NARUBAS & PARUBAS

Uitbreidingen van de associatieve classificatietechniek ARUBAS

Kristof Alentijns

promotor :

Prof. dr. Koen VANHOOF

co-promotor :

De heer Benoit DEPAIRE

Eindverhandeling voorgedragen tot het bekomen van de graad
master in de toegepaste economische wetenschappen :
handelsingenieur in de beleidsinformatica

universiteit hasselt

Voorwoord

Deze masterproef vormt het sluitstuk van mijn opleiding tot Handelsingenieur in de Beleidsinformatica aan de Universiteit Hasselt. De totstandkoming ervan zou onmogelijk geweest zijn zonder de steun en hulp van een aantal personen die ik via deze weg graag mijn dank wil betuigen.

Eerst en vooral wil ik mijn co-promotor, de Heer Benoit Depaire, bedanken voor zijn uiterst deskundige begeleiding. Ondanks zijn eigen drukke agenda had hij altijd tijd om feedback te geven, mij in de juiste richting te sturen en samen te zitten om problemen op te lossen. Verder wil ik mijn dank betuigen aan mijn promotor, Prof. dr. Koen Vanhoof, voor zijn deskundig advies bij tegengekomen problemen.

Tevens zou ik mijn ouders willen bedanken voor de morele en financiële steun tijdens mijn gehele opleiding.

Tenslotte wil ik ook mijn vrienden, kotgenoten en medestudenten bedanken waarmee ik, tussen de studiemomenten door, vele fijne tijden heb beleefd.

Mei 2009,

Kristof Alentijns

Samenvatting

Onderzoeksvraag

Deze masterproef behandelt de ontwikkeling en toetsing van twee uitbreidingen op de associatieve classificatietechniek ARUBAS (Depaire, Vanhoof & Wets, 2008). ARUBAS werd ontwikkeld aan het onderzoeksinstituut IMOB door de onderzoeksgroep Data Analyse en Modellering. Het is een associatieve classificatietechniek die de classificatie van ongekende data instanties in twee fasen uitvoert. De eerste fase betreft een transformatie van de featureruimte waarin trainingsinstanties zich bevinden. Deze transformatie wordt uitgevoerd aan de hand van gegenereerde *class assocation rules* (CAR's). Het aantal gegenereerde CAR's kan ingesteld worden. De getransformeerde trainingsinstanties worden in de tweede fase geütiliseerd om nieuwe instanties te classificeren met behulp van instantiegebaseerde classificatie. Hierbij wordt de gelijkenis van alle trainingsinstanties met de nieuwe instantie berekend en op basis hiervan een klasse toegewezen. In deze masterproef worden de prestaties van een aanpassing van het ARUBAS algoritme, namelijk de introductie van een pruningstap in de tweede fase van het algoritme, onderzocht. We onderzoeken of ARUBAS betere classificatieresultaten bekomt, indien slechts met een subset N of fractie P van alle trainingsinstanties vergeleken wordt. N stelt hierbij een vast getal en P een percentage voor. De aangepaste algoritmen worden respectievelijk NARUBAS en PARUBAS benoemd.

Onderzoeksaanpak

In eerste instantie werden tijdens een exploratief onderzoek preliminaire resultaten gegenereerd om een inzicht te verkrijgen in het potentieel van de aangepaste algoritmen. Tevens

werd hierbij gezocht naar een methode om parameters N en P per dataset te optimaliseren. Vervolgens werd een vergelijkende studie tussen NARUBAS, PARUBAS, ARUBAS en bestaande classificatietechnieken uitgevoerd. Vier hypothesen werden hierbij opgesteld en getoetst.

Resultaten

Geen significante prestatieverschillen tussen NARUBAS en PARUBAS werden gevonden. Op basis van andere overwegingen werd echter beslist om verder onderzoek op PARUBAS te focussen.

Bij eenzelfde, random gegenereerde aantal regels aangeleerd, bekwam PARUBAS met geoptimaliseerde P-waarden per dataset (PARUBAS-opt.), alsook PARUBAS met $P = 30\%$ en $P = 50\%$ significant betere resultaten als ARUBAS.

PARUBAS-opt. met een random gegenereerd aantal regels aangeleerd presteerde significant beter als classificatietechniek 1R, maar significant minder goed als C4.5.

PARUBAS-opt., met het optimale aantal regels per dataset dat gevonden werd voor ARUBAS, bekwam een hogere gemiddelde rangschikking als ARUBAS. Dit verschil was echter niet significant op 10%. In een vergelijking met geprunede en niet geprunede versies van bestaande classificatietechnieken C4.5, CBA, RIPPER en 1R, behaalde PARUBAS-opt. de vierde plaats, net voor ARUBAS en na C4.5 *pruned*, C4.5 *unpruned* en RIPPER *pruned*. C4.5 *pruned*, C4.5 *unpruned* en RIPPER *pruned* en PARUBAS-opt. presteerden significant beter als 1R. Onderling verschilden ze echter niet significant op 10%.

Conclusies

PARUBAS behaalde in elk van de uitgevoerde experimenten een hogere gemiddelde rangschikking als ARUBAS. Dit verschil was echter niet steeds significant op 10%. Op basis hiervan kan een voorkeur voor PARUBAS worden geuit, maar geen sterke aanbeveling. Verder onderzoek kan zich focussen op de gezamenlijke optimalisatie per dataset van zowel het aantal aan-

geleerde regels als de P-waarde, wat hoogstwaarschijnlijk de prestaties van PARUBAS-opt. verder zou verbeteren.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Inleiding

1.1 Probleemstelling

Tegenwoordig wordt overal rondom ons data vergaard. Tijdens het afrekenen aan de kassa in de supermarkt wordt informatie over onze aankopen opgeslagen. Telkens we een snelheidsovertreding begaan, komt dit in een database bij de politie terecht. Wanneer we online een aankoop begaan moeten we vaak eerst een registratieproces doorlopen waarbij onze gegevens worden opgeslagen en waaraan vervolgens informatie omtrent aankoopgedrag gelinkt kan worden.

Deze ontwikkelingen hebben ertoe geleid dat bedrijven, overheden en onderzoekers met een overvloed aan data te kampen hebben. Dit luxeprobleem maakt dat het veel moeilijker en tijdrovender wordt om deze grote hoeveelheden data manueel te verwerken. Een gevolg hiervan is dat veel meer gegevens opgeslagen worden dan daadwerkelijk verwerkt worden tot betekenisvolle kennis. Intelligent geanalyseerde en verwerkte data kunnen nochtans leiden tot nieuwe inzichten in bepaalde problemen en situaties. Zo kan een onderneming bijvoorbeeld sterk gebaat zijn bij het verkrijgen van nieuwe inzichten in zijn bedrijfsprocessen en hier eventueel een competitief voordeel uit halen. Het is hierin dat het onderzoeksdomein van *data mining* op de voorgrond komt.

Binnen het domein van *data mining* wordt getracht, met behulp van algoritmes, bruikbare

kennis uit grote hoeveelheden data te halen. De laatste decennia zijn tal van methoden, zogenaamde classificatietechnieken, ontwikkeld om objecten (personen, situaties, ...) te identificeren volgens een op voorhand gedefinieerde set van categorieën. Bij een ander type data mining technieken, associatieregel-algoritmen, wordt gezocht naar alle interessante relaties (associaties) die in een database aanwezig zijn, waarbij in tegenstelling tot bij classificatietechnieken geen op voorhand gedefinieerde set van categorieën bepaald is. Deze relaties, uitgedrukt als associatieregels, vormen de output van associatieregel-algoritmen.

Recent zijn classificatietechnieken ontwikkeld op basis van deze associatieregels, i.e. associatieve classificatie (AC). Associatieve classificatietechnieken integreren associatieregel-algoritmen en traditionele classificatietechnieken. Aangepaste associatieregel-algoritmen gaan hierbij op zoek naar specifieke associatieregels waarmee ongekende data instanties geclassificeerd worden.

Binnen IMOB is recent een nieuwe associatieve classificatietechniek ontwikkeld door de onderzoeksgroep Data Analyse en Modellering, genaamd ARUBAS. Deze techniek omvat een geheel nieuwe aanpak om associatieregels te gebruiken voor classificatiedoeleinden.

Ze bestaat uit twee stappen. De eerste stap betreft een transformatie van de feature-ruimte waarin de data instanties zich bevinden. Op basis van gevonden associatieregels worden instanties getransformeerd van de attribuutruimte naar een nieuwe, krachtigere patroonruimte. In een tweede stap wordt in deze nieuwe feature-ruimte een ongeclassificeerde case vergeleken met alle getransformeerde trainingscases. De ongeclassificeerde case wordt vervolgens toegewezen aan de klasse waarmee hij gemiddeld genomen de grootste gelijkenis vertoont. Deze gelijkenis met een klasse wordt berekend door de gemiddelde gelijkheid met alle cases van die klasse te nemen (Depaire, Vanhoof & Wets, 2008).

De huidige onderzoeksresultaten omtrent de prestaties van ARUBAS zijn reeds veelbelovend. In deze thesis willen we echter nagaan of een kleine aanpassing aan het algoritme tot nog betere prestaties leidt. Zo zullen we onderzoeken of het vergelijken met slechts een subset van N cases of een fractie P van alle cases per klasse tijdens de classificatiefase een betere accuraatheid oplevert.

1.2 Onderzoeksopzet

Het eerste deel van deze thesis beslaat een literatuurstudie aangaande data mining, associatieregeltechnieken en meer specifiek associatieve classificatietechnieken. Deze literatuurstudie heeft als doel de lezer meer inzicht te verschaffen in het specifieke domein waarin het ARUBAS raamwerk gesitueerd is. Zo introduceert Hoofdstuk 2 de lezer tot het domein van data mining en bevat een bondige bespreking van essentiële begrippen zoals associatieregels en *class association rules*. Hoofdstuk 3 focust op associatieve classificatietechnieken, waarvan het achterliggend idee grondig behandeld wordt. De belangrijkste bestaande associatieve classificatietechnieken worden hier ook kort besproken en vergeleken. Hoofdstuk 4 vervolgens, omvat een grondige bespreking aangaande het huidige ARUBAS raamwerk.

Hoofdstuk 5 behandelt de opbouw van het praktijkonderzoek. Vooreerst wordt de statistische methodologie, die in deze thesis gevolgd wordt om de prestaties van verscheidene data mining algoritmes onderling te vergelijken, uitgelijnd. Hierop volgt een bespreking van het uitgevoerde exploratief onderzoek. Hierbij werden preliminaire resultaten gegenereerd met als objectief inzicht te verkrijgen in de prestaties van het aangepaste algoritme bij diverse parameterinstellingen. In een volgend onderdeel formuleren we de onderzochte hypothesen en wordt uitgelijnd hoe de hypothesetesting werd aangepakt. De bekomen resultaten worden tevens besproken. Het hoofdstuk wordt afgesloten met de formulering van de conclusies gevormd uit het praktijkonderzoek.

Hoofdstuk 2

Data Mining in het KDD-proces - Associatieregels

Als gevolg van de huidige trend naar digitalisering en de beschikbaarheid van goedkope computerhardware, worden over een grote verscheidenheid van onderzoeksdomeinen immense hoeveelheden aan data gegenereerd en opgeslagen. Dit fenomeen heeft, over al deze domeinen, een grote interesse naar methoden en tools om bruikbare kennis uit deze gegevens te halen veroorzaakt. Het domein van data mining levert de tools en technieken die gebruikt kunnen worden om zulke bruikbare kennis te extraheren uit grote hoeveelheden gegevens. Data mining wordt toegepast in een resem wetenschappelijke domeinen zoals astronomie, geologie enzovoort. Ook in de bedrijfseconomische context kent data mining vele toepassingen. Marketing, financiën, fraude-opsporing, telecommunicatie en e-business vormen hierbij de voornaamste applicatiegebieden (Fayyad, Piatetsky-Shapiro & Smyth, 1996).

We duiden het potentieel van data mining met behulp van een veelgebruikt voorbeeld ontleend uit Witten & Frank (2005). Veronderstel dat een onderneming, operationeel in een zeer competitieve markt, problemen heeft met klantloyaliteit. De oplossing voor deze problemen, zij het in de vorm van ruwe data, bevindt zich in een beschikbare database met gegevens omtrent aankopen van klanten samen met klantenprofielen. Patronen in het gedrag van klanten kunnen met behulp van data mining geanalyseerd worden. Hiermee wordt getracht te identificeren welke karakteristieken loyale en minder loyale klanten bezitten. Deze informatie

kan dan gebruikt worden om huidige klanten te identificeren die hoogstwaarschijnlijk minder loyaliteit zullen vertonen. De onderneming kan zich vervolgens specifiek richten op deze potentieel minder loyale klanten met aangepaste marketingtools en speciale behandelingen, die te kostelijk zouden zijn om op zijn volledige klantenbestand toe te passen. Data mining technieken kunnen tevens toegepast worden op dezelfde database om klanten te identificeren die mogelijk geïnteresseerd zouden zijn in andere diensten/producten die de onderneming aanbiedt. Specifieke marketingacties, gericht op een beperkte groep klanten, kunnen vervolgens ondernomen worden.

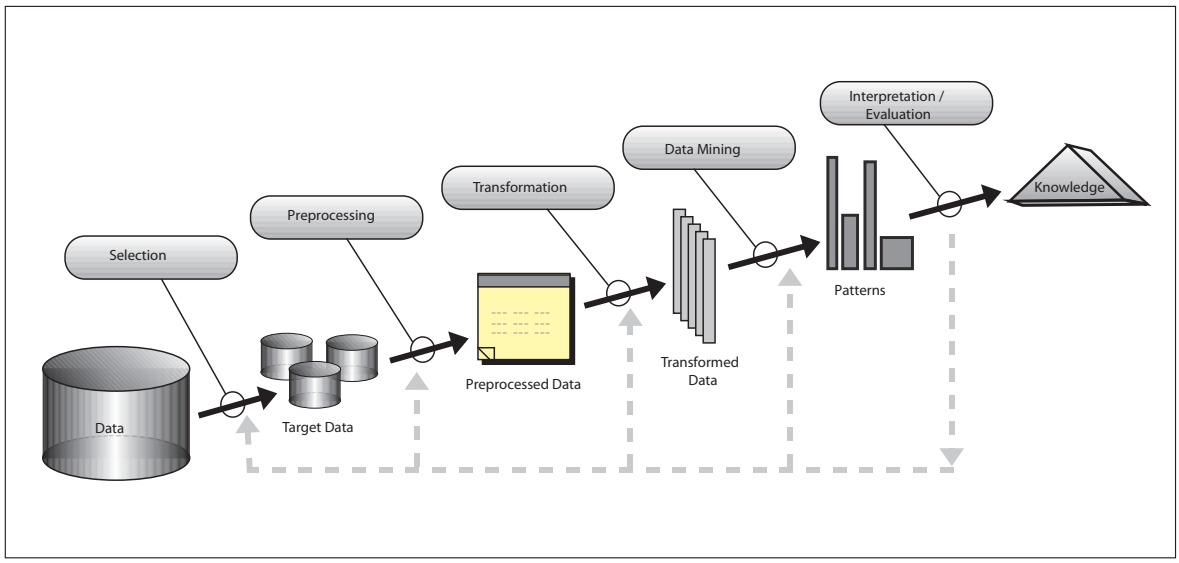
Witten & Frank (2005) definiëren data mining als een automatisch of semi-automatisch proces dat tracht patronen te ontdekken in (grote hoeveelheden) data. Met behulp van algoritmen wordt getracht bruikbare kennis uit deze data te halen.

Data mining maakt deel uit van het breder onderzoeksgebied van *knowledge discovery in databases* (KDD). Dit hoofdstuk vat aan met een korte bespreking van de functie van data mining in KDD. Een verdere introductie tot het domein van data mining volgt, waarna begrippen zoals associatieregels en *class association rules* worden verduidelijkt, waarvan het begrip essentiëel is om de verder besproken materie in deze thesis te vatten.

2.1 Het KDD-proces

Fayyad *et al.* (1996) plaatsen data mining in het bredere *knowledge discovery in databases* (KDD) proces, uitgebeeld in Figuur 2.1. Deze figuur toont het volledige KDD-proces waarvan data mining één van de stappen, die doorlopen worden bij het extraheren van bruikbare informatie uit data, uitmaakt. De additionele stappen in dit proces dragen allen bij tot het verkrijgen van zinvolle en juiste informatie uit de gegevens. Zonder het juist prepareren, selecteren en opkuisen van de ruwe data in combinatie met de aanwending van relevante expertise, leidt het toepassen van een data mining algoritme mogelijk tot de ontdekking van onbruikbare en/of onjuiste patronen.

De eerste stap in het KDD-proces behelst het samenstellen van de dataset die gehanteerd



Figuur 2.1: Het KDD proces (Fayyad *et al.*, 1996)

wordt voor *knowledge discovery*. Deze stap omvat het selecteren van de dataset of een subset van gegevens uit een dataset. De volgende stap betreft preparatie en *cleaning* van de data. Dit omvat het omgaan met onvolledige gegevens, inconsistenties, dubbele gegevens en redundante variabelen. De data worden tevens in een standaard formaat gezet, wat nodig is omdat gegevens vaak uit verscheidene bronnen voortkomen. Tijdens de transformatiestap worden de data getransformeerd en eventueel gereduceerd zodat ze de juiste structuur hebben om geanalyseerd te kunnen worden door het geselecteerde data mining algoritme. Continue en nominale gegevens worden bijvoorbeeld vaak omgezet in discrete gegevens omdat vele data mining algoritmes enkel discrete gegevens kunnen verwerken (Fayyad *et al.*, 1996).

Hierna volgt de data mining stap. De dataset verkregen uit voorgaande stappen wordt geanalyseerd, waarbij gezocht wordt naar patronen. De gevonden patronen moeten praktisch toepasbaar (bruikbaar) zijn en zo mogelijk tot een (economisch) voordeel leiden. Bruikbare patronen laten toe betekenisvolle voorspellingen te maken op basis van nieuwe data. De uitdrukking van patronen gebeurt meestal op een structurele wijze. Structurele patronen worden gedefinieerd als patronen die een achterliggende structuur — welke bestudeerd en besproken kan worden — bezitten die zou verklaren hoe deze patronen werden verkregen. Dit soort

patronen draagt bij tot het begrijpen van de data (Witten & Frank, 1999). In de volgende sectie zal met behulp van een voorbeeld worden verduidelijkt wat juist bedoeld wordt met een structureel patroon. Tevens zullen enkele elementaire begrippen met betrekking tot databases en data mining worden omschreven.

De laatste stap in het KDD-proces omvat de evaluatie en interpretatie van de resultaten. De analyse van een database met verkeersgegevens aan de hand van data mining technieken kan zo bijvoorbeeld uitwijzen dat alleenstaande personen vaker betrokken zijn bij dodelijke ongevallen. De uitkomst van deze analyse geeft echter geen verklaring voor dit gevonden patroon. Geïdentificeerde patronen moeten zodus geïnterpreteerd en geëvalueerd worden door een domeinexpert om betekenisvolle kennis te bekomen.

2.2 Structurele Patronen

In Tabel 2.1 wordt voor verschillende waarden van weergegevens aangegeven of een niet nader gedefinieerd spel zal gespeeld worden. Elke regel in deze tabel vertegenwoordigt een record (instantie). Zulke instantie bestaat uit waarden voor verscheidene attributen. De weergegevens in Tabel 2.1 worden bepaald door volgende attributen, met de mogelijke attribuutwaarden tussen haakjes: ***outlook*** *(sunny, overcast, rainy),* ***temperature*** *(hot, mild, cool),* ***humidity*** *(high, normal) en* ***windy*** *(true, false).* Tenslotte wordt ook de klasse ***play*** (met mogelijke waarden: *yes, no*) gegeven, die de uitkomst van de verschillende attribuutcombinaties weergeeft. Het is duidelijk dat niet alle mogelijke combinaties van attribuutwaarden voorkomen in Tabel 2.1. Dit zal bij echte databases ook meestal het geval zijn en het is net deze onvolledigheid die een uitdaging vormt bij het opstellen van regels om data te beschrijven. Tegenstrijdige en foutieve data vormen een tweede moeilijkheid voor data mining technieken om correcte regels op te stellen die de data zo goed mogelijk beschrijven. Zo is het zeer goed mogelijk dat een databank instanties met eenzelfde attribuutwaarden, maar toch verschillende klasselabels bevat. Dit kan het gevolg zijn van fouten die in de database geslopen zijn, evenwel is het soms ook mogelijk dat bepaalde attribuutcombinaties verscheidene klasselabels kunnen bezitten. Hierdoor worden zelfs op de trainingsdata zelf vaak foutieve classificaties gemaakt.

Hieronder worden enkele classificatieregels weergegeven die uit Tabel 2.1 kunnen gehaald worden (Witten & Frank, 1999):

```
if outlook = rainy and windy = false then play = yes
if outlook = overcast then play = yes
```

Deze regels stellen structurele patronen voor. Een deel van de data worden in feite gesynthetiseerd met behulp van deze twee regels, namelijk alle instanties die voldoen aan het antecedent van één van deze regels. Structurele patronen worden niet altijd in de vorm van regels uitgedrukt. Beslissingbomen drukken patronen ook op een structurele wijze uit, door een opeenvolging van beslissingen, die in een bepaalde uitkomst/aanbeveling resulteren, grafisch voor te stellen. Beslissingsbomen worden later kort besproken.

Tabel 2.1: The weather data (Witten & Frank, 1999) .

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

2.3 Machine Learning

De technieken gebruikt in data mining worden vaak omschreven als *machine learning*. Het domein van *machine learning* bestudeert het ontwerp en de ontwikkeling van algoritmen die verbeteren door *experience*. Dit *learning* proces heeft als objectief, het verbeteren van de prestaties van het algoritme op nieuwe data.

In data mining toepassingen worden twee vormen van *machine learning* onderscheiden: *supervised* en *unsupervised learning*. Een min of meer gelijklopende opdeling kan gemaakt worden op basis van het data mining objectief. Enerzijds kan dit objectief voorspellen, anderzijds beschrijven betreffen (Fayyad *et al.*, 1996). Bij *supervised learning* leert het *learning* algoritme regels aan op basis van een trainingsset, waarbij voor elke individuele instantie in de trainingsset de klasse waartoe deze instantie behoort, geweten is. Met de regels die hieruit voortkomen wordt vervolgens getracht te voorspellen tot welke klasse een ongekende instantie behoort. *Supervised learning* tracht dus te voorspellen. Indien bijvoorbeeld een *learning* algoritme wordt toegepast op Tabel 2.1 en dit algoritme regels opstelt voor het voorspellen van klasse *play*, met behulp van de gegeven waarden van *play* bij de attribuutcombinaties hierboven, spreken we van *supervised learning*. Hierbij zullen classificatieregels gegenereerd worden van de vorm hierboven weergeven. Bij *unsupervised learning* zoekt het *learning* algoritme naar patronen/associaties in de data waarbij geen klasselabels gegeven zijn voor de instanties in de database. De data worden beschreven aan de hand van een structuur die erin gezocht wordt. Het data mining objectief betreft hierbij aldus beschrijving van de data. Passen we nu een techniek van *unsupervised learning*, zoals een associatieregelalgoritme, toe op Tabel 2.1 verkrijgen we mogelijk regels van de vorm:

```
if temperature = cool then humidity = normal
if outlook = sunny and temperature is hot then humidity = high and play = no
```

Beslissingsboom-algoritmen, classificatieregeltechnieken en instantiegebaseerde classificatietechnieken zijn voorbeelden van *supervised learning*. Zij wijzen instanties toe aan verschillende klassen. De toepassing van het simpelste classificatieregelalgoritme —1R— op Tabel 2.1 levert volgende regels op:

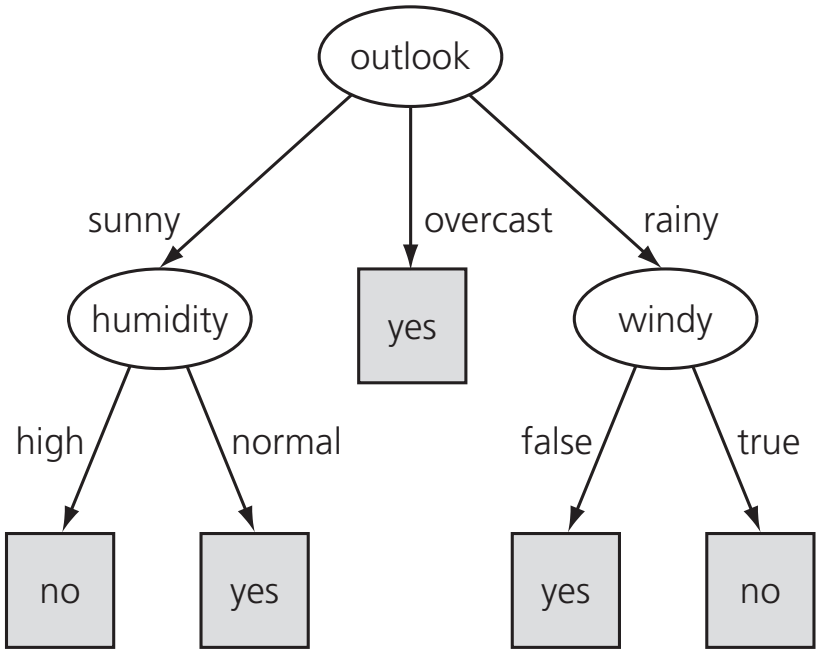
```
if outlook = sunny then play = no
if outlook = overcast then play = yes
if outlook = rainy then play = yes
```

Deze twee sets met regels tonen duidelijk het verschil tussen *supervised* en *unsupervised learning*. In de eerste set kan elk attribuut in het antecedent of het consequent van de regel voorkomen. Het consequent kan tevens meerdere attributen bevatten. Er wordt dus gezocht naar alle mogelijke relaties tussen alle attributen. Bij de classificatieregels daarentegen bevat het consequent telkens slechts één enkel attribuut, namelijk het klasse-attribuut (*play* in dit geval). Relaties tussen alle attributen en het klasse-attribuut worden hierbij gezocht.

De meeste classificatiemethoden gebruiken gegeven data om een classificatiemodel (*classifier*) te construeren waarna dit model gebruikt wordt om nieuwe data te classificeren. Instantiegebaseerde classificatie echter stelt dit verwerkingsproces uit tot wanneer een nieuwe instantie geclassificeerd wordt. Nieuwe instanties worden vergeleken met gekende trainingsinstanties, hun gelijkheid berekend met behulp van een afstandsmaat en op basis hiervan een klasse toegewezen. Het bekendste instantiegebaseerde classificatiealgoritme is het *K-nearest-neighbour* algoritme. De te classificeren testinstantie wordt hierbij vergeleken met de K meest gelijkaardige trainingsinstanties en de classificatie hierop gebaseerd (Han & Kamber, 2006).

Beslissingsboomalgoritmes delen de data op aan de hand van de attribuutwaarden die de instanties bezitten. Bovenaan de beslissingboom (aan de zogenaamde *root node*) worden de data opgesplitst op basis van één attribuut, geselecteerd op basis van een bepaald criterium, waarbij voor elke mogelijke attribuutwaarde een afsplitsing gemaakt wordt. Groepen data (voorgesteld in de beslissingsboom als *nodes*) worden zo bekomen die vervolgens weer op éénzelfde wijze worden opgesplitst. Dit herhaalt zich voor elke *node* tot elke gecreëerde groep instanties (*node*) dezelfde klassewaarde bezit of tot opsplitsing niet meer mogelijk is. Classificatieregels kunnen makkelijk worden afgeleid uit beslissingsbomen (Han & Kamber, 2006). Figuur 2.2 toont een mogelijke beslissingsboom voor de weerdata uit Tabel 2.1. Bij de *root node* wordt hier gesplitst op attribuut *outlook*. De grijze *nodes* zijn de zogenaamde *leaf nodes*, die de waarde voor het klasse-attribuut (*play*) aangeven. De pijlen (*branches*) geven

de opsplitsingen per attribuutwaarde weer.



Figuur 2.2: Beslissingsboom voor de weerdata uit Tabel 2.1 (Witten & Frank, 2005).

Clustering is een voorbeeld van *unsupervised learning* en wordt toegepast wanneer geen klasse voorhanden is om te voorspellen, maar instanties in groepen (clusters) ingedeeld moeten worden. Clusters met gelijkaardige instanties worden gevormd, waarbij getracht wordt een zo sterk mogelijke gelijkenis binnen de clusters te verkrijgen en een zo laag mogelijke gelijkenis tussen de verschillende clusters (Witten & Frank, 2005). Associatieregelalgoritmes, welke later besproken worden, zijn een tweede voorbeeld van *unsupervised learning*. ARUBAS, een associatieve classificatietechniek, behoort tot het domein van *supervised learning*.

