, is dat eenvoudig verschillende predicatoren kunnen getest worden. Meer concreet kan dit gezien worden als een preprocessing stap, waarbij in de volgende stap het model op verscheidene manier getest kan worden, zoals SVM, kNN, ...

4.2 Model

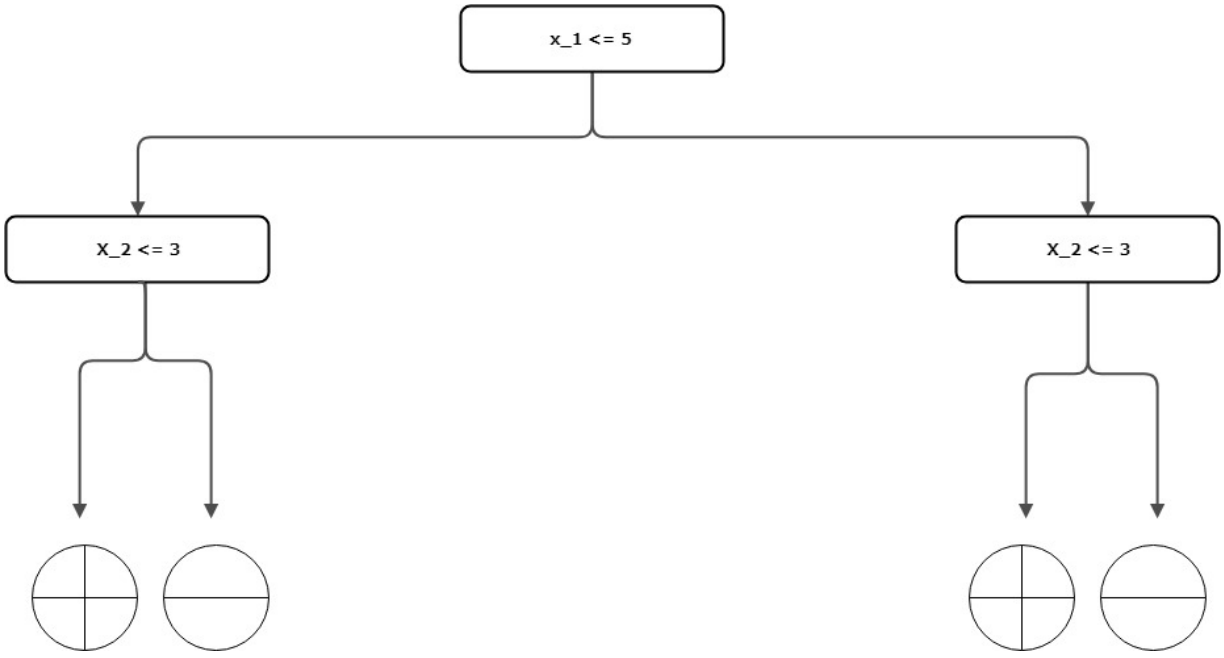
Via een leidraad willen we verklaren welke stappen er noodzakelijk zijn, en met welke elementen we rekening moeten houden bij het opstellen van het model en dit voor zowel het super- als het unsupervised gedeelte. Voor een gedetailleerde flow, kan afbeelding 5.10 erbij genomen worden.

1. Opsplitsen van de dataset in labeled, ongelabelde en testset
2. Het aanmaken van de verschillende beslissingsbomen
3. Het transformeren van de gelabelde voorbeelden naar hun binaire transformatie
4. Het samenbrengen van voorgaande elementen naar een representatie, bruikbaar voor een predictor

In een eerste stap is het belangrijk om de volledige dataset in beschouwing te nemen, waarbij we initieel een onderdeel aan de kant zetten ter evaluatie. Belangrijk hierbij is dat deze set een target heeft, anders heeft het geen nut om er de accuraatheid op te bepalen. Daarnaast mogen we deze set niet inschakelen om het model te trainen. Dit kan

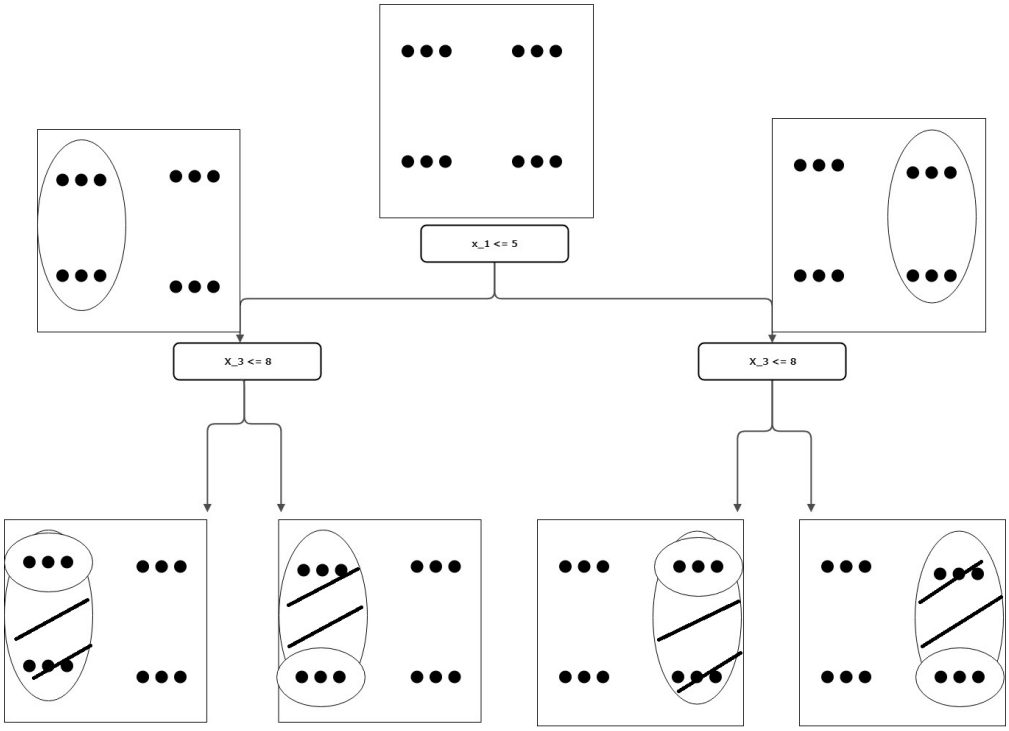
leiden tot het fenomeen overfitting, waarbij het model de testset begint te memoriseren. Naast het creëren van een testset dient een in het geval van semi-supervised learning, een ongelabelde dataset beschikbaar zijn. Tevens is het belangrijk dat de set van alle gegevens niet gesorteerd zijn volgens een bepaald attribuut en dus alle voorbeelden at random voorkomen.
Via Random Forests en Clus (applicatie die toelaat RF toe te passen op de dataset), worden de bomen gegenereerd en start de preproccessing stap. De laatste duidt op het toepassen van Clustering op de dataset. Het is uitermate belangrijk dat de labeled data ook opgenomen zijn in de ongelabelde data omdat voor het voorbeeld het pad van de clustering, op zijn beurt, moet opgenomen worden in het vooropgestelde model.

Als eerste stap bespreken we hoe het supervised gedeelte werkt en vervolgens verduidelijken we het unsupervised gedeelte en hoe deze kan opgenomen worden in de representatie voor semi-supervised learning. Dit verloopt analoog eenmaal het concept duidelijk is. Wanneer een boom opgesteld wordt, is de labeled data van toepassing. Van deze data worden telkens sub-partities genomen om redenen van de noodzaak aan het bewaren van de distributie van de voorbeelden en uitzonderingen een kans te geven. Volgende numerieke waardes zijn ter ondersteuning van de methodiek en hebben geen vaste waarde. Veronderstel dat men beschikt over een 100 tal trainingsvoorbeelden, dan kan een subset voor RF bestaan uit 5 attributen. Vanuit deze subset wordt voor elk attribuut de "information gain" berekend, waarbij het attribuut met de hoogste waarde als kandidaat naar voor komt als de root node van de boom.[3] Dit proces wordt herhaald tot alle voorbeelden geclassificeerd zijn en resulteert in de volledige boom. Vervolgens stellen we dat de root controleert of een attribuut $x_1 \leq 5$ is, en bijvoorbeeld bij $x_1 = 2$ dat $x_2 = 3$ is, figuur 4.1 illustreert dit. Tevens zorgen deze testen ervoor dat alle voorbeelden mee geclassificeerd zijn en resulteert zo in onderstaande boom, waarbij met het voorbeeld de representatie van [1,1,0,1,0,0,0,+], beschikbaar wordt:



Figuur 4.1: Voorbeeld van boom in RF

Voor het unsupervised gedeelte gaan we ongeveer hetzelfde tewerk, met als grootste verschil dat er geen classificatie van de target label wordt gedaan, maar dat men clustering aanwendt. Een ander belangrijk verschil bij unsupervised learning is dat zowel de gelabelde als de ongelabelde voorbeelden ingezet worden om een model te maken. In het eerste aangehaalde verschil is het objectief het maken van een root op de trainingsdata data, waar de intra-afstand in elke cluster wordt geminimaliseerd of de inter-afstand gemaximaliseerd. Een voorbeeld is figuur 4.2.

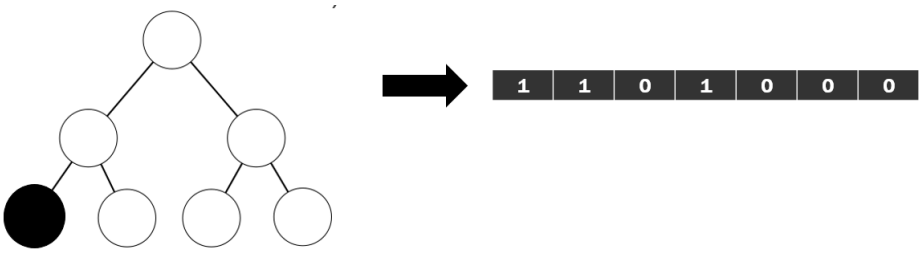


Figuur 4.2: Voorbeeld van unsupervised boom in RF

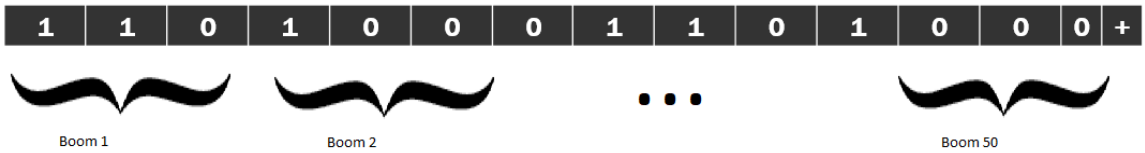
Voor beide cases herhalen we dit proces van Random Forests een n-tal keer met telkens een andere subset van de features als trainingsdata. In de volgende fase wordt voor elk voorbeeld uit de trainingsdata het pad doorlopen doorheen elke boom. Met het pad komt het aantal opeenvolgende testen van de beslissingsboom overeen. Voor elke test of knoop verkrijgen we een evaluatiefunctie, die evolueert naar 1 als het pad de conditie heeft gecontroleerd, of 0 als het voorbeeld de node niet heeft doorlopen. Met andere woorden bestaat het pad uit de te volgen stappen vanuit de rootnode tot een leaf, en stemt overeen met de classificatie of de cluster die bij de gegenereerde boom hoort.

Zoals reeds aangehaald bestaat de transformatie eruit om elk voorbeeld uit de trainingsdata door alle gegenereerde bomen te laten lopen en telkens het pad te concateneren in bijvoorbeeld een array representatie zoals in afbeelding 4.3. Voor de eenvoud werd gekozen om alle supervised bomen eerst te concateneren en vervolgens alle unsupervised bomen. Ook werd bepaald dat de trainingsdata in de ongelabelde data moet voorkomen. Dit komt door de voorgaande regel, waarbij zo de ongelabelde informatie over de gelabelde voorbeelden kan opgenomen worden als extensie, om deze expressiever te maken, zoals te zien is in afbeelding 4.4. De laatste index van de array bestaat dan uiteindelijk uit de originele target label. Dit proces wordt ook toegepast op de testdata, waarbij uiteindelijk zo

het model vorm krijgt. Daarnaast stopt de preprocessing stap en zit zowel de informatie van super- en unsupervised vervat in een representatie waarop verschillende predicatoren het model kunnen evalueren.

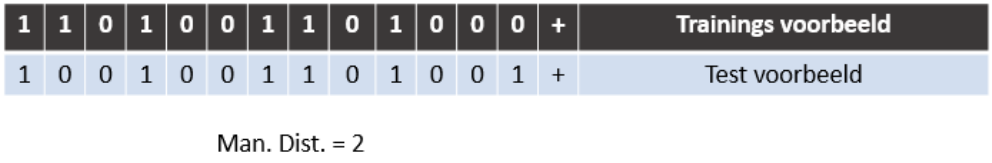


Figuur 4.3: Transformatie naar binair



Figuur 4.4: Voorbeeld concateneren alle bomen gegenereerd door Random Forests

Wanneer dit proces is toegepast op alle trainingsvoorbeelden en alle testdata, berekenen we vervolgens bij kNN als predictor de afstand van elk trainingsvoorbeeld ten opzichte van een bepaald testvoorbeeld, zoals bij figuur 4.5. Via majority voting wordt dan een predictie classificatie gemaakt voor elk testvoorbeeld en controleren we hoe vaak het model correct of fout is.



Figuur 4.5: Voorbeeld toegepast model

Hoofdstuk 5

Implementatie

5.1 Inleiding

Om overfitting te vermijden wordt Random Forests gebruikt binnenin Clus. Dit is een java jar waarin dergelijke methodieken geïmplementeerd zijn. De output van Clus gebruiken we voor verdere verwerking. Een belangrijk voordeel van deze applicatie, is dat het configureerbaar is om zowel supervised- als unsupervised beslissingsbomen te genereren. Om de verwerking en de implementatie te stroomlijnen werd geopteerd om de Java omgeving verder in te zetten.

5.2 Plan van Aanpak

1. De dataset omzetten naar ARFF: labeled, labeled + unlabeled en testset
2. Het configuratie bestand aanmaken
3. Automatisch perl script inzetten
4. De binaire conversie van de attributen uitvoeren
5. De gegenereerde output bestanden sequentieel inlezen in Java
6. Machine learning methodes toepassen

5.2.1 Conversie naar ARFF

De conversie naar ARFF is een voorwaarde om de data te kunnen inlezen in Clus. Als voorbeeld kiezen we de paddenstoelen databank als referentie, om zo de verschillende eigenschappen van dit bestand te verduidelijken. Indien we experimenten willen uitvoeren met de ontwikkelde applicatie, dan raden we de volgende structuur aan:

```
1  @RELATION mushrooms
2
3  @ATTRIBUTE cap_shape {b,c,x,f,k,s}
4  @ATTRIBUTE cap_surface {f,g,y,s, n,b,c,g,r}
5  @ATTRIBUTE cap_color {n,b,c,g,r,p,u,e,w,y}
6  @ATTRIBUTE bruises {t,f}
7  @ATTRIBUTE odor {a,l,c,y,f,m,n,p,s}
8  @ATTRIBUTE gill_attachment {a,d,f,n}
9  @ATTRIBUTE gill_spacing {c,w,d}
10 @ATTRIBUTE gill_size {b,n}
11 @ATTRIBUTE gill_color {k,n,b,h,g,r,o,p,u,e,w,y}
12 @ATTRIBUTE stalk_shape {e,t}
13 @ATTRIBUTE stalk_root {b,c,u,e,z,r,?}
14 @ATTRIBUTE stalk_surface_above_ring {f,y,k,s}
15 @ATTRIBUTE stalk_surface_below_ring {f,y,k,s}
16 @ATTRIBUTE stalk_color_above_ring {n,b,c,g,o,p,e,w,y}
17 @ATTRIBUTE stalk_color_below_ring {n,b,c,g,o,p,e,w,y}
18 @ATTRIBUTE veil_type {p,u}
19 @ATTRIBUTE veil_color {n,o,w,y}
20 @ATTRIBUTE ring_number {n,o,t}
21 @ATTRIBUTE ring_type {c,e,f,l,n,p,s,z}
22 @ATTRIBUTE spore_print_color {k,n,b,h,r,o,u,w,y}
23 @ATTRIBUTE population {a,c,n,s,v,y}
24 @ATTRIBUTE habitat {g,l,m,p,u,w,d}
25 @ATTRIBUTE Class {e,p}
26
```

Figuur 5.1: Voorbeeld van data bestand

De eerste regel, zoals te zien op afbeelding 5.1 , moet starten met de @RELATION tag. Deze duidt de start en de naam van het bestand aan. Voor de rest heeft deze regel geen verdere invloed. De volgende regels duiden de attributen en hun eigenschappen aan.