2.4 Associatieregels (AR)

Gegeven is een database D met m attributen $A_1, A_2, ..., A_m$ waarbij $a_1, a_2, ..., a_m$ de bijhorende attribuutwaarden vertegenwoordigen. Deze attributen kunnen categorisch of continu zijn. In geval van continue attributen moeten deze eerst gediscretiseerd worden. $\mid D \mid$ stelt het aantal instanties in D voor. Associatieregels beschrijven relaties tussen attributen en

zijn van de vorm: $X \Rightarrow Y(conf)$, waarbij X het antecedent en Y het consequent wordt genoemd. X en Y stellen combinaties van attribuut–waarde paren voor, waarbij $X \cap Y = \emptyset$ en $Y \neq \emptyset$. Een attribuut-waarde paar wordt ook een *item* of *literal* genoemd en voorgesteld door: $((A_i, a_i))$. Combinaties van attribuut-waarde paren worden itemsets genoemd en voorgesteld door: $((A_{i1}, a_{i1}), ..., (A_{ik}, a_{ik}))$. Y volgt uit de voldoening aan X, $conf$ is de betrouwbaarheidsfactor (*confidence*). Een instantie i uit database D voldoet aan X indien in i voldaan wordt aan de attribuutwaarden in X. We verduidelijken dit aan de hand van een voorbeeld. Tabel 2.2 toont een fictief (zeer beperkt) voorbeeld van transactiedata die mogelijk uit de analyse van het aankoopgedrag in een supermarkt voortkomen. De attributen hier zijn de volgende: bier, chips, brood, boter met elk als waarde 0 indien de transactie het attribuut niet bevat, 1 als dit wel zo is. Hieruit kan volgende associatieregel gehaald worden: $bier \Rightarrow chips(67\%)$, waarmee aangegeven wordt dat twee van de drie personen die bier aankopen ook chips zullen aankopen. Dit kan ook als volgt voorgesteld worden: $(bier, 1) \Rightarrow (chips, 1)(67\%)$. Een instantie (transactie in dit geval) voldoet aan het precedent van deze regel indien ze het attribuut bier bevat, wat bij transactie 1, 3 en 4 het geval is.

Tabel 2.2: Vereenvoudigde supermarkt transactiedata

Transactie	bier	chips	brood	boter
1	1	1	0	0
2	0	1	1	0
3	1	0	0	1
4	1	1	1	0
5	0	1	0	0

Associatieregeltechnieken hebben als doel interessante regels/associaties te ontdekken in een gegeven dataset. Verscheidene statistieken zijn voorhanden om de interessantheid van een associatieregel uit te drukken. De meest gebruikte hierbij zijn *support* en *confidence*. De *support* van een associatieregel $s(X \Rightarrow Y)$ kan worden gedefinieerd als het aantal instanties in de database die voldoen aan de regel — de *support count* — gedeeld door het totale aantal

items aanwezig in de database. De *support* van een regel $X \Rightarrow Y$ stelt de fractie voor van het aantal instanties in database D die zowel X als Y bevatten. Zodus,

$$support = P(X \cap Y) = \frac{\text{aantal instanties die zowel } X \text{ als } Y \text{ bevatten}}{\text{totaal aantal transacties}}.$$

Passen we dit toe op *bier* $\Rightarrow$ *chips* en de data in Tabel 2.2 verkrijgen we:

$$support = \frac{2}{5} = 40\%.$$

De *confidence*, $conf(X \Rightarrow Y)$ van associatieregel $X \Rightarrow Y$ geeft het percentage van instanties in D met antecedent X weer die ook aan het consequent Y voldoen. We definiëren zo,

$$confidence = P(Y \mid X) = \frac{P(X \cap Y)}{P(X)} = \frac{\text{aantal instanties die zowel } X \text{ als } Y \text{ bevatten}}{\text{aantal instanties die } X \text{ bevatten}}.$$

Toegepast op *bier* $\Rightarrow$ *chips*:

$$confidence = P(Y \mid X) = \frac{2}{3} = 67\%$$

waarbij de *confidence* de accuuraatheid van de regel weergeeft.

Kijken we terug naar de gegevens in Tabel 2.1 dan heeft regel $humidity = normal \Rightarrow play = yes$ in deze dataset een *support* van 6/14 en een *confidence* van 6/7.

2.4.1 Class association rules

Indien we associatieregels willen gebruiken voor classificatiedoeleinden moeten we hun vorm wat aanpassen. Bij een gewone associatieregel $X \Rightarrow Y$ stelt consequent Y een itemset voor. Elk item dat niet in het antecedent voorkomt mag in het consequent voorkomen. Wanneer we associatieregels voor classificatiedoeleinden willen gebruiken hebben we regels nodig die klassen kunnen voorspellen. Hiervoor moeten we Y beperken tot slechts één item, waarbij het attribuut van dit attribuut-waarde paar het klasseattribuut is.

Zo verkrijgen we een *class association rule* (CAR) van de vorm $X \Rightarrow C$ waarbij C een bepaalde klasse voorstelt en X een itemset van de vorm $((A_{i1}, a_{i1}), ..., (A_{ik}, a_{ik}))$. De definities van *support* en *confidence* van CAR's zijn dezelfde als die bij gewone associatieregels.

Hoofdstuk 3

Classificatie met behulp van Associatieregels

Dit hoofdstuk bevat een vergelijkende literatuurstudie aangaande bestaande associatieve classificatietechnieken. Associatieve classificatietechnieken integreren twee data mining technieken, namelijk classificatieregeltechnieken en associatieregeltechnieken. De CAR's, in Hoofdstuk 2 behandeld, liggen aan de basis van deze integratie. Door te focussen op associatieregels die één enkele klasse bepalen, kunnen (aangepaste) associatieregeltechnieken gebruikt worden voor classificatiedoeleinden.

Een groot voordeel geassocieerd met het gebruik van associatieregels voor classificatie betreft hun theoretische mogelijkheid om alle interessante relaties in een database te ontdekken. Het aantal regels dat gevonden wordt in sommige datasets kan echter zeer hoog oplopen waardoor het belangrijk is dat een goede *pruning* strategie gehanteerd wordt teneinde een accurate en compacte set classificatieregels te bekomen. Hoe compacter de set regels hoe makkelijker ze te interpreteren is door de gebruiker. *Pruning* is dan ook een belangrijke stap bij associatieve classificatie (AC).

We vatten dit hoofdstuk aan met een opdeling in stappen van het AC-proces. Vervolgens bespreken we de uitvoering van deze stappen bij verscheidene belangrijke AC-techieken. Het hoofdstuk wordt afgesloten met een kort besluit dat we uit deze vergelijkende literatuurstudie

opmaken.

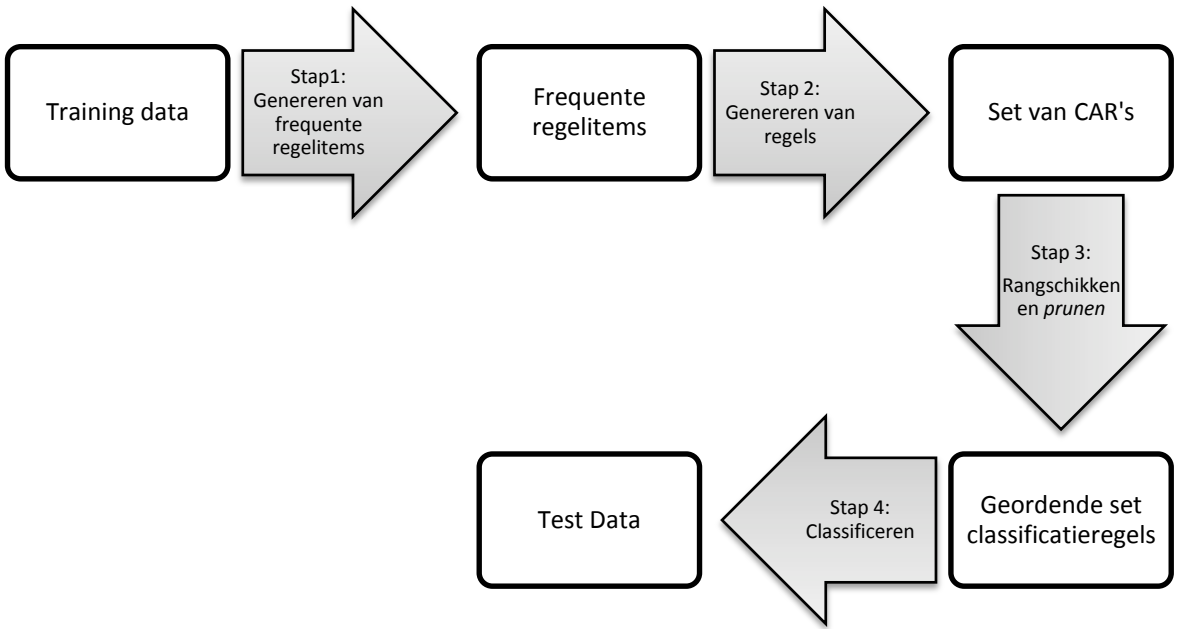
3.1 Associatieve classificatie, een opdeling in stappen

Thabtah (2007) definieert vier veralgemeende stappen, geïllustreerd in Figuur 3.1, die doorlopen worden bij de classificatie op basis van associatieregels.

- Stap 1: Het zoeken naar alle frequente regelitems
- Stap 2: De generatie van alle CAR's die voldoen aan een vooropgestelde minimum *confidence* uit de frequente regelitems in Stap 1 geproduceerd.
- Stap 3: De selectie van een subset van CAR's waarop de classificatie zal gebaseerd zijn.
- Stap 4: Classificatie van testdata met de bekomen classificatieregels en meting van de prestaties.

Een regelitem wordt voorgesteld door $\langle itemset, c\rangle$ waarbij $c \in C$ de klassenwaarde voorstelt. De eerste stap komt dan ook overeen met het ontdekken van frequente itemsets bij associatieregelalgoritmen. Het is computationeel de meest intensieve stap van het gehele proces. De meeste methoden om frequente regelitems te genereren zonderen alle mogelijk frequente regelitems af, waarna wordt geteld hoe vaak ze voorkomen in de trainingsdata. Hierna wordt voor elk frequent regelitem dat voldoet aan een bepaalde minimum *confidence* één enkele regel van de vorm $X \Rightarrow C$ opgesteld. Hierbij stelt C de klasse voor die het vaakst bij itemset X voorkomt in de trainingsdata (Stap 2). Verscheidene AC-methoden hanteren een verschillende aanpak om in Stap 3 de gevonden CAR's te gebruiken voor de classificatie van instanties. In Stap 4 wordt de ontwikkelde AC-techniek toegepast op een onafhankelijke test dataset om vervolgens de behaalde prestaties te meten (Thabtah, 2007).

Een simpel voorbeeld, overgenomen en aangepast uit Thabtah (2007), verduidelijkt de eerste drie stappen van het associatieve classificatieproces. Tabel 3.1 toont een simpele training dataset met drie attributen, A_1, A_2 en A_3 met respectievelijke mogelijke waarden (a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3) en twee klassen (y_1, y_2). Minimum *support* (*minsupp*) wordt bepaald



Figuur 3.1: Veralgemeende stappen bij associatieve classificatie (Thabtah, 2007).

op 20% en minimum *confidence* (*minconf*) op 80%. Gevonden frequente regelitems (bevattende één, twee of drie items) worden uitgebeeld in Tabel 3.2. Enkel de eerste vijf frequente regelitems in Tabel 3.2 voldoen aan *minsupp* en *minconf*. Omzetting van deze regelitems in regels en ordening evenals pruning op basis van bepaalde criteria levert een classificatiemodel voor de data in Tabel 3.1 op.

rij-id	A_1	A_2	A_3	klasse
1	a_1	a_2	b_3	y_1
2	a_1	a_2	c_3	y_2
3	a_1	b_2	b_3	y_1
4	a_1	b_2	b_3	y_2
5	b_1	b_2	a_3	y_2
6	b_1	a_2	b_3	y_1
7	a_1	b_2	b_3	y_1
8	a_1	a_2	b_3	y_1
9	c_1	c_2	c_3	y_2
10	a_1	a_2	b_3	y_1

Tabel 3.1: Training dataset (Thabtah, 2007)

Frequente Regelitems			
Antecedent	Consequent (klasse)	Supp	Conf
$\langle a_2, b_3 \rangle$	y_1	4/10	4/4
$\langle a_1, a_2, b_3 \rangle$	y_1	3/10	3/3
$\langle b_3 \rangle$	y_1	6/10	6/7
$\langle a_1, b_3 \rangle$	y_1	5/10	5/6
$\langle a_2 \rangle$	y_1	4/10	4/5
$\langle a_1, a_2 \rangle$	y_1	3/10	3/4
$\langle a_1 \rangle$	y_1	5/10	5/7

Tabel 3.2: Mogelijk classificatiemodel voor de data in Tabel 3.1 (Thabtah, 2007)

3.2 Frequente regelitems minen en CAR's genereren

Om CAR's te minen worden aangepaste associatieregelalgoritmen gebruikt. Vele AC-technieken zoals onder andere CBA (Liu, Hsu & Ma, 1998) en 2SARC (Antonie, Zaiane & Holte, 2006) hanteren hiervoor een aangepaste versie van het Apriori algoritme (Agrawal, Imieliński & Swami, 1993). Predictive Apriori, een aanpassing van Apriori door Scheffer (2005), wordt door ARUBAS(Depaire *et al.*, 2008) gebruikt om frequente regelitems te vinden. Andere AC-algoritmen zoals CMAR (Li, Han & Pei, 2001), CPAR (Yin & Han, 2003) utiliseren respectievelijk de *FP-growth* aanpak, ontwikkeld door Han, Pei & Yin (2000) en een *greedy strategy* gebaseerd op FOIL (Quinlan & Cameron-Jones, 1993). In volgende secties worden enkel Apriori en Predictive Apriori besproken, aangezien ARUBAS in zijn huidige implementatie het Predictive Apriori algoritme implementeert voor de generatie van CAR's. Voor een bespreking van de andere associatieregelalgoritmen verwijzen we door naar hun respectievelijke auteurs alsook naar Thabtah (2007).

3.2.1 Frequente itemsets genereren, Apriori en Predictive Apriori

Het mogelijk aantal associatieregels dat kan gevonden worden in een dataset stijgt exponentieel met het aantal attributen. In vele datasets kan dit aantal behoorlijk hoog liggen. Dit maakt het zoeken naar frequente itemsets computationeel intensief. Om de zoektijd te reduceren is het zodus van belang dat het zoekdomein op een efficiënte wijze beperkt wordt.

Bij het genereren van frequente itemsets maken Apriori (en Predictive Apriori) gebruik van de zogenaamde *downward-closure* eigenschap om het zoekdomein te verkleinen. Deze eigenschap stelt dat indien een itemset Z niet frequent is de additie van een item A, Z niet frequenter zal maken. Met andere woorden, indien Z niet frequent is dan zal $Z \cap A$ ook niet frequent zijn. Hierbij stelt $Z \cap A$ elke mogelijke superset van Z (itemset die Z bevat) voor (Larose, 2005). Uit deze eigenschap volgt tevens dat elke subset (deelverzameling) van een frequente itemset ook frequent is (Thabtah, 2007).

Het algoritme start met het zoeken naar alle frequente itemsets F_1 die uit één item bestaan. Deze itemsets voldoen aan een op voorhand bepaalde minimum *support*. Vervolgens wordt

gezocht naar itemsets bestaande uit twee items. Om alle frequente itemsets met k items te vinden, gaat het Apriori algoritme als volgt te werk. Eerst creëert het een set C_k van kandidaat k-itemsets door alle mogelijke combinaties van F_{k-1} met zichzelf te vormen. C_k wordt hierna in eerste instantie gepruned met behulp van de *downward-closure* eigenschap, de *support* van de overblijvende itemsets wordt vervolgens nagegaan en resterende infrequente itemsets verwijderd. Het algoritme wordt beëindigd wanneer de set die alle frequente itemsets van grootte k bevat leeg is of wanneer k gelijk is aan het totaal aantal attributen (Larose, 2005).

3.2.2 Apriori

Het Apriori algoritme (Agrawal *et al.*, 1993) genereert associatieregels die voldoen aan bepaalde *support* en *confidence* waarden. De regels worden geordend op basis van *confidence*, waarbij regels met gelijke *confidence* volgens *support* worden geordend. De generatie van associatieregels verloopt als volgt.

In eerste instantie produceert het algoritme frequente itemsets, op de wijze uiteengezet in vorige sectie. Hierbij worden enkel itemsets die voldoen aan een minimum *support* in aanmerking genomen, waardoor deze stap zodus een soort op *support* gebaseerde pruning behelst. Tijdens een tweede stap genereert het algoritme regels op basis van de gevonden frequente itemsets, waarbij pruning op basis van *confidence* wordt toegepast. Regels worden gevormd door voor elke frequente itemset f en elke niet lege subset s van f, regels te construeren van de vorm: $s \mapsto (f - s)$. Deze regel moet voldoen aan de vooropgestelde minimum *confidence*. Minimum *support* en *confidence* waarden kunnen door de gebruiker worden ingesteld (Agrawal *et al.*, 1993).

Eén nadeel gerelateerd aan deze methode van associatieregel-mining bevindt zich in het feit dat het altijd wel mogelijk is heel algemene regels met hoge *support* en lage *confidence* en zeer specifieke regels met hoge *confidence* en lage *support* te vinden. Zulke regels weerspiegelen echter meestal geen relaties die ook in de realiteit opgaan en geven zo geen goed of zelfs een incorrect beeld van de achterliggende data. *Support* en *confidence* limieten moeten dan ook

met dit in het achterhoofd ingesteld worden.

3.2.3 Aanpassing Apriori voor Associatieve Classificatie

Het CBA algoritme (Liu *et al.*, 1998), wat staat voor *Classificiation Based on Assocations*, is nog steeds één van de belangrijkste AC-technieken. Het *rule generator* gedeelte van het algoritme (CBA-RG) gebruikte als één van de eerste AC-technieken een aangepaste versie van de Apriori kandidaat generatiestap om regels te genereren.

Zoals reeds besproken genereren associatieve classificatietechnieken een speciaal type associatieregels, *class association rules*, om classificatie mogelijk te maken. Het CBA-RG algoritme gebruikt een virtuele opdeling van de training dataset om het minen van CAR's met behulp van Apriori mogelijk te maken. De trainingset wordt zo opgedeeld dat elke subset slechts instanties van één klasse bevat. Per subset worden vervolgens de frequente itemsets gezocht zoals in de vorige sectie uiteengezet. Regelitems worden gevormd door de frequente itemsets met hun bijhorende klassen te combineren. CAR's worden uiteindelijk gevormd uit de set met alle gevonden regelitems. De *support count* van de volledige regel ($X \cap C$) wordt bijgehouden alsook de *support count* van het antecedent X met behulp van twee tellers. Dit laat toe de *confidence* van een regel te berekenen. De tellers worden tevens gebruikt bij het prunen van regels (Liu *et al.*, 1998).

3.2.4 Predictive Apriori

Het ARUBAS algoritme, zoals geïmplementeerd door Depaire *et al.* (2008), utiliseert het Predictive Apriori algoritme, ontwikkeld door Scheffer (2005), om frequente regelitems te vinden. Net zoals bij Apriori wordt gezocht naar frequente itemsets op basis van *support* en de *downward closure* eigenschap toegepast om het zoekdomein te verkleinen. Apriori prefereert regels met een hogere *confidence*, waarop de rankschikking van gevonden regels gebaseerd wordt. Het Predictive Apriori algoritme echter implementeert de notie dat de aantrekkelijkheid (*interestingness*) van een regel wordt bepaald door een *trade-off* die bestaat tussen de *support* en *confidence* van die regel. Regels met een hoge *support* hebben vaak een lagere *confidence*, de hoge *support* vormt evenwel een additioneel bewijs voor de accuraatheid van

de gevonden *confidence*. Predictive Apriori evalueert zo de *confidence* van regels afhankelijk van hun *support*.

Als maat van aantrekkelijkheid (*interestingness measure*) voor een regel definieert Scheffer de *expected accuracy* van regels op nieuwe data, welke berekend wordt in een bayesiaans raamwerk, uit *confidence* en *support* waarden van een regel. De *expected accuracy* van een regel op nieuwe data is in feite een gecorrigeerde *confidence* waarde van deze regel, waarbij de *confidence* gecorrigeerd wordt op basis van de *support* waarde. Hoe hoger de *support*, hoe dichter de *expected accuracy* score ligt bij de *confidence* waarde. Deze aanpak wordt *Bayesian frequency correction* benoemd (Scheffer, 2005).

In tegenstelling tot bij Apriori moet bij het Predictive Apriori algoritme slechts één parameter ingesteld worden, namelijk het aantal regels n dat het algoritme moet produceren. Het algoritme genereert zo de n regels met de hoogste *expected accuracy* scores. Predictive Apriori bevat verder ook nog een additionele ingebedde *pruning* stap. Bij gelijke *predictive accuracy* scores selecteert het algoritme de meest algemene regel. Verscheidene redenen kunnen aangehaald worden waarom algemene regels te prefereren zijn.

Data mining omvat het extraheren van kennis uit gegevens. Specifieke regels die uit meer algemene regels gevormd worden leveren geen additionele kennis over de data en zijn zodus redundant in een data mining setting. Uit algemene regels kunnen tevens vele specifieke regels geëxtraheerd worden, dit terwijl het classificatiemodel compact gehouden wordt door enkel de meer algemene regels op te nemen. Tenslotte verbetert de generaliseerbaarheid van een classificatiemodel naar nieuwe data toe wanneer algemenere regels, die minstens even accuraat zijn als hun meer specifieke equivalenten, worden gebruikt (Scheffer, 2005).

3.2.5 Aanpassing Predictive Apriori voor CAR Mining

Net als bij Apriori moeten enkele aanpassingen worden uitgevoerd aan het algoritme om CAR mining mogelijk te maken. Twee stappen van het algoritme moeten hiervoor aangepast worden. Een volledige bespreking van het algoritme ligt echter buiten het bereik van deze thesis. Zie hiervoor Scheffer (2005).

3.3 Pruning technieken

Zoals te zien op Figuur 3.1 vormt pruning de verbindende stap tussen het genereren van AC-regels en de classificatie van testdata op basis van het gecreëerde classificatiemodel. Tijdens de pruning-stap worden de classificatieregels geselecteerd die deel zullen uitmaken van het finale classificatiemodel.

Associatieregelalgoritmen genereren over het algemeen zeer veel regels, wat enerzijds de calculatietijd vergroot alsook de interpretatie van de regels door experts bemoeilijk. Hierom is het van groot belang dat in AC-algoritmen efficiënte en effectieve pruning-technieken worden geadopteerd met als objectief een compacte en accurate set regels te bekomen. Zo tracht vermeden te worden dat de uiteindelijke *classifier* redundante of foutieve regels utiliseert voor het classificeren van de testdata. Het beperkt houden van het aantal regels in het finale classificatiemodel voorkomt ook het veel voorkomend probleem van *overfitting*, waarbij het classificatiemodel de originele data wel goed beschrijft, maar veel minder toepasbaar is op nieuwe data (Thabtah, 2007). In volgende secties worden kort enkele pruning technieken, gebruikt bij associatieve classificatie, besproken.

3.3.1 χ^2 test

Statistische testen zoals de χ^2 test kunnen ook toegepast worden voor pruning bij AC. Li *et al.* (2001) hanteren de χ^2 test voor pruning in het CMAR algoritme.

Met behulp van de χ^2 test kan getest worden op positieve en negatieve correlaties tussen antecedent en consequent (de klasse) van gegenereerde regels op basis waarvan pruning kan uitgevoerd worden. Het CMAR algoritme utiliseert de χ^2 test voor pruning als volgt. Bij elke gevonden regel, van de vorm $R : x \mapsto c$, test CMAR of antecedent x gecorreleerd is met klasse c. Indien de test duidt op een positieve correlatie dan wordt regel R opgeslagen om gebruikt te worden in het classificatiemodel, is dit niet het geval wordt de regel verwijderd Li *et al.* (2001). Meer informatie omtrent de χ^2 test kan in elke basis handboek statistiek gevonden worden.

3.3.2 Database *coverage*

De database *coverage* pruning methode, gecreëerd door Liu *et al.* (1998) en toegepast in hun CBA algoritme, zorgt ervoor dat elke regel opgenomen in het finale classificatiemodel minstens één trainingsinstantie correct classificeert en dat elke trainingsinstantie tevens door de hoogst gerangschikte regel, die van toepassing is op de instantie, bepaald wordt. Voor elke geordende regel, startende bij de hoogst gerangschikte regel (r_1), gaat de methode éénmaal doorheen alle trainingsdata om alle instanties te vinden die voldoen aan het antecedent van r_1. Wanneer al deze instanties gevonden zijn worden ze verwijderd en r_1 toegevoegd aan het classificatiemodel. Dit wordt herhaald tot alle traininginstanties bepaald zijn door een regel of alle geordende regels onderzocht zijn. Regels die geen enkele trainingsinstantie bepalen worden altijd verwijderd (Thabtah, 2007). De database *coverage* methode wordt ook toegepast in andere AC-algoritmem zoals bijvoorbeeld het CMAR algoritme (Li *et al.*, 2001).

3.3.3 Pruning van redundante regels

Bij het vormen van regels onderzoeken AC-technieken alle mogelijke attribuutcombinaties als mogelijk regel antecedent. Enorm grote aantallen regels, waaronder regels die specifiekere gevallen van meer algemene regels vertegenwoordigen, kunnen zo worden gegenereerd. Om te voorkomen dat zulke redundante regels opgenomen worden in het finale classificatiemodel werd door Li *et al.* (2001) de zogenaamde *redundant rule pruning* techniek ontwikkeld en toegepast in hun CMAR algoritme.

De *redundant rule pruning* techniek prefereert meer algemene regels met hogere *confidence* en verwijdert meer specifieke regels met lagere *confidence*. Algemene regels worden geprefereerd omwille van twee problematische eigenschappen gerelateerd aan specifieke regels. Eerst en vooral bevatten specifieke regels vaak overbodige informatie die ook terug te vinden is in meer algemene regels. Ten tweede, omdat specifieke regels vaak een lage *support* bezitten, is de kans groter dat deze regels gevonden werden als gevolg van ruis in de data. Pruning wordt uitgevoerd als volgt. Nadat alle regels gegenereerd zijn en gerangschikt op basis van bepaalde criteria wordt de *redundant rule pruning* stap uitgevoerd. Hierbij wordt elke regel van de

vorm $I' \mapsto c$ gepruned, indien een algemene regel $I \mapsto c$ bestaat die een hogere rank heeft en waarbij I een subset van I' vertegenwoordigt (Thabtah, 2007).

Het CMAR algoritme past *redundant rule pruning* toe wanneer een regel in de *CR-tree* wordt opgenomen. Tijdens de invoeging van een regel in de *CR-tree* (zie Li *et al.* (2001)) wordt de *CR-tree* doorlopen en nagegaan of de regel gepruned kan worden of dat regels reeds opgenomen, kunnen verwijderd worden (Li *et al.*, 2001). Ook andere algoritmen zoals in Antonie *et al.* (2006) utiliseren *redundant rule pruning*.

3.3.4 Pruning gebaseerd op de *pessimistic error rate*

De *pessimistic error rate* die gebruikt wordt bij pruning in beslissingsbomen kan ook in AC toegepast worden om regels te prunen. Bij beslissingsbomen worden doorgaans twee soorten pruning technieken onderscheiden, postpruning en prepruning waarbij postpruning frequenter toegepast wordt. Postpruning houdt in dat de beslissingsboom eerst volledig wordt opgebouwd, waarna beslist wordt aan elke *node* van de boom of de *node* vervangen wordt door een *leaf* (*subtree replacement*), of door een *node* eronder gelegen (*subtree raising*). Deze beslissing wordt gebaseerd op een vergelijking van de *pessimistic error rates* aan de potentieel te vervangen *node* en de mogelijk vervangende *leaf* of *node* (Witten & Frank, 2005).

De toepassing van de *pessimistic error rate* voor pruning in AC gebeurt als volgt. Voor elke gegenereerde regel r wordt getest of de *pessimistic error rate* van r hoger is dan de *pessimistic error rate* van regel r', waarbij r' verkregen werd door het verwijderen van één van de items in het precedent van regel r. Indien de *pessimistic error rate* van regel r hoger is dan die van regel r' dan wordt r vervangen door r' (Thabtah, 2007). Pruning op basis van de *pessimistic error rate* wordt onder andere toegepast in CBA (Liu *et al.*, 1998).

3.3.5 *Lazy pruning*

Lazy pruning gelijkt sterk op de *database coverage* pruning methode, echter worden bij *lazy pruning* enkel de regels verwijderd die trainingsinstanties foutief classificeren. *Lazy pruning* creëert in feite een classificatiemodel dat uit twee niveaus bestaat. Regels die bij *database*

coverage pruning verwijderd zouden worden, worden opgeslagen op een tweede niveau van het classificatiemodel. Dit verloopt als volgt.

Nadat de regels gegenereerd zijn wordt elke trainingsinstantie één voor één geselecteerd en de eerste regel uit de set gerangschikte regels die van toepassing is op de instantie eraan toegewezen. De geselecteerde trainingsinstantie wordt vervolgens verwijderd, waarna wordt gecontroleerd of de instantie correct geklasseerd is. Wanneer de volledige trainingsdataset doorlopen is, worden enkel de regels die een verkeerde classificatie veroorzaakten verwijderd en de instanties die ze (incorrect) klasseerden doorlopen het proces opnieuw. Dit herhaalt zich tot alle instanties correct geklasseerd zijn. Uiteindelijk levert deze vorm van pruning een tweeledig classificatiemodel op. Het eerste niveau bevat de regels die minstens één trainingsinstantie correct klasseerden, het tweede niveau bestaat uit de regels die niet gebruikt werden tijdens de trainingsfase. Het tweede niveau regels wordt gebruikt wanneer geen enkele van de regels in de eerste set toepasbaar zijn op een te classificeren instantie. Zodoende gaat geen potentieel bruikbare kennis verloren door het prunen van te veel regels (Thabtah, 2007). Thabtah wijst erop dat ondanks de kleine verbetering in *accuracy* die *lazy pruning* oplevert ten opzichte van *database coverage* pruning, het gebruik van deze techniek ook negatieve gevolgen met zich meebrengt. Zo resulteert de toepassing van *lazy pruning* vaak in enorm grote classificatiemodellen die moeilijk te begrijpen of te bestuderen zijn door domeinexperts. Verder utiliseren *lazy pruning* algoritmen meer geheugen dan andere AC-technieken en indien het aantal te verwerken regels zeer hoog is, kan het algoritme zelfs falen.

3.3.6 Significantie van pruning

Aangezien in AC het aantal potentiële regels exponentieel stijgt met het aantal attributen, produceren AC-algoritmen enorme classificatiemodellen. Indien geen restricties ingesteld worden tijdens de regelgeneratiefase, of geen pruning wordt toegepast kan het aantal regels opgenomen in het classificatiemodel enorm hoge aantallen bereiken. Dit heeft als gevolg dat het finale model ondoorgrondelijk en onbegrijpelijk wordt voor de eindgebruiker.

3.4 Regelsortering

De ordening van gevonden CAR's speelt een belangrijke rol in het AC-proces omdat vele AC-algoritmen (e.g. CBA, CPAR en CMAR) de rangschikking van regels hanteren als basis voor het selecteren van het classificatiemodel en het prunen van CAR's.

Meest voorkomend in AC is regelranking op basis van *support*, *confidence* en kardinaliteit van het regelantecedent. Deze regelranking methode prefereert algemene regels. Andere methoden zijn ontwikkeld zoals in L^3 (Baralis, Chiusano & Garza, 2004) die meer specifieke regels hoger rangschikken. Regelsortering wordt hier niet verder behandeld. Voor meer informatie hieromtrent verwijzen we naar Thabtah (2007).

3.5 Classificatie

Classificatie op basis van gevonden CAR's vormt de finale stap in het AC-proces. In deze sectie worden verscheidene methoden die gebruikt worden voor het uitvoeren van de classificatiestap globaal besproken. Resulterend uit deze laatste stap wordt het finale classificatiemodel (de *classifier*) bekomen. Pruning, besproken in vorige sectie, heeft als voorbereidende stap de taak de regelset compacter te maken om zo enkel de beste regels te behouden. Tijdens de classificatie moet nu beslist worden hoe deze regels toegepast worden om ongekende instanties te classificeren. De methoden gebruikt voor classificatie in AC kunnen in twee groepen worden ingedeeld: methoden die voorspellen op basis van meerdere regels en methoden die slechts de beste regel die van toepassing is op een bepaalde instantie hanteren voor het toewijzen van een klasselabel.

3.5.1 Classificatie op basis van één enkele regel

Het CBA algoritme (Liu *et al.*, 1998) voert de classificatie uit op basis van één regel met behulp van een beslissingslijst. De geprunede en geordende lijst regels, verkregen uit voorgaande AC-stappen, wordt doorlopen op zoek naar de eerste regel die van toepassing is op de te classificeren instantie. Het consequent van deze regel wordt toegewezen als klasse-label. Is geen enkele regel van toepassing dan wordt de default klasse (de meest voorkomende klasse

in de trainingsset) toegewezen.

3.5.2 Classificatie op basis van meerdere regels

Bij het classificeren kunnen meerdere regels van toepassing zijn op één ongekende instantie. Bepaalde AC-algoritmen implementeren dan ook classificatiealgoritmen die meerdere regels gebruiken voor het toewijzen van een klassenlabel. Enkele in AC gebruikte methoden voor classificatie op basis van meerdere regels worden hier kort besproken.

Weighted vote **methoden**

Weighted vote algoritmen hanteren een bepaalde procedure om gewichten toe te kennen aan de regels en zo met inachtneming van meerdere regels en bijbehorende gewichten te classificeren. CMAR (Li *et al.*, 2001) hanteert zulke procedure waarbij een subset van regels met hoge *confidence* die van toepassing zijn op de testinstantie wordt geanalyseerd, gebruik makend van een gewogen χ^2 analyse. Bevat de subset slecht regels die één enkele klasse aanduiden, wordt deze klasse toegewezen aan de instantie. Indien de subset van regels verschillende klassen aanduidt, worden de regels per klasse gegroepeerd en de sterkte van de groep op basis van *support* en interne correlatie (op basis van een gewogen χ^2 analyse) berekend. Het klasse-label van de sterkste groep wordt vervolgens aan de ongekende instantie toegewezen. Een uitgebreidere bespreking van dit algoritme valt buiten het bereik van deze thesis. We verwijzen hiervoor naar Thabtah (2007).

Classificatie op basis van de Laplace *accuracy*

Classificatie op basis van meerdere regels kan worden uitgevoerd met behulp van de Laplace *accuracy*. Deze methode wordt in CPAR toegepast. CPAR past de Laplace *expected accuracy* toe om de accuraatheid van regels te berekenen. De *expected accuracy* wordt als volgt berekend:

$$LaplaceAccuracy = (n_c + 1)/(n_{tot} + k) \tag{3.1}$$

waarbij k het aantal klassen voorstelt, n_{tot} het totale aantal instanties dat voldoet aan het antecedent van de regel en n_c het aantal instanties die tot klasse c, de klasse die de regel

aanduidt, behoren. Het CPAR algoritme berekent de *expected accuracy* van elke regel voor het classificeren van nieuwe data. Gegeven een set regels selecteert CPAR de beste k regels die toepasbaar zijn op de ongekende instantie. De gemiddelde *expected accuracy* van de k beste regels per klasse worden vergeleken en de klasse met de hoogste *expected accuracy* toegewezen aan de instantie (Yin & Han, 2003).

3.6 Besluit

In dit hoofdstuk werden de algemene stappen in AC uiteengezet en de verschillende aanpakken voor het uitvoeren van deze stappen, toegepast door verscheidene bestaande AC-algoritmen, kort besproken. Dit hoofdstuk beoogde geen alomvattende vergelijkende studie van bestaande AC-technieken neer te zetten, maar eerder een schets te maken van verscheidene bestaande technieken in het domein. Zo vormt de vergelijkende literatuurstudie in dit hoofdstuk een inleiding op de uiteenzetting van het ARUBAS raamwerk in volgend hoofdstuk, alsook op het praktijkonderzoek dat besproken wordt in Hoofdstuk 5. Voor een uitgebreidere bespreking en vergelijking van bestaande AC-technieken verwijzen we naar Thabtah (2007).

Hoofdstuk 4

ARUBAS

ARUBAS, ontwikkeld door Depaire *et al.* in 2008 kan omschreven worden als een raamwerk om associatieve classificatiemodellen te construeren. Deze techniek omvat een geheel nieuwe aanpak om associatieregels te gebruiken voor classificatiedoeleinden, die gelijkenissen vertoont met het werk van Antonie *et al.* (2006). In tegenstelling tot eerder besproken associatieve classificatietechnieken gebruikt ARUBAS gevonden CAR's niet om nieuwe instanties toe te wijzen aan een bepaalde klasse, maar in een eerste fase om de huidige feature-ruimte van de trainingsinstanties te transformeren naar een nieuwe, krachtigere feature-ruimte. In een tweede stap vervolgens, wordt instantiegebaseerde classificatie toegepast om nieuwe instanties te classificeren.

Dit hoofdstuk behandelt de werking van het huidige ARUBAS-algoritme. We besluiten met het vormen van een onderzoeksvraag voor de potentiële verbetering van het algoritme. Deze onderzoeksvraag vormt de basis voor het uitgevoerde Masterproef onderzoek dat in Hoofdstuk 5 uitgebreid uiteengezet wordt.

4.1 Het ARUBAS raamwerk

4.1.1 Patroonruimte

In de originele feature-ruimte, genaamd de attribuutruimte, bestaat elke instantie X_d uit een combinatie van attribuutwaarden en een klassewaarde, voorgesteld door $X_d = (a_1, ..., a_m, c)$.

Elke dimensie in deze attribuutruimte bestaat uit één attribuut. Antonie *et al.* (2006) introduceerden een feature-ruimte transformatie in hun 2SARC2 algoritme waarbij instanties opnieuw worden gedefinieerd op basis van regelkenmerken. In de nieuwe feature-ruimte, in het ARUBAS raamwerk benoemd patroonruimte, omvat elke dimensie een combinatie van attributen, zogeheten patronen, genoteerd als $P_p = ((A_{i1}, a_{i1}), ..., (A_ik, a_ik))$.

Aangezien classificatie van nieuwe instanties hier het doel is, zijn slechts de patronen die sterk geassocieerd zijn met één enkele klasse-waarde van belang. Elke bestaande *class association rule* techniek heeft echter als doel patronen te zoeken die sterk gecorreleerd zijn met een klasse-waarde. Hierdoor kan in het ARUBAS raamwerk eender welke CAR mining techniek toegepast worden om CAR's te vinden op basis waarvan de transformatie van de feature-ruimte bewerkstelligd wordt. De antecedenten van de gevonden CAR's vormen de patronen die de nieuwe dimensies van de patroonruimte uitmaken. In de patroonruimte worden records uitgedrukt als combinaties van patronen met een klassewaarde. De transformatie van attribuutruimte naar patroonruimte kan als volgt worden weergegeven:

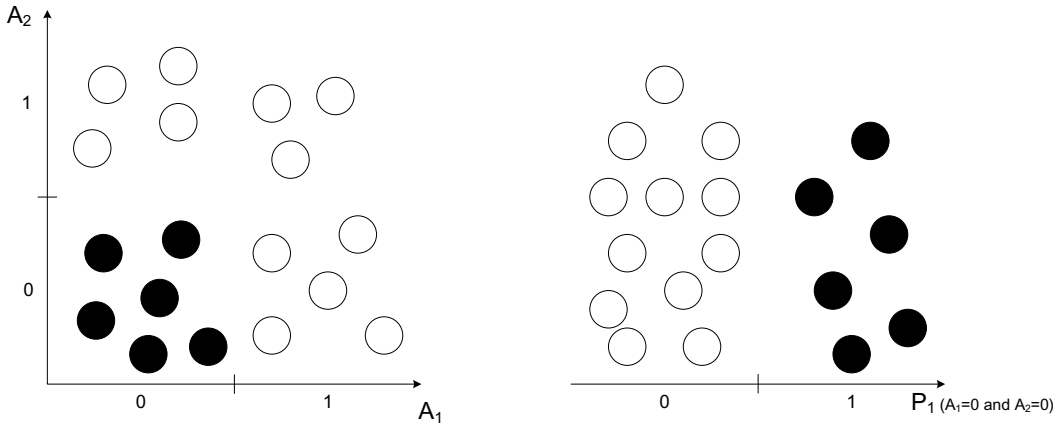
$$X_t = (A_1, ..., A_m, C) \mapsto X_t = (P_1, ..., P_p, C) \tag{4.1}$$

Indien X_t een patroon P_p bevat, heeft de instantie hiervoor waarde 1, zo niet heeft ze waarde 0. Hierbij stel p het aantal gevonden CAR's voor (Depaire *et al.*, 2008).

Stel nu dat de originele feature-ruimte twee attributen A_1 en A_2 en een klasse C bevat. Beide attributen zijn continu en moeten daardoor gediscretizeerd worden. Discretizering leidt tot een transformatie naar binaire attributen waarbij A_1 gediscretizeerd wordt tot $A_1^* = \{0, 1\}$ en A_2 tot $A_2^* = \{0, 1\}$. Klasse $C = \{0, 1\}$ is reeds binair.

De linkerzijde van Figuur 4.1 toont de verdeling van de instanties in de originele feature-ruimte. Instanties met klassewaarde 1 (zwart) bevinden zich linksonder op de figuur. De toepassing van een CAR mining algoritme op deze data levert waarschijnlijk volgende regel op: $\langle(A_1, 0), (A_2, 0)\rangle \Rightarrow 1$, met een *support* en *confidence* van respectievelijk 33% en 100%. Veronderstellend dat dit de enige gevonden CAR is, bekomen we patroon $P_1 = \langle(A_1, 0), (A_2, 0)\rangle$. De patroonruimte heeft zodoende slechts twee dimensies, namelijk het patroon en de klasse.

Door de transformatie van attribuutruimte naar patroonruimte wordt de rechterzijde van Figuur 4.1 verkregen. Instanties in deze patroonruimte worden enkel onderscheiden op basis van het voldoen aan patroon P_1 of het niet voldoen. Een enkele scheidingslijn verdeelt de instanties in hun respectievelijke klassen, terwijl in de attribuutruimte, zoals te zien op de linkerzijde van Figuur 4.1, twee scheidingslijnen nodig zijn om hetzelfde te bereiken. Op deze manier vereenvoudigt de transformatie het classificatieprobleem. Depaire *et al.* (2008) wijzen erop dat dit voorbeeld vrij academisch is en een versimpelde versie van reële datasets voorstelt, desondanks illustreert het duidelijk de potentiële voordelen van de feature-ruimte transformatie.



Figuur 4.1: Feature-ruimte transformatie (Depaire *et al.*, 2008)

In essentie is de patroonruimte krachtiger dan de attribuutruimte omdat de gegevens in de patroonruimte worden omschreven door middel van attribuutcombinaties die sterk met elkaar en met een klassewaarde gecorreleerd zijn. Omdat enkel attributen met een beperkt aantal mogelijke waarden reeds een groot aantal patronen kunnen voortbrengen (drie attributen met elke vier mogelijk attribuutwaarden kunnen tot 64 patronen opleveren), kan de patroonruimte echter ook complexer worden dan de attribuutruimte. Om dit te voorkomen moet er voor gezorgd worden dat enkel de sterkste correlaties als patronen worden gebruikt, wat bereikt kan worden door aangepaste limieten voor *support* en *confidence* te stellen (Depaire *et al.*, 2008).

4.1.2 Classificatie

Na de transformatie van alle beschikbare traininginstanties naar de patroonruimte worden deze getransformeerde instanties geütiliseerd om nieuwe instanties aan een klasse toe te wijzen. Deze classificatiestap wordt uitgevoerd met behulp van instantiegebaseerde classificatie. Een te classificeren nieuwe case X_N wordt in deze fase vergeleken met elke trainingcase X_T waarvan de klasse reeds geweten is. De gelijkenis tussen de twee cases wordt berekend met behulp van een gekozen gelijkheidsmaat (*fitness measure*). Er wordt hierbij gekeken naar het aantal patronen die beide instanties overeenkomstig bezitten. Met andere woorden het aantal overeenkomstige patronen waarvoor beide instanties waarde 1 hebben. Aangezien patronen atrribuutcombinaties gecorreleerd met een klassewaarde zijn, worden enkel patronen vergeleken uit CAR's die de klasse van X_T correct voorspelden. Het vergelijken van patronen tussen X_T en X_N die naar een andere klasse wijzen dan die waartoe X_T behoort heeft geen zin namelijk, omdat het overeenkomen van zulke patronen geen additioneel bewijs levert dat X_N tot dezelfde klasse behoort als X_T. Met het oog ze te onderscheiden van gewone patronen duiden Depaire *et al.* patronen, uit regels gehaald die de klasse van trainingsinstantie X_T correct voorspellen, aan met P^T.

In hun bespreking van het ARUBAS raamwerk stellen Depaire *et al.* vijf verschillende *fitness measures* voor die gebruikt kunnen worden om de gelijkheid tussen een nieuwe case en een trainingscase te berekenen. Depaire *et al.* zetten een experiment op waarbij het ARUBAS algoritme vijf keer werd toegepast, telkens met de implementatie van een verschillende *fitness measure*, en de resulterende *accuracies* statistisch werden vergeleken. Dit experiment toonde aan dat een *fitness measure* gebaseerd op gewogen gemiddelde *similarity* significant beter presteerde dan vier andere *fitness measures*.

De *similarity* tussen trainingcase X_T en nieuwe case X_N wordt als volgt berekend:

$$s(X_T, X_N) = \frac{\sum_{i=1}^{p} P_{Ti}^T P_{Ni}^T}{max\left[\sum_{i=1}^{p} P_{Ti}^T, \sum_{i=1}^{p} P_{Ni}^T\right]}. \tag{4.2}$$

Deze formule berekent het percentage van patronen dat de instantie met de minste patronen

deelt met de instantie die de meeste patronen bevat. Stel dat $X_T = (1, 1, 1, 1, c_1)$ en $X_N = (1, 0, 0, 0, ?)$. Hierbij bevat X_N één van de vier patronen die op X_T van toepassing zijn. Het toepassen van vergelijking 4.2 resulteert in: $s(X_T, X_N) = 0,25$. De ongeclassificeerde case wordt toegewezen aan de klasse waarmee de case gemiddeld genomen de grootste gelijkenis vertoont. Vergelijking 4.2 bezit tevens volgende eigenschappen:

$$\forall i \in 0, ..., p : P^T_{Ti} = P^T_{Ni} \Leftrightarrow s(X_T, X_N) = 1, \tag{4.3}$$

$$\forall i \in 0, ..., p : P^T_{Ti} \neq P^T_{Ni} \Leftrightarrow s(X_T, X_N) = 0. \tag{4.4}$$

Vergelijking 4.2 heeft echter ook te kampen met een beperking. Ze houdt namelijk geen rekening met het aantal patronen dat de nieuwe en de gekende instantie bevatten. Dit feit leidt potentieel tot misleidende *similarity* waarden, wat met behulp van Tabel 4.1 geïllustreerd wordt. Instanties X_{T1} en X_{N1} zijn hier identiek, hoewel ze slechts één patroon gemeen hebben. Instanties X_{T2} en X_{N2} bezitten drie dezelfde patronen, maar zijn niet identiek. Hierdoor zal hun *similarity* waarde lager dan die van (X_{T_1}, X_{N1}). Omdat ze echter meer overeenkomstige patronen bezitten lijkt het intuïtief gezien aannemelijk dat er meer bewijs is dat instanties X_{T2} en X_{N2} tot dezelfde klasse behoren. Om dit mee in rekening te brengen wordt een wegingsfactor geïntroduceerd. Deze wegingsfactor bepaalt het percentage van het aantal

Tabel 4.1: Voorbeeld Data (Depaire *et al.*, 2008)

X_{T1}	(1, 0, 0, 0, c_1)
X_{N1}	(1, 0, 0, 0, c_1)
X_{T2}	(1, 0, 1, 1, c_1)
X_{N2}	(1, 1, 1, 1, c_1)

patronen — geassocieerd met de klasse waartoe trainingsinstantie X_T behoort — waaraan X_T of X_N voldoen. De wegingsfactor wordt als volgt uitgedrukt, waarbij p^T het aantal patronen verkregen uit regels die de klasse-waarde van instantie X_T voorspellen voorstelt:

$$\alpha_{TN} = \frac{\sum_{i=1}^{p} 1 - (1 - P^T_{Ti})(1 - P^T_{Ni})}{p^T}. \tag{4.5}$$

De *similarity* van een nieuwe instantie met betrekking tot een trainingsinstantie (berekend op basis van vergelijking 4.2) en de wegingsfactor uit vergelijking 4.5 kunnen samen toegepast worden om de overeenstemming tussen de nieuwe instantie en een specifieke klasse te berekenen, wat als volgt uitgedrukt wordt:

$$S_c(X_N) = \frac{\sum_{t=0}^{|T_c|} \alpha_{tN} s(X_t, X_N)}{|T_c|}. \tag{4.6}$$

$|T_c|$ stelt hierbij het aantal trainingsinstanties met klassewaarde c voor. De teller drukt het gewogen gemiddelde tussen nieuwe instantie X_N en elke trainingsinstantie X_t van klasse c uit. Het is vergelijking 4.6 die significant beter presteerde dan de andere vier *fitness measures* bij een statistische vergelijking over 23 UCI datasets van de *accuracies* van vijf ARUBAS implementaties met telkens een andere gebruikte *fitness measure* (Depaire *et al.*, 2008).

4.1.3 Het algoritme

Het ARUBAS raamwerk biedt de mogelijkheid om eender welk CAR mining algoritme te hanteren om de benodigde regels te genereren. Om het raamwerk empirisch te testen gebruikten Depaire *et al.* een aangepaste versie van Scheffer's (2005) Predictive Apriori algoritme. Scheffer's algoritme steunt op het idee dat de aantrekkelijkheid (*interestingness*) van een regel wordt bepaald door een *trade-off* die bestaat tussen de *support* en *confidence* van die regel. Als maat van aantrekkelijkheid (*interestingness measure*) voor een regel definieert Scheffer de zogenaamde *expected accuracy* van regels op nieuwe data, gebaseerd op *confidence* en *support*. Het selecteren van de beste regels wordt gebaseerd op hun *expected accuracy* score. Voor een uitgebreidere bespreking van het Predictive Apriori algoritme verwijzen we terug naar Hoofdstuk 3. Een voordeel van dit algoritme is dat de onderzoeker geen *support* en *confidence* limieten meer moet bepalen maar enkel het aantal gewenste regels. Op basis hiervan bepaalt het algoritme automatisch de *expected accuracy* limiet en worden regels die hier niet aan voldoen verwijderd.

Het ARUBAS raamwerk zelf bezit geen enkele parameter, waardoor, uit de vereniging ervan met Scheffer's Predictive Apriori, een algoritme ontstaat met slechts één parameter, namelijk

het aantal CAR's die gegenereerd moeten worden. Depaire *et al.* definiëren dit als het ARUBAS-Scheffer AC algoritme.

Experimenteel onderzoek, uitgevoerd door Depaire *et al.*, toonde verder aan dat de gemiddelde *accuracy*, verkregen bij tenfold cross validation toegepast op de originele dataset, overeenkomstig verloopt met de evolutie van de accuracy die verkregen wordt indien trainen en testen uitgevoerd wordt op de trainingsset. Dit heeft als gevolg dat de originele dataset gebruikt kan worden voor de optimalisatie van de enige parameter van het algoritme. Alzo wordt een AC-techniek zonder parameters verkregen.

4.2 Empirische Resultaten

(Depaire *et al.*, 2008) testten het potentieel van het ARUBAS raamwerk door een experiment op te zetten waarbij ARUBAS-Scheffer vergeleken werd met andere bestaande classificatietechnieken, namelijk met C4.5, Ripper, 1R en CBA algoritmen. De *accuracies* van de algoritmen over 23 UCI datasets werden vergeleken. De resultaten van dit experiment toonden aan dat het ARUBAS-Scheffer algoritme, op een significantieniveau van 10%, niet significant minder presteert dan C4.5, RIPPER en CBA en significant betere prestaties behaalt als 1R. ARUBAS-Scheffer kwam tijdens dit experiment naar voren als de tweede beste classificatietechniek (een gedeelde tweede plaats met de 'geprunede' versie van het RIPPER algoritme). Enkel de 'geprunede' versie van C4.5 ging ARUBAS-Scheffer vooraf. Dit verschil was echter niet significant op 10%. Een belangrijk gegeven dat Depaire *et al.* hierbij verder opmerken is het feit dat, in tegenstelling tot de andere classificatietechnieken, ARUBAS-Scheffer een parametervrije classificatietechniek is.

De huidige onderzoeksresultaten omtrent de prestaties van ARUBAS zijn dus reeds veelbelovend. In deze thesis willen we echter nagaan of een kleine aanpassing aan het algoritme tot nog betere prestaties leidt. Zo zullen we onderzoeken of het vergelijken met slechts een subset van N cases per klasse, tijdens de instantiegebaseerde classificatiefase, een betere classificatie oplevert.

4.3 Onderzoeksvraag

Leidt het aanpassen van ARUBAS, door in de vergelijkingsfase slechts met een subset van N cases per klasse te vergelijken, tot betere prestaties?

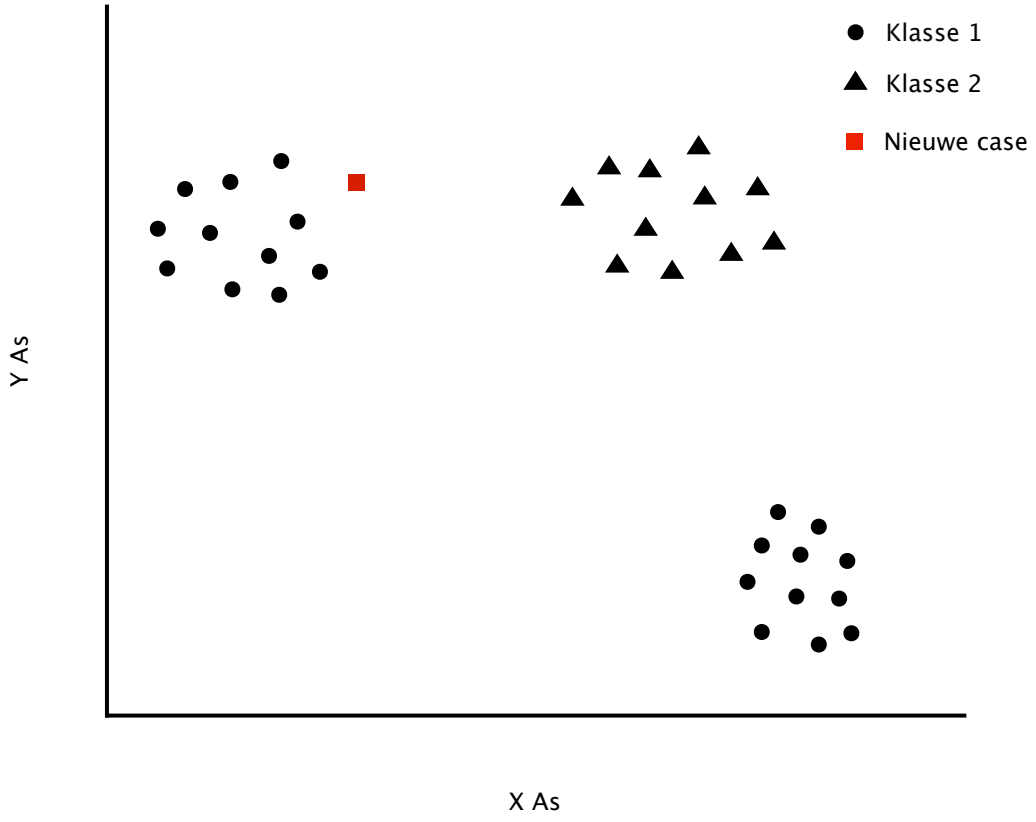
Deze aanpassing aan het algoritme zou eventueel betere resultaten opleveren omdat bij de cases met eenzelfde klasse mogelijk groepen met uiteenlopende kenmerken bestaan die, wanneer vergeleken wordt met alle cases per klasse, verkeerde classificaties kunnen veroorzaken. Dit wordt zeer eenvoudig in Figuur 4.3 geïllustreerd.