Wanneer we de @ATTRIBUTE aanroepen, verwacht Clus een éénduidige naamgeving voor elk typerend kenmerk van de databank. Eénmaal de naamgeving gedefinieerd is, moet een specificatie van het type opgegeven worden. In het voorbeeld werken we met een set van nominale waardes die de eigenschap kan aannemen. Naast de nominale waarden kan ook gekozen worden voor numerieke waarden.

De specificatie is zeer belangrijk, want het heeft invloed op de verdere verwerking van het classificatie algoritme. Numerieke waarden moeten gekozen worden indien er sprake is van totale ordering van het attribuut. Eenvoudig gezegd komt dit neer op wanneer over temperaturen gesproken wordt, dat 35°C groter is dan 30°C. Anderzijds kan bij nominale waarden geen ordening geplaatst worden. Zo is het kleur van de kap van een paddenstoel niet groter of kleiner dan een ander kleur kap. Hoe deze vergelijking uitgevoerd wordt verduidelijken we bij de toepassing van de machine learning algoritmes.

De zelfgeschreven applicatie verwacht dat het target attribuut op het einde van de lijst van eigenschappen komt. Ook voor de naam van deze target is verwacht dat deze de naam Class krijgt.

Wanneer alle attributen gespecificeerd zijn, verwacht het bestand een @DATA tag. Deze annotatie duidt de start van de databank aan, zoals te zien is in het onderstaand voorbeeld van de paddenstoelen databank.

```
@DATA
x,s,e,f,s,f,c,n,b,t,?,k,s,p,w,p,w,o,e,w,v,d,p
x,f,y,f,f,f,c,b,p,e,b,k,k,p,p,p,w,o,l,h,y,g,p
f,y,n,t,l,f,c,b,w,e,r,s,y,w,w,p,w,o,p,n,s,p,e
k,s,g,f,n,f,w,b,p,e,?,k,k,w,w,p,w,t,p,w,n,g,e
x,f,w,f,n,f,w,b,p,t,e,f,f,w,w,p,w,o,e,n,s,g,e
f,s,y,t,l,f,w,n,p,t,b,s,s,w,w,p,w,o,p,n,v,d,e
x,y,e,f,f,f,c,n,b,t,?,k,s,p,w,p,w,o,e,w,v,d,p
x,y,y,t,l,f,c,b,g,e,c,s,s,w,w,p,w,o,p,k,n,m,e
f,f,g,f,n,f,w,b,h,t,e,f,s,w,w,p,w,o,e,k,a,g,e
b,y,y,t,l,f,c,b,w,e,c,s,s,w,w,p,w,o,p,n,s,m,e
x,y,e,f,s,f,c,n,b,t,?,k,k,p,p,p,w,o,e,w,v,p,p
x,s,w,t,p,f,c,n,k,e,e,s,s,w,w,p,w,o,p,k,s,g,p
x,y,g,f,f,f,c,b,h,e,b,k,k,b,b,p,w,o,l,h,y,d,p
b,y,y,t,a,f,c,b,k,e,c,s,s,w,w,p,w,o,p,k,s,m,e
f,y,n,t,n,f,c,b,w,t,b,s,s,g,p,p,w,o,p,k,v,d,e
x,s,g,f,n,f,w,b,p,t,e,s,s,w,w,p,w,o,e,k,a,g,e
x,f,n,t,n,f,c,b,n,t,b,s,s,w,p,p,w,o,p,n,v,d,e
f,y,n,t,n,f,c,b,p,t,b,s,s,p,g,p,w,o,p,n,v,d,e
```

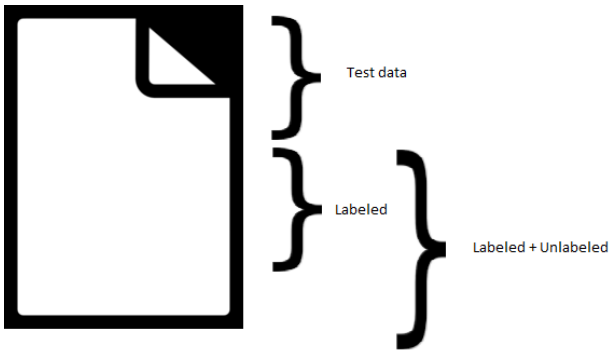
Figuur 5.2: Voorbeeld van data bestand

Elke regel duidt een observatie of example aan. Daarnaast is elk attribuut gesplitst door een ”,”. Om een mogelijk probleem bij het inlezen in de applicatie te voorkomen, is het belangrijk om tussen elke regel en elk attribuut een onderscheid te maken of er nu een numerieke waarde of een nominale waarde verwacht wordt. Om via de meegegeven code te werken, is aangewezen om de volgende set van regels te respecteren:

- @ATTRIBUTE moet in hoofdletters staan (Deze vereiste is voor het geautomatiseerd Perl script)
- De target moet het woord Class bevatten, waarbij de rest dit niet mag
- Tussen de data mogen na de ”,”geen spaties staan

5.2.2 Keuze van de test, labeled, unlabeled dataset

Een belangrijke keuze in de opmaak van het originele databestand, is het opsplitsen van deze data in verschillende onderdelen. Een databestand kan namelijk gesorteerd zijn volgens een bepaald attribuut. Dit kan negatieve gevolgen hebben in het trainen van het model. Het model zou in dit geval slechte voorspellingen maken op ongeziene attributen/data. Voor de eerste testen van dit onderzoek werd voor de $75\% - 25\%$ regel gekozen. Er kan achter ook gekozen worden om cross-validation toe te passen. Deze regel suggereert dat het aantal originele examples wordt opgesplitst in 75% voor het trainen van het model en 25% voor het testen of verifiëren van het model. Omdat in ons geval zowel labeled, unlabeled als test data nodig is, werd er gekozen om van het originele 20% te kiezen voor de testen en deze test data ook te gebruiken bij het evalueren van de random forests. Van de overige 80% werd opnieuw gekozen voor 20% te nemen voor de pure labeled data, die verder ook opgenomen wordt in de labeled + unlabeled dataset. Waarbij de rest als extra wordt toegevoegd aan het voorgaande bestand.



Figuur 5.3: Voorbeeld van data bestand verdeling

K-Fold cross-validation is een andere techniek die de data in k−subsets verdeelt om het model te trainen waarbij telkens één andere subset wordt weggelaten om als testdata te fungeren. Dit wordt K keer uitgevoerd en aangewend om het model op te stellen. Voor de uitkomst kan dan een gemiddelde worden genomen over de verschillende accuracy's heen. Deze methode wordt echter niet gebruikt in deze thesis, maar kan wel als verder onderzoek gebruikt worden. In deze scriptie werd gekozen om alle regels in te lezen en te shuffelen. Zo wordt het probleem uit de eerste paragraaf verholpen, waarbij het getrainde model enkel een klein deel van een attribuut verkend heeft en de distributie behouden wordt.

Om de invloed van het unsupervised onderdeel te beïnvloeden, zal ook voor elke dataset 5% van de resterende data, na het reduceren van de test voorbeelden genomen worden, om te zien hoe zwaar unsupervised learning bijdraagt tot het verbeteren van het model.

5.2.3 Configuratie bestand

Naast de databank verwacht Clus ook een configuratie bestand zodat het duidelijk is met welke methodes en parameters er zal gewerkt worden bij elk algoritme. Dit bestand is onderverdeeld in de volgende categorieën te gebruiken voor de proefopstelling.

1. Data: bevat de bestandsnaam, testset
2. Attributes: bevat de target attribuut, clustering, beschrijvende attributen, ...
3. Model: Aantal elementen voor subdivisie in clustering of aantal voorbeelden dat een bladknoop mag omvatten
4. Tree: pruning methodes
5. Output: WritePredictions: schrijf testen naar een bestand
6. Ensemble: RandomForests gebruiken, aantal bomen die gegenereerd moeten worden, schrijf bomen naar bestand, ...
7. Constraints: MaxDepth, de diepte die een boom mag aannemen bij het trainen van een model

De voorgestelde methode van Levatić et al. [13] laat toe om extra parameters specifiek voor SSL aan te passen. Hieronder is een overzicht van de mogelijke opties bij de tag van SemiSupervised:

1. Unlabeled data : de naam van het ongelabelde bestand
2. SemiSupervisedMethod : PCT of Predictive Clustering Trees

Voor de standaard proefopstelling hebben we geopteerd om de volgende eigenschappen te verwerken. Zoals op afbeelding 5.4 te zien is, hebben we de maxDepth op Infinity gezet. Een nadeel hiervan is dat overfitting kan ontstaan. De MinimalWeight hebben we op 1.0 ingesteld, wat aanduidt dat alle voorbeelden tot een unieke bladknoop resulteren. Dit zijn twee parameters die aanduiden dat het model nog verder configureerbaar is, maar kan leiden tot overfitting. De optie SelectRandomSubspaces definieert hoeveel features er genomen worden voor Bagging en het opstellen van de tree.

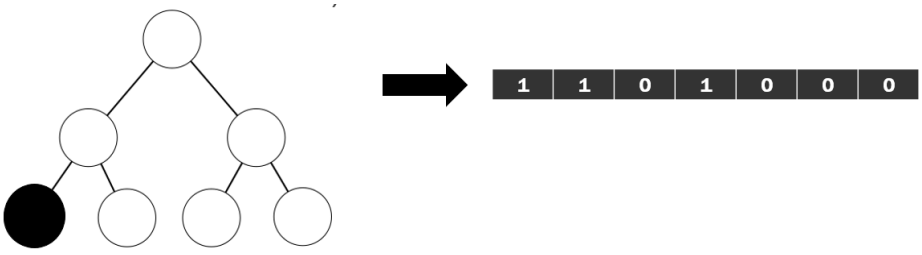
```
[Model]
MinimalWeight = 1.0

[Tree]
PruningMethod = None

[Constraints]
MaxDepth = Infinity

[Output]
WritePredictions = Test

[Ensemble]
EnsembleMethod = RForest
Iterations = 50
OOBestimate = No
Optimize = No
PrintPaths = Yes
PrintAllModels=Yes
SelectRandomSubspaces = 5

[Data]
File = labeled.arff
TestSet = test.arff
```

Figuur 5.4: Voorbeeld van configuratie bestand

5.2.4 Automatiseren van deeltaken via perl script

Een belangrijke deeltaak is het automatiseren van deze deelstappen via scripts. Dit laat toe dat de conversie naar de binaire attributen eenvoudig en snel kan verlopen. Het meegekregen script maakt gebruik van Unix commando's en moet dus, zonder enige vertaling naar DOS, in een unix omgeving uitgevoerd worden. Voor deze automatisering hebben we een virtuele Ubuntu machine ingeschakeld. Deze scripts genereren telkens drie ARFF bestanden, om via een Java applicatie te laten verwerken. Het meegekregen script bestaat uit twee subdelen:

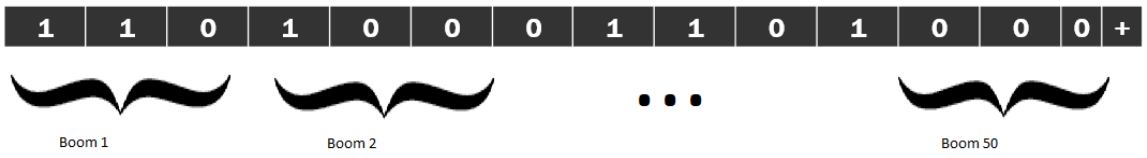
1. Het toepassen van Random Forest met de opgestelde configuratie
2. Het omzetten van deze resultaten naar binaire vorm



Figuur 5.5: Voorbeeld transformatie naar binair

Zoals op afbeelding 5.5 te zien is, verloopt de conversie van een tree naar binaire representatie als volgt:
Via een subset van de features genereren we telkens bomen. Elke boom wordt in een volgende stap binair geconverteerd naar een array van 0'en en 1'en. Het target attribuut wordt echter niet getransformeerd. De reden hiervoor is dat we de classificatie moeten bewaren en dat dit geen invloed mag hebben bij de reconstructie van de boom zelf. Deze stap herhalen we en passen we toe voor het aantal bomen dat we moeten genereren via het configuratie bestand van Clus.

Wanneer voor elk voorbeeld, en dit voor alle bomen, de binaire transformatie uitgevoerd werd, zorgt de concatenatie hiervan tot de uiteindelijke binaire representatie van de gelabelde data. Een probleem is dat voor elke boom het volledig pad binair voorgesteld moet worden. De reden voor deze keuze is dat we anders geen onderscheid kunnen maken waar elke boom effectief start en op welke positie deze zou stoppen. Een nadeel daarentegen is dat soms onnodige takken hierin opgeslagen worden, of met andere woorden, wanneer een boom bestaat uit 1500 nodes, dan moet voor elk example een array voorzien worden van lengte 1500, wat in het geval van een 50 tal bomen, dit snel resulteert tot een opslag van 75.000 0'en en 1'en. Bij een 10.000 aantal examples kan dit snel leiden tot memory overflow.
In de afbeelding hieronder is een voorbeeld concatenatie te zien. Deze vorm is echter nog niet volledig om te gebruiken in de semi-supervised omgeving.