Deze figuur toont dat de instanties, die in de trainingsset geclassificeerd zijn als klasse 1, in twee groepen verspreid zitten naargelang eigenschappen. Dit wordt in de figuur voorgesteld door de grote afstand tussen de twee groepen. Indien we nu een nieuwe case trachten te classificeren en we deze vergelijken met alle trainingsinstanties van klasse 1 en alle instanties van klasse 2, zal deze hoogstwaarschijnlijk toegewezen worden aan klasse 2. Dit doordat de gemiddelde gelijkenis van de nieuwe case met alle instanties van klasse 1 sterk negatief beïnvloed zal worden door de groep instanties rechtsonder. We zien echter dat de ongeclassificeerde case veel dichter bij een deel van de instanties van klasse 1 ligt en waarschijnlijk dus ook tot deze klasse behoort. Indien we nu slechts met een subset N van de trainingsinstanties, degene die het dichtst bij de nieuwe case liggen, vergelijken, zal de nieuwe case hoogstwaarschijnlijk wel aan klasse 1 worden toegewezen. Door deze aanpassing aan het algoritme zullen sommige verkeerde classificaties dus mogelijk voorkomen worden.

In feite wordt hierdoor een soort van na-pruning uitgevoerd. Uitschieters, met betrekking tot de te classificeren instantie, worden buiten beschouwing gelaten en enkel de sterkst overeenkomstige trainingsinstanties worden gebruikt voor classificatie.

Bij het vergelijken van de algoritmen hanteren we de verkregen *accuracies* om significante verschillen tussen de prestaties te onderzoeken. In het domein van data mining bestaat echter controverse rond het gebruik van *accuracies* om de prestaties van classificatie-algoritmen te vergelijken. Provost, Fawcett & Kohavi (1998) tonen aan dat de argumenten, die naar voren

geschoven worden om het gebruik van *accuracies* voor de vergelijking van classificatietechnieken te rechtvaardigen, in vele gevallen betwistbaar zijn. Zij adviseren daarom het uitvoeren van een ROC (*receiver operating characteristic*) analyse om de prestaties van classificatie algoritmen onderling te vergelijken. Aangezien ROC analyse echter moeilijker uit te voeren is en geen eenduidige resultaten oplevert (zie Provost *et al.* (1998)), zullen we tijdens onze vergelijkende studie toch behaalde *accuracies* hanteren als prestatiemaat. Andere recente studies van classificatietechnieken tonen tevens aan dat dit nog steeds de standaard betreft in het domein.



Figuur 4.2: De classificatie van nieuwe instanties

4.3.1 Deelvragen

Bovenstaande onderzoeksvraag delen we hier op in enkele specifiekere deelvragen, welke vervolgens kort besproken.

i) Moet N absoluut of procentueel bepaald worden?

We zullen bij verschillende datasets verscheidene waarden voor N testen, eerst absoluut bepaald, daarna procentueel (P), om de resultaten van beide methoden te vergelijken. Op basis van deze resultaten in combinatie met andere overwegingen, zullen we opteren verdere experimenten met procentuele of absolute waarden uit te voeren.

ii) Hoe groot moet N/P zijn om de beste classificatieresultaten te bekomen?

Bij het aanpassen van het algoritme zal een bepaalde grootte voor N of P moeten gekozen worden. Hierbij is het van belang dat onderzocht wordt welke invloed verschillende waarden voor N en P hebben op de classificatieresultaten. Hieruit zal dan getracht worden een optimale waarde voor N/P te bepalen. Een belangrijke vraag die hierbij gesteld moet worden is of de optimale waarde voor N/P gelijk blijft voor verschillende datasets of telkens anders is. Indien deze varieert zullen we onderzoeken of mogelijk is de optimale waarde telkens per dataset te berekenen en op basis hiervan de classificatie uit te voeren.

iii) Indien deze varieert, kunnen we per dataset de optimale waarde voor N/P bepalen?

Zoals hiervoor reeds aangehaald, is het mogelijk dat de optimale grootte van N/P afhankelijk is van de dataset waarop we het ARUBAS-algoritme toepassen. Als dit uit onderzoek zo blijkt te zijn, zullen we nagaan of het mogelijk is de optimale waarde telkens per dataset te berekenen en op basis hiervan de classificatie uit te voeren.

Hoofdstuk 5

Experimentele Methodologie en Bespreking Praktijkonderzoek

5.1 Statistische vergelijking van classificatiemethoden

In deze thesis volgen we de methodologie uitgelijnd door Demšar (2006) om prestaties van verscheidene classificatiemethoden te vergelijken. Deze methodologie beschrijft hoe getest kan worden of de prestatiescores (in dit onderzoek de *accuracies*) van verschillende algoritmen significant verschillen. Demšar adviseert hiervoor de Friedman test toe te passen, waarbij hij zich baseert op het werk van Friedman (1937, 1940). Om, indien meer dan twee algoritmen vergeleken worden, te onderzoeken welke specifieke methoden onderling significant verschillen stelt hij de Nemenyi test voor, zich baserend op Nemenyi (1963).

Statistische technieken die gebruikt worden om significante prestatieverschillen tussen twee algoritmen te testen zijn over het algemeen niet geschikt om de prestaties van meerdere algoritmen onderling te vergelijken. Indien meerdere algoritmen paarsgewijs worden vergeleken stijgt de kans dat de nulhypothese, stellende dat geen prestatieverschillen bestaan tussen de verschillende algoritmen, onterecht wordt verworpen. We verduidelijken dit aan de hand van een voorbeeld ontleend uit het werk van Jensen & Cohen (2000).

Stel dat een investeringsadviseur moet aangenomen worden. De taak van deze adviseur

bestaat erin te voorspellen of de aandelenmarkt omhoog of omlaag gaat. Bij het aannemen van deze adviseur wil men dat deze betere voorspellingen maakt dan wanneer random gegokt zou worden. Om de beste adviseur te kiezen wordt een test opgesteld. Gedurende 14 dagen moet een kandidaat voorspellingen doen, waarbij vanaf 11 juiste voorspellingen geconcludeerd wordt dat de kandidaat kennis van zaken bezit. De grens van 11 juiste voorspellingen wordt gekozen omdat, indien de kandidaat niet weet wat hij doet (en dus 50% kans heeft op een gegeven dag om een juiste voorspelling te maken), slechts een kans van 2,87% bestaat dat hij 11 of meer juiste voorspellingen zou maken. Zo wordt de kans om een onbekwame adviseur aan te nemen beperkt tot 2,87%.

Wanneer meerdere kandidaten worden vergeleken gaat deze logica echter niet meer op. Indien toegepast op één kandidaat, houdt deze test inderdaad een kans van 2,87% in om de foute beslissing te nemen. Wordt echter vergeleken tussen n kandidaten dan stijgt deze kans in verhouding met n. Zo kan de kans dat een onbekwaam adviseur wordt aangenomen hier als volgt berekend worden: $1 - (1 - 0,0287)^n$. Worden 10 kandidaten vergeleken loopt deze kans op tot 25,3% (Jensen & Cohen, 2000). Dezelfde logica gaat ook op bij het vergelijken van meerdere classificatietechnieken. Worden twee technieken vergeleken en wijst dit uit dat één van de twee technieken beter presteert als de andere op een significantieniveau van 5% wil dit daadwerkelijk zeggen dat slechts een kans van 5% bestaat dat dit prestatieverschil toevallig is. Worden echter vier classificatietechnieken paarsgewijs vergeleken, omvat elk van deze vergelijkingen een kans van 5% dat het gevonden prestatieverschil in werkelijkheid niet bestaat. Zes paarsgewijze vergelijkingen zijn nodig om alle algoritmen onderling te vergelijken. Bij toepassing van bovenvermelde formule verkrijgen we: $1 - (1 - 0,05)^6 = 0,26 = 26\%$. De kans dat eender welk van de onderlinge prestatieverschillen niet de werkelijkheid vertegenwoordigt, bedraagt dus 26%.

In zijn bespreking van technieken om meerdere classificatietechnieken te testen laat Demšar technieken die ontwikkeld zijn voor de paarsgewijze vergelijking van twee algoritmen links liggen. ANOVA en de Friedman test (met bijhorende Nemenyi post-hoc test) worden echter wel besproken.

ANOVA is een parametrische test die steunt op enkele assumpties voor het vergelijken van *sample* gemiddelden (in dit geval zijnde de *accuracies*) zoals de normale verdeling van de *accuracies* over de verscheidene datasets en de *sphericity* assumptie die stelt dat de *random variables* een gelijke variantie bezitten. Bij het vergelijken van de prestaties van *machine learning* algoritmen worden deze twee assumpties echter hoogstwaarschijnlijk geschonden. Voornamelijk het schenden van de tweede assumptie levert mogelijk problemen op, aangezien het niet voldoen aan de sphericity assumptie leidt tot een significante stijging van de Type 1 foutratio. Op basis hiervan concludeert Demšar dat ANOVA geen geschikte test lijkt te zijn om de prestaties van meerdere classificatiemethoden te vergelijken. Hij adviseert daarom de non-parametrische Friedman test te hanteren, die in de volgende sectie kort besproken wordt.

5.1.1 Friedman Test

De Friedman test is een non-parametrische test die de te vergelijken algoritmen voor elke aparte dataset rangschikt op basis van hun prestaties. In geval van gelijke prestaties worden gemiddelde rangschikkingen toegewezen. Als r_i^j de rangschikking van het j-de van in totaal k algoritmen op de i-de van N datasets voorstelt, wordt de gemiddelde rankschikking van het j-de algoritme als volgt berekend: $R_j = \frac{1}{N}\sum_i r_i^j$. De nulhypothese bij de Friedman test stelt dat alle algoritmen gelijke prestaties vertonen en dus dat alle gemiddelde rankschikkingen R_j gelijk zijn. Om deze hypothese te testen wordt de Friedman statistiek

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right]$$

berekend welke verdeeld is volgens χ_F^2 met $k-1$ vrijheidsgraden indien N en k groot genoeg zijn ($N > 10$ en $k > 5$). Demšar verwijst echter naar het werk van Iman en Davenport (1980) die aantoonden dat χ_F^2 te conservatief is en een betere statistiek

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}$$

voorstelden die verdeeld is volgens de F-distributie met $k-1$ en $(k-1)(N-2)$ vrijheidsgraden. Indien deze statistiek uitwijst dat de nulhypothese wordt verworpen, kan de post-hoc Nemenyi test uitgevoerd worden. Deze wordt gebruikt voor het onderling vergelijken van de verschillende algoritmen. Een significant verschil in prestaties van twee classificatietechnieken

kan besloten worden indien hun gemiddelde rangschikkingen tenminste verschillen met het kritisch verschil (*critical difference*)

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

waarbij kritische waarde q_α gebaseerd is op de *Studentized range* statistiek gedeeld door $\sqrt{2}$.

5.2 Exploratief Onderzoek

De eerste stap in het onderzoek bestond erin de huidige Java-code van het ARUBAS-Scheffer algoritme aan te passen. Zo werd Java-code geschreven om het mogelijk te maken tijdens de instantiegebaseerde classificatiefase slechts met een subset van de meest gelijkaardige instanties te vergelijken, hetzij gebruik makend van een vast getal N of een percentage P. Nemen we een percentage dan wordt een ongekende instantie vergeleken met P percent van alle trainingsinstanties van een bepaalde klasse. Zo varieert het aantal instanties waarmee vergeleken wordt niet enkel per dataset maar ook per klasse. Kiezen we voor een vast getal N dan blijft dit getal gelijk voor elke dataset alsook voor alle klassen die de respectievelijke dataset bevat. In het vervolg van deze thesis duiden we het algoritme met vast getal N aan met NARUBAS en dit met percentage P als PARUBAS. De uitgevoerde aanpassingen aan de Java-code zijn terug te vinden in Bijlage A, listing A.1.

We hebben er voor gekozen om de gelijkenis tussen instanties te baseren op de gewogen *similarity measure*, die in het oorspronkelijke ARUBAS onderzoek als beste *fitness measure* naar voren kwam en besproken werd in Hoofdstuk 4.

5.2.1 Experiment 1

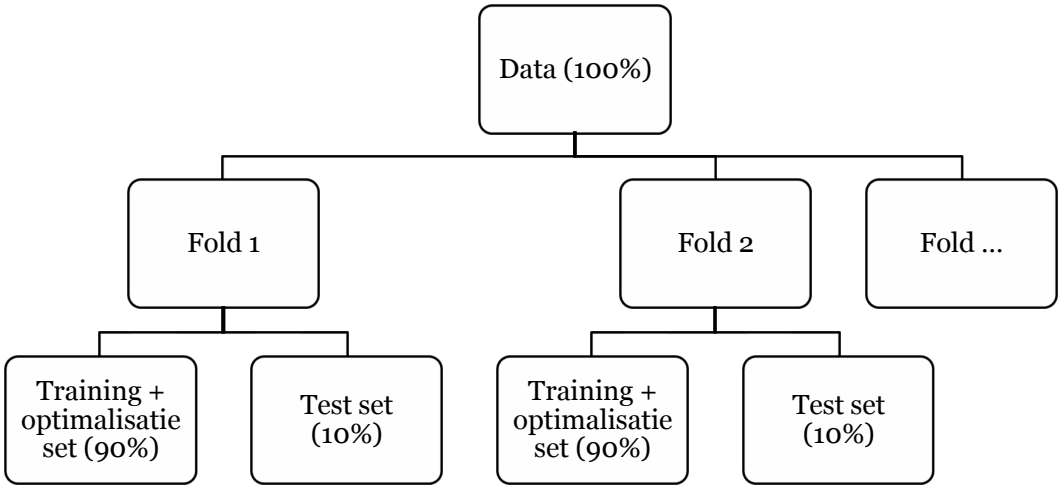
In eerste instantie hebben we dit onderzoek aangevat met het uitvoeren van enkele exploratieve experimenten, dit met twee objectieven voor ogen. Eerst en vooral was het nodig een inzicht te verkrijgen in het mogelijke effect van de aanpassing in het ARUBAS-Scheffer algoritme op de prestaties. Tevens wilden we onderzoeken of het mogelijk was de trainingsset te gebruiken als optimalisatie dataset om parameters N en P evenals het aantal CAR's te optimaliseren

en zo het algoritme parameter-vrij te houden. De optimalisatie van parameters mag namelijk niet uitgevoerd worden op testdata — de data die gebruikt worden voor de validatie (het meten van de prestaties) van het algoritme.

Figuur 5.1 geeft de gemaakte opdeling van de originele datasets in dit experiment weer. In eerste instantie wordt de dataset opgesplitst in 10 *folds* met *tenfold cross validation*. *Tenfold cross validation* is een statistische techniek die gebruikt wordt om de toekomstige performantie van een data mining algoritme op een onafhankelijke dataset te voorspellen. *Tenfold cross validation* deelt de originele dataset op in 10 gestratificeerde subsets, waarna telkens één deel gebruikt wordt als testset en de negen andere sets samen als trainingsset (dit stelt de eerste *fold* voor). Dit proces wordt tien maal uitgevoerd zodat elk van de 10 sets éénmaal als testset wordt gebruikt. Vervolgens wordt het algoritme voor elke *fold* getraind op de trainingsset en gevalideerd op de testset. Het gemiddelde van de 10 bekomen *accuracies* (één per testset bij elk van de 10 *folds*) wordt berekend. Deze gemiddelde *accuracy* stelt de geschatte accuraatheid voor die het algoritme verwacht wordt te bekomen op ongekende data. Bij de toepassing van *tenfold cross validation* worden alle instanties voor zowel training als validatie gebruikt. Elke instantie wordt tevens exact één maal gebruikt voor validatie. Hierin ligt de kracht van *tenfold cross validation*. Alle instanties worden namelijk gebruikt voor de training van het algoritme, maar validatie wordt nooit uitgevoerd op instanties die in een bepaalde *fold* voor training gebruikt werden.

Figuur 5.1 toont dat we in dit experiment de trainingsset voor elke *fold* ook als optimalisatieset willen gebruiken. Dit is echter enkel mogelijk indien de gemiddelde *accuracy*, verkregen over de trainingssets voor verschillende N/P - waarden en een verschillend aantal CAR's, een gelijkaardig verloop vertoont met de gemiddelde *accuracy* verkregen over de testsets. Indien dit het geval is zullen de N/P - CAR combinaties die de hoogste gemiddelde *accuracy* bekomen op de trainingsset ook de hoogste waarden bekomen op de testset. Zo zouden we deze trainingsset kunnen hanteren om optimale waarden voor N/P, alsook het aantal aan te leren CAR's te calculeren.

Het experiment werd praktisch als volgt opgezet. Het aantal aangeleerde CAR's lieten we



Figuur 5.1: Opdeling van de data voor optimaliseren en testen van het algoritme.

variëren tussen 100 en 1000 met een interval van 100. Bij elk regelinterval lieten we tevens ook N of P variëren. Bij het NARUBAS algoritme werd N gevarieerd tussen 1 en 56 met interval 5. Voor PARUBAS lieten we P variëren tussen 10% en 100% met interval 10%. Dit proces werd eerst uitgevoerd met de normale *tenfold cross validation* procedure. Training van het algoritme werd dus op de trainingsset uitgevoerd en validatie op de testset. Vervolgens werd hetzelfde experiment uitgevoerd, echter met gebruik van de trainingsset/optimalisatieset als testset. De Java-code voor dit experiment is terug te vinden in Bijlage A, listing A.2. Dit experiment werd voor zowel NARUBAS als PARUBAS toegepast op 5 UCI datasets Asuncion & Newman (2007). In volgende paragrafen bespreken we kort de resultaten van dit experiment.

NARUBAS

Figuur 5.2 toont de resultaten bekomen voor het NARUBAS algoritme op UCI dataset 'hayes-roth'. De gemiddelde *accuracy* wordt hier aangegeven op de y-as en de verscheidene geteste waarden voor N zijn uitgezet op de x-as. Per aantal aangeleerde CAR's werd een lijn geplot op de grafiek. Figuur 5.2 vertoont duidelijk een aanzienlijke variatie van de *accuracy*, afhankelijk van het aantal instanties waarmee vergeleken wordt tijdens de classificatiefase. Dit duidt op een potentiële verbetering van het algoritme, aangezien bij deze dataset de vergelijking met

minder instanties betere resultaten lijkt op te leveren dan wanneer we met een hoger aantal instanties vergelijken. De overige gegenereerde grafieken, terug te vinden in Bijlage B.1.1, tonen aan dat de *accuracy* vaak varieert met het aantal instanties waarmee vergeleken wordt, hoewel dit niet bij elke dataset zo is. Dit versterkt het vermoeden dat de aanpassing van het ARUBAS-Scheffer algoritme mogelijk betere prestaties kan opleveren.

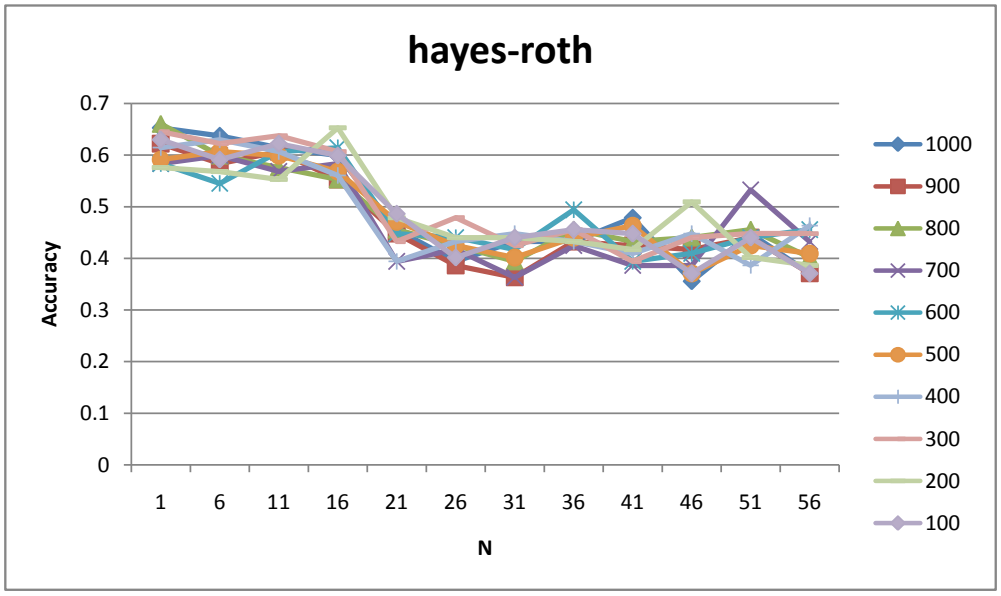
Figuur 5.3 toont een gelijkaardige grafiek, evenwel werden de resultaten, uitgezet op deze grafiek, verkregen met het aanleren van CAR's zowel als het classificeren van instanties op de trainingsset/optimalisatieset. Vergelijken we Figuur 5.2 met Figuur 5.3 nemen we waar dat de *accuracy* geen volledig overeenkomstig verloop vertoont. De grafieken in Bijlage B.1.1 tonen aan dat hetzelfde fenomeen zich bij verscheidene datasets voordoet. Indien we zodus de trainingsset zouden gebruiken om parameter N te optimaliseren, zal dit niet altijd de optimale waarden opleveren voor N, aangezien uit het experiment blijkt dat deze vaak verschillend zijn bij validatie op trainingsdata en op testdata.

Tengevolge concluderen we dat het niet mogelijk is de trainingsset te hanteren om per dataset een optimale waarde voor N te berekenen, welke we vervolgens zouden kunnen aanwenden om de uiteindelijke *classifier* voor de dataset te produceren.

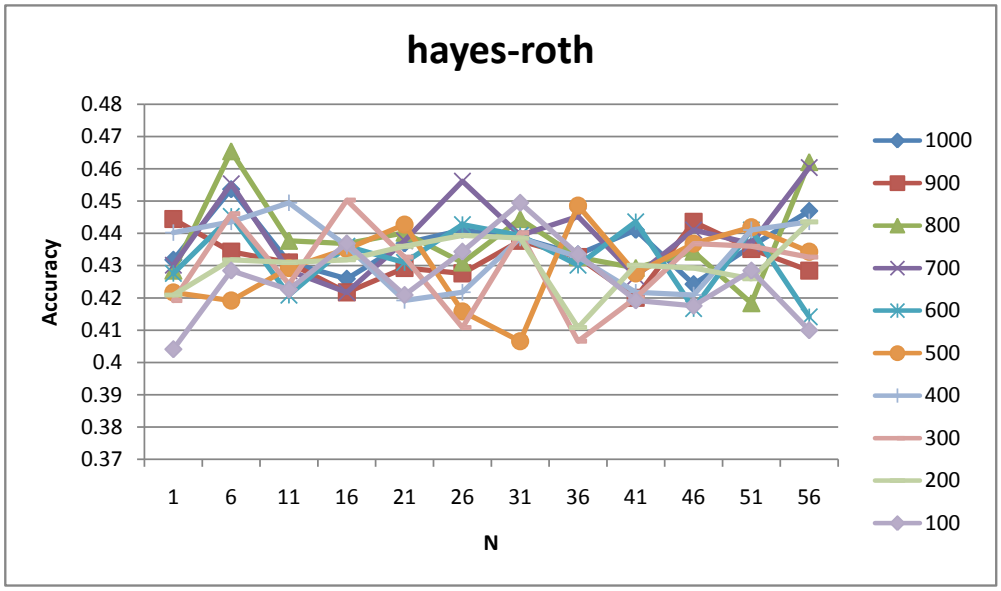
PARUBAS

De grafieken geproduceerd voor het PARUBAS algoritme zijn terug te vinden in Bijlage B.1.2. Deze grafieken vertonen gelijkaardige resultaten als die gevonden voor NARUBAS. De *accuracy* varieert bij alle geteste datasets voor verschillende P - waarden, wat weer duidt op een potentiële prestatieverbetering ten opzichte van ARUBAS-Scheffer. De *accuracies* bekomen op de trainingsset en testset vertonen evenwel ook hier vaak geen gelijkaardig verloop.

Ook voor PARUBAS kunnen we bijgevolg concluderen dat het niet mogelijk is de trainingsset te hanteren als optimalisatie dataset.



Figuur 5.2: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



Figuur 5.3: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.

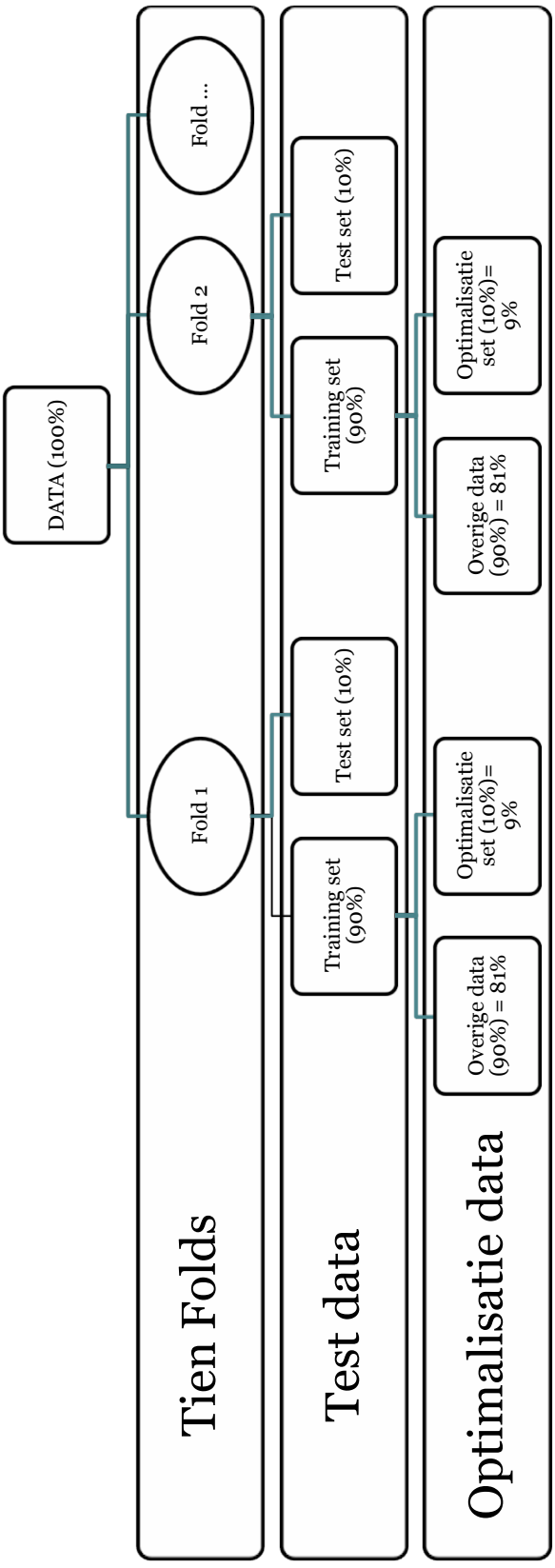
5.2.2 Experiment 2

Uit experiment 1 hebben we kunnen afleiden dat het optimaliseren van parameter N of P op basis van de volledige trainingsset niet het gewenste resultaat zou opleveren. Om dit probleem op te lossen, hebben we getracht de optimalisatie dataset op een andere wijze te produceren. Vervolgens werd nagegaan of de *accuracy* bij deze optimalisatieset een gelijkaardig verloop kent met die verkregen op de testdata. Twee methoden werden getest, welke uitgebeeld zijn in Figuur 5.4 en 5.5. In eerste instantie worden deze twee methoden besproken waarna de uitgevoerde experimenten en bekomen resultaten behandeld worden. Dit experiment werd enkel uitgevoerd voor PARUBAS aangezien het werd opgezet na de toetsing van Hypothese 1 (zie sectie 5.3.1, pagina 57), waarna beslist werd om enkel met PARUBAS verder te werken.

Methode 1

Figuur 5.4 geeft de samenstelling van de test dataset en optimalisatie dataset weer die werd onderzocht. Bij deze methode wordt eerst *tenfold cross validation* uitgevoerd op de volledige dataset. Hierdoor bekomen we voor 10 *Folds* een trainingsset en testset die telkens respectievelijk 90% en 10% van de volledige data uitmaken. Elk van de tien trainingssets wordt vervolgens opgedeeld in een dataset van 90% ('Overige data' benoemd op Figuur 5.4) en een optimalisatieset van 10%. Bij het optimaliseren van de parameters wordt het model getraind op de 'overige data' en vervolgens gevalideerd op de optimalisatieset. Zo verkrijgen we 10 optimalisatie trainingssets, telkens bestaande uit 81% van de oorspronkelijke dataset en 10 optimalisatie testsets telkens bestaande uit 9% van alle data. Per *fold* wordt zodus een optimalisatieset gecreëerd.

Het op deze wijze samenstellen van de optimalisatiedata zorgt ervoor dat de samenstelling ervan zeer sterk samenhangt met die van de testdata. Bij het gebruik van deze methode behouden we per *fold* tevens 90% van de data om regels op te leren, echter hebben we telkens slechts 81% van de data voorhanden om de parameters te optimaliseren. Aangezien de optimalisatiedata telkens samengesteld zijn uit enkel de instanties van de trainingsset gebruiken we nooit data uit de testset om onze parameters te optimaliseren en vermijden we zo overfitting.

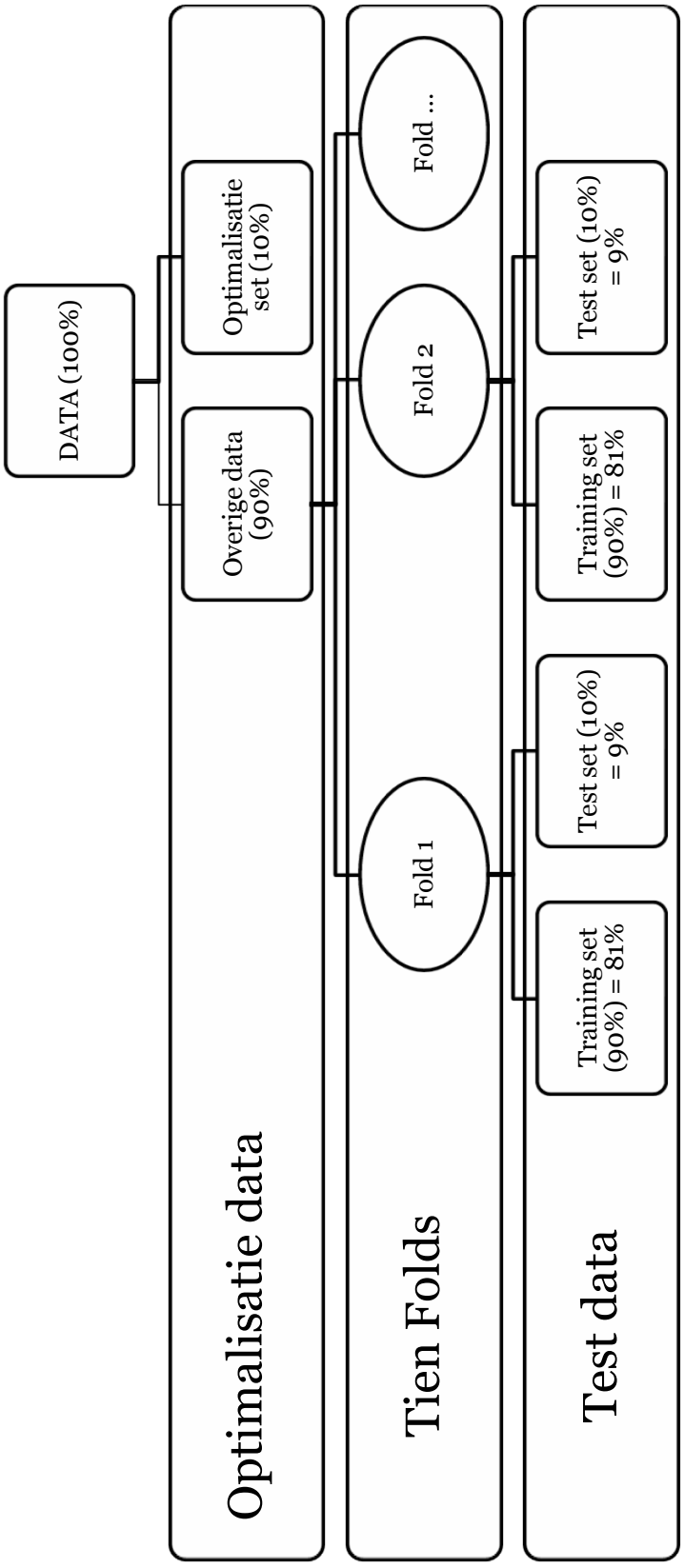


Figuur 5.4: Eerste methode voor de creatie van een optimalisatieset

Methode 2

Figuur 5.5 toont de tweede methode die onderzocht werd om de optimalisatiegegevens samen te stellen. Methode 2 deelt de dataset in eerste instantie op in een dataset ('Overige data') die 90% van alle data bevat en een optimalisatieset, samengesteld uit 10% van alle data. De 'overige data' wordt hierbij, net als bij methode 1, gebruikt als trainingsset tijdens de optimalisatiefase, waarna gevalideerd wordt op de optimalisatieset. Per dataset wordt zo slechts één optimalisatieset gecreëerd. Dit in tegenstelling tot bij methode 1 waarbij per *fold* een optimalisatieset werd geproduceerd. Vervolgens wordt *tenfold cross validation* toegepast op de 'overige data' om zo de testdata te genereren. Tien *folds* worden gecreëerd, bestaande uit telkens een trainingsset en testset die respectievelijk 90% en 10% van de data in de *fold* omvatten. De trainingsset en testset bevatten zo uiteindelijk per *fold* achtereenvolgens 81% en 9% van de volledige dataset.

Ook het op deze wijze produceren van de optimalisatiedata zorgt voor een zeer overeenkomstige samenstelling ervan met die van de testdata. De instanties in de optimalisatieset worden tevens nooit gebruikt voor het samenstellen van de testset waardoor overfitting vermeden wordt.



Figuur 5.5: Tweede methode voor de creatie van een optimalisatieset

Experimentele opzet en resultaten - Methode 1

De Java code in Bijlage A listing A.3 bevat de code die gebruikt werd voor het testen van methode 1. Het experiment werd uitgevoerd op 10 UCI datasets (Asuncion & Newman, 2007). De praktische uitvoering van het experiment wordt in volgende paragrafen toegelicht.

Net als bij Experiment 1 werd per dataset het aantal CAR's gevarieerd tussen 100 en 1000 met interval 100. Per regelinterval werd parameter P tevens gevarieerd tussen 10% en 100% met interval 10%. De gemiddelde *accuracies* werden vervolgens berekend met *tenfold cross validation*. We genereerden voor elke *fold* in Figuur 5.4 het classificatiemodel uit de trainingsset om vervolgens de *accuracy* te berekenen op de testset. Het gemiddelde van de tien verkregen *accuracies* stelt de *accuracy* verkregen op de dataset voor. Het verloop van de *accuracy* per P - waarde, werd voor elk regelinterval uitgezet op een grafiek. Dit voor 10 UCI datasets. Figuur 5.6 toont zulke grafiek voor UCI dataset 'balance-scale'.

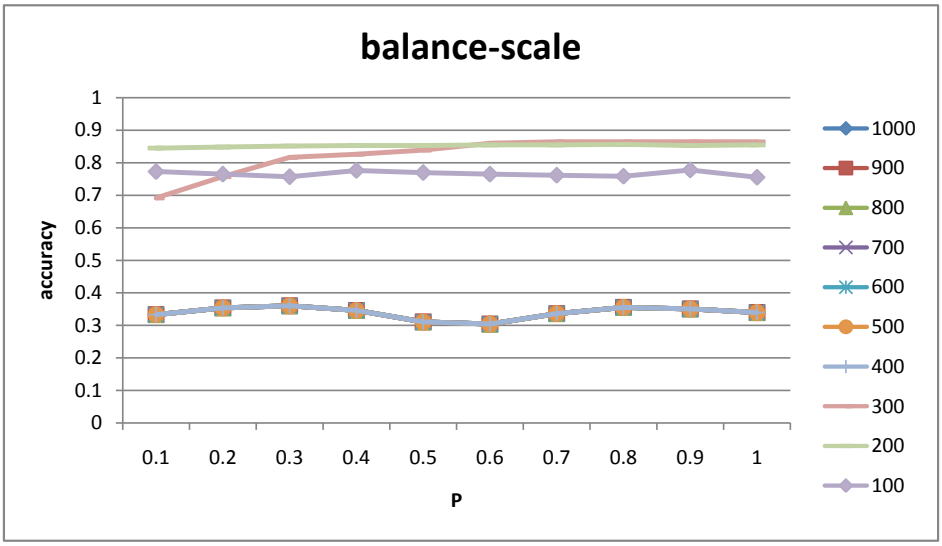
Ook de *accuracies* over verscheidene P - waarden, verkregen op de optimalisatieset in Figuur 5.6 werden per regelinterval uitgezet op grafieken. Figuur 5.7 toont zulke grafiek voor UCI dataset 'balance-scale'.

We vergelijken de twee grafieken per dataset met het objectief te onderzoeken of de *accuracies* een gelijkaardig verloop vertonen. Bij Figuur 5.6 en 5.7 lijkt dit min of meer het geval te zijn. De figuren in bijlage B.2 tonen echter aan dat bij enkele datasets nog steeds discrepanties bestaan tussen het verloop van de *accuracy* bij optimalisatie- en testdata.

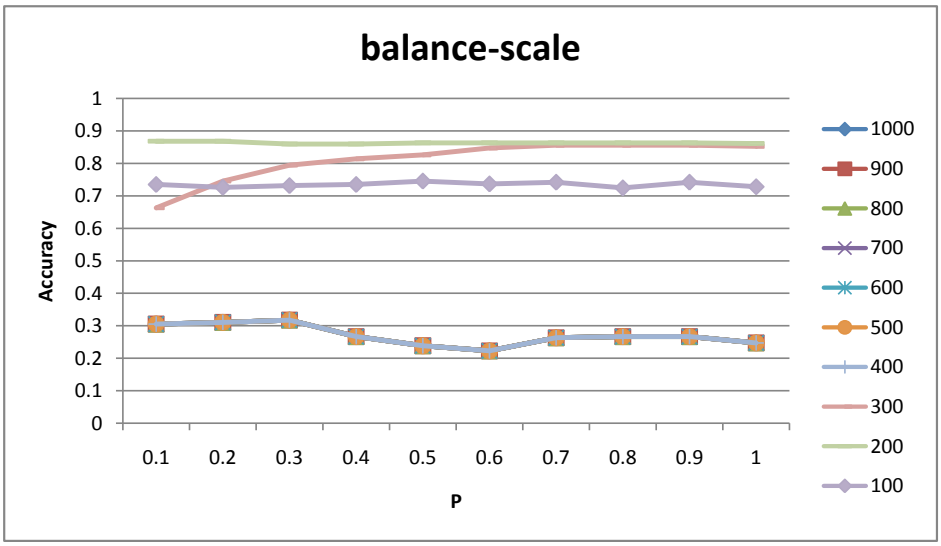
Concluderend kunnen we stellen dat de optimalisatiedata, samengesteld met methode 1, niet in alle gevallen representatief zijn ten opzichte van de testdata. Het gebruik van deze optimalisatiedata zou bijgevolg niet altijd de werkelijke optimale waarden voor parameter P aanduiden.

Experimentele opzet en resultaten - Methode 2

De Java code in Bijlage A listing A.4 bevat de code die gebruikt werd voor het testen van methode 2, beschreven in vorige sectie. Het experiment werd uitgevoerd op 5 UCI datasets



Figuur 5.6: Accuracies verkregen met tenfold cross validation toegepast op UCI dataset 'balance-scale', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



Figuur 5.7: Accuracies verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4) toegepast op UCI dataset "balance-scale", het aantal regels aan geleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

(Asuncion & Newman, 2007). De praktische uitvoering van het experiment wordt in volgende paragrafen toegelicht.

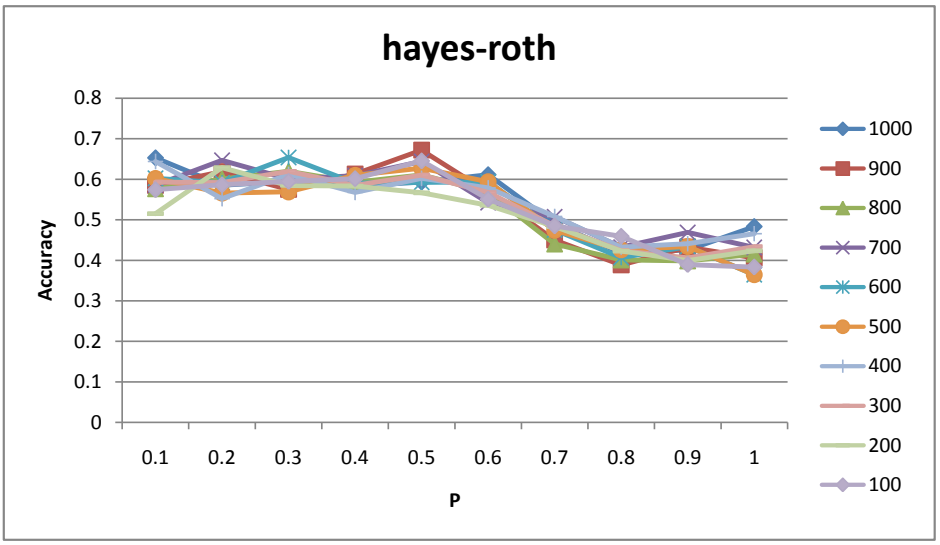
Per dataset werd het aantal CAR's gevarieerd tussen 100 en 1000 met interval 100. Per regelinterval werd parameter P tevens gevarieerd tussen 10% en 100% met interval 10%. Vervolgens werden voor de verschillende parameter-settings gemiddelde *accuracies* berekend op de testdata. Per *fold* werd zodus getraind op de trainingsset en *accuracies* berekend op de testset zoals aangegeven op Figuur 5.5. Het gemiddelde van de tien bekomen *accuracies* wordt berekend en vormt de voorspelde *accuracy* op een bepaalde dataset. Het verloop van de *accuracy* per P - waarde werd voor elk regelinterval uitgezet op een grafiek. Figuur 5.8 toont zulke grafiek voor UCI dataset 'hayes-roth'.

Ook de *accuracies* over verscheidene P - waarden, verkregen op de optimalisatieset (zie Figuur 5.5), werden per regelinterval uitgezet op grafiek. Figuur 5.9 toont zulke grafiek voor UCI dataset 'hayes-roth'.

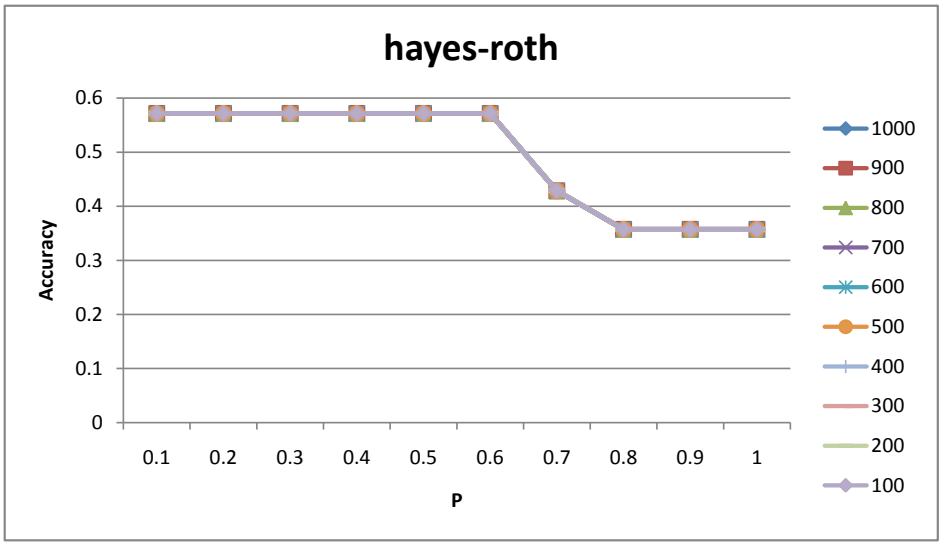
We vergelijken de twee grafieken per dataset met het objectief te onderzoeken of de *accuracies* een gelijkaardig verloop vertonen. Bij Figuur 5.8 en 5.9 lijkt dit min of meer het geval te zijn. De figuren in bijlage B.3 tonen echter aan dat bij enkele datasets nog steeds discrepanties bestaan tussen het verloop van de *accuracy* bij optimalisatie- en testdata.

Concluderend kunnen we stellen dat de optimalisatiedata, samengesteld met methode 2, niet in alle gevallen representatief zijn ten opzichte van de testdata. Het gebruik van deze optimalisatiedata zou bijgevolg niet altijd de werkelijke optimale waarden voor parameter P aanduiden. Uit de bestudering van de grafieken horende bij methode 1 en 2 kunnen we verder ook concluderen dat methode 1 meer representatieve optimalisatiedata produceert dan methode 2.

Een mogelijke verklaring voor de suboptimale resultaten van beide geteste methoden voor de samenstelling van een optimalisatie dataset vinden we in de geringe omvang van enkele UCI datasets. Dit veroorzaakt potentieel te grote discrepanties tussen de samenstelling van optimalisatie - en testdata bij bepaalde datasets.



Figuur 5.8: Accuracies verkregen op de testdata samengesteld met Methode 2 (zie Figuur 5.5), toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



Figuur 5.9: Accuracies verkregen op de optimalisatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5) toegepast op UCI dataset 'balance-scale', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

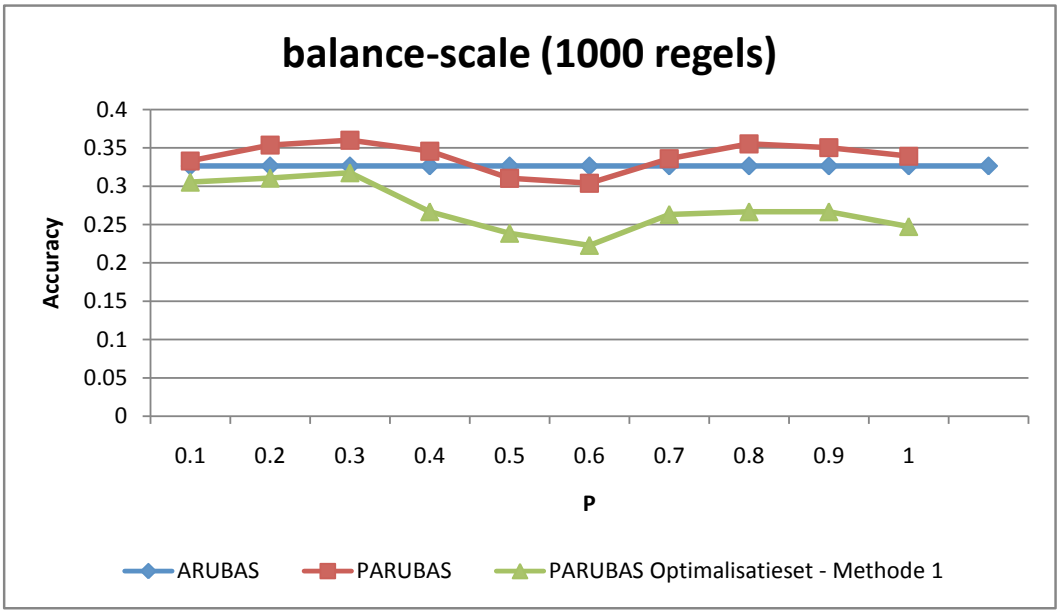
5.2.3 Grafische analyse

Ter afronding van het exploratief onderzoek hebben we een grafische analyse uitgevoerd om een beter inzicht te verkrijgen in de gegenereerde experimentele gegevens. We zetten zo de behaalde *accuracies* van het PARUBAS algoritme bij een vast aantal regels en een variërend percentage (dit toegepast op zowel de test- als de optimalisatiedata) op een grafiek en vergelijken dit met de *accuracy* bekomen met het originele ARUBAS-Scheffer algoritme. De optimalisatiedata werden hierbij gecreëerd met toepassing van methode 1, besproken in vorige sectie, aangezien deze de meest representatieve optimalisatiesets produceert.

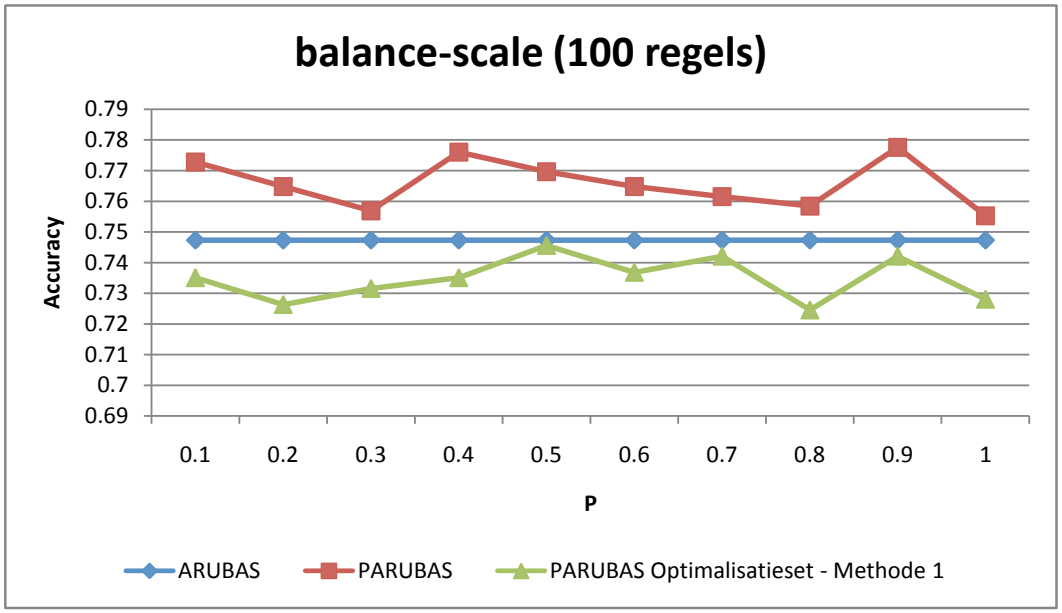
Figuur 5.10 en 5.11 indiceren dat PARUBAS potentieel betere prestaties levert dan ARUBAS-Scheffer. De *accuracy* van PARUBAS bij dataset 'balance-scale' waarbij 1000 regels werden aangeleerd (Figuur 5.10) ligt namelijk bij 7 van de 10 geteste percentages hoger dan of gelijk aan die bekomen met ARUBAS-Scheffer. Indien voor dezelfde dataset 100 regels aangeleerd worden (Figuur 5.11), presteert PARUBAS zelfs voor elke geteste P - waarde beter [1]. In Bijlage B.4 merken gelijkaardige resultaten op bij andere datasets. Ongeacht of we 100 of 1000 regels aanleren scoort PARUBAS vaak beter dan ARUBAS-Scheffer. Op Figuur 5.10 nemen we ook een vrij gelijk verloop waar tussen PARUBAS toegepast op de testdata en de optimalisatiedata. Nemen we hier de beste waarde voor P gebaseerd op de gegevens van de optimalisatieset dan zal dit ook op de testset de hoogste *accuracy* opleveren. Figuur 5.11 daarentegen vertoont een grotere discrepantie in verloop, waardoor de optimale waarde van P, geïndiceerd bij de optimalisatieset, niet de beste *accuracy* waarde oplevert bij de testdata.

Uit dit exploratief onderzoek concluderen we dat PARUBAS en NARUBAS een potentiële verbetering met zich meebrengen ten opzichte van het originele ARUBAS-Scheffer algoritme. Om een optimale verbetering te bekomen, alsook het algoritme parameter-vrij te houden zou de optimalisatie van parameters N of P wenselijk zijn. Het produceren van een representatieve dataset voor optimalisatie-doeleinden blijkt echter moeilijk te zijn bij enkele van de UCI

[1] Figuur 5.11 geeft aan dat PARUBAS waarbij 100% van de instanties gebruikt worden beter presteert als ARUBAS, dit terwijl ARUBAS identiek is aan PARUBAS-100%. Deze inconsequentie wordt veroorzaakt door het random toewijzen van een klasse ingeval van gelijke scores voor twee of meerdere klassen.



Figuur 5.10: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset 'balance-scale' waarbij 1000 regels aangeleerd werden.



Figuur 5.11: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset 'balance-scale' waarbij 100 regels aangeleerd werden.

datasets. De gegenereerde data en hieruit geproduceerde grafieken tonen echter aan dat PARUBAS en NARUBAS vaak bij verscheidene waarden voor parameters N of P beter presteren dan ARUBAS-Scheffer. Op basis van deze voorlopige conclusies formuleren we in volgende sectie enkele hypothesen die vervolgens statistisch onderzocht worden.

5.3 Hypothesetoetsing

In deze sectie worden de hypothesen geformuleerd die getoetst werden met het oog een antwoord op de deelvragen geformuleerd in sectie 4.3.1 te verkrijgen. Bij de toetsing van de hypothesen werd de methodologie, uitgelijnd in sectie 5.1, toegepast.

5.3.1 Hypothese 1

De toetsing van de eerste hypothese tracht een antwoord te vinden op deelvraag ***i*** , welke geformuleerd werd in Hoofdstuk 4 sectie 4.3.1, pagina 38, waarbij we willen onderzoeken of N procentueel of absoluut bepaald moet worden. De nulhypothese en alternatieve hypothese worden als volgt geformuleerd.

H0 : Bij éénzelfde aantal regels aangeleerd produceert NARUBAS dezelfde resultaten als PARUBAS.

H1 : Bij éénzelfde aantal regels aangeleerd produceert NARUBAS niet dezelfde resultaten als PARUBAS.

Hypothese 1 werd getest door NARUBAS met verschillende waarden voor N en PARUBAS met verscheidene waarden voor P uit te voeren op 15 UCI datasets (Asuncion & Newman, 2007). Voor beide algoritmen werden 200 CAR's aangeleerd. Het gemiddelde van de behaalde *accuracies* voor de verschillende N alsook P waarden werd per dataset berekend, welke terug te vinden zijn in Tabel 5.1. Gemiddelde *accuracies* werden berekend voor N-waarden: $(150, 100, 50, 20, 10)$ en P-waarden: $(0.7, 0.5, 0.3, 0.2, 0.1)$. Calculatie van deze gemiddelde *accuracies* werd als volgt uitgevoerd. Voor elk van de N/P - waarden werden met *tenfold cross validation* gemiddelde *accuracies* berekend voor elke dataset. Vervolgens werd per dataset

het gemiddelde genomen van alle *accuracies* bekomen over de geteste N/P - waarden. Deze gemiddelde *accuracy* stelt de score van NARUBAS, respectievelijk PARUBAS voor op een bepaalde dataset. De Java-code gebruikt voor het uitvoeren van dit experiment is terug te vinden in Bijlage A, listing A.5.

Tabel 5.1 toont de berekende χ^2 en F_F statistieken die aanwijzen dat de nulhypothese, op een significantieniveau van 10%, niet verworpen kan worden. Verder werden ook andere waardecombinaties voor N en P uitgetest. Deze leverden telkens zeer gelijkaardige resultaten. We concluderen vervolgens dat geen significant verschil bestaat tussen de prestaties van NARUBAS en PARUBAS.

We hebben geopteerd om bij verdere experimenten met een percentage als argument te werken, omdat dit intuïtief gezien logischer is. Zo kunnen we het percentage ook telkens tot 100% laten oplopen, waarbij het algoritme hetzelfde functioneert als ARUBAS-Scheffer. Indien we P per dataset optimaliseren en onze optimalisatieset volledig representatief is (wat uit sectie 5.2 echter niet zo bleek te zijn) verkrijgen we telkens de beste *classifier*, hetzij PARUBAS met een percentage kleiner dan 100%, hetzij PARUBAS 100%, zijnde het originele ARUBAS-Scheffer algoritme.

5.3.2 Hypothese 2

De toetsing van hypothese 2 heeft als doel na te gaan of de prestaties van PARUBAS, bij het aanleren van een willekeurig aantal regels betere prestaties oplevert dan ARUBAS-Scheffer. Tevens onderzoeken we welke waarde voor P de beste resultaten oplevert om zo een antwoord te bekomen op deelvraag *ii* (zie Hoofdstuk 4, sectie 4.3.1 pagina 38). De nulhypothese en alternatieve hypothese worden als volgt geformuleerd.

H₀ : Bij éénzelfde, random gegenereerde aantal regels aangeleerd, produceert PARUBAS dezelfde resultaten als ARUBAS-Scheffer.

H₁ : Bij éénzelfde, random gegenereerde aantal regels aangeleerd, produceert PARUBAS niet dezelfde resultaten als ARUBAS-Scheffer.