Figuur 5.6: Afbeelding concatenatie

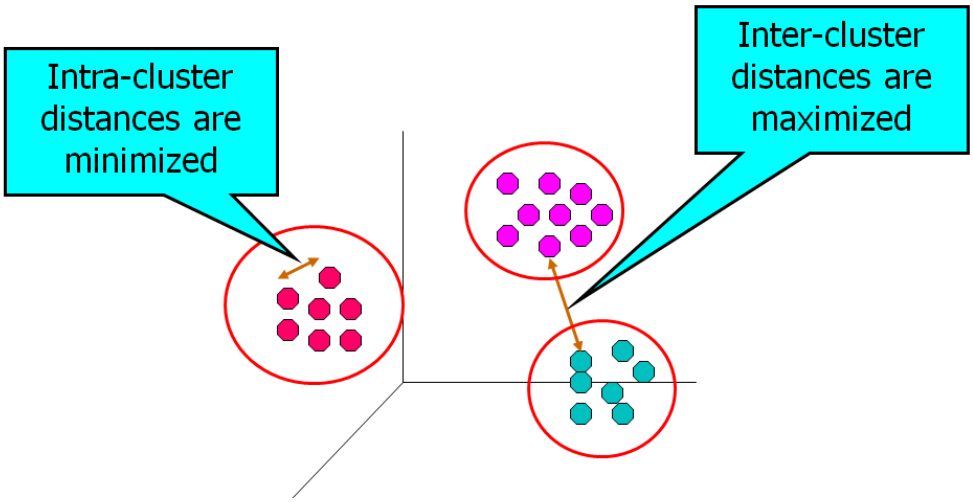
Voordat het model bruikbaar is voor verdere verwerking, dienen we de labeled + unlabeled dataset mee in evaluatie te nemen, door een tweede model te bouwen gebaseerd op Clustering. Zoals vermeld in Hoofdstuk 3, is naast de labeled, ook de unsupervised informatie nodig om volledig te spreken over semi-supervised. Voor deze stap raden we de volgende configuratie aan:

```
[Model]
MinimalWeight = 1.0

[Tree]
PruningMethod = None

[Constraints]
MaxDepth = Infinity

[Output]
WritePredictions = Test

[Ensemble]
EnsembleMethod = RForest
Iterations = 50
OOBestimate = No
Optimize = No
PrintPaths = Yes
PrintAllModels=Yes
SelectRandomSubspaces = 5

[Attributes]
Descriptive=1-22
Disable=23
Target=1-22
[Data]
File = lu.arff
TestSet = test.arff
```

Figuur 5.7: Afbeelding LU configuratie bestand

In dit bestand kunnen we de minimalweight eveneens aanpassen. Deze parameter heeft invloed op het aantal elementen die tenminste nodig zijn voor een cluster kan bestaan. De maxDepth op infinity laat toe om een cluster voor elk uniek example te schetsen. Het tweaken of aanpassen van deze parameters kan grote gevolgen hebben op het model dat gegenereerd wordt. De oorzaak van deze grote aanpassingen, is dat bomen "weak learners"zijn.

Deze methodiek genereert opnieuw 50 bomen waarbij de target disabled is. Hierbij opteren we om niet de classificatie accuracy zo hoog mogelijk te zetten, maar eerder om de afstand binnen elke cluster te minimaliseren. Zoals te zien is op afbeelding 5.8. In dit geval kan aangenomen worden dat elke blad een subset is van een grotere boom. Hierdoor hoeven enkel de blader knopen opgeslagen te worden.
Bemerk dat opnieuw er wel onderscheid nodig is tussen elke boom. Zo kan de eerste boom groter zijn dan de tweede boom van Random Forests. Om het onderscheid te kunnen maken is nogmaals een 'sparse tree' of volledige structuur nodig. Voor de proef werd de intra-cluster afstand alvast geminimaliseerd [afb. 5.8].



Figuur 5.8: Voorbeeld clustering [1]

5.2.5 Conversie in Java

Om deze arrays te kunnen aanwenden, worden de bestanden sequentieel ingelezen, met name de test voorbeelden, de labeled, als de combinatie met de unlabeled. Deze stap wordt hierna kort uitgelegd, hoewel deze in principe irrelevant is tot het bijbrengen van SSL. Voor de implementatie in verband met het inlezen van deze zaken, is hoofdstuk 9 relevant.

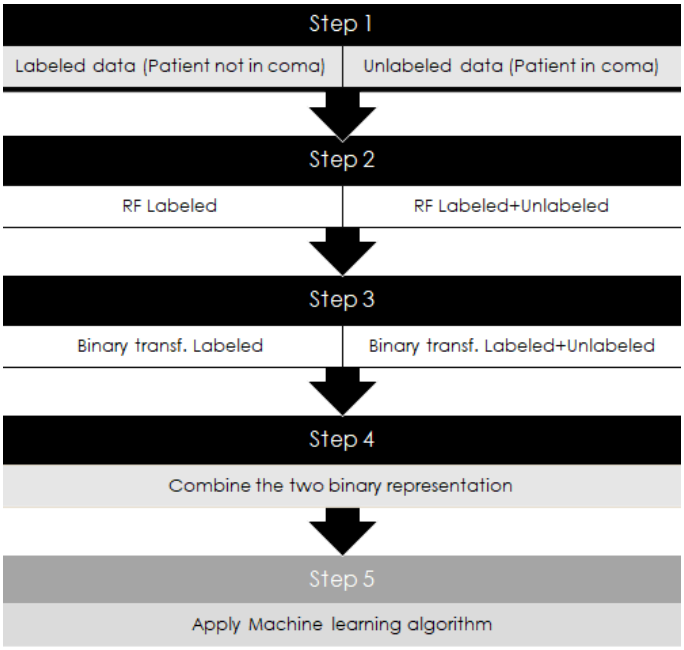
5.2.6 Toepassen van machine learning methodes

Een andere cruciale stap is het toepassen van een predictor op de voorbeelden in hun binaire vorm met dus de gelabelde en de ongelabelde informatie en vervolgens de classificatie te bepalen voor elk van hen. Daarna dienen we deze te vergelijken met de target-label. De totale accuraatheid kunnen we dan bepalen door het aantal correcte classificaties te tellen. Deze methodiek vergelijken we nu met andere technieken, wat ons toelaat de empirische studie te concluderen en vast te stellen of het gevonden model een verbetering is al dan niet.

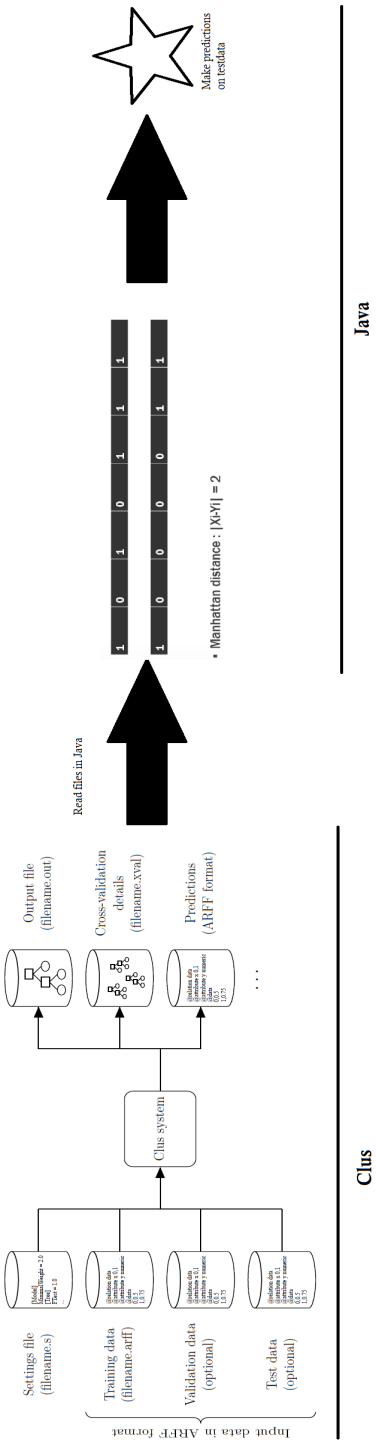
Wanneer de data (labeled, labeled + unlabeled en testdata) van alle drie de onderverdelingen binair is omgezet, wordt in de meeste gevallen het kNN-algoritme gebruikt. Dit algoritme gaat bepalen hoever elk test voorbeeld ligt van de trainingsdata. Het totaal aantal verschillende waardes binnen de array wordt aangewend als de afstand (Manhattan Distance). Wanneer voor alle trainingsdata de afstand berekend is voor een gegeven test case, dan kunnen de K-dichtste elementen bepaald worden voor het voorspellen van de klasse van de nieuwe test. Een belangrijk punt hierbij is dat K steeds een oneven aantal moet zijn om zeker een majority klasse te identificeren.

5.3 Architectuur en Tools

De tools binnen deze scriptie bestaan voornamelijk uit Netbeans en Clus. Netbeans helpt bij het implementeren van de verschillende machine learning algoritmes en het sequentieel inlezen van de bestanden gegenereerd door Clus. De keuze is tevens gebaseerd op de Java achtergrond kennis verworven in de voorbije jaren. Ook Perl is ingelast voor de automatisatie van de scripts binnen Clus. De onderstaande afbeeldingen bieden de lezers een gedetailleerde achtergrond hoe elk proces verloopt tot het uiteindelijke model en hoe goed het model scoort op de verschillende datasets. Voor de datasets zelf verwijs ik dan ook door naar sectie 6.1.



Figuur 5.9: Stappenplan van het model



Figuur 5.10: Flowchart van het proces [11]

Hoofdstuk 6

Evaluatie

6.1 Datasets

De gedefinieerde methode uit sectie 5 hebben we toegepast op 10 datasets. Deze datasets geven ons inzicht of het gevonden model toegepast kan worden. Daarnaast laat het toe om bepaalde kenmerken te typeren die noodzakelijk zijn in de preprocessing van elke set. De data verzameld voor deze proefopstelling, is afkomstig van de UCI [7]. Deze bron omvat een aantal datasets, waar een beschrijving staat wat het target attribuut is en welke eigenschappen er effectief opgenomen zijn.
Een overzicht van de opgenomen datasets, met de bijbehorende target:

- Abalone: Waarbij de target bestaat uit de leeftijd van het dier.
- Bank: Waarin het doel bestaat uit het uitmaken of een klant een termijndeposito zal doen.
- Bank note: Met als target of een banknote geauthenticeerd zal worden.
- Breast cancer: Heeft als target of een tumor goed- of kwaadaardig is.
- Biodegradation: Waarin de target bestaat uit het bepalen of iets bio-afbreekbaar is al dan niet.
- Car: Heeft als target een auto classificatie.
- Horses: Waarbij de target probeert te bepalen of een aandoening te wijten is aan een operatie.
- Ionosphere: Waarbij de target bestaat of ionen goed of slecht waren
- Mushroom: De target heeft hierin de classificatie of een paddenstoel eetbaar of giftig is.
- Musk-molecules: Waarin de target bestaat uit musk of niet-musk.
- Nursing: Heeft als target of een persoon ouderenzorg vereist.

Al snel blijkt dat in de eerste implementatie (zonder de voorgestelde preprocessing methode) van het kNN algoritme een typerend kenmerk naar boven komt. Namelijk het onderscheid tussen de nominale en numerieke waardes, die willekeurig in de datasets attributen voorkomen. In de eerste testen blijkt dat datasets waarin alle waardes nominaal of numeriek waren, de beste zijn. Dit komt duidelijk naar voor bij het bepalen van de afstand, met name, bij de nominale waarden die een afstand van 1 krijgen, indien ze verschillen of numerieke waarden aantonen, en waarin de afstand bepaald wordt tussen de specifieke waarden. Een oplossing om dit onderscheid te bewaren verkrijgen we door het bijhouden van welke type elke eigenschap heeft.
Deze bevinding resulteerde vanuit het zoeken naar andere datasets die duidelijk volledig uit nominale of numerieke waardes bestonden. Later in het onderzoek werden dan ook de eigenschappen van elke feature bijgehouden om beter de Manhattan afstand te kunnen meten tussen de verschillende voorbeelden. Volgende datasets zijn een collectie van "pure datasets":

- Balance: Waarbij de target bestaat uit waar een weegschaal zich zal bevinden.
- Income: De target bestaat of de persoon meer of minder zal verdienen dan 50.000$ op jaarbasis.

De finale oplossing om geen onderscheid te moeten maken tussen de verschillende types van attributen, was het toepassen van verschillende technieken. Zo is het onder meer mogelijk om nominale attributen te converteren naar een vector. De vector neemt dan een 1 aan wanneer een waarde van dit nominaal attribuut specifiek van toepassing is voor een example, en 0 bij de andere waardes van deze nominale feature. Bij numerieke waarden, wordt aangeraden om normalisatie te gebruiken, en laat toe de afstand te generaliseren over verschillende meeteenheden heen. Dit proces kan eenvoudig toegepast worden met de Weka tool.[19]

6.1.1 Invloed van SSL gedeelte

In deze paragraaf bespreken we hoe we de invloed van het unsupervised gedeelte kunnen bepalen. om dit te realiseren bestaan er vier verschillende werkwijzen. Een eerste werkwijze is specifiek voor SSL Clus [13] of de methode voorgesteld door:

$$Impurity_{SSL}(E) = w \cdot Impurity(E_l, Y) + \frac{1-w}{D} \cdot \sum_{i=1}^{D} Impurity(E, X_i)$$

waarin we met de w bepalen hoe zwaar elk gedeelte in consideratie moet genomen worden. Tijdens de uitvoering wordt 3-fold cross validation gedaan, waarbij de weight telkens varieert tussen $[0-1]$. Deze start bij 0 en wordt per sprong met 0.1 verhoogd. De andere werkwijzen om invloed te hebben zijn specifiek voor de methode voorgesteld binnen deze thesis.

- de hashing methode
- de te nemen attributen bij het opstellen van Random Forests
- de maximale diepte van de boom
- het aantal bomen

Een nadeel aan de twee laatste werkwijzen is dat de accuraatheid afneemt bij het maken van clusters in het unsupervised gedeelte. De fout die verminderd wordt door meer bomen in consideratie te houden, neemt hiermee terug toe. Daarnaast lijdt het beperken van de maxDepth tot pruning, waarbij ook de nauwkeurigheid afneemt. [3]

6.2 Resultaten

In de volgende sectie bespreken we de verschillende resultaten. Een overzicht verduidelijkt hoe goed elk algoritme bij elke dataset scoort. Naast de verschillende methodes spelen we ook met de parameters in Clus, onder meer zal de minimalWeight aangepast worden. Deze eigenschap heeft invloed op een vereiste wanneer een leaf verwacht wordt in de boom, of wanneer de dataset verder opgesplitst. Daarnaast heeft deze eigenschap invloed op de unsupervised methodiek. Het aantal attributen zichtbaar in de laatste twee kolommen werd bekomen bij een maxDepth van Infinity. Voor de onderverdeling van de data in de 3 groepen verwijzen we terug naar sectie 5.2.2. Wanneer voor andere onderzoeken een andere onderverdeling gebruikt wordt, lichten we dit verder toe.

6.2.1 Parameters

De standaard waarden voor de parameters, ingesteld voor de proeven, zijn de volgende:

- RF: 50 bomen
- maxDepth: Infinity (Maximale diepte van de bomen gegenereerd door Random Forests)
- kNN: 5 buren
- Threshold: 80% voor de semi-supervised learning technieken
- Kernel SVM: radial basis function kernel
- Methode van Levatić et al. [13] : SSL-RF

(Specifieke parameters worden verder in het document toegelicht.)

6.2.2 Resultaten kNN

Dataset	#training	#test	#attributen	#L bin. attr.	#L+U bin. attr.
Abalone	668	835	12	10292	233074
Balance	100	125	22	2182	15544
Bank	723	904	50	7122	110772
BankNote	219	274	6	1010	3136
Biodegradation	168	211	43	1980	51366
BreastCancer	112	139	11	642	20888
Car	276	345	8	4434	49382
Horses	47	59	58	868	14030
Income	5209	6511	106	40333	*MO*
Ionosphere	56	70	36	492	17560
Mushroom	1300	1624	128	2514	30228
Musk Molecules	76	95	168	914	23972
Nursing	2073	2592	28	20722	243394

Tabel 6.1: Overzicht datasets
L : Labeled
L+U : Labeled + Unlabeled

In de eerste testen komt naar voor dat voor bepaalde sets er een enorme hoeveelheid aan attributen gegenereerd wordt. Vanaf het aantal features bij L+U over de 100.000 optekent, gaf dit tijdens de trainingsfase een memory overflow. Dit komt overeen met het niet kunnen opslaan van het getrainde model. Voor de testen op de accuraatheid resulteerde dit in een error, die in onderstaande tabel te zien is. Voor het kNN onderdeel opteerden we voor: 5 dichtste buren.