Tabel 5.1: NARUBAS versus PARUBAS

Datasets	*Accuracies* PARUBAS	*Accuracies* NARUBAS	*Ranks* PARUBAS	*Ranks* NARUBAS
balance-scale	0.85859703	0.864004096	2	1
car	0.721995564	0.717021777	1	2
cmc	0.444024637	0.444024637	1.5	1.5
ecoli	0.769893048	0.708467023	1	2
glass	0.719480519	0.696753247	1	2
haberman	0.740344086	0.741612903	2	1
iris	0.932	0.94	2	1
mammographic_masses	0.804544674	0.802244416	1	2
monk1	0.564318182	0.616688312	2	1
monk2	0.650907104	0.65557377	2	1
monk3	0.799961039	0.843279221	2	1
pima-indians-diabetes	0.746664388	0.74475393	1	2
tae	0.49	0.444166667	1	2
tic-tac-toe	0.966622807	0.966622807	1.5	1.5
yeast	0.584647198	0.582232904	1	2
Average	0.719600018	0.717829714	1.46666667	1.53333333
Hypothesetoetsing	Waarde	significantie		
χ^2	0.066666667	0.796253415		
F_F	0.0625	0.806218498		

Tabel 5.2: Gemiddelde rankschikking op 19 UCI datasets voor ARUBAS-Scheffer, PARUBAS met verscheidene vaste P-waarden en PARUBAS opt. Elk algoritme werd tien maal gerund met verschillende random gegenereerde waarden voor het aantal regels aangeleerd.

	ARUBAS	PA - 70%	PA - 50%	PA - 30%	PA - 10%	PA - 1%	PA - opt.
Avg Rank	4.55	4.09	3.85	3.64	4.2	4.54	3.13

De toetsing van hypothese 2 werd uitgevoerd door het tien keer toepassen van ARUBAS-Scheffer, met telkens een andere waarde voor het aantal gegenereerde CAR's, op 19 UCI datasets (Asuncion & Newman, 2007). De toegepaste waarden voor deze CAR-parameter werden random gegenereerd en bedroegen: (326, 27, 156, 948, 299, 616, 399, 445, 852, 354).

Ook PARUBAS werd voor eenzelfde CAR-waarden toegepast op de 19 datasets. Per regelinterval werden tevens volgende waarden voor parameter P getest: (0,70; 0,50; 0,30; 0,10; 0,01). Zo werden algoritmen: PA-70%, PA-50%, PA-30%, PA-10%, PA-1% verkregen.

Tenslotte werd ook PARUBAS met optimale P-waarden per dataset (PA-opt.) uitgevoerd op de 19 datasets. De berekening van deze optimale P-waarden wordt in volgende paragraaf verduidelijkt. De Java-code horende bij dit experiment is terug te vinden in Bijlage A, listing A.6.

Zoals besproken in sectie 5.2.1 werd geen representatieve optimalisatie dataset gevonden met de beproefde technieken. Echter hebben we verder geëxperimenteerd met methode 1 uit Figuur 5.4, waarbij in plaats van 10 CV, 5 CV tot en met 9 CV werd toegepast. Gegenereerde data en bijbehorende grafieken toonden aan dat geen van deze methoden ideale optimalisatiesets produceren. Applicatie van 6 CV leverde evenwel de meest representatieve optimalisatieset op van alle geteste methoden. Voor het berekenen van een optimale P-waarde per dataset hebben we zodus methode 1 uit Figuur 5.4 met 6 CV toegepast en op basis hiervan optimale P-waarden per dataset gecalculeerd. De optimale P-waarden per dataset werden berekend met behulp van de Java-code in Bijlage A listing A.7.

De gemiddelde rangschikking van de verscheidene algoritmen over de 19 UCI datasets wordt getoond in Tabel 5.2. PARUBAS-opt. is hierbij gemiddeld genomen ongeveer de derde beste

classificatietechniek en gaat alle andere geteste algoritmen vooraf. De Friedman test leverde een χ_F^2 waarde van 63,49 met 5 vrijheidsgraden en een F_F waarde van 11,15 met 5 en 1.128 vrijheidsgraden op, welke beide een p-waarde kleiner dan 0,01 hebben. Dit resultaat indiceert dat er significante verschillen in accuraatheid bestaan tussen de zeven geteste classificatiealgoritmen.

Vervolgens werd de Nemenyi test toegepast om onderlinge significante verschillen tussen de verschillende classificatietechnieken te identificeren. De resultaten worden grafisch gerepresenteerd in Figuur 5.12. De AC-technieken die minder van elkaar verschillen dan de *critical difference (CD)* waarde en dus niet significant van elkaar verschillen bij $p = 0.1$, zijn met elkaar verbonden. De Nemenyi test wijst aan dat PARUBAS-opt., bij een willekeurig aantal aangeleerde regels, significant beter presteert dan de andere geteste *classifiers* met uitzondering van PARUBAS-30%. ARUBAS presteert significant slechter dan PARUBAS-opt., PA-30% en PA-50%.

Hiermee wordt aangetoond dat bij een willekeurig aantal aangeleerde regels en de vergelijking met een willekeurig percentage van de instanties tijdens de classificatiefase, dat PARUBAS minstens even goed presteert als ARUBAS. PARUBAS-opt., PARUBAS-30% en PARUBAS-50% behalen tevens significant betere resultaten dan het ARUBAS-Scheffer algoritme.

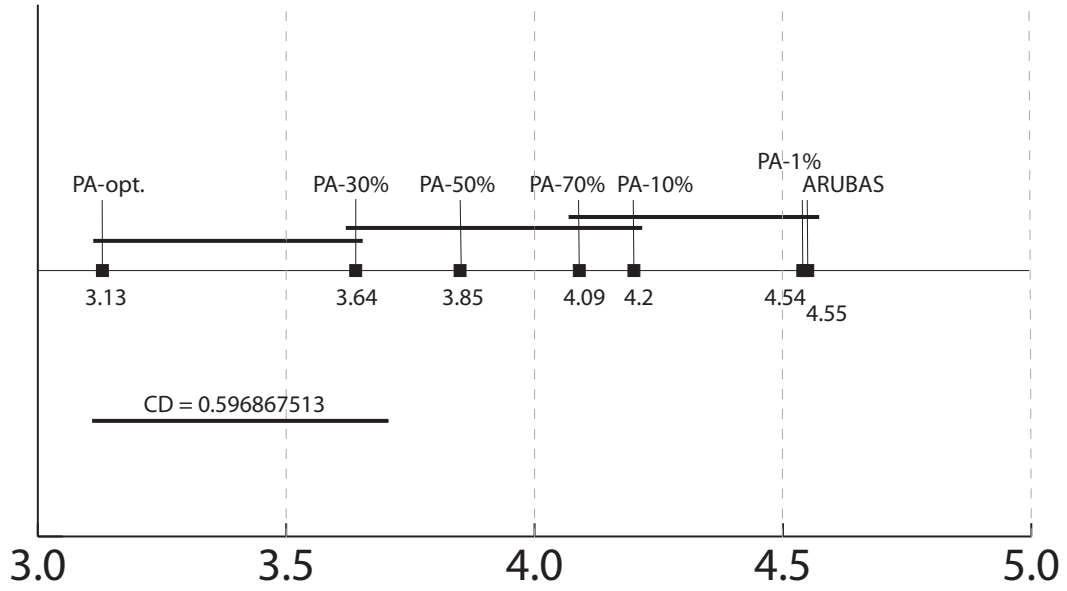
5.3.3 Hypothese 3

Hypothese 3

Hypothese 3 wordt getoetst om na te gaan of de prestaties van algoritme PARUBAS-opt. bij het aanleren van een willekeurig aantal regels, gelijke of betere prestaties oplevert als bestaande AC-algoritmen.

H₀ : Bij éénzelfde aantal regels aangeleerd, produceert PARUBAS-opt. dezelfde resultaten als bestaande AC-technieken.

H₁ : Bij éénzelfde aantal regels aangeleerd, produceert PARUBAS-opt. niet dezelfde resultaten als bestaande AC-technieken.



Figuur 5.12: Nemenyi Test - Hypothese 2

Bij de toetsing van hypothese 3 werden de resultaten bekomen door het toepassen van PARUBAS en ARUBAS-Scheffer op 19 UCI datasets voor 10 random gegenereerde waarden (dezelfde als gebruikt voor het testen van Hypothese 2) als instelling voor het aantal CAR's. Voor PARUBAS werden per dataset en per CAR-waarde de optimale waarden voor P berekend. Deze optimale waarden werden net als bij de toetsing van vorige hypothese bekomen op basis van een optimalisatieset, gegenereerd door middel van toepassing van methode 1 uit Figuur 5.4 met 6 CV geïmplementeerd.

De gemiddelde rangschikking van de verscheidene algoritmen over de 19 UCI datasets wordt getoond in Tabel 5.3. Met een gemiddelde rangschikking van ongeveer 1,7 produceert het C4.5 algoritme hier duidelijk de beste resultaten. Toepassing van de Friedman test leverde een χ_F^2 waarde van 115,25 op met 3 vrijheidsgraden en een F_F waarde van 47,90 met 3 en 564 vrijheidsgraden, welke beide een p-waarde kleiner dan 0,01 hebben. Uit de resultaten van de Friedman test concluderen we dat er significante verschillen bestaan tussen de geteste classificatietechnieken.

Figuur 5.13 geeft een grafische voorstelling van de resultaten van de Nemenyi test weer. AC-

Tabel 5.3: Gemiddelde rankschikking op 19 UCI datasets voor ARUBAS-Scheffer, PARUBAS opt., 1R en C4.5. ARUBAS-Scheffer en PARUBAS werden tien maal gerund met verschillende random gegenereerde waarden voor het aantal regels aangeleerd.

	C4.5	PARUBAS opt.	ARUBAS	1R
Avg Rank	1.73	2.41	2.86	3.00

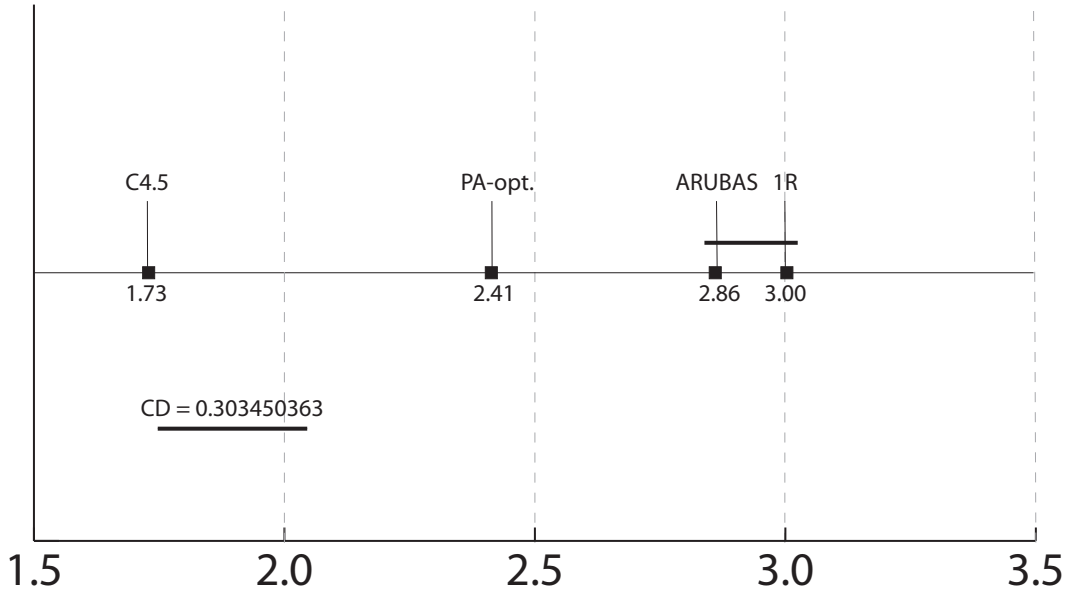
technieken waarbij het verschil tussen de gemiddelde rangschikkingen groter is dan de CD-waarde zijn met elkaar verbonden. Deze AC-technieken zijn niet significant verschillend bij $p = 0,1$. De resultaten van de Nemenyi test tonen aan dat PARUBAS-opt., met een random aantal aangeleerde regels, significant betere resultaten behaalt dan ARUBAS-Scheffer en 1R. Beslissingsboom algoritme C4.5 presteert significant beter dan de drie andere AC-algoritmen. Het feit dat ARUBAS-Scheffer evenals PARUBAS-opt. hier significant minder presteren dan C4.5 kan verklaard worden met het gegeven dat bij de toepassing van beide algoritmen random waarden voor het aantal regels gekozen werden. Gegevens uit het verkennend onderzoek toonden aan dat het aantal regels dat aangeleerd wordt, tijdens de regelgeneratie-fase van het algoritme, bij de meeste datasets een significante invloed uitoefent op de accuraatheid van het algoritme.

5.3.4 Hypothese 4

Tenslotte willen we nagaan hoe PARUBAS, met geoptimaliseerde parameters per dataset, presteert ten opzichte van ARUBAS-Scheffer met het optimale aantal regels ingesteld, alsook ten opzichte van andere bestaande classificatietechnieken.

$\boldsymbol{H_0}$: Alle geteste classificatietechnieken, met inclusie van ARUBAS-Scheffer met geoptimaliseerde CAR-parameter en PARUBAS-opt., leveren gelijke prestaties.
$\boldsymbol{H_1}$: Niet alle geteste classificatietechnieken, met inclusie van ARUBAS-Scheffer met geoptimaliseerde CAR-parameter en PARUBAS-opt., leveren gelijke prestaties.

We hebben aangetoond dat, bij een willekeurig aantal aangeleerde regels, PARUBAS met vast



Figuur 5.13: Nemenyi Test - Hypothese 3

percentage 50%, 30% en PARUBAS-opt. significant beter presteren als ARUBAS-Scheffer. In een laatste fase van het onderzoek wordt getest hoe PARUBAS met optimale parameters per dataset (zowel voor P als de CAR-parameter) presteert ten opzichte van ARUBAS-Scheffer, met geoptimaliseerde CAR-parameter per dataset, evenals ten opzichte van andere bestaande classificatietechnieken.

De hypothesetoetsing werd als volgt uitgevoerd. In eerste instantie werd per dataset het optimale aantal regels voor het ARUBAS-Scheffer algoritme berekend. Deze waarden werden gebruikt bij het berekenen van de gemiddelde *accuracies* voor ARUBAS-Scheffer. Om de PARUBAS-opt. *classifier* op te bouwen, werden per dataset de optimale CAR-parameter-waarden die voor ARUBAS-Scheffer berekend werden, ingesteld. Op basis van deze settings werden per dataset de optimale P-waarden berekend. PARUBAS werd vervolgens toegepast met de per dataset gecalculeerde optimale parameter-waarden. De pseudo-code van PARUBAS-opt. zoals deze werd uitgevoerd voor deze hypothesetoetsing wordt gegeven door Algorithm 1.

ARUBAS-Scheffer en PARUBAS-opt. werden vergeleken met 'geprunede' en 'ongeprunede'

Algorithm 1 PARUBAS-opt.

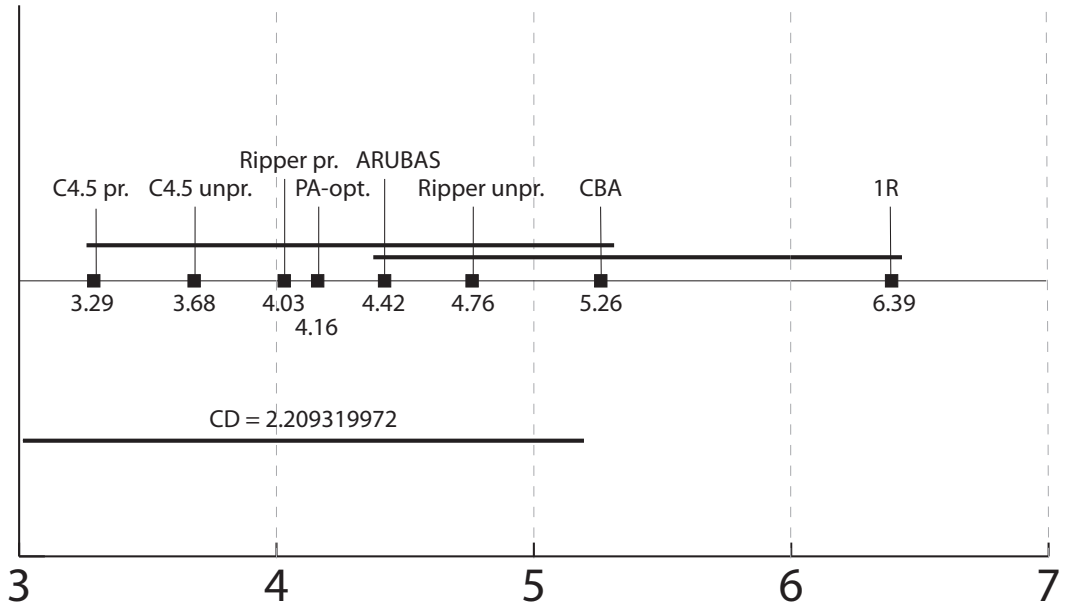
1: $maxCars = optNumCars = 1000$
2: $maxAccuracy = 0$
3: **repeat**
4: Leer $numCars$ CAR's uit de trainingsdata met behulp van Predictive Apriori
5: Voer het ARUBAS raamwerk uit en evalueer de bekomen *accuracy* op de trainingsdata
6: **if** *current accuracy* $>$ $maxAccuracy$ **then**
7: $maxAccuracy = current\ accuracy$
8: $optNumCars = numCars$
9: **end if**
10: $numCars = numCars$ -10
11: **until** $numCars = 0$
12: $P = optP = 1$
13: $maxAccuracy = 0$
14: Leer $optNumCars$ CAR's aan met behulp van Predictive Apriori
15: **repeat**
16: Classificeer ongekende instanties uit de optimalisatieset met behulp van P percent van de trainingsinstanties
17: **if** *current accuracy* $>$ $maxAccuracy$ **then**
18: $maxAccuracy = current\ accuracy$
19: $optP = P$
20: **end if**
21: $P = P$ - 0,01
22: **until** $P = 0{,}01$
23: Leer $optNumCars$ CAR's uit de trainingsdata met behulp van Predictive Apriori
24: Classificeer instanties uit de testset met behulp van $optP$ percent van de trainingsinstanties
25: Evalueer de *accuracy* op de testdata

versies van classificatietechnieken Ripper, C4.5, CBA en 1R. De gemiddelde rangschikkingen van de algoritmen over 19 UCI datasets (Asuncion & Newman, 2007) worden getoond in Tabel 5.4. Met een gemiddelde rangschikking van ongeveer 3,3 produceert het C4.5 *pruned* algoritme de beste resultaten. Toepassing van de Friedman test leverde een χ_F^2 waarde van 21,28 op met 7 vrijheidsgraden en een F_F waarde van 3,43 met 7 en 17 vrijheidsgraden, welke beide een p-waarde kleiner dan 0,01 hebben. We concluderen hieruit dat significante verschillen bestaan tussen de verscheidene classificatietechnieken.

Tabel 5.4: Gemiddelde rangschikking op 19 UCI datasets voor ARUBAS-Scheffer, PARUBAS opt. en 6 andere classificatietechnieken.

	C4.5 pr.	C4.5 unpr.	Ripper pr.	PARUBAS opt.	ARUBAS	Ripper unpr.	CBA	1R
Avg Rank	3.29	3.68	4.03	4.16	4.42	4.76	5.26	6.39

Figuur 5.14 geeft een grafische voorstelling van de resultaten van de Nemenyi test weer. AC-technieken waarbij het verschil tussen de gemiddelde rangschikkingen groter is dan de CD-waarde zijn met elkaar verbonden. Deze AC-technieken zijn niet significant verschillend bij $p = 0,1$. De resultaten van de Nemenyi test tonen aan dat PARUBAS-opt. wel een hogere rangschikking behaalt dan ARUBAS-Scheffer, echter is dit verschil niet significant op 10%. Over alle geteste classificatietechnieken behaalt PARUBAS-opt. de vierde plaats, net voor ARUBAS en na C4.5 *pruned*, C4.5 *unpruned* en RIPPER *pruned*. De verschillen tussen deze algoritmen zijn echter niet significant op 10%. Wat verder opvalt is dat ARUBAS-Scheffer over de 19 UCI datasets niet significant beter presteert als 1R. PARUBAS-opt. daarentegen behaalt wel significant betere resultaten als 1R op significantieniveau 10%.



Figuur 5.14: Nemenyi Test - Hypothese 4

5.4 Conclusies

In deze masterproef werden twee uitbreidingen van de associatieve classificatietechniek ARUBAS voorgesteld, NARUBAS en PARUBAS. Na een uitgebreid exploratief onderzoek werden vier hypothesen getoetst aangaande de prestaties van NARUBAS en PARUBAS ten opzichte van elkaar, ARUBAS-Scheffer en andere bestaande classificatietechnieken.

Het uitgevoerde praktijkonderzoek toonde aan dat geen significante prestatieverschillen bestaan tussen NARUBAS en PARUBAS. Voor deelvraag ***i*** kan dus geen eenduidig antwoord geformuleerd worden. Op basis van andere overwegingen werd echter beslist verder onderzoek te focussen op PARUBAS.

Bij eenzelfde, random gegenereerde aantal regels aangeleerd, bekwam PARUBAS met geoptimaliseerde P-waarden per dataset (PARUBAS-opt.), alsook PARUBAS met $P = 30\%$ en $P = 50\%$ significant betere resultaten als ARUBAS-Scheffer. Indien één P-waarde gekozen moet worden voor alle datasets wordt zodus best geopteerd voor 30% of 50%, waarmee het antwoord op deelvraag ***ii*** gegeven wordt. Optimalisatie van parameter P per dataset levert evenwel nog betere resultaten op. Het exploratief onderzoek toonde echter aan dat het pro-

duceren van representatieve optimalisatiedata voor sommige datasets moeilijk bleek te zijn. Hierdoor werd bij het opbouwen van PARUBAS-opt. niet voor elke dataset de optimale P-waarde verkregen (gegenereerde resultaten toonden dit aan). Verder onderzoek kan zich focussen op het produceren van betere optimalisatiedata. Als antwoord op deelvraag ***iii***, zijn we er dus niet in geslaagd voor elke dataset optimale P-waarden te calculeren.

PARUBAS-opt. met een random gegenereerd aantal regels aangeleerd presteerde significant beter dan classificatietechniek 1R, maar significant minder goed dan C4.5.

PARUBAS-opt., met het optimale aantal regels per dataset dat gevonden werd voor ARUBAS-Scheffer, bekwam een hogere gemiddelde rangschikking als ARUBAS. Dit verschil was echter niet significant op 10%. In een vergelijking met geprunede en niet geprunede versies van bestaande classificatietechnieken C4.5, CBA, RIPPER en 1R, behaalde PARUBAS-opt. de vierde plaats, net voor ARUBAS en na C4.5 *pruned*, C4.5 *unpruned* en RIPPER *pruned*. C4.5 *pruned*, C4.5 *unpruned* en RIPPER *pruned* en PARUBAS-opt. presteerden significant beter als 1R. Onderling verschilden ze echter niet significant op 10%.

PARUBAS behaalde in elk van de uitgevoerde experimenten een hogere gemiddelde rangschikking als ARUBAS-Scheffer. Dit verschil was echter niet steeds significant op 10%. Op basis hiervan kan een voorkeur voor PARUBAS worden geuit, maar geen sterke aanbeveling. In dit onderzoek werden de CAR- en P-parameters apart geoptimaliseerd per dataset (zie algorithm 1). Verder onderzoek kan zich dan ook focussen op de gezamenlijke optimalisatie per dataset van zowel het aantal aangeleerde regels als het percentage van instanties waarmee vergeleken wordt. Dit zou hoogstwaarschijnlijk de prestaties van het algoritme verder verbeteren.

Bibliografie

R. Agrawal, T. Imieliński & A. Swami (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 207–216. ACM New York, NY, USA.

M. L. Antonie, O. R. Zaiane & R. C. Holte (2006). Learning to use a learned model: A two-stage approach to classification. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006)*, pp. 33–42.

A. Asuncion & D. Newman (2007). UCI machine learning repository. URL `http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html`.

E. Baralis, S. Chiusano & P. Garza (2004). On support thresholds in associative classification. In Haddad *et al.* (2004), pp. 553–558.

N. Cercone, T. Y. Lin & X. Wu, editors (2001). *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*. IEEE Computer Society. ISBN 0-7695-1119-8.

W. Chen, J. F. Naughton & P. A. Bernstein, editors (2000). *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*. ACM. ISBN 1-58113-218-2.

J. Demšar (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

B. Depaire, K. Vanhoof & G. Wets (2008). Arubas: An association rule based similarity framework for associative classifiers. In *ICDM Workshops*, pp. 692–699.

C. Drummond (2006). Machine learning an experimental science (revisited). In *Evaluation Methods for Machine Learning Workshop of the Twenty-First National Conference on Artificial Intelligence*.

T. Fawcett & P. A. Flach (2005). A response to webb and ting's on the application of roc analysis to predict classification performance under varying class distributions. *Mach. Learn.*, 58(1):33–38.

U. M. Fayyad, G. Piatetsky-Shapiro & P. Smyth (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54.

H. Haddad, A. Omicini, R. L. Wainwright & L. M. Liebrock, editors (2004). *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*. ACM. ISBN 1-58113-812-1.

J. Han & M. Kamber (2006). *Data mining: concepts and techniques*. Elsevier, Amsterdam, 2nd ed edition. ISBN 1558609016. URL `http://www.loc.gov/catdir/enhancements/fy0664/2006296324-d.html`.

J. Han, J. Pei & Y. Yin (2000). Mining frequent patterns without candidate generation. In Chen *et al.* (2000), pp. 1–12.

D. J. Hand & R. J. Till (2001). A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

D. D. Jensen & P. R. Cohen (2000). Multiple comparisons in induction algorithms. *Machine Learning*, 38(3):309–338.

D. T. Larose (2005). *Discovering knowledge in data: an introduction to data mining*. Wiley-Interscience, Hoboken, N.J. ISBN 0471666572 (cloth). URL `http://www.loc.gov/catdir/description/wiley042/2004003680.html`.

W. Li, J. Han & J. Pei (2001). Cmar: Accurate and efficient classification based on multiple class-association rules. In Cercone *et al.* (2001), pp. 369–376.

B. Liu, W. Hsu & Y. Ma (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 80–86.

F. Provost, T. Fawcett & R. Kohavi (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 445–453.

J. R. Quinlan & R. M. Cameron-Jones (1993). FOIL: A midterm report. In *Machine Learning: ECML-93, European Conference on Machine Learning, Proceedings*, volume 667 of *Lecture Notes in Artificial Intelligence*, pp. 3–20. Springer-Verlag.

T. Scheffer (2005). Finding association rules that trade support optimally against confidence. *Intelligent Data Analysis*, 9(4):381–395.

F. A. Thabtah (2007). A review of associative classification mining. *Knowledge Eng. Review*, 22(1):37–65.

S. Vanderlooy & E. Hüllermeier (2008). A critical analysis of variants of the auc. *Machine Learning*, 72(3):247–262.

G. I. Webb & K. M. Ting (2005). On the application of roc analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):25–32.

I. H. Witten & E. Frank (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. ISBN 1-55860-552-5.

I. H. Witten & E. Frank (2005). *Data mining: practical machine learning tools and techniques*. Morgan Kaufman, Amsterdam, 2nd ed edition. ISBN 0120884070. URL `http://www.loc.gov/catdir/enhancements/fy0624/2005043385-d.html`.

X. Yin & J. Han (2003). Cpar: Classification based on predictive association rules. In *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA*.

A. Zimmermann & L. De Raedt (2004). Corclass: Correlated association rule mining for classification. *Lecture Notes in Computer Science*, pp. 60–72.

Bijlage A

Java Code

Listing A.1: Java code om Similarity met N instanties of P percent van de instanties te kunnen berekenen tijdens de classificatiefase

```
private double getNSimilarity (Instance testcase, int Number) {
    double[] NSArray = getNSimilarityArray(testcase, Number);
    double nSimilarity = 0;
    for(int j = 0; j < Number; j++){
        nSimilarity = nSimilarity + NSArray[j];
    }
    nSimilarity = nSimilarity / Number;
    return nSimilarity;
}
private double getNSimilarity (Instance testcase, double percentage){
    double p = percentage*m_data.numInstances();
    int n = (int)Math.ceil((p));
    double[] NSArray = getNSimilarityArray(testcase, n);
    double nSimilarity = 0;
    for(int j = 0; j < n; j++){
        nSimilarity = nSimilarity + NSArray[j];
    }
    nSimilarity = nSimilarity / n;
    return nSimilarity;
    }
private double[] getSimilarityArray(Instance testcase){
    int[] transformedTestcase = transformInstance(testcase);
```

```
        double[] similarityArray = new double[m_data.numInstances()];
        int numRules = m_ruleset[0].size();
        //Compare the rules covering the testcase with the rules
        //covering a trainingcase (for each trainingcase (j))

        for(int j = 0; j < m_data.numInstances(); j++){
            double similarity = getEquation2(transformedTestcase, m_matrix[j])*(
                double)getEquation5Nominator(transformedTestcase,m_matrix[j])/((
                double)numRules);
            similarityArray[j] = similarity;
        }
        return similarityArray;
    }
    private double[] getNSimilarityArray (Instance testcase, int numberInstances
        ){
        int n = numberInstances;
        double[] nSimArray = new double[n];
        DoubleArrayList simArray = new DoubleArrayList(getSimilarityArray(
            testcase));
        simArray.sort();
        simArray.reverse();
        for(int i = 0; i < n && i < simArray.size() ; i++ ){
            nSimArray[i] = simArray.get(i);
        }
    return nSimArray;
    }
```

Listing A.2: Exploratief Onderzoek, Experiment 1 waarbij de percentages gevarieerd worden, de JAVA-code voor de variatie van N is gelijkaardig

```
package applications;
import java.io.BufferedReader;
public class Experiment_tenfold {
    static int seed = 12536245;
    static int numFolds = 10;
    /**
     * @param args
     */
```

```
public static void main(String[] args) throws Exception{
    String[] datasets = {"lenses"};
    String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
        Workspace/Arubas/data/discretized datasets/";
    String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
        Experiments/Comparing/";
    FileWriter outputSummary = new FileWriter(outputloc+"Experiment tenfold 0
        .1p.xls");
    BufferedWriter sum = new BufferedWriter (outputSummary);
    sum.write("<table><tr><td>NArubas</td></tr>");
    for(int i = 0; i < datasets.length; i++) {
        sum.write("<tr><td>" +datasets[i] + "</td></tr>");
        Instances data = new Instances(new BufferedReader (new FileReader (
            dataloc+datasets[i]+"_discretizedclean.arff")));
        data.setClassIndex(data.numAttributes()-1);
        data.deleteWithMissingClass();
        //Generate the folds
        Random rand = new Random (seed);
        Instances randData = new Instances(data);
        randData.randomize(rand);
        randData.stratify(numFolds);
        Instances[] train = new Instances[numFolds];
        Instances[] test = new Instances[numFolds];
        for(int f = 0; f < numFolds; f++){
            train[f] = randData.trainCV(numFolds, f);
            test[f] = randData.testCV(numFolds, f);
        }
        PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
        FastVector[][] ruleset = new FastVector [numFolds][];
        for (int f = 0; f < numFolds; f++){
            System.out.println("Learn rules for fold: "+f);
            rsg[f] = new PredictiveAprioriRSG(train[f]);
            rsg[f].updateParameter(RuleSetGeneratorProperties.NUMBER_OF_RULES,
                new Integer(1000));
            ruleset[f] = rsg[f].buildRuleSet();
        }
        for(int k = 1000; k > 0; k-=100){
            sum.write("<tr><td>aantal regels</td><td>" + (k) + "</td></tr>");
```

```
            FastVector[][] trimmedRuleset = new FastVector[numFolds][];
            for(int f = 0; f < numFolds; f++){
               trimmedRuleset [f] = RuleSetUtil.trimRuleSet(ruleset[f], k);
            }
            for (double l = 0.1; l <=1.00 ; l+=0.1){
               sum.write("<tr><td>" + l + "</td>");
               double[] accuracies = new double[numFolds];
               for(int f = 0; f < numFolds; f++){
                   ArubasClassifier AC = new ArubasClassifier(train[f],
                       trimmedRuleset[f], FitnessMeasures.NSIMILARITY, l);
                   AC.buildClassifier();
                   AC.classifyInstancesPercentage(test[f]);
                   accuracies[f] = AC.getAccuracy();
               }
               double accuracy = Utils.mean(accuracies);
               sum.write("<td>" + accuracy + "</td>");
            }
         }
      }
       sum.write("</table>");
       sum.close();
   }
}
```

Listing A.3: Experiment 2, Methode 1

```
package applications;
import java.io.BufferedReader;
public class Experiment_double_tenfold {
   static int seed = 12536245;
   static int numFolds = 10;
   /**
    * @param args
    */
   public static void main(String[] args) throws Exception{
      String[] datasets = {"lenses"};
      String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
          Workspace/Arubas/data/discretized datasets/";
```

```
String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
    Experiments/Aangepaste double tenfold/";
FileWriter outputSummary = new FileWriter(outputloc+"Experiment tenfold 0
    .1p afterCV.xls");
BufferedWriter sum = new BufferedWriter (outputSummary);
sum.write("<table><tr><td>NArubas</td></tr>");
for(int i = 0; i < datasets.length; i++) {
   sum.write("<tr><td>" +datasets[i] + "</td></tr>");
    Instances data = new Instances(new BufferedReader (new FileReader (
        dataloc+datasets[i]+"_discretizedclean.arff")));
    data.setClassIndex(data.numAttributes()-1);
    data.deleteWithMissingClass();

    //Generate the folds
    Random rand = new Random (seed);
    Instances randData = new Instances(data);
    randData.randomize(rand);
    randData.stratify(numFolds);
    Instances[] train = new Instances[numFolds];
    Instances[] test = new Instances[numFolds];
    Instances[] trainOpt = new Instances[numFolds];
    Instances[] testOpt = new Instances[numFolds];
    for(int f = 0; f < numFolds; f++){
       train[f] = randData.trainCV(numFolds, f);
       test[f] = randData.testCV(numFolds, f);
       train[f].stratify(10);
       trainOpt[f] = train[f].trainCV(10, 0);
       testOpt[f] = train[f].testCV(10, 0);
    }
    PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
    FastVector[][] ruleset = new FastVector [numFolds][];
    for (int f = 0; f < numFolds; f++){
          System.out.println("Learn rules for fold: "+f);
          rsg[f] = new PredictiveAprioriRSG(trainOpt[f]);
          rsg[f].updateParameter(RuleSetGeneratorProperties.
              NUMBER_OF_RULES,new Integer(1000));
          ruleset[f] = rsg[f].buildRuleSet();
    }
```

```
            for(int k = 0; k < 901; k+=100){
                sum.write("<tr><td>aantal_regels</td><td>" + (1000-k) + "</td></tr
                    >");
                FastVector[][] trimmedRuleset = new FastVector[numFolds][];
                for(int f = 0; f < numFolds; f++){
                    trimmedRuleset [f] = RuleSetUtil.trimRuleSet(ruleset[f], 1000-k)
                        ;
                }
                for (double l = 0.1; l <=1.00 ; l+=0.1){
                    sum.write("<tr><td>" + l + "</td>");
                    double[] accuracies = new double[numFolds];
                    for(int f = 0; f < numFolds; f++){
                        ArubasClassifier AC = new ArubasClassifier(trainOpt[f],
                            trimmedRuleset[f], FitnessMeasures.NSIMILARITY, l);
                        AC.buildClassifier();
                        AC.classifyInstancesPercentage(testOpt[f]);
                        accuracies[f] = AC.getAccuracy();
                    }
                    double accuracy = Utils.mean(accuracies);
                    sum.write("<td>" + accuracy + "</td>");
                }
            }
        }
        sum.write("</table>");
            sum.close();
    }
}
```

Listing A.4: Experiment 2, Methode 2

```
package applications;
import java.io.BufferedReader;
public class Experiment_double_tenfold2 {
    static int seed = 12536245;
    static int numFolds = 10;
    /**
     * @param args
     */
```

```
public static void main(String[] args) throws Exception{
    String[] datasets = {"haberman", "hayes-roth", "iris", "lenses", "
        mammographic_masses"};
    String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
        Workspace/Arubas/data/discretized datasets/";
    String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
        Experiments/Aangepaste double tenfold/";
    FileWriter outputSummary = new FileWriter(outputloc+"Experiment 0.1
        p Tenfold Opt Before optimization.xls");
    BufferedWriter sum = new BufferedWriter (outputSummary);
    sum.write("<table><tr><td>PArubas</td></tr>");
    for(int i = 0; i < datasets.length; i++) {
       sum.write("<tr><td>" +datasets[i] + "</td></tr>");
       Instances data = new Instances(new BufferedReader (new FileReader (
           dataloc+datasets[i]+"_discretizedclean.arff")));
       data.setClassIndex(data.numAttributes()-1);
       data.deleteWithMissingClass();

       //Generate the folds
       Random rand = new Random (seed);
       Instances randData = new Instances(data);
       randData.randomize(rand);
       randData.stratify(numFolds);
       Instances train = randData.trainCV(numFolds, 0);
       Instances test = randData.testCV(numFolds, 0);
       train.stratify(numFolds);
       Instances[] train2 = new Instances[numFolds];
       Instances[] test2 = new Instances[numFolds];
       for(int f = 0; f < numFolds; f++){
          train2[f] = train.trainCV(numFolds, f);
          test2[f] = train.testCV(numFolds, f);
       }
       PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
       FastVector[][] ruleset = new FastVector [numFolds][];
       for (int f = 0; f < numFolds; f++){
          System.out.println("Learn rules for fold: "+f);
          rsg[f] = new PredictiveAprioriRSG(train);
          rsg[f].updateParameter(RuleSetGeneratorProperties.NUMBER_OF_RULES,
```

```
                new Integer(1000));
            ruleset[f] = rsg[f].buildRuleSet();
        }
        for(int k = 0; k < 901; k+=100){
            sum.write("<tr><td>aantal_regels</td><td>" + (1000-k) + "</td></tr
                >");
            FastVector[][] trimmedRuleset = new FastVector[numFolds][];
            for(int f = 0; f < numFolds; f++){
                trimmedRuleset [f] = RuleSetUtil.trimRuleSet(ruleset[f], 1000-k)
                    ;
            }
            for (double l = 0.1; l <=1.00 ; l+=0.1){
                sum.write("<tr><td>" + l + "</td>");
                double[] accuracies = new double[numFolds];
                for(int f = 0; f < numFolds; f++){
                    ArubasClassifier AC = new ArubasClassifier(train,
                        trimmedRuleset[f], FitnessMeasures.NSIMILARITY, l);
                    AC.buildClassifier();
                    AC.classifyInstancesPercentage(test);
                    accuracies[f] = AC.getAccuracy();
                }
                double accuracy = Utils.mean(accuracies);
                sum.write("<td>" + accuracy + "</td>");
            }
        }
    }
    sum.write("</table>");
    sum.close();
  }
}
```

Listing A.5: Experiment Hypothese 1

```
package applications;
import java.io.BufferedReader;
public class Experiment_P_VS_N {
  static int seed = 12536245;
  static int numFolds = 10;
```

```
/**
 * @param args
 */
public static void main(String[] args) throws Exception{
    String[] datasets = {"balance-scale","car","cmc","ecoli","glass","
        haberman","iris","mammographic_masses","monk1","monk2","monk3","pima-
        indians-diabetes","tae","tic-tac-toe","yeast"};
    String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
        Workspace/Arubas/data/discretized datasets/";
    String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
        Experiments/Comparing/";
    FileWriter outputSummary = new FileWriter(outputloc+"
        Experiment_numbersVSpercentages_15sets_anderePN.xls");
    BufferedWriter sum = new BufferedWriter (outputSummary);
    sum.write("<table><tr><td></td><td>PArubas</td><td>NArubas</td></tr>");
    for(int i = 0; i < datasets.length; i++) {
       sum.write("<tr><td>" +datasets[i] + "</td>");
       Instances data = new Instances(new BufferedReader (new FileReader (
           dataloc+datasets[i]+"_discretizedclean.arff")));
       data.setClassIndex(data.numAttributes()-1);
       data.deleteWithMissingClass();
       //Generate the folds
       Random rand = new Random (seed);
       Instances randData = new Instances(data);
       randData.randomize(rand);
       randData.stratify(numFolds);
       Instances[] train = new Instances[numFolds];
       Instances[] test = new Instances[numFolds];
       for(int f = 0; f < numFolds; f++){
          train[f] = randData.trainCV(numFolds, f);
          test[f] = randData.testCV(numFolds, f);
       }
       PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
       FastVector[][] ruleset = new FastVector [numFolds][];
       for (int f = 0; f < numFolds; f++){
          System.out.println("Learn rules for fold: "+f);
          rsg[f] = new PredictiveAprioriRSG(train[f]);
          rsg[f].updateParameter(RuleSetGeneratorProperties.NUMBER_OF_RULES,
```

```
                new Integer(200));
            ruleset[f] = rsg[f].buildRuleSet();
        }
        double[] PAccuracies = new double[5];
        double[] percentages = {0.7, 0.5, 0.3, 0.2, 0.1};
        for (int j = 0; j < percentages.length; j++){
            double[] accuracies = new double[numFolds];
            for(int f = 0; f < numFolds; f++){
                ArubasClassifier AC = new ArubasClassifier(train[f], ruleset[f],
                    FitnessMeasures.NSIMILARITY, percentages[j]);
                AC.buildClassifier();
                AC.classifyInstancesPercentage(test[f]);
                accuracies[f] = AC.getAccuracy();
            }
            PAccuracies[j] = Utils.mean(accuracies);
        }
        double PAccuracy = Utils.mean(PAccuracies);
        sum.write("<td>" +PAccuracy+ "</td>");
        double[] NAccuracies = new double[5];
        int[] numbers = {150, 100, 50, 20, 10};
        for (int k = 0; k < numbers.length; k++){
            double[] accuracies = new double[numFolds];
            for(int f = 0; f < numFolds; f++){
                ArubasClassifier AC = new ArubasClassifier(train[f], ruleset[f],
                    FitnessMeasures.NSIMILARITY, numbers[k]);
                AC.buildClassifier();
                AC.classifyInstances(test[f]);
                accuracies[f] = AC.getAccuracy();
            }
            NAccuracies[k] = Utils.mean(accuracies);
        }
        double NAccuracy = Utils.mean(NAccuracies);
        sum.write("<td>" +NAccuracy+ "</td></tr>");
    }
    sum.write("</table>");
    sum.close();
  }
}
```

Listing A.6: Experiment Hypothese 2

```
package applications;
import java.io.BufferedReader;
public class Experiment_compare_Arubas_NARUBAS_random_rules {
   static int seed = 12536245;
   static int numFolds = 10;
   /**
    * @param args
    */
   public static void main(String[] args) throws Exception{
      String[] datasets = {"balance-scale","car","cmc","echocardiogram","ecoli"
          ,"glass","haberman","hayes-roth","iris","lenses","mammographic_masses"
          ,"monk1","monk2","monk3","pima-indians-diabetes","post-operative","tae
          ","tic-tac-toe","yeast"};
      String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
          Workspace/Arubas/data/discretized datasets/";
      String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
          Experiments/Comparing/";
      FileWriter outputSummary = new FileWriter(outputloc+"
          Experiment_Comparing_random_rules_6intervals_19Sets.xls");
      BufferedWriter sum = new BufferedWriter (outputSummary);
      sum.write("<table><tr><td>Dataset</td><td>Arubas</td><td>NArubas - 70%</
          td><td>NArubas - 50%</td><td>NArubas - 30%</td><td>NArubas - 10%</td><
          td>NArubas - 1%</td></tr>");
      for(int i = 0; i < datasets.length; i++) {
         System.out.println("processing dataset " + i);
         sum.write("<tr><td>" +datasets[i] + "</td>");
         Instances data = new Instances(new BufferedReader (new FileReader (
             dataloc+datasets[i]+"_discretizedclean.arff")));
         data.setClassIndex(data.numAttributes()-1);
         data.deleteWithMissingClass();
         //Generate the folds
         Random rand = new Random (seed);
         Instances randData = new Instances(data);
         randData.randomize(rand);
         randData.stratify(numFolds);
         Instances[] train = new Instances[numFolds];
```

```
        Instances[] test = new Instances[numFolds];
        for(int f = 0; f < numFolds; f++){
            train[f] = randData.trainCV(numFolds, f);
            test[f] = randData.testCV(numFolds, f);
        }
        PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
        FastVector[][] ruleset = new FastVector [numFolds][];
        for (int f = 0; f < numFolds; f++){
                System.out.println("Learn rules for fold: "+f);
                rsg[f] = new PredictiveAprioriRSG(train[f]);
                rsg[f].updateParameter(RuleSetGeneratorProperties.
                    NUMBER_OF_RULES,new Integer(1000));
                ruleset[f] = rsg[f].buildRuleSet();
        }
        int[] regels = {891, 155, 93, 653, 999, 792, 567, 390, 327, 842};
        for(int r : regels){
            sum.write("<tr><td>aantal regels</td><td>" + (r) + "</td></tr>");
            FastVector[][] trimmedRuleset = new FastVector[numFolds][];
            for(int f = 0; f < numFolds; f++){
            trimmedRuleset [f] = RuleSetUtil.trimRuleSet(ruleset[f], r);
            }
        }
        double[] percentages = {1.00, 0.7, 0.5, 0.3, 0.1, 0.01};
        for (double l : percentages){
            double[] accuracies = new double[numFolds];
            for(int f = 0; f < numFolds; f++){
                ArubasClassifier AC = new ArubasClassifier(train[f],
                    trimmedRuleset[f], FitnessMeasures.NSIMILARITY, l);
                AC.buildClassifier();
                AC.classifyInstancesPercentage(test[f]);
                accuracies[f] = AC.getAccuracy();
            }
            double accuracy = Utils.mean(accuracies);
            sum.write("<td>" + accuracy + "</td>");
        }
        sum.write("</tr>");
    }
     sum.write("</table>");
```

```
        sum.close();
    }
}
```

Listing A.7: Optimalisatie van P per dataset.

```
package applications;
import java.io.BufferedReader;
public class Optimalnumberofinstances {
    static int seed = 12536245;
    static int numFolds = 10;
    static int[] rules4arubas =
        {280,160,1000,20,120,180,20,20,20,20,60,40,200,20,140,80,160,540,240};
    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        String[] datasets = {"balance-scale","car","cmc","echocardiogram","ecoli
            ","glass","haberman","hayes-roth","iris","lenses","
            mammographic_masses","monk1","monk2","monk3","pima-indians-diabetes",
            "post-operative","tae","tic-tac-toe","yeast"};
        String dataloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/Eclipse
            Workspace/Arubas/data/discretized datasets/";
       String outputloc = "/Users/Kristof/Documents/BI/2e Master/Thesis/
           Experiments/Optimal number of rules/";
        FileWriter outputSummary1 = new FileWriter(outputloc+"
            optimalNinstances19sets_354rules.xls");
        BufferedWriter sum1 = new BufferedWriter(outputSummary1);
        sum1.write("<table><tr><td>Data set</td>");
        for(double i = 1.00; i > 0; i-= 0.010){
          sum1.write("<td>"+i+ "</td>");
        }
        sum1.write("</tr>");
        for(int k = 0; k < datasets.length; k++){
          System.out.println("processing dataset " + k);
           Instances data = new Instances(new BufferedReader(new FileReader(
               dataloc+datasets[k]+"_discretizedclean.arff")));
           data.setClassIndex(data.numAttributes()-1);
          sum1.write("<tr><td>"+datasets[k]+"</td>");
```

```
//Generate the folds
Random rand = new Random (seed);
Instances randData = new Instances(data);
randData.randomize(rand);
randData.stratify(numFolds);
Instances[] train = new Instances[numFolds];
Instances[] test = new Instances[numFolds];
Instances[] trainOpt = new Instances[numFolds];
Instances[] testOpt = new Instances[numFolds];
for(int f = 0; f < numFolds; f++){
   train[f] = randData.trainCV(numFolds, f);
   test[f] = randData.testCV(numFolds, f);
   train[f].stratify(6);
   trainOpt[f] = train[f].trainCV(6, 0);
   testOpt[f] = train[f].testCV(6, 0);
}
PredictiveAprioriRSG[] rsg = new PredictiveAprioriRSG[numFolds];
FastVector[][] ruleset = new FastVector [numFolds][];
for (int f = 0; f < numFolds; f++){
       System.out.println("Learn rules for fold: "+f);
       rsg[f] = new PredictiveAprioriRSG(trainOpt[f]);
       rsg[f].updateParameter(RuleSetGeneratorProperties.
           NUMBER_OF_RULES,new Integer(354));
       ruleset[f] = rsg[f].buildRuleSet();
   }
  for(double i = 1.00; i >0; i-=0.01){
   double[] accuracies = new double[numFolds];
   for(int f = 0; f < numFolds; f++){
       ArubasClassifier AC = new ArubasClassifier(trainOpt[f], ruleset[
           f], FitnessMeasures.NSIMILARITY, i);
       AC.buildClassifier();
       AC.classifyInstancesPercentage(testOpt[f]);
       accuracies[f] = AC.getAccuracy();
   }
   double accuracy = Utils.mean(accuracies);
    sum1.write("<td>"+accuracy+"</td>");
  }
  sum1.write("<td>=max(B"+(k+2)+":CW"+(k+2)+")</td><td>=VERGELIJKEN(CX
```

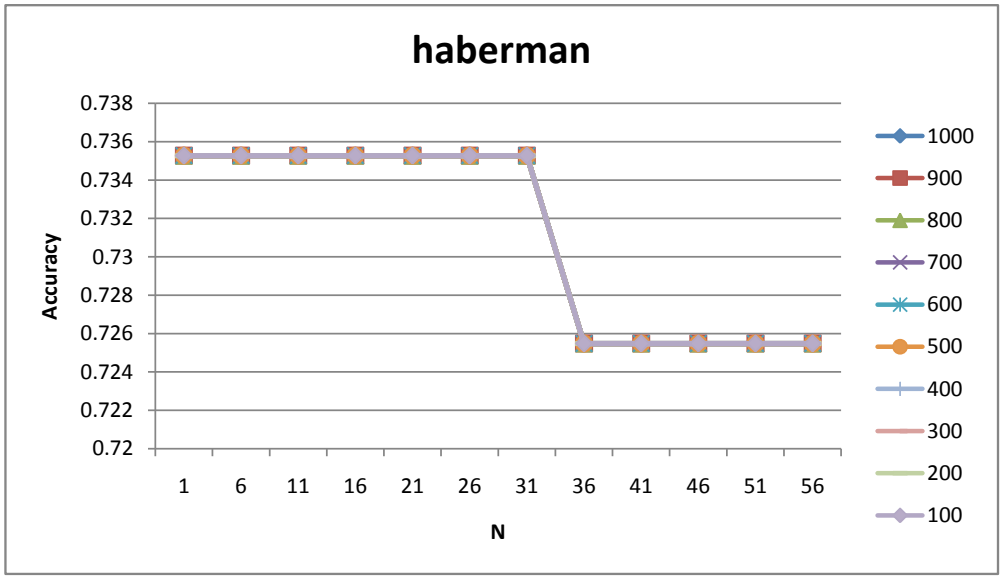
```
                " +(k+2)+" ,B" +(k+2) +" :CW" +(k+2)+ " ,0)</td><td>=1.01-(CY" + (k
                +2) +" /100)</td></tr>" );
        }
        sum1.write("</table>");
        sum1.close();
    }
}
```

Bijlage B

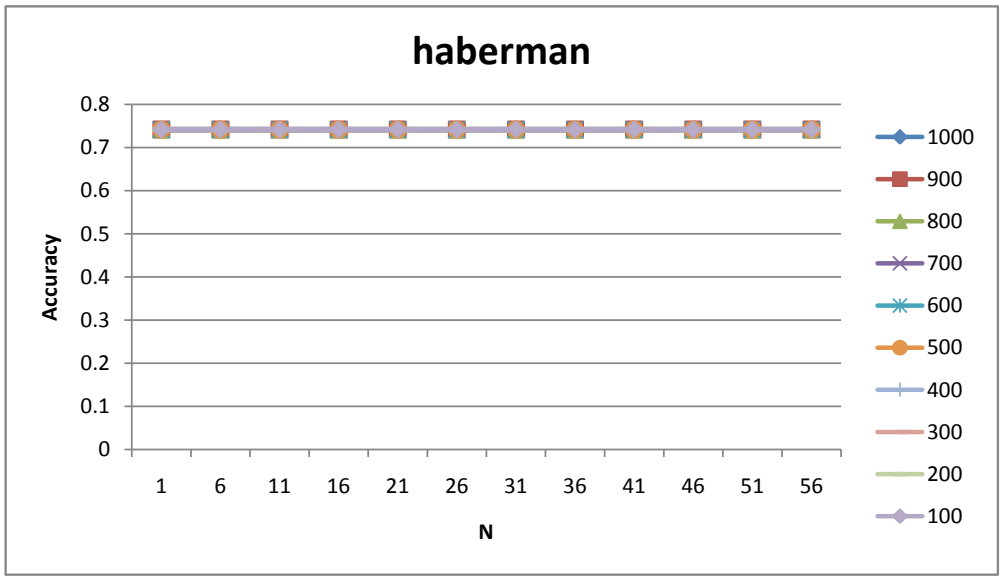
Exploratief Onderzoek

B.1 Experiment 1

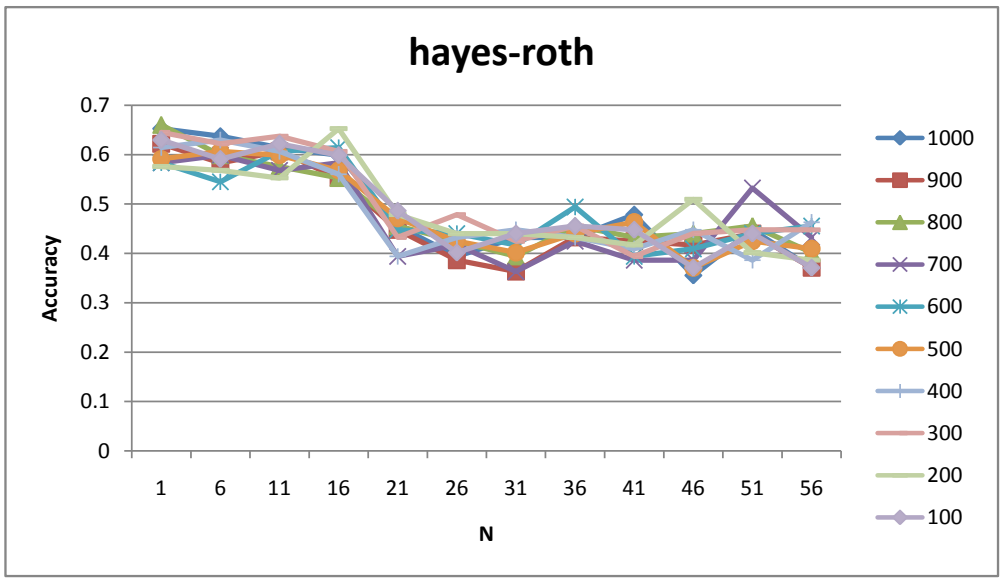
B.1.1 NARUBAS



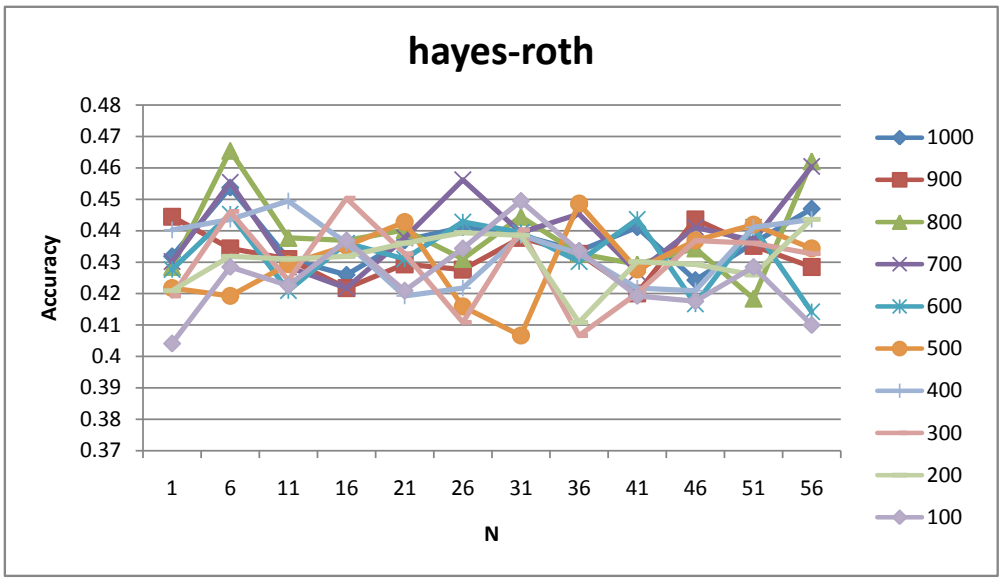
Figuur B.1: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