Dataset	Sup. Or.	$Self.Or_{80\%}$.
Abalone	75%	72%
Balance	68%	76%
Bank	89%	87%
BankNote	96%	100%
Biodeg	79%	87%
BreastCancer	95%	96%
Car	84%	86%
Horses	79%	71%
Income	82%	82%
Ionosphere	77%	88%
Mushroom	99%	100%
Musk Molecules	67%	76%
Nursing	89%	92%

Tabel 6.2: Accuracy bij verschillende algoritmes maxDepth: Infinity
S.Or.: Deze methodiek stemt overeen met supervised learning en dit zonder enige transformatie op de data.
Self Or.: Self-learning is hier van toepassing op de originele data.

Uit tabel 6.2 zien we dat in sommige gevallen het opnemen van ongelabelde data, waarbij een grote zekerheid bestaat dat een voorspelling kan gedaan worden, leidt tot een verbetering van het model. Het is belangrijk in deze test dat op de ongelabelde data geen clustering wordt toegepast.

Dataset	Sup. binair	SSL binair	SSL Clus	SSL Clus binair
Abalone	77%	*MO*	78%	77%
Balance	76%	76%	80%	75%
Bank	90%	89%	94%	89%
BankNote	96%	97%	95%	95%
Biodeg	81%	81%	82%	81%
BreastCancer	96%	95%	97%	97%
Car	89%	82%	89%	88%
Horses	81%	79%	80%	83%
Income	84%	*MO*	86%	84%
Ionosphere	90%	91%	87%	87%
Mushroom	99%	99%	100%	99%
Musk Molecules	77%	73%	73%	71%
Nursing	94%	*MO*	94%	93%

Tabel 6.3: Accuracy bij verschillende algoritmes maxDepth: Infinity
Supervised binair: voert met de binaire representatie van de trainingsdata een classificatie uit
SSL binaire: voert aan de hand van de uitbreiding van de trainingsdataset met de clusterinformatie een classificatie uit
SSL Clus: voert een classificatie uit waarin de beslissingsbomen opgebouwd zijn via een combinatie van supervised en unsupervised technieken
SSL Clus binair: voert een classificatie uit waarin de data eerst getransformeerd wordt in zijn binaire transformatie voordat de methode van Levatić toegepast wordt

Om een vergelijking van SSL methodes toe te laten, hebben we de kolom van SSL Clus toegevoegd. Deze werkwijze werd extern geïmplementeerd en vrijgemaakt voor het publiek in 2018.[13] Tevens leiden we met de huidige verdeling van testen (20% test, 20% labeled en 80% labeled + unlabeled) een grotere verscheidenheid af uit de resultaten. Via het spelen met de verdeling in test, labeled en unlabeled, vonden we uiteindelijk een verdeling die optimaal is. Hierin moeten we concluderen dat in de meeste gevallen het voorgestelde transformatie model slechter scoort, maar anderzijds is op de methode van Levatić et al.[13] geen kNN toegepast. Hiervoor werd gekeken naar het output bestand en meer bepaald de accuraatheid gegeneerd door Clus.

De volgende tabel gebruikt dezelfde verdeling van de data. De maxDepth werd echter sterk gereduceerd om het aantal attributen fors te doen dalen, zodat hetzelfde model ook toegepast kan worden op datasets die memory overflow genereren. Tevens laat dit toe de invloed van het Unsupervised gedeelte te gaan beperken. In tabel 6.1 is duidelijk te zien dat dit gedeelte de grootste invloed heeft op de afstand, waardoor de verlaging van de accuraatheid kan te wijten liggen.

Dataset	#training	#test	#attributen	#S bin. attr.	#SSL bin. attr.
Abalone	668	835	10	2724	3076
Income	5209	6511	106	1316	2646
Nursing	2073	2592	28	1524	3146

Tabel 6.4: Overzicht datasets na aanpassing maxDepth: 5 bij super- en unsupervised learning

Een belangrijk nadeel met deze beperking is dat pruning in een vroeg stadium voorkomt. Dit is niet direct zichtbaar bij de resultaten van onderstaande tabel. Slechts één dataset gaat er effectief op achteruit. Uit verdere testen bleek dat de afwijkingen van het Unsupervised gedeelte afkomstig zijn en dit enkel bij de voorgestelde methode is en niet bij SSL Clus. Door het aanwenden van de maxDepth, wordt namelijk vroeger een leaf geplaatst, waar anders de boom eerder opteerde om nog een opsplitsing te maken. Waar we een opsplitsing verwachten van zowel een indeling op positieve als negatieve voorbeelden, wordt nu echter een leaf gevormd, en wordt dus vanaf nu mogelijks aan generalisatie gedaan.

Dataset	Sup. Binair	SSL Binair
Abalone	78%	78%
Income	89%	87%
Nursing	93%	89%

Tabel 6.5: Accuracy bij verschillende algoritmes na aanpassing maxDepth: 5 bij super- en unsupervised

Zoals reeds vermeld kan de accuracy verlagen wanneer het unsupervised gedeelte te zwaar doorweegt. Om dit te beperken werd gekozen om de maxDepth bij het unsupervised gedeelte te verkleinen, zodat telkens het supervised gedeelte een zwaardere invloed kan hebben, met als doel de accuraatheid terug te verbeteren. In onderstaande tabel hebben we dit toegepast op de datasets waarbij na het SSL gedeelte dit een negatieve impact had.

Dataset	#training	#test	#attributen	#L bin. attr.	#L+u bin. attr.
Bank	723	904	50	7122	3016
BreastCancer	112	139	11	642	350
Car	276	345	23	4434	3118
Horses	47	59	58	868	350
Musk Molecules	76	95	168	914	350

Tabel 6.6: Overzicht attributen maxDepth: Infinity Supervised, Unsupervised beperkt

Dataset	S. binair	SSL binair	#S. bin	#SSL bin	maxDepth Unsupervised
Bank	90%	90%	814/904	814/904	5
BreastCancer	96%	96%	134/139	134/139	2
Car	89%	83%	308/345	288/345	5
Horses	81%	81%	48/59	48/59	2
Musk Molecules	77%	75%	74/95	72/95	2

Tabel 6.7: Accuracy bij verschillende algoritmes maxDepth: Infinity bij Supervised, beperkt bij Unsupervised

Bij tabel 6.7 werd gekozen voor een maxDepth van 5 of 2 naarmate het aantal attributen bij Clustering nog steeds groter was dan het aantal features bij enkel de gelabelde data. Het beoogde effect om het SSL gedeelte beter te krijgen dan het pure supervised, is nog altijd niet aanwezig.
Een reden hiervoor kan liggen bij het voorgestelde model de "Curse of dimensionality". Om dit verder te onderzoeken hebben we geopteerd om de maxDepth incrementeel toe te laten nemen en na te gaan of een bepaald gewicht van het unsunsupervised daardoor in een verbetering resulteert (wat gelijkaardig is aan het tweaken met het SSL gedeelte bij Clus of de methode van Levatić et al.).
De volgende set van testen geven aan hoe we incrementeel de maxDepth hebben aangepast om zo een verhouding te vinden waarin het unsupervised gedeelte een effectieve verbetering vormt. Bij de tests werd gekozen voor $5, 10, 20$ als diepte.

Dataset	S. binair	SSL binair
Abalone	77%	76%
Balance	76%	76%
Bank	90%	89%
BankNote	96%	95%
Biodeg	81%	81%
BreastCancer	96%	97%
Car	88%	83%
Horses	81%	79%
Income	*MO*	82%
Ionosphere	90%	90%
Mushroom	99%	99%
Musk Molecules	74%	72%
Nursing	*MO*	*MO*

Tabel 6.8: Accuracy bij verschillende algoritmes maxDepth:5 Unsupervised

Dataset	S. binair	SSL binair
Abalone	77%	76%
Balance	76%	74%
Bank	90%	90%
BankNote	96%	95%
Biodegradation	81%	83%
BreastCancer	96%	94%
Car	88%	81%
Horses	81%	76%
Income	*MO*	*MO*
Ionosphere	90%	90%
Mushroom	99%	99%
Musk Molecules	74%	74%
Nursing	*MO*	*MO*

Tabel 6.9: Accuracy bij verschillende algoritmes maxDepth:10 Unsupervised

De dataset R = Mushroom had een toename in het aantal correcte predicties, maar bleef procentueel hetzelfde.

Dataset	S. binair	SSL binair
Abalone	77%	76%
Balance	76%	76%
Bank	90%	89%
BankNote	96%	97%
Biodegradation	81%	81%
BreastCancer	96%	95%
Car	88%	82%
Horses	81%	79%
Income	*MO*	*MO*
Ionosphere	90%	91%
Mushroom	99%	99%
Musk Molecules	74%	72%
Nursing	*MO*	*MO*

Tabel 6.10: Accuracy bij verschillende algoritmes maxDepth:20 Unsupervised

Uit de testen blijkt dat elke dataset een weight heeft waarbij de accuraatheid tot een lokaal maximum stijgt. Bij elke dataset lijkt het dus belangrijk om dit maximum te vinden. Ook de opgenomen SSL Clus [13] methode probeert dit te doen door het optimale gewicht te vinden tussen het interval $[0-1]$. Hieruit zien we dat het probleem zich gedraagt als een maximalisatie probleem. Verder onderzoek is nodig om dit te bewijzen of te ontkrachten, wat echter buiten de scope van deze thesis zich situeert.

Een andere test, die ook een beeld kan geven hoe invloedrijk het unsupervised gedeelte weldegelijk is, verkrijgen we via het spelen met het percentage van het aantal labeled/

unlabeled examples. In deze proefopstelling werd gekozen om 5% te nemen van de totale dataset en deze te gebruiken als labeled data. De gevallen waar er in eerste instantie memory overflow was, werden beperkt tot een maxDepth van 5, om zodoende toch enig beeld te krijgen hoe invloedrijk het unsupervised wel degelijk is. Tabel 6.12 en 6.11 toont ons de resultaten van deze proefopstelling, waarin te zien is dat self-learning en de binaire transformatie in een verbetering resulteert ten opzichte van de originele representatie, maar waar ook uit af te leiden is dat het toevoegen van clustering informatie niet leidt tot een optimalisatie.

Dataset	Sup. Or.	$Self.Or_{80\%}$.
Abalone	75%	65%
Balance	63%	75%
Bank	89%	89%
BankNote	95%	99%
Biodeg	82%	87%
BreastCancer	62%	71%
Car	72%	86%
Horses	62%	71%
Income	81%	MO
Ionosphere	81%	91%
Mushroom	98%	100%
Musk Molecules	62%	82%
Nursing	82%	93%

Tabel 6.11: Accuracy datasets met 5% als labeled data

Dataset	Sup. binair	SSL binair	SSL Clus	SSL Clus binair
Abalone	75%	74%	76%	TD
Balance	55%	61%	61%	TD
Bank	90%	90%	90%	TD
BankNote	91%	91%	91%	TD
Biodeg	78%	75%	80%	TD
BreastCancer	64%	62%	97%	TD
Car	77%	78%	79%	TD
Horses	64%	57%	64%	TD
Income	83%	82%	86%	TD
Ionosphere	92%	74%	88%	TD
Mushroom	99%	98%	99%	TD
Musk Molecules	56%	56%	61%	TD
Nursing	91%	87%	93%	TD

Tabel 6.12: Accuracy datasets met 5% als labeled data

6.2.3 Resultaten SVM

Bij de toepassing van SVM's voor de proefopstelling hebben gebruik gemaakt van LIBSVM [15]. Deze tool is een library ontwikkeld door Chih-Chung Chang and Chih-Jen Lin, en bevat een voorgedefinieerd script 'easy.py',die bestaat uit een reeks van generale stappen die via een eenvoudige voorstelling uitermate eenvoudig aan te wenden is. Voor de proefopstelling zelf hebben we drie testcases uitgewerkt, waaronder de originele representatie, de voorgestelde methode, en de methode van SSL Clus.

Zoals reeds vermeld voert het script een tal van stappen uit voordat de classificatie kan uitgevoerd worden. Een belangrijk onderdeel hiervan is het scalen van de attributen. De reden om dit nauwgezet te doen, is dat het vinden van een scheidingslijn zonder scaling, voor verschillende groottes zorgt voor elke feature. Dit normaliseren geeft een natuurlijker beeld weer waarin elke feature evenveel invloed heeft. Eénmaal deze stap afgewerkt is, wordt cross-validatie toegepast om de parameters behorende bij de kernel function te optimaliseren. Als kernel gebruiken we de RBF of Radial Basis Function bij de proefopstelling. Bij de correcte uitvoering van deze stap, kan een model bestand gecreëerd worden als evaluatie op het test dataset.

Bij het uitvoeren van het experiment bleek echter dat de predictie bij de voorgestelde methode en bij SSL Clus, slecht te scoren. De reden was voornamelijk te wijten aan het scalen van de binaire representatie, waarbij sommige attributen tijdens het trainen van het model niet toegevoegd werden aan de scaling file. Dit bestand geeft eigenlijk de extrema weer van elk attribuut. Wanneer dus een bepaalde branch nooit doorlopen wordt, verkrijgt dit de waarde 0. Om dit probleem op te lossen hebben we de scaling bij de binaire representatie weggelaten. Feitelijk is dit logisch en het heeft ook niet direct scaling nodig. De extrema hierbij kunnen enkel de waardes 0 en 1 aannemen. Na het weglaten van deze stap, bekwamen we een enorme aanwinst van de accuraatheid bij de binaire modellen($\pm$20%). Tabel 6.13 toont de accuraatheid bij SVM met de RBF:

Dataset	Sup. Or.	Sup. binair	SSL binair	SSL Clus binair
Abalone	73%	74%	74%	77%
Balance	74%	78%	50%	70%
Bank	88%	88%	88%	88%
Banknote	93%	95%	57%	95%
Biodeg	65%	80%	65%	79%
BreastCancer	95%	96%	96%	96%
Car	78%	80%	70%	83%
Horses	68%	69%	68%	68%
Income	83%	MO	MO	84%
Ionosphere	77%	91%	64%	65%
Mushroom	100%	100%	MO	100%
Musk Molecules	62%	80%	61%	69%
Nursing	94%	89%	MO	89%

Tabel 6.13: Accuracy SVM : maxDepth= Infinity

Uit bovenstaande tabel blijkt de transformatie van de gelabelde data naar zijn binaire representatie, tot een verhoging van de accuraatheid leidt. Echter indien we de ongelabelde data bij clustering toevoegen, dan treedt er een waarneembare daling op. Ook het voorgestelde model van Levatić et al.[13] doet niet beter dan de transformatie. Om het verschil aan te duiden tussen het gebruik van SVM's en kNN hebben we beiden tabellen nog eens samengevoegd. Hieruit kunnen we ook afleiden dat de keuze van het learning proces duidelijk invloed heeft op de accuraatheid. Beide elementen vormen zeker onderwerp om grondig te controleren op de dataset van ICU.

Dataset	SSL Clus SVM	SSL Clus kNN
Abalone	77%	77%
Balance	70%	75%
Bank	88%	89%
Banknote	95%	95%
Biodeg	79%	81%
BreastCancer	96%	97%
Car	83%	88%
Horses	68%	83%
Income	84%	84%
Ionosphere	65%	87%
Mushroom	100%	99%
Musk Molecules	69%	71%
Nursing	89%	93%

Tabel 6.14: Accuracy SVM & kNN

6.2.4 SVM's en self-learning

Naast de traditionele manier van SVM's, bestaat er ook een parameter binnen libsvm, die probabiliteiten als uitvoer geeft bij vector machines. Deze optie hebben we meegenomen in het onderzoek, omdat een vergelijking mogelijk moet zijn met de self-learning bij kNN. Zo heeft de library een functie genaamd probability estimate, die aangeeft met welke zekerheid een klasse wordt toegekend aan een example. Indien een voorbeeld zich ver van de hyperplane bevindt, dan heeft deze een grote zekerheid, dat hij een bepaalde target krijgt. Echter de examples die zich dicht bij het hypervlak bevinden, worden met een kleinere zekerheid geclassificeerd. In het geval van self-learning moet een bepaalde threshold gekozen worden. Voor de proefopstelling in tabel 6.15 is geopteerd voor een threshold van 80%.

Bij het uitvoeren van deze functie wordt voor elke example een target label berekent, met de kans dat deze tot deze klasse hoort en zijn complementaire kans. Om self-learning toe te passen is de volgende strategie ontwikkeld, die dient toegepast te worden op de ongelabelde data.

1. Maak een model van de gelabelde data en maak een predictie met de bijbehorende probabiliteiten op de ongelabelde data
2. Voeg de examples waarin de kans groter is dan de threshold toe aan de gelabelde data.
3. Creëer het model met de toegevoegde ongelabelde voorbeelden
4. Test het finale model op de uiteindelijke testset

Het is dus de bedoeling dat in de eerste stap van het proces via libSVM [6] een model wordt gemaakt door middel van de trainingsdata en in samenwerking met de probability estimation. Eens het model opgesteld, kunnen we het toepassen op de ongelabelde data. Daardoor verkrijgen we voor elke example zijn bijbehorende kansen voor elke klasse, waarbij de library ook een predictie bestand genereert, zoals te zien is op figuur 6.1.

```
labels 0 1
0 0.923155 0.0768448
0 0.903041 0.0969588
0 0.910185 0.0898155
0 0.913041 0.0869587
0 0.907415 0.0925852
0 0.913118 0.0868823
0 0.906681 0.0933191
0 0.911354 0.0886464
0 0.902782 0.0972182
0 0.911926 0.088074
0 0.912948 0.0870521
0 0.906872 0.0931282
0 0.922968 0.0770322
0 0.903297 0.0967033
0 0.930491 0.0695087
0 0.898402 0.101598
0 0.922148 0.0778521
0 0.92009 0.0799105
0 0.935471 0.0645295
0 0.902412 0.0975885
0 0.907454 0.0925458
0 0.908412 0.0915878
```

Figuur 6.1: Afbeelding voorbeeld predictie bestand

Eens het bestand aangemaakt, kan het inlezen starten. Het verwerken van dit bestand hebben we opnieuw gedaan via een java klasse die het bestand sequentieel inleest en verwerkt. Deze applicatie, gaat voor elke regel beide kansen extraheren. Indien één van de twee kansen de voorgestelde threshold overstijgt wordt de bijbehorende example geselecteerd. In de tweede stap passen we de target van de ongelabelde data aan met de gesuggereerde label en nbemen we dit mee op in de trainingsdata. Dit is gelijkaardig aan de self-learning van kNN mits een reeds van aanpassingen van de data.

Wanneer deze stap compleet is, produceren we met de tool opnieuw een model op de nieuwe aangepaste trainingset en dit zonder de probability estimation. Vervolgens wordt dan het finale model geëvalueerd tegenover de originele testset en wordt een accuracy teruggegeven.

Dataset	$Self.Binair_{80\%}$	S.Or.
Abalone	74%	73%
Balance	81%	74%
Bank	89%	88%
BankNote	97%	93%
Biodeg	78%	65%
BreastCancer	96%	95%
Car	83%	78%
Horses	68%	68%
Income	83%	83%
Ionosphere	89%	77%
Mushroom	98%	100%
Musk Molecules	58%	62%
Nursing	94%	94%

Tabel 6.15: Accuracy op datasets met SVM en self-learning
S.Or. : Supervised learning op de orginele representatie
Self.Binair;: Self-learning met binaire transformatie

Uit de tabel is zichtbaar dat self-learning in samenwerking met support vector machines een verbetering is ten opzichte om enkel met de gelabelde data te werken. Wanneer we de voorgestelde transformatie er bij nemen, zorgt dit voor een afname van de accuraatheid. Waarschijnlijk is de curse of dimensionality opnieuw de speler die invloed heeft op de accuraatheid van het model. Op basis van deze uitspraak kunnen we concluderen dat het toevoegen van de ongelabelde data in de binaire representatie ervoor zorgt dat de accuraatheid achteruit gaat. Bij self-learning zorgt het toevoegen van de ongelabelde data dan eerder wel voor een verbetering van het model, maar dan wel in zijn originele representatie. De achteruitgang van voorgaande tabel bij $R = Musk$ is dat het model zeer slecht scoort op de ongelabelde data, waardoor vervolgens weinig examples worden opgenomen. Bij alle gevallen van deze dataset is er een grote onzekerheid bij de indeling van zijn classificatie.

Wanneer we de tabel 6.13 er bij nemen, kunnen we afleiden dat de transformatie van de originele naar de binaire reeds een verbetering markeert. Bij een volgend experiment trachten we de accuraatheid opnieuw te verhogen door eerst de transformatie uit te voeren op zowel de trainings en de ongelabelde data. In plaats van de concatenatie te gebruiken zoals eerst voorgesteld is, passen we de gegenereerde bomen van Random Forests toe op de ongelabelde data, waardoor deze data dezelfde lengte verkrijgt als de labeled dataset. Daarbij passen we dan self-learning toe zoals in de voorgaande stappen is voorgesteld.

6.3 Combinatie van Self-learning en de binaire transformatie

Voor de laatste test hebben we verondersteld dat semi-supervised learning het best werkt door geen clustering data toe te voegen, maar eerder via self-learning een threshold te bepalen en deze te gebruiken om ongelabelde data op te nemen in de originele trainingsdataset. Dit proces kunnen we iteratief uitbreiden waarbij we voorbeelden na het doorlopen van het proces mee opnemen. Wanneer een aantal voorbeelden opgenomen zijn, kan het proces uiteindelijk herhaald worden tot geen enkel voorbeeld meer kan toegevoegd worden (Origineel werd proces 1x doorlopen). Bij het doorlopen van een enkele cyclus blijken de resultaten bij sommige datasets zeer slecht en bij sommige zeer goed. Ook hieruit gaan we er van uit dat het toevoegen van ongelabelde data opnieuw een optimalisatieproces is.

Dataset	$Self.Binair_{80\%}$
Abalone	53%
Balance	81%
Bank	88%
BankNote	71%
Biodeg	67%
BreastCancer	70%
Car	70%
Horses	67%
Income	MO
Ionosphere	62%
Mushroom	64%
Musk Molecules	64%
Nursing	MO

Tabel 6.16: Accuracy bij datasets met binaire transformatie & self-learning met kNN & predictor kNN

De combinatie van deze technieken zorgt voor een grote detoriatie in de accuraatheid van de modellen. We mogen daaruit concluderen dat de combinatie van self-learning met de binaire transformatie bij kNN niet tot een verbetering van de modellen komt.

Dataset	$Self.Binair_{80\%}$
Abalone	MO
Balance	50%
Bank	88%

Tabel 6.17: Accuracy bij datasets met binaire transformatie & self-learning met kNN & predictor SVM

Uit de twee voorgaande testen volgt dat het model geen verbetering met zich meebrengt. De laatste manier waarop we dit toch willen doen werken is het proces omgekeerd te doen en opnieuw te valideren. Zo kunnen we kiezen om eerst SVM de kansberekening te laten uitvoeren op de ongelabelde data. Indien deze de threshold overstijgt, dan kunnen deze voorbeelden toegevoegd worden aan de trainingsdata, waarbij we vervolgens de transformatie toepassen en het model verder testen op de testdata.

Dataset	$Self.Binair_{80\%}$ predictor kNN	$Self.Binair_{80\%}$ predictor SVM
Abalone	75%	75%
Balance	80%	77%
Bank	89%	88%
BankNote	94%	93%
Biodeg	81%	77%
BreastCancer	96%	96%
Car	85%	82%
Horses	74%	68%
Income	*MO*	*MO*
Ionosphere	94%	93%
Mushroom	99%	98%
Musk Molecules	70%	63%
Nursing	*MO*	*MO*

Tabel 6.18: Accuracy bij datasets met self-learning met SVM & binaire transformatie

Deze manier van werken toont een grote verbetering met de vorige test. Hieruit kunnen we besluiten dat SVM beter geschikt is om te gebruiken bij self-learning. De laatste test die we uitvoeren is SVM als predictor te nemen na de transformatie. Deze resultaten zijn zichtbaar in de tweede kolom van de voorgaande tabel. In de volgende sectie geven we een in een overzicht/samenvatting welke testen we allemaal uitgevoerd hebben en welke conclusies we uit de studie mogen trekken.

6.4 Samenvatting

Na het uitvoeren van deze set van testen, is duidelijk af te leiden dat de keuze van de methode invloed heeft op de accuracy. Bij het bouwen van deze bomen is het belangrijk indachte te zijn in welke mate het unsupervised gedeelte invloed mag hebben in het model. In deze thesis hebben we 2 specifieke modellen onder de loep genomen.

1. SVM (RBF)
2. kNN

De eerste methode gaat aan de hand van een kernel functie een hyperplane construeren. In deze scriptie werd gekozen om RBF kernel te gebruiken. De reden hiertoe is dat in het algemeen deze kernel de beste resultaten verschaft. Voor de proeven werden de parameters voor de functie geoptimaliseerd en werd vervolgens het model toegepast op de testdataset.

De tweede manier of kNN, gaat voor elke example uit de testdata de afstand berekenen met de trainingdata en zal vervolgens a.d.h.v. van zijn 5 dichtste buren, majority voting toepassen om zijn target te bepalen. Om het unsupervised gedeelte te laten variëren bij deze methode worden twee manieren voorgesteld:

- Het aantal trees beperken (werd niet uitgewerkt in deze scriptie)
- De maxDepth vooraf te definiëren

Het doel bestaat erin om de beste verhouding te vinden tussen de gelabelde data en de ongelabelde data. Deze verhouding komt overeen met de methode van SSL Clus voorgesteld door Levatić et al.[13], waarin een gewicht wordt gegeven aan de entropy van het unsupervised gedeelte.

Bij de testen hebben we ook gekeken in welke mate het unsupervised gedeelte effectief een verbetering biedt. Wanneer de setup zo gekozen wordt dat van de totale dataset slechts 5% genomen mag worden als training, kan vervolgens door de conversie en kNN toe te passen, geverifieerd worden of er een toename is van de accuraatheid door SSL bomen. Dit is een element die we meenemen naar de dataset van ICU.

In het algemeen is vast te stellen dat de methodiek voorgesteld door Levatić et al.[13] de beste is bij kNN. Dit model werd grondig getest tegen het zelf voortgebrachte model, waarbij de resultaten duidelijk spreken. De aanpassing van de Entropy formule is momenteel een belangrijke keuze om semi-supervised voorspellingen te maken.

Een cruciaal besluit voortvloeiend uit de testen, is dat voor elke dataset we een optimale verhouding moeten vinden hoe zwaar het unsupervised gedeelte een rol mag spelen. Dit herleidt zich tot een optimalisatie proces.

6.4.1 Overzicht testen/uitbreidingen

Op basis van de vele uitgevoerde testen, en vooral omdat in deze scriptie het empirisch onderzoek centraal staat, is er een persoonlijke voorkeur om alle zaken nog eens helder samen te vatten. Zo werden er onder meer uitbreidingen gedaan op zowel het kNN algoritme als SVM's. In eerste instantie werden alle datasets gepreprocessed door de Weka tool [19].

- Normalisatie van de numerieke waarden, zodat éénheden geen invloed hebben op de afstand
- Transformatie bij de nominale waarden tot vectoren

Na het preprocessen van de datasets kan zowel kNN als SVM's worden toegepast. Dit resulteerde tot de eerste testen, waarbij het enkel mogelijk was supervised te testen en nog geen rekening te houden met de beperking van het aantal attributen. Om semi-supervised te werken werd een eigen methodiek voorgesteld, waarbij elk example wordt voorgesteld als zijn pad binnen de bomen gegenereerd door Random Forests. Daarnaast werd voor de ongelabelde data hetzelfde gedaan maar nu voor clustering, waarbij SSL tot stand komt via de concatenatie van beide zaken.[18][16] Om te zien hoe invloedrijk het unsupervised gedeelte effectief is, werd na deductie van de testset gekozen om 5% te nemen als de gelabelde data en de rest te gebruiken als ongelabelde data. Samengevat komt dit neer voor kNN tot de volgende proefopstellingen:

- Normale representatie van de data zonder ongelabelde data
- Self-learning waarin de ongelabelde data werd gebruikt en waarbij majority voting boven een bepaalde threshold moet zijn
- Binaire transformatie enkel van toepassing op de gelabelde data
- Uitbreiding van de omzetting met ongelabelde data onder de vorm van Clustering
- Voorstelling van de methode door Levatić et al. [13]
- Incrementeel laten toenemen van de maxDepth om een goede verhouding te vinden waarin de niet-gelabelde data het meest bijdraagt

Voor SVM's is geopteerd om libSVM te gebruiken waarin een optimalisatie van parameters mogelijk is. Om self-learning hierop te kunnen toepassen, dienden we vooraleerst een probability estimation model te maken, en die vervolgens op de ongelabelde data aan te wenden. Deze kansen werden dan gebruikt, en wanneer deze de threshold overstegen, hebben we ze vervolgens mee opgenomen in de trainingsdataset (6.2.4). Vooraleer de Levatić et al. [13] manier kan opgenomen worden binnen SVM's, moet het configuratie bestand aangepast worden zodat alle trees van Random Forests worden gegenereerd. In de volgende stap zijn deze bomen toegepast op alle examples naar hun respectievelijke binaire representatie. Voor SVM's hebben we de volgende zaken onderzocht:

- Originele representatie
- Binaire vorm met enkel de gelabelde data
- SSL, waarbij dus ook de niet-gelabelde data aanwezig is
- Levatić et al. methode
- Self-learning

Uit deze grote set van testen komt naar voor dat er verschillen zitten in de keuze van het model. Bij kNN kan de ongelabelde data onder de vorm van Clustering worden toegevoegd, waarin een goeie maxDepth gevonden moet worden specifiek voor elke dataset. Bij SVM's leidt het toevoegen van clustering informatie echter tot een afname van de accuraatheid en is het beter om Self-learning toe te passen op zijn originele vorm.

Hoofdstuk 7

Inzetbaarheid van het Model

7.1 Inleiding

Het UZ Leuven beschikt over een afdeling met naam intensive care unit (ICU) waar er mensen met levensbedreigende ziekten of problemen zijn opgenomen, met in het bijzonder kanker en zenuwaandoeningen. De behandeling hiervan heeft meestal zware gevolgen op de levenskwaliteit van de patiënt. Machine learning kan hiervoor in de toekomst een belangrijke rol spelen, met name, het zo snel mogelijk een kwalitatieve analyse opstellen voor de patiënt in kwestie. Dit deel van de scriptie focust zich in het opnemen van een specifieke casus, meer bepaald in het domein van de medische wereld. Fouten zijn in deze omgeving per definitie niet tolereerbaar, wat ook de reden is waarom er zoveel testen werden uitgevoerd bij het opstellen van het model.

7.2 Toepassing in Ziekenhuisomgeving

Vanuit het ziekenhuis werd een dataset met niet-fictieve gegevens overgemaakt. Wegens de wet van privacy en persoonsgegevens, kunnen we deze dataset niet zomaar publiekelijk aanwenden. Voor het gebruik ervan werd dan ook een non-disclosure agreement ondertekend.

Het model aangenomen in deze thesis heeft een predictie uitgevoerd op de ontvangen dataset omtrent de intensive care unit acquired weakness of kortweg ICUAW. Deze dataset bestaat uit 598 voorbeelden en een totaal van 478 relevante features. In eerste instantie, was er bij UZ Leuven nog geen voorkennis of er wel bepaalde verbanden afleidbaar waren uit de gemeten testen. Daarnaast waren er heel wat attributen die onvolledig waren of dubbels bevatte. Hierdoor werd gekozen om in eerste instantie alle duplicaten te verwijderen en enkel de eigenschappen te behouden die volledig waren of waarvoor elke patiënt een waarde was geregistreerd. Deze data bevat onder meer de BMI, het aantal kilocalorieën en andere zaken die een invloed hebben op de target. Het doel van deze dataset of de target bestaat erin om de MRC score te bepalen. Deze score kan een waarde aannemen tussen $0 - 60$. Binnen de geneeskunde wordt afgesproken wanneer de MRC score onder de 48 valt, de persoon in kwestie een speciale behandelingen vereist, terwijl bij 60 de persoon alles nog kan uitvoeren of eerder als de normaal beschouwd wordt. Het bepalen van de score zelf is zeer moeilijk, maar er kan wel een binaire classificatie opge-

steld worden, waarin we kunnen stellen dat de waarde 0 overeenstemt met een score < 48 en een 1, wanneer de score ≥ 48.

Bij de eerste resultaten, waar nog geen rekening gehouden werd met missing values noch met ongelabelde data, werd de dataset onderverdeeld in de volgende categoriën:

1. 20% testdata
2. 80% labeled data

Methode	Accuraatheid in %
Supervised met originele representatie zonder Weka preprocessing	72%
Supervised met binaire transformatie zonder Weka preprocessing	97%

Tabel 7.1: Accuracy ICUAW dataset

Op basis van de voorgaande tabel kunnen we afleiden dat er effectief verbanden bestaan tussen de waargenomen attributen en de te bepalen classificatie. Voor UZ Leuven vormt dit een indicatie dat er verder onderzoek moet gebeuren om deze soort van modellen te evalueren om te intraheren binnen de business logica van ziekenhuizen. Daarnaast bieden de genereerde beslissingsbomen binnen Random Forests aan om een model te maken dat interpreteerbaar is voor dokters en als ondersteunend middel kan ingezet worden om verdere beslissingen te maken in de behandeling van de patiënt. Met een interpreteerbaar model bedoelen we dat de testen die genomen worden voor de vertakkingen van de beslissingsboom, overeenstemmen met resultaten van testen bij de patiënt.

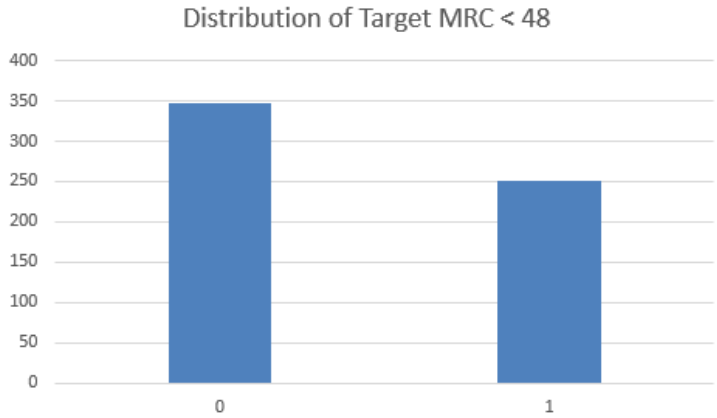
Een ander belangrijk aspect voor het UZ Leuven, is dat voor elk gegeneerde boom de bovenste n-knopen beduidend zijn. Deze knopen vertellen namelijk welke attributen er verantwoordelijk zijn voor de grootste "information gain" of welke attributen een grote impact hebben in de classificatie van het voorbeeld. Daarnaast kunnen we onvolledige data eventueel verder opnemen, maar hierin kan de missing values niet vervangen worden door mean averages, waarmee we de waarde van de attributen bedoelen die niet kunnen geëvalueerd worden door fysieke proeven.

Na de eerste testen was er inderdaad reeds een indicatie van attributen die vaak zichtbaar waren in de top van de bomen. Hieronder vindt u een lijst van deze features.

- BMI 20 uit de 50 beslissingsbomen
- MaxKalium 13 uit de 50 beslissingsbomen
- CorticoidenDripOfBolus 11 uit de 50 beslissingsbomen

Verder bleek dat er attributen opgenomen werden die reeds een indicatie gaven van de target label, met in het bijzonder onder meer "WorstMCRMeting" en "Eerste meting van MRC". Voor deze elementen was verder onderleg nodig met het UZ om te bepalen indien deze elementen al dan niet in rekeningen mochten genomen worden. Een tweede aspect is dat de distributie van de target label ook een belangrijke rol speelt om te mogen

concluderen dat een bepaalde dataset voldoet in het voorspelbaar zijn. Stel dat alle voorbeelden een target hebben van 1, dan is het niet moeilijk om ook een 1 te voorspellen op alle voorbeelden in de testdataset.[8]



Figuur 7.1: Histogram van de target

Uit de histogram 7.1 valt af te leiden dat de dataset voldoet, omdat er zowel voorbeelden tussen zitten met een MRC < 48 en ≥ 48. Voor de tweede test hebben we geopteerd om alle attributen op te nemen. De reden hiervoor is dat Clus overweg kan met deze missing values doordat ze zowel in de linkertak als de rechtertak van de beslissingsboom doorlopen. Omwille van het feit dat er meer attributen opgenomen zijn, wordt verondersteld dat het model zal verslechteren naar accuraatheid. Onderstaande tabel toont de resultaten, waarbij alle attributen opgenomen werden. Bovendien zijn alle testen uitgevoerd met cross-validatie, waarbij elke opdeling van de data in een gelabelde en een test set, tien keer werd uitgevoerd. Het gemiddelde over de accuraatheid heen, werd verder mee in rekening genomen. Wanneer de preproccessing is toegepast, zijn alle numerieke waarden genormaliseerd en zijn alle nominale waarden getransformeerd naar een vector.

Methode	Accuraatheid in %
Supervised met originele representatie zonder Weka preprocessing	65%
Supervised met originele representatie met Weka preprocessing	83%
Supervised met binaire transformatie met Weka preprocessing	97%

Tabel 7.2: Accuracy ICUAW dataset met missing values

Uit tabel 7.2, die alle attributen en missing values bevat, blijkt dat het preprocessen van de data in de originele representatie tot verbetering leidt. Enerzijds was dit te verwachten, maar anderzijds heeft dit geen invloed op de methode met de binaire transformatie. Naast deze trend duidt dit erop dat preprocessing geen vereiste is bij de binaire transformatie.

Een andere test bestaat uit het weglaten van de attributen die iets zeggen over de MRC score. Deze kunnen invloed hebben op de predictie van de target label. Het weglaten kan dus een beeld geven of er voorspelling mogelijk is zonder het resultaat te weten van voorgaande testen over deze score. Het resultaat zal ons informeren of er daadwerkelijk

verbanden zijn tussen de gemeten attributen zonder kennis te hebben van MRC. Ook hier is er sprake van 10 cross-validatie.

Methode	Accuraatheid in %
Supervised met originele representatie met Weka preprocessing	82%
Supervised met binaire transformatie met Weka preprocessing	95%

Tabel 7.3: Accuracy ICUAW dataset zonder MRC attributen

Uit deze resultaten is af te leiden, dat de voorkennis van MRC scores geen vereiste is om een accurate voorspelling te doen. Dit is een positief element van toepassing voor het UZ.

In de volgende testen zal het gelabelde gedeelte terug geschroefd worden, en dit om aan te tonen dat ook hier het opnemen van de ongelabelde data tot een verbetering zal leiden. Om dit effect waar te nemen, nemen we nu ook aan dat de accuraatheid gelijkwaardig achteruit zal gaan en deze zal toenemen naar het resultaat van de vorige tabellen die ongeveer 97% bedraagt (wanneer de niet gelabelde data wordt opgenomen of met SSL te werk wordt gegaan). Verder hebben we terug de dataset genomen zonder voorkennis van MRC. De volgende test is gebaseerd op het opnemen van ongelabelde data in de trainings data.

1. 20% testdata
2. 20% labeled data van de overblijvende 80%
3. 80% niet-gelabelde data van de reeds overblijvende 80%

Methode	Accuraatheid
Supervised met originele representatie met Weka preprocessing	75%
Supervised met binaire transformatie met Weka preprocessing	79%
Methode van Levatić et al.	76%
Self-learning met SVM (80%) en kNN als predictor met originele representatie	74%
Self-learning met SVM (80%) en kNN als predictor met binaire transformatie	76%
Self-learning met SVM (80%) en Levatić et al.	62%

Tabel 7.4: Accuracy ICUAW dataset zonder MRC attributen

Via de testen uit sectie 6.2, blijkt reeds dat via het toevoegen van clusterings data aan de gelabelde data, dit niet bijdraagt in het verbeteren van het model. De verbetering is enkel te zien, wanneer de methode van Levatić et al. [13] of Self-learning wordt gebruikt op de niet gelabelde data. Vooral SVM's zijn goed hierin, maar deze kan niet overweg met missing data, waarbij "features" niet kunnen vervangen worden met averages. Na het zoeken van een oplossing vertelt de handleiding van libSVM dat missing values wel vervangen kunnen worden door 0. De reden hiervoor is dat libSVM deze zaken als sparse ziet en ze zo niet opslaat [14]. Verder blijkt ook dat het gebruiken van self-learning niet direct bijdraagt tot het verbeteren van de accuraatheid. De laatste rij van tabel 7.4 is

een test waarbij SVM gebruikt wordt om de trainingsdata uit te breiden met ongelabelde voorbeelden en waarop uiteindelijk de methode van Levatić wordt toegepast.

7.3 Conclusie

Uit de casus is een model af te leiden dat begrijpbaar en ondersteunend kan zijn voor de dokter. De methode van de transformatie blijkt in de meeste gevallen het best te scoren. Bij de methode van self-learning was er een afname van de accuraatheid merkbaar, waarbij het opnemen van bepaalde ongelabelde data niet bijdraagt in het creëren van een model dat een betere accuraatheid geeft.

Het voordeel van de getransformeerde data in zijn binaire representatie is dat preprocessing niet noodzakelijk is. De reden hiervoor is dat voor elk voorbeeld de gemeten waarde niet mee telt als afstand, maar eerder het pad binnen de beslissingsboom. Anders gezegd zijn de meeteenheden weggewerkt, waardoor ze ook geen invloed meer hebben op de predictor.

Wanneer enerzijds de accuraatheid in beschouwing wordt genomen, dan moeten we rekening houden met welke attributen er nu effectief mogen meespelen in de predictie van de MRC score. Wanneer anderzijds de metingen over de MRC opgenomen worden, dan blijkt dit tot een verbetering van het model te leiden. Voor het UZ Leuven is dit een eerste stap om te bepalen of er specifieke verbanden merkbaar zijn tussen de waargenomen attributen en de voorspelling van de MRC hoger of lager gesitueerd is dan 48. Uit de testen kunnen we afleiden dat verder onderzoek nog nodig is.

Hoofdstuk 8

Conclusie

8.1 Conclusie

In deze scriptie zijn we op zoek gegaan naar een model om voorspellingen te kunnen doen in een semi-supervised omgeving en dat via een empirische studie. Hiermee bedoelen we dat zowel gelabelde en ongelabelde data wordt in rekening gebracht om een predictie te doen. Supervised gaat aan de hand van een aantal voorbeelden waarbij de target gekend is, de dichtste buren opzoeken door de "Manhattan distance" te berekenen of anderzijds een separatie te zoeken in een bepaalde dimensie, om zo bij ongeziene gevallen een waarde te suggereren. Unsupervised probeert aan de hand van gemeenschappelijke eigenschappen van de voorbeelden, groepen te maken, waarbij de intra-afstand tussen de voorbeelden minimaal is, en de inter-afstand van elke cluster maximaal is. Dit fenomeen wordt ook wel clustering genoemd. Specifiek gaat semi-supervised beide zaken gaan combineren, omdat er in de realiteit vaak gevallen zijn waarbij de hoeveelheid voorbeelden bij supervised veel schaarser zijn dan de voorbeelden waarbij de target niet gekend is. De reden hiervoor is vaak dat een bepaalde test in de praktijk kostelijk, of soms gewoon niet toepasselijk is, zoals in de casus van ICU. Intensive Care Unit of kortweg ICU is een onderdeel van de geneeskunde waarin patiënten continue gemonitord worden en waarbij een grote hoeveelheid data gegenereerd wordt. De niet-gelabelde data komt vaak voor bij patiënten die zich in een coma bevinden. Het grootste probleem hier is dat bepaalde testen moeilijk te evalueren zijn ingevolge de toestand van de patiënt. Feedback van de patiënt is echter essentieel in de verdere aanpak. [9]

Het voorgestelde model dat we hebben onderzocht, bestaat uit het opstellen van n-beslissingsbomen door middel van Random Forests voor zowel de labeled data, waarbij de target gekend is, als voor ongelabelde data via clustering. Vervolgens wordt voor elk voorbeeld zijn pad opgeslagen doorheen elke boom, waarbij een knoop een 1 krijgt indien deze doorlopen wordt, en een 0 indien dit niet zo is. De concatenatie van alle paden doorheen elke boom, laat toe een representatie voor te stellen waarbij zowel gelabelde als ongelabelde data opgeslagen wordt. In de volgende stappen passen we een predictor toe zoals in de voorgaande alinea werd aangehaald.

Vervolgens hebben we de voorgestelde methode grondig onderworpen aan een variantie van testcases. De conclusie hieruit is dat de methode een verbetering is ten opzichte van de originele representatie. Dit was positief voor het verdere verloop van het onder-

zoek. Daarnaast werd de methode van Levatić et al. aangewend, als alternatief voor de zelf voorgestelde representatie. Bij de toepassing van Levatić kwam naar voor dat het voorgestelde model moet onderdoen in accuraatheid.

Na verdere analyse blijkt de variant, waarbij enkel rekening gehouden werd met de paden van de bomen in het supervised gedeelte, het beste scoort. In sommige gevallen resulteert dit zelfs beter dan de methode van Levatić, waarbij semi-supervised learning moest onderdoen voor supervised modellen. Naast deze conclusie hebben we ontdekt dat deze representatie geen preprocessing vereist, doordat de eenheden van "features" of eigenschappen wegvallen.

De target in de dataset is de MRC score, waarbij de classificatie bestaat uit of die < 48 of ≥ 48 is. Voor het UZ Leuven was het vooral een eerste test om te onderzoeken of er bepaalde verbanden zijn tussen de waargenomen features en de te bepalen MRC score. Uit de dataset, waarbij rekening gehouden wordt met enkel de features die volledig zijn, behaalde het voorgestelde model een score van 97%. Wanneer we de attributen die spreken over de MRC score, weglaten, dan behaalt het model een score van 79%. Deze resultaten duiden erop dat er een zekere samenhang terug te vinden is, enerzijds tussen de target, en anderzijds tussen de gemeten waarden.

8.2 Adviezen

In verband met de casus blijken de resultaten positief om verder onderzoek te promoten en meer bepaald in het verder verfijnen van het bestaande model, ofwel dat er een totaal ander pad moet gekozen worden. Om de set van attributen te selecteren is verder overleg nodig met het UZ Leuven. Dat was spijtig genoeg niet te evalueren binnen de gegeven tijd, omdat de dataset slechts beschikbaar was vanaf eind april 2018.

Belangrijk om aan te halen is dat het model reeds in proef kan gesteld worden binnen het ziekenhuis zelf. De methode kan een predictie uitvoeren die dan vervolgens in de toekomst kan geverifieerd worden. Dit geeft een verdere indicatie of het model al dan niet geldig is. Voor de patiënt heeft dit geen gevolgen en voor het ziekenhuis kan het een bevestiging zijn dat verder onderzoek noodzakelijk is.

8.3 Verder onderzoek

Er zijn nog een aantal testen die we verder kunnen evalueren en daarmee de conclusie kunnen bevestigen of ontkrachten. Zo zijn er twee testen die we hadden kunnen uitvoeren, maar door tijdsgebrek niet mogelijk waren.

- Self-learning gebruikmakend van de voorgestelde methode, waarmee we bedoelen dat de labeled dataset kan uitgebreid worden met de clusterings informatie in zijn binaire vorm en waarop vervolgens self-learning kan toegepast worden om de trainingsdataset uit te breiden met de ongelabelde voorbeelden.
- Supervised learning kan gebruikt worden om attributen te identificeren die belangrijk zijn ten opzichte van de target. Wanneer deze attributen ontdekt zijn kan clustering aangewend worden op deze subset om zo hoopvol een betere representatie te krijgen.

8.4 Subvragen

Naast de centrale onderzoeksvraag, hebben we verschillende subvragen opgesteld die meer gericht zijn in het verder optimaliseren en eenvoudiger maken van het model, waarbij we tevens getracht hebben om bepaalde verbeteringen te vinden specifiek voor de casus, met name:

1. Welke specifieke technieken bestaan er om predicties te maken, en zijn ze wel toegankelijk?
2. Hoe doeltreffend is een SSL techniek in het classificeren, en welke is de betere in de setting van beslissingsbomen?
3. Zijn er trends afleidbaar uit de data van patiënten die langdurig op ICU verblijven?

Wat de eerste vraag betreft, zijn er verschillende technieken toegepast op de data, waaronder de reeds besproken Random Forests en Support Vector Machines. Voor deze technieken zijn reeds een tal van bibliotheken beschikbaar die deze zaken vrij toegankelijk maken. Zo is de applicatie van Clus eenvoudig te gebruiken bij RF, en laten deze gegenereerde beslissingsbomen toe om eenvoudig de transformatie toe te passen op de te onderzoeken dataset. Bij SVM's hebben we libSVM aangewend, die bij het maken van het model zowel parameteroptimalisatie doet voor de kernel functie als scaling automatisch uitvoert.

De tweede deelvraag gaat meer specifiek in op de toegepaste predictor en hoe invloedrijk het unsupervised gedeelte mag zijn. Beide predictoren vereisen dat duidelijk geëvalueerd moet worden in welke mate ongelabelde data mag meegenomen worden in de trainingsdata. Om dit te garanderen werden twee methodes voorgesteld. De ene stelt dat de maxDepth van de beslissingsbomen gereduceerd moet worden om deze data te beperken, ofwel kan er gekozen worden om het aantal bomen te verminderen. In deze scriptie werd enkel het eerste getest en de tweede dus niet.

De derde deelvraag gaat dieper in op de casus. Een belangrijk onderzoek van deze thesis was het onderzoeken of er een samenhang was tussen de gemonitorde attributen in de

dataset van het Universitair Ziekenhuis in Leuven. Deze vraag zal dus ja beantwoorden wanneer het model effectief een kwaliteitsvolle predictie kan doen op de externe dataset, wat aan de hand van de resultaten dus het geval is. Belangrijk hierbij is dat we evalueren via Random Forests welke eigenschappen het meeste bijdrage tot de voorspelling.

De conclusie bij het onderzoek luidt dus dat semi-supervised beslissingsbomen duidelijk mogelijk zijn. Tijdens het opstellen van deze thesis hebben we reeds de methode van Levatić et al.[13] voorgesteld, die aan de hand van de IG formule, naast de supervised ook rekening houdt met de unsupervised gegevens. Het onderzoek startte zelf met een methodiek om aan de hand van een voorbeeld en zijn bijbehorend pad in elke beslissingsboom (supervised en clustering) een representatie te maken die door verschillende predictors toegepast kunnen worden.

Uit de numerieke testen bleek dat het bijbrengen van ongelabelde data, leidt tot het optimaliseren in welke mate deze data opgenomen kan/mag worden bij de trainingsdata. Het toepassen van de predictors zelf zijn zeer toegankelijk, omdat er reeds een tal van tools beschikbaar zijn die deze zaken eenvoudig bruikbaar maken voor verdere extensies.

De verbetering in vergelijking met de originele methodieken zijn verschillend van dataset tot dataset. Vaak is het voorgekomen dat wanneer een supervised model zeer slecht scoort, er een merkbare verbetering is, maar indien deze reeds goed scoort dit slechts leidt tot een kleine verbetering.

Hoofdstuk 9

Appendix

De eerste klasses beschrijven hoe de waardes ingelezen worden in een array binnen Java om vervolgens kNN op toe te passen, dit voor zowel de originele als de binaire representatie. Voor de volledige code kan de repository geraadpleegd worden: `https://bitbucket.org/RubenGuillemyn/masterthesis_ml_knn/src/master/`

```
// ReadIn.java
/*
 */
package readIn;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 *
 * @author Ruben
 */
public class ReadIn {

    private ArrayList<Object[]> binaryAttrListLabeled = new ArrayList<>();

    public ArrayList<Object[]> getTraining() {
        return binaryAttrListLabeled;
    }

    public Set<Object> getClassification() {
        return classification;
    }
```

```
    private Set<Object> classification = new HashSet<>();

    public ReadIn(String fileNameLabeled) {
        readFile(fileNameLabeled);
    }

    public List<Object[]> readFile(String fileNameLabeled) {
        BufferedReader br = null;
        FileReader fr = null;
        int totNrLAttr = 0;
        try {
            File labeled = new File("toArrayRep/" + fileNameLabeled);
            System.out.println(labeled.getAbsolutePath());
            fr = new FileReader(labeled.getAbsolutePath());
            br = new BufferedReader(fr);
            String sCurrentLine = br.readLine();
            while (!sCurrentLine.toLowerCase().contains("@attribute class")) {
                totNrLAttr++;
                sCurrentLine = br.readLine();
            }
            System.out.println("Total Number of labeled attributes: " +
                (totNrLAttr - 1));

            boolean dataRule = true;
            //Skips empty lines
            while (dataRule) {
                sCurrentLine = br.readLine();

                if (sCurrentLine.toLowerCase().contains("data")) {
                    sCurrentLine = br.readLine();
                    dataRule = false;
                }
            }
            //averages = new HashMap<>();
            while (sCurrentLine != null) {
                Object[] exampleEntries = new Object[totNrLAttr - 1];
                List<String> itemsL = new
                    ArrayList<>(Arrays.asList(sCurrentLine.split(",")));

                for (int i = 0; i < itemsL.size(); i++) {
                    //Calculate numeric values

                    if (!sCurrentLine.contains("?")) {
                        exampleEntries[i] = itemsL.get(i);
                        //Stores the unique values
                        if (i == itemsL.size() - 1) {
                            classification.add(itemsL.get(i));
                            binaryAttrListLabeled.add(exampleEntries);
                        }
                    }
```

```
                }
                sCurrentLine = br.readLine();
            }
        } catch (IOException e) {
            e.printStackTrace();

        } finally {
            try {
                br.close();
                fr.close();

            } catch (Exception exc) {
                exc.printStackTrace();
            }

        }
        System.out.println("Nr of training: " + binaryAttrListLabeled.size());

        //Determining the averages
        return binaryAttrListLabeled;
    }
}
```

```
\\ReadInTransformed.java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package readIn;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 *
 * @author Ruben
 */
public class ReadInTransformed {
    private ArrayList<Object[]> binaryAttrListBoth = new ArrayList<>();
```

```
    private ArrayList<Object[]> binaryAttrList = new ArrayList<>();
    private ArrayList<Object[]> binaryAttrListLu = new ArrayList<>();

    private Set<Object> classification = new HashSet<>();

    private int kNN;

    public ArrayList<Object[]> getTrainingBoth() {
        return binaryAttrListBoth;
    }

    public ArrayList<Object[]> getTainingL() {
        return binaryAttrList;
    }

    public ArrayList<Object[]> getTrainingUnlabeled() {
        return binaryAttrListLu;
    }


    public Set<Object> getClassification() {
        return classification;
    }

    public ReadInTransformed(String fileNameLabeled, String fileNameUnlabeled)
        {
        this.kNN = kNN;
        readRFBIFiles(fileNameLabeled, fileNameUnlabeled);

    }

    public List<Object[]> readRFBIFiles(String fileNameLabeled, String
        fileNameUnlabeled) {

        BufferedReader br = null;
        FileReader fr = null;
        BufferedReader br2 = null;
        FileReader fr2 = null;

        int totNrLAttr = 0;
        int totNrLUAttr = 0;
        try {

            File labeled = new File("toArrayRep/" + fileNameLabeled);
            System.out.println(labeled.getAbsolutePath());
            fr = new FileReader(labeled.getAbsolutePath());
            br = new BufferedReader(fr);
            String sCurrentLine = br.readLine();
            while (!sCurrentLine.toLowerCase().contains("class")) {
                try {
```

```
                totNrLAttr =
                    Integer.parseInt(sCurrentLine.replaceAll("[\\D]", ""));

            } catch (NumberFormatException nExc) {

            }
            sCurrentLine = br.readLine();
        }
        System.out.println("Total Number of labeled attributes: " +
            totNrLAttr);
        File unLabeled = new File("toArrayRep/" + fileNameUnlabeled);
        fr2 = new FileReader(unLabeled.getAbsolutePath());
        br2 = new BufferedReader(fr2);

            //String sCurrentLineUn="";
        String sCurrentLineUn = br2.readLine();
        while (!sCurrentLineUn.toLowerCase().contains("class")) {
            try {
                totNrLUAttr =
                    Integer.parseInt(sCurrentLineUn.replaceAll("[\\D]", ""));

            } catch (NumberFormatException nExc) {

            }
            sCurrentLineUn = br2.readLine();
        }
        System.out.println("Total Number of Unlabeled attributes: " +
            totNrLUAttr);
//
//          //Skips empty lines
        for (int i = 0; i < 3; i++) {
            sCurrentLine = br.readLine();
            sCurrentLineUn = br2.readLine();

        }

        while (sCurrentLine != null && sCurrentLineUn != null) {

            //Removes the "{}"
            sCurrentLine = sCurrentLine.substring(1, sCurrentLine.length()
                - 1);
            sCurrentLineUn = sCurrentLineUn.substring(1,
                sCurrentLineUn.length() - 1);

            Object[] exampleEntries = new Object[totNrLAttr + totNrLUAttr +
                1];
            Object[] exampleEntriesL = new Object[totNrLAttr+1];
            Object[] exampleEntriesLu = new Object[totNrLUAttr+1];
            Arrays.fill(exampleEntries, "0");
            Arrays.fill(exampleEntriesL, "0");
```

```
            Arrays.fill(exampleEntriesLu, "0");

            List<String> itemsL = new
                ArrayList<>(Arrays.asList(sCurrentLine.split(", ")));
            List<String> itemsLU = new
                ArrayList<>(Arrays.asList(sCurrentLineUn.split(", ")));
            for (int i = 0; i < itemsL.size(); i++) {
                int index = Integer.parseInt(itemsL.get(i).split(" ")[0]);
                Object value = itemsL.get(i).split(" ")[1];
                exampleEntriesL[index-1]= value;
                if(i < itemsL.size()-2){
                    exampleEntries[index - 1] = value;
                }

                //Stores the unique values
                if (i == itemsL.size() - 1) {
                    classification.add(value);
                    exampleEntries[exampleEntries.length-1]=value;
                }
            }

            binaryAttrList.add(exampleEntriesL);
            for (int i = 0; i < itemsLU.size()-1; i++) {
                int index = Integer.parseInt(itemsLU.get(i).split(" ")[0]);
                Object value = itemsLU.get(i).split(" ")[1];
                exampleEntries[(index + totNrLAttr) - 1] = value;
                exampleEntriesLu [index-1]=value;
                //Stores the unique values
                if (i == itemsLU.size() - 1) {
                    classification.add(value);
                }
            }
            binaryAttrListLu.add(exampleEntriesLu);
            binaryAttrListBoth.add(exampleEntries);
            sCurrentLine = br.readLine();
            sCurrentLineUn = br2.readLine();

        }

    } catch (IOException e) {
        e.printStackTrace();

    } finally {
        try {
            br.close();
            fr.close();
            br2.close();
            fr2.close();
        } catch (Exception exc) {
            exc.printStackTrace();
```

```
            }

        }
        System.out.println("The nr of examples in trainingdata: " +
            binaryAttrListBoth.size());
        return binaryAttrListBoth;
    }
}
```

```
// BinaryTransformationLabeledUnlabeled.java
package semisupervisedtrees.algorithms;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;


public class BinaryTransformationLabeledUnlabeled {

    private ArrayList<Object[]> binaryAttrList = new ArrayList<>();
    private ArrayList<Object[]> binaryAttrListTest = new ArrayList<>();
    private Set<Object> classification = new HashSet<>();
    private int[][] closestN;

    private int kNN;

    public BinaryTransformationLabeledUnlabeled(ArrayList<Object[]>
        training,ArrayList<Object[]> test, Set<Object> classification, int kNN)
        {
        this.kNN = kNN;
        this.binaryAttrList = training;
        this.binaryAttrListTest = test;
        this.classification=classification;
        findClosestNeighbors();
        HashMap<Integer, Object[]> model = evaluateNeighborsAndTest();
        calculateEffect(model);
    }



    private int[][] findClosestNeighbors() {
        closestN = new int[binaryAttrListTest.size()][kNN];
        for (int i = 0; i < binaryAttrListTest.size(); i++) {
```

```
            Map<Integer, Integer> distances = new HashMap<>();
            for (int j = 0; j < binaryAttrList.size(); j++) {

                int manDist = calculateDist(binaryAttrListTest.get(i),
                    binaryAttrList.get(j));
                distances.put(j, manDist);

            }
            //Sort list by incremented distance
            distances = sortByValue(distances);
            Set<Integer> entries = distances.keySet();
            Iterator<Integer> it = entries.iterator();
            for (int k = 0; k < kNN; k++) {
                closestN[i][k] = it.next();
            }

        }
        return closestN;
    }

    private int calculateDist(Object[] fromEntry, Object[] otherEntry) {
        int distance = 0;
        for (int i = 0; i < fromEntry.length-1; i++) {
            if (!fromEntry[i].equals(otherEntry[i])) {
                distance++;
            }
        }
        return distance;
    }

    //This allows to sort on entries
    public static <K, V extends Comparable<? super V>> Map<K, V>
        sortByValue(Map<K, V> map) {
        return map.entrySet()
                .stream()
                .sorted(Map.Entry.comparingByValue(/*Collections.reverseOrder()*/))
                .collect(Collectors.toMap(
                        Map.Entry::getKey,
                        Map.Entry::getValue,
                        (e1, e2) -> e1,
                        LinkedHashMap::new
                ));
    }

    private HashMap<Integer, Object[]> evaluateNeighborsAndTest() {
        HashMap<Integer, Object[]> evaluate = new HashMap<>();
        for (int i = 0; i < binaryAttrListTest.size(); i++) {
            Object counV = countVotes(i);
            Object[] classEntry = binaryAttrListTest.get(i);
            Object classValue = classEntry[classEntry.length - 1];
```

```
            evaluate.put(i, new Object[]{classValue, counV});
        }
        return evaluate;
    }


    private Object countVotes(int index) {

        HashMap<Object, Integer> cVotes = new HashMap<>();
        for (Object myClassification : classification) {
            cVotes.put(myClassification, 0);
        }
        int[] neighBors = closestN[index];
        //algorithm for majority voting
        for (int i = 0; i < kNN; i++) {
            Object[] vector = binaryAttrList.get(neighBors[i]);
            cVotes.put(vector[vector.length - 1],
                cVotes.get(vector[vector.length - 1]) + 1);

        }
        Map.Entry<Object, Integer> maxEntry = null;
        for (Map.Entry<Object, Integer> entry : cVotes.entrySet()) {
            if (maxEntry == null ||
                entry.getValue().compareTo(maxEntry.getValue()) > 0) {
                maxEntry = entry;
            }
        }

        return maxEntry.getKey();
    }
    /*
    * Calculates the percentage of the correct predictions
    */
    private void calculateEffect(HashMap<Integer, Object[]> model) {
        int nrOFCorrectPredictions = 0;
        for (Map.Entry<Integer, Object[]> entry : model.entrySet()) {
            if (entry.getValue()[0] .equals(entry.getValue()[1])) {
                nrOFCorrectPredictions++;
            }
        }
        int percent = (nrOFCorrectPredictions * 100) / model.size();
        System.out.println("The kNN has an efficiency of:" + percent + " %
            binary or " + nrOFCorrectPredictions+ " of "+
            binaryAttrListTest.size() + " nr. of examples");
    }

}
```

```
// SelfLearning.java
```

```
package semisupervisedtrees.algorithms;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;


public class SelfLearningTree {

    private ArrayList<Object[]> binaryAttrListLabeled = new ArrayList<>();
    private ArrayList<Object[]> binaryAttrListUnlabeled = new ArrayList<>();
    private ArrayList<Object[]> binaryAttrListTest = new ArrayList<>();
    private ArrayList<Object[]> attrListLabeled = new ArrayList<>();
    private ArrayList<Object[]> attrListUnlabeled = new ArrayList<>();
    private ArrayList<Object[]> attrListTest = new ArrayList<>();
    private Set<Object> classification = new HashSet<>();
    private int[][] closestN;

    private int kNN;
    private double threshHold;

    //Original representation

    public SelfLearningTree(ArrayList<Object[]> trainingOriginal,
        ArrayList<Object[]> unLabeled, ArrayList<Object[]> test, Set<Object>
        classification, int kNN, double threshHold) {
        this.kNN = kNN;
        this.threshHold = threshHold;
        this.classification = classification;
        this.attrListLabeled = trainingOriginal;
        this.attrListUnlabeled = unLabeled;
        this.attrListTest = test;
        selfLearningNoTrans(0);
        int[][] closest = findClosestNeighbors();
        HashMap<Integer, Object[]> model = evaluateNeighborsAndTest(closest,
            attrListTest);
        calculateEffect(model);
    }

    private int[][] findClosestNeighbors() {
        closestN = new int[attrListTest.size()][kNN];
        for (int i = 0; i < attrListTest.size(); i++) {
            Map<Integer, Double> distances = new HashMap<>();
            for (int j = 0; j < attrListLabeled.size(); j++) {
```

```
            double manDist = calculateDist(attrListTest.get(i),
                attrListLabeled.get(j));
            distances.put(j, manDist);

        }
        //Sort list by distance
        distances = sortByValue(distances);
        Set<Integer> entries = distances.keySet();
        Iterator<Integer> it = entries.iterator();
        for (int k = 0; k < kNN; k++) {
            closestN[i][k] = it.next();
        }

    }
    return closestN;
}

private double calculateDist(Object[] fromEntry, Object[] otherEntry) {
    double dDistance = 0.0;
    for (int i = 0; i < fromEntry.length - 1; i++) {
        if (!fromEntry[i].equals(otherEntry[i])) {
            try {
                double fromE = Double.parseDouble((String) fromEntry[i]);
                double otherE = Double.parseDouble((String) otherEntry[i]);

                dDistance += Math.abs(fromE - otherE);
            } catch (Exception exc) {
                dDistance += 1;
            }
        }
    }
    return dDistance;
}

//This allows to sort on entries
public static <K, V extends Comparable<? super V>> Map<K, V>
    sortByValue(Map<K, V> map) {
    return map.entrySet()
            .stream()
            .sorted(Map.Entry.comparingByValue(/*Collections.reverseOrder()*/))
            .collect(Collectors.toMap(
                    Map.Entry::getKey,
                    Map.Entry::getValue,
                    (e1, e2) -> e1,
                    LinkedHashMap::new
            ));
}

private HashMap<Integer, Object[]> evaluateNeighborsAndTest(int[][]
    evaluateSet, ArrayList<Object[]> AttrList) {
```

```
        HashMap<Integer, Object[]> evaluate = new HashMap<>();
        for (int i = 0; i < AttrList.size(); i++) {
            Object counV = countVotes(i);
            Object[] classEntry = AttrList.get(i);
            Object classValue = classEntry[classEntry.length - 1];
            evaluate.put(i, new Object[]{classValue, counV});
        }
        return evaluate;
    }

    private Object countVotes(int index) {

        HashMap<Object, Integer> cVotes = new HashMap<>();
        for (Object myClassification : classification) {
            cVotes.put(myClassification, 0);
        }

        int[] neighBors = closestN[index];
        //algorithm for voting
        for (int i = 0; i < kNN; i++) {
            Object[] vector = attrListLabeled.get(neighBors[i]);
            // System.out.println(vector[vector.length-1]);
             if(vector==null || vector[vector.length-1]==null ||
                cVotes.get(vector[vector.length - 1])==null){
                   System.out.println("Null or Target is not known of example");
              }
             else{
                cVotes.put(vector[vector.length - 1],
                    cVotes.get(vector[vector.length - 1]) + 1);
             }


        }
        Map.Entry<Object, Integer> maxEntry = null;
        for (Map.Entry<Object, Integer> entry : cVotes.entrySet()) {
            if (maxEntry == null ||
                entry.getValue().compareTo(maxEntry.getValue()) > 0) {
                maxEntry = entry;
            }
        }

        return maxEntry.getKey();
    }

    private void calculateEffect(HashMap<Integer, Object[]> model) {
        int nrOFCorrectPredictions = 0;
        for (Map.Entry<Integer, Object[]> entry : model.entrySet()) {
            if (entry.getValue()[0].equals(entry.getValue()[1])) {
                nrOFCorrectPredictions++;
            }
```

```
        }

        int percent = (nrOFCorrectPredictions * 100) / model.size();
        System.out.println("The kNN has an efficiency of:" + percent + " % or
            " + nrOFCorrectPredictions + " of " + attrListTest.size() + " nr.
            of examples");
    }

    private void selfLearningNoTrans(int step) {
        closestN = new int[attrListUnlabeled.size()][attrListLabeled.size()];
        for (int i = 0; i < attrListUnlabeled.size(); i++) {
            Map<Integer, Double> distances = new HashMap<>();
            for (int j = 0; j < attrListLabeled.size(); j++) {
                Double manDist = calculateDist(attrListUnlabeled.get(i),
                    attrListLabeled.get(j));
                distances.put(j, manDist);

            }
            //Sort list by distance
            distances = sortByValue(distances);
            Set<Integer> entries = distances.keySet();
            Iterator<Integer> it = entries.iterator();
            for (int k = 0; k < attrListLabeled.size(); k++) {
                closestN[i][k] = it.next();
            }

        }
        boolean changed = false;
        for (int i = 0; i < attrListUnlabeled.size(); i++) {
            HashMap<Object, Integer> cVotes = new HashMap<>();
            for (Object myClassification : classification) {
                cVotes.put(myClassification, 0);
            }

            int[] neighBors = closestN[i];
            //algorithm for voting

            for (int j = 0; j < kNN; j++) {
                Object[] vector = attrListLabeled.get(neighBors[j]);
                // System.out.println(vector[vector.length-1]);
                //Classification is not known by training
                if(vector==null || vector[vector.length-1]==null ||
                    cVotes.get(vector[vector.length - 1])==null){
                    //System.out.println("Null");
                }
                else {
                     cVotes.put(vector[vector.length - 1],
                        cVotes.get(vector[vector.length - 1]) + 1);
                }
```

```
            }

            for (Object entry : cVotes.keySet()) {
                double percent = (cVotes.get(entry) * 100) / kNN;
                if (percent > threshHold) {
                    try {
                        attrListLabeled.add(attrListUnlabeled.get(i));
                        attrListUnlabeled.remove(attrListUnlabeled.get(i));
                        changed = true;
                    } catch (Exception exc) {

                    }
                }
            }
        }
        if (changed) {
            step += 1;
            selfLearningNoTrans(step);
        } else {
            System.out.println("Nr of steps: " + step);
        }

    }


     private void selfLearningTrans(int step) {
        closestN = new
            int[binaryAttrListUnlabeled.size()][binaryAttrListLabeled.size()];
        for (int i = 0; i < binaryAttrListUnlabeled.size(); i++) {
            Map<Integer, Double> distances = new HashMap<>();
            for (int j = 0; j < binaryAttrListLabeled.size(); j++) {
                Double manDist = calculateDist(binaryAttrListUnlabeled.get(i),
                    binaryAttrListLabeled.get(j));
                distances.put(j, manDist);

            }
            //Sort list by distance
            distances = sortByValue(distances);
            Set<Integer> entries = distances.keySet();
            Iterator<Integer> it = entries.iterator();
            for (int k = 0; k < binaryAttrListLabeled.size(); k++) {
                closestN[i][k] = it.next();
            }

        }
        boolean changed = false;
        for (int i = 0; i < binaryAttrListUnlabeled.size(); i++) {
            HashMap<Object, Integer> cVotes = new HashMap<>();
            for (Object myClassification : classification) {
                cVotes.put(myClassification, 0);
```

```
            }

            int[] neighBors = closestN[i];
            //algorithm for voting
            int top = (neighBors.length / 4);
            for (int j = 0; j < top; j++) {
                Object[] vector = binaryAttrListLabeled.get(neighBors[j]);
                // System.out.println(vector[vector.length-1]);
                cVotes.put(vector[vector.length - 1],
                    cVotes.get(vector[vector.length - 1]) + 1);

            }

            for (Object entry : cVotes.keySet()) {
                double percent = (cVotes.get(entry) * 100) / 20;
                if (percent > threshHold) {
                    try {
                        binaryAttrListLabeled.add(binaryAttrListUnlabeled.get(i));
                        binaryAttrListUnlabeled.remove(binaryAttrListUnlabeled.get(i));
                        changed = true;
                    } catch (Exception exc) {

                    }
                }
            }
        }
        if (changed) {
            step += 1;
            selfLearningNoTrans(step);
        } else {
            System.out.println("Nr of steps: " + step);
        }

    }

}
```

Bibliografie

[1] Irfy Bangalore. How many types of cluster analysis and techniques using r. Internet, November 2016. URL: https://analyticsbuddhu.wordpress.com/2016/11/01/types-of-cluster-analysis-and-techniques-using-r/.

[2] Jo Best. Ibm watson: The inside story of how the jeopardy-winning supercomputer was born, and what it wants to do next. Internet, 2013. URL: https://www.techrepublic.com.

[3] Prof. H. Blockeel. *Machine Learning and Inductive Inference*. Acco, 2010.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[5] Nikki Castle. Supervised vs. unsupervised machine learning. Internet, July 2017. URL: https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms.

[6] Chih-Chung Chang Chih-Wei Hsu and Chih-Jen Lin. *A Practical Guide to Support Vector Classication*. Department of Computer Science National Taiwan University, Taipei 106, Taiwan, May 2016.

[7] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[8] Carlos R. González and Yaser S. Abu-Mostafa. Mismatched training and test distributions can outperform matched ones. *Neural Computation*, 27(2):365–387, 2015.

[9] Fabian Guiza. *Predictive data mining in intensive care*. PhD thesis, KUL, 2009.

[10] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 986–996, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[11] Hendrik Blockeel Celine Vens Saso Dzeroski Jan Struyf, Bernard Zenko. *Clus: User's Manual*, September 2011.

[12] Christopher L. Kramer. Intensive care unit–acquired weakness. *Neurologic Clinics*, 35(4):723 – 736, 2017. Neurocritical Care. URL: http://www.sciencedirect.com/science/article/pii/S0733861917300695, doi:https://doi.org/10.1016/j.ncl.2017.06.008.

[13] Jurica Levatić, Michelangelo Ceci, Dragi Kocev, and Sašo Džeroski. Semi-supervised classification trees. *Journal of Intelligent Information Systems*, 49(3):461–486, Dec 2017.

[14] Chih-Jen Lin. Libsvm faq. Internet, January 2018. URL: `https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html`.

[15] David Meyer. *Support Vector Machines The Interface to libsvm in package e1071.* FH Technikum Wien, Austria, February 2017.

[16] K Pliakos and C Vens. Feature induction and network mining with clustering tree ensembles. volume 10312 LNCS, pages 3–18. Springer, 2017. URL: `https://lirias.kuleuven.be/retrieve/439772`.

[17] A. Ultsch and J. Lötsch. *A Data Science Based Standardized Gini Index as a Lorenz Dominance Preserving Measure of the Inequality of Distributions.* Universitätsbibliothek Johann Christian Senckenberg, 2017.

[18] C. Vens and F. Costa. Random forest based feature induction. In *2011 IEEE 11th International Conference on Data Mining*, pages 744–753, Dec 2011. `doi:10.1109/ICDM.2011.121`.

[19] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.

[20] Xiaojin Zhu. Semi-supervised learning, 2006.

DTAI Research Group
Celestijnenlaan 200A bus 2402
3001 HEVERLEE, BELGIË
tel. + 32 16 327 700
fax: + 32 16 327 996
https://wms.cs.kuleuven.be/cs