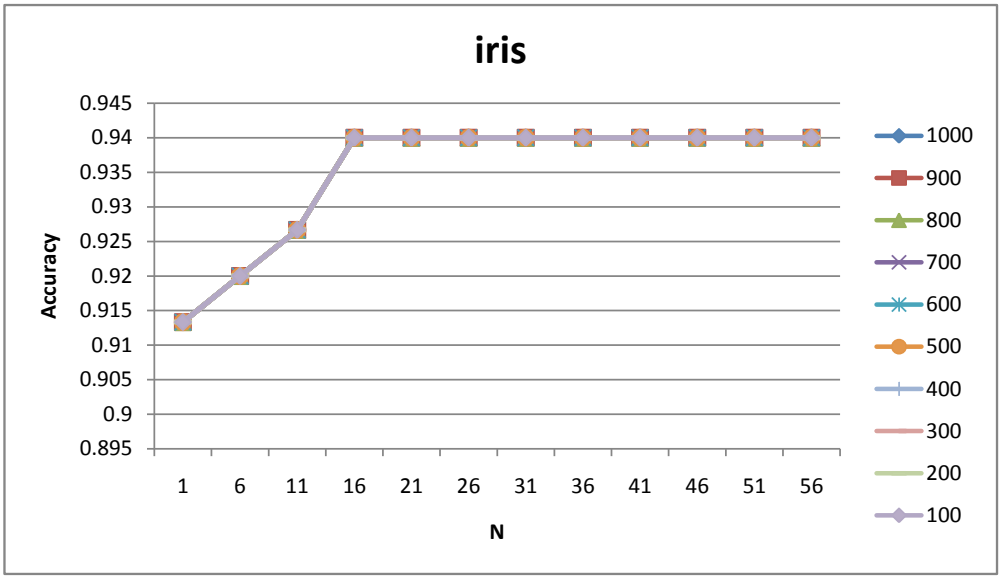
Figuur B.2: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'haberman' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



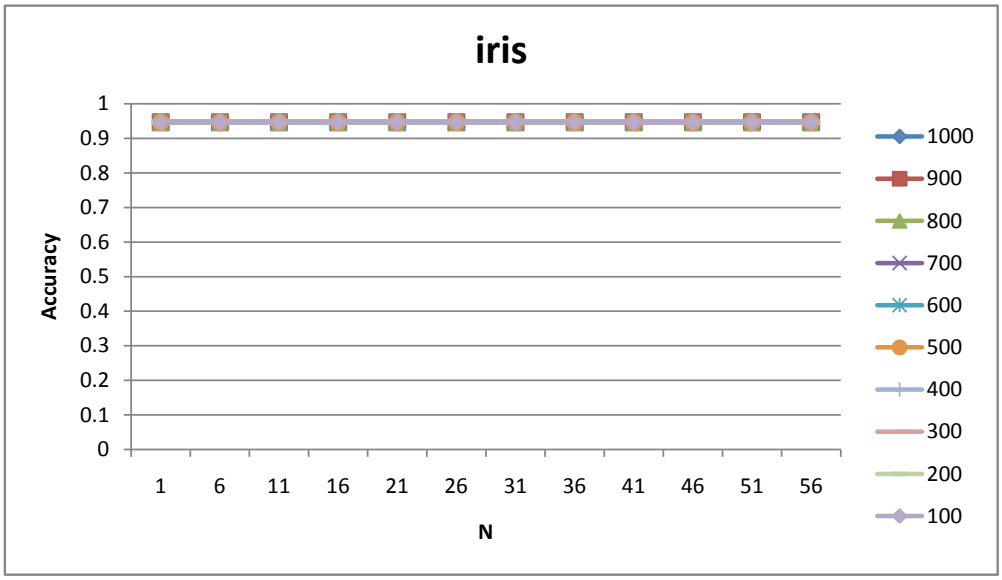
Figuur B.3: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



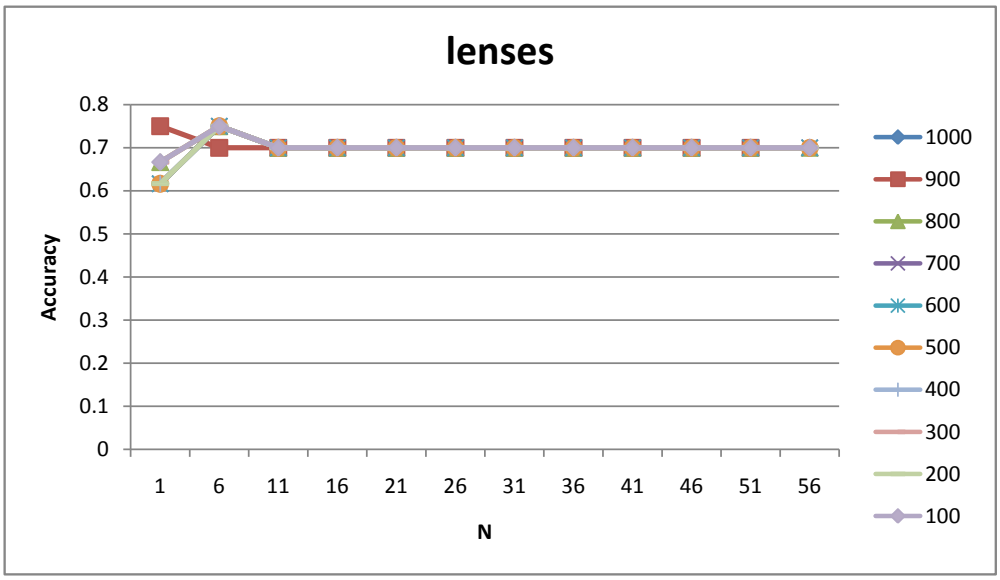
Figuur B.4: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



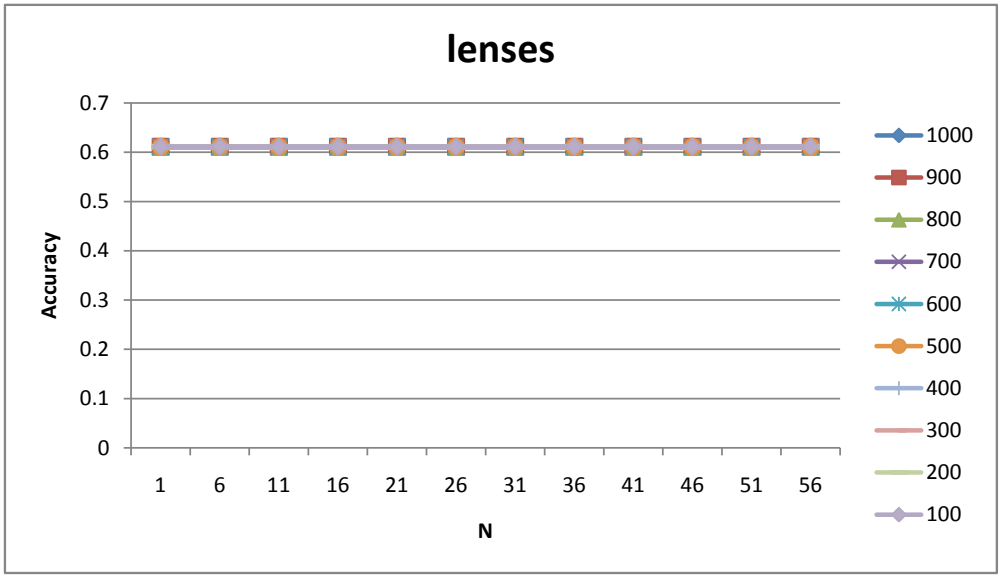
Figuur B.5: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



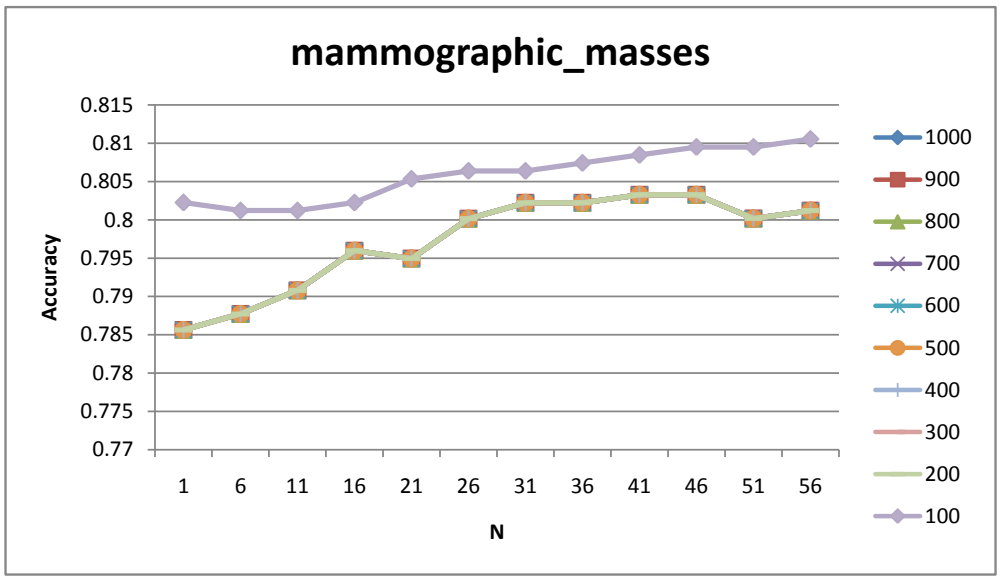
Figuur B.6: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'iris' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



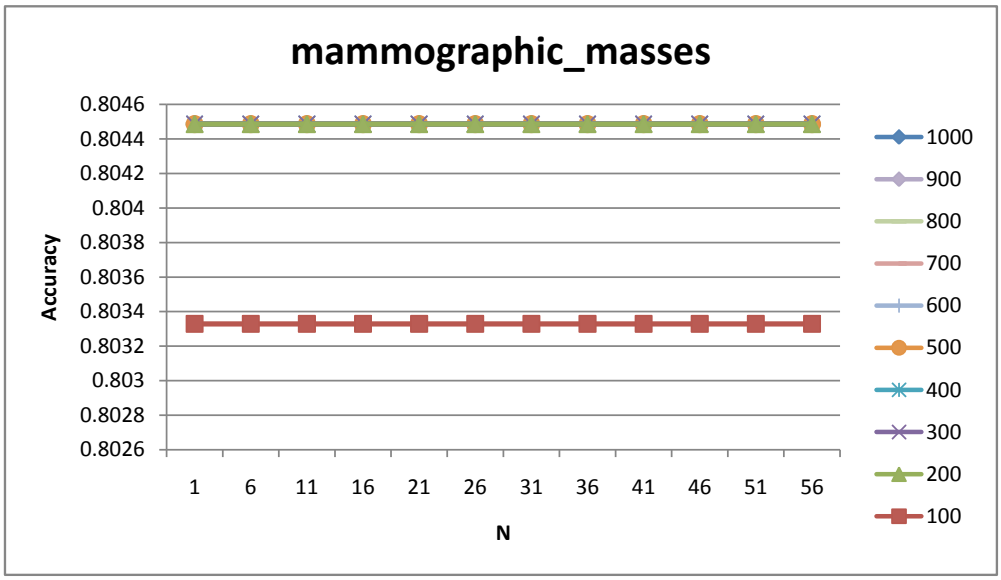
Figuur B.7: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.



Figuur B.8: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'lenses' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.

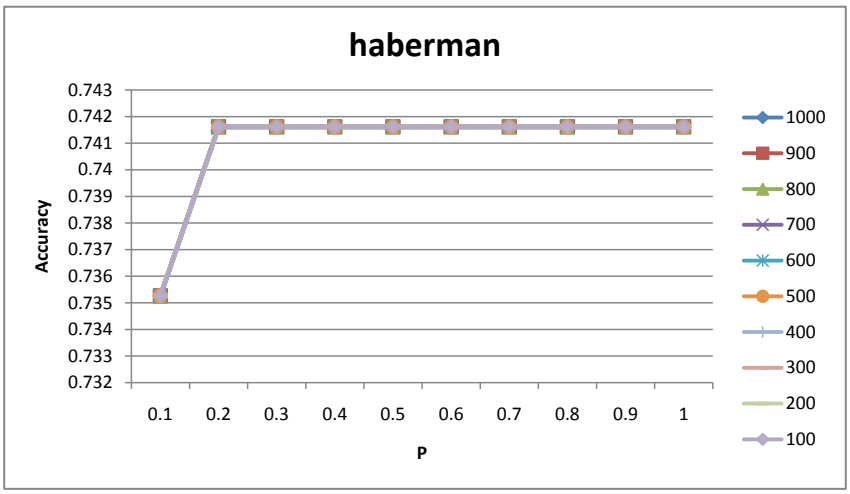


Figuur B.9: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'mammographic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.

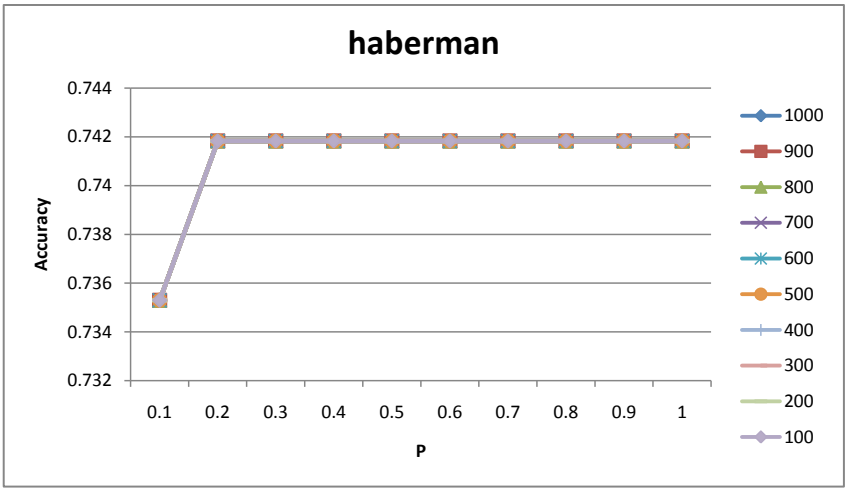


Figuur B.10: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'mammographic_masses' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval het aantal instanties waarmee vergeleken wordt variërend van 1 tot 56 met interval 5.

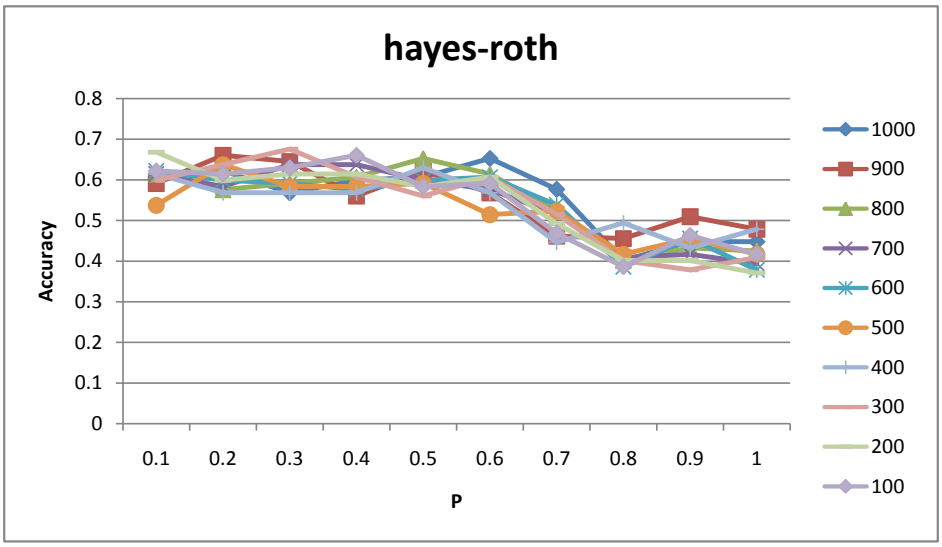
B.1.2 PARUBAS



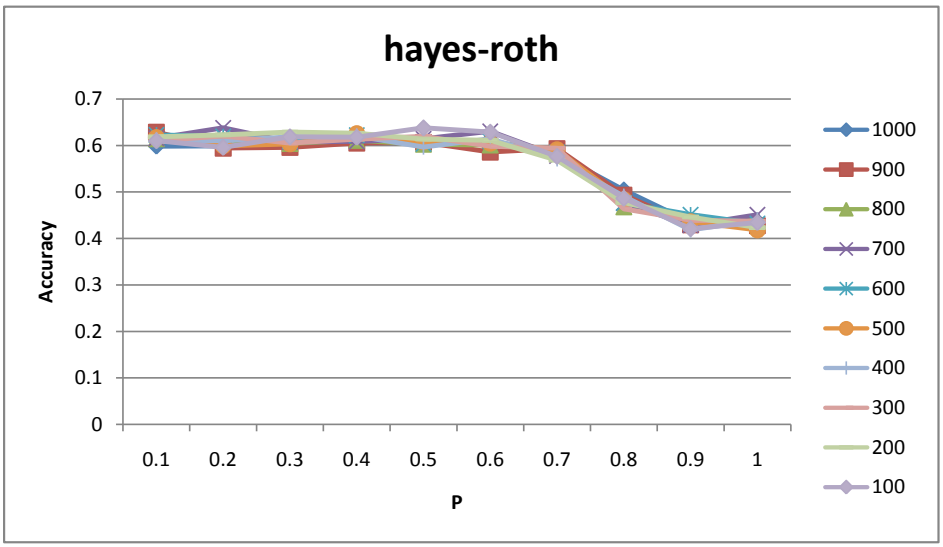
Figuur B.11: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



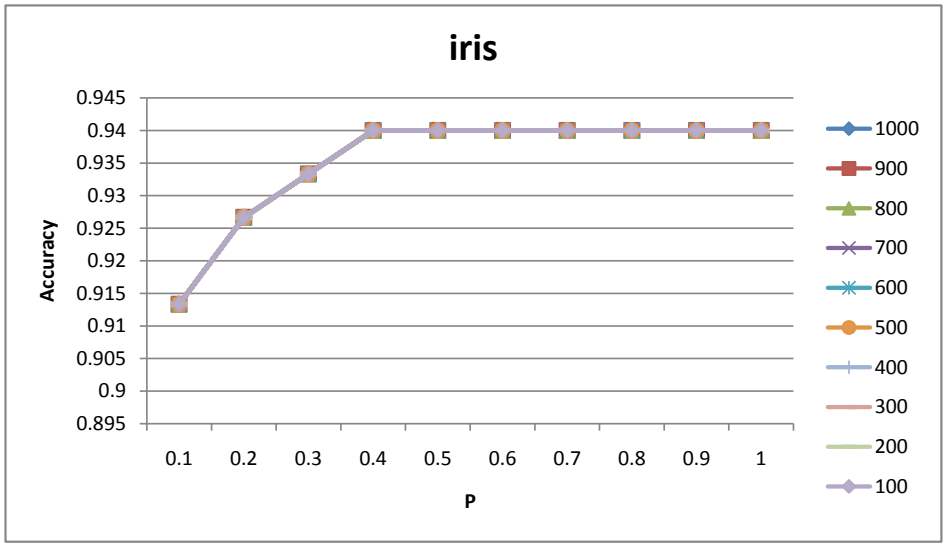
Figuur B.12: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'haberman' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van van 0.1 tot 1 met interval 0.1.



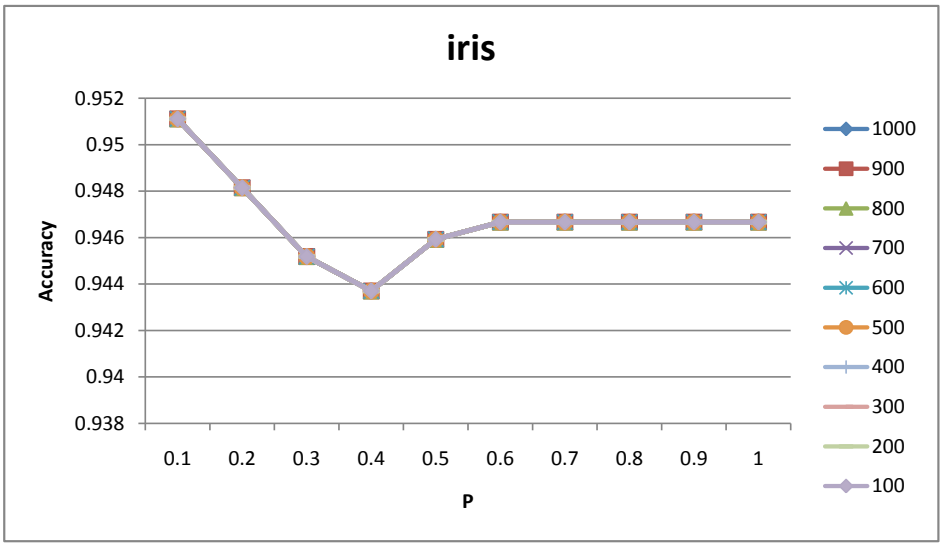
Figuur B.13: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



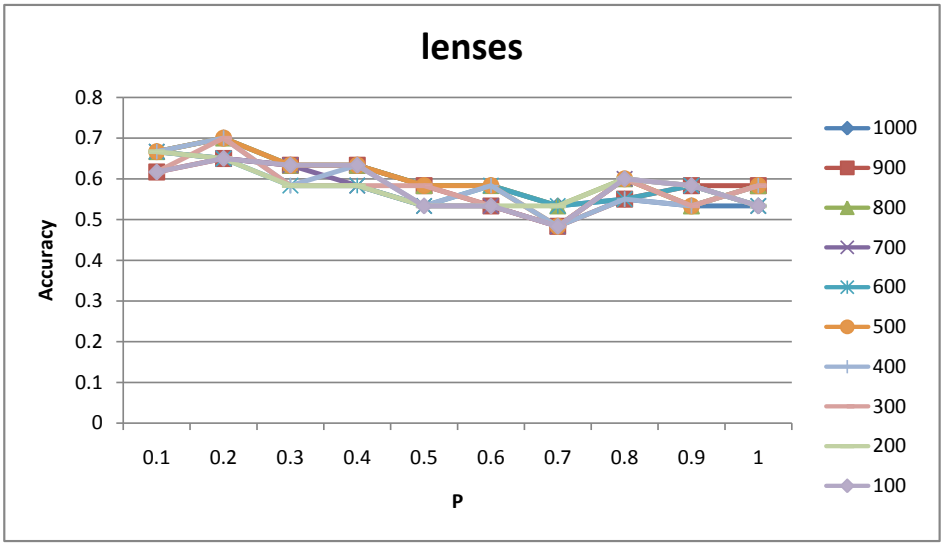
Figuur B.14: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van van 0.1 tot 1 met interval 0.1.



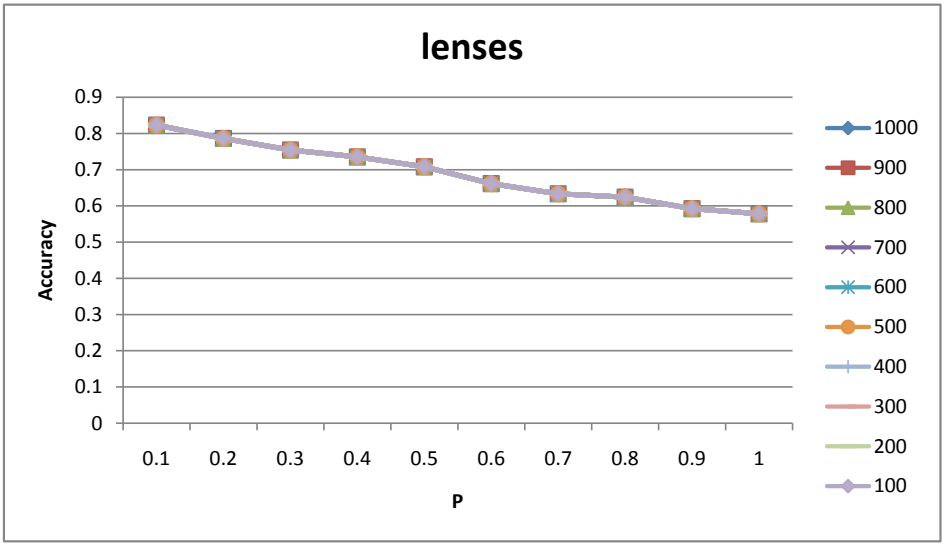
Figuur B.15: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



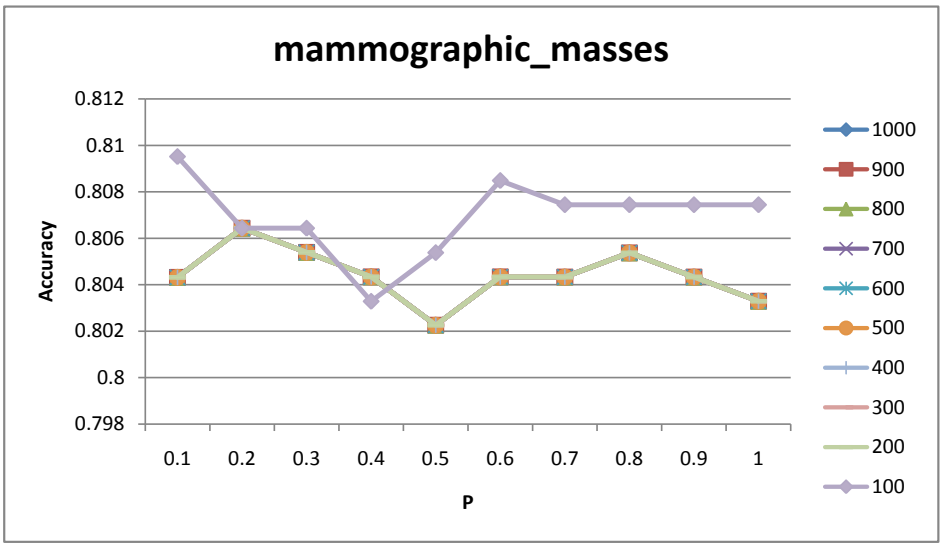
Figuur B.16: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'iris' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van van 0.1 tot 1 met interval 0.1.



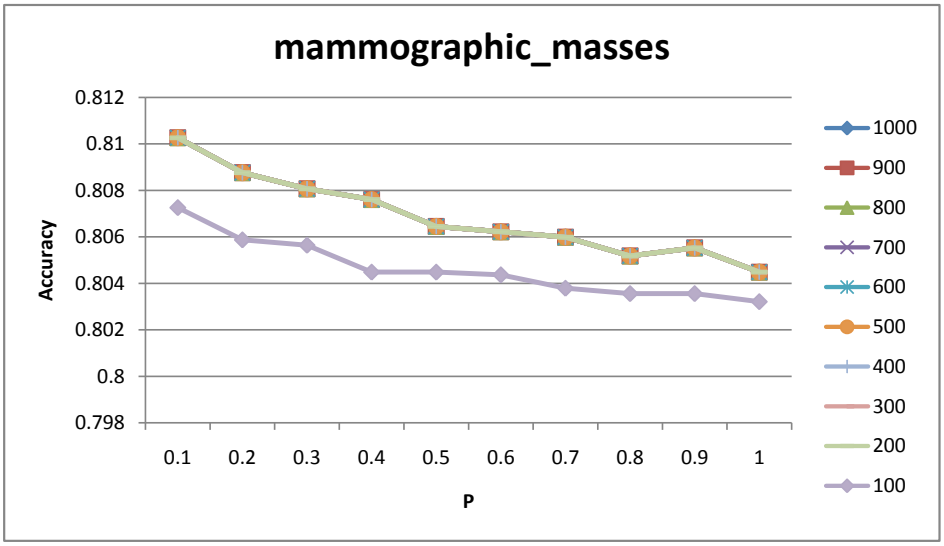
Figuur B.17: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



Figuur B.18: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'lenses' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van van 0.1 tot 1 met interval 0.1.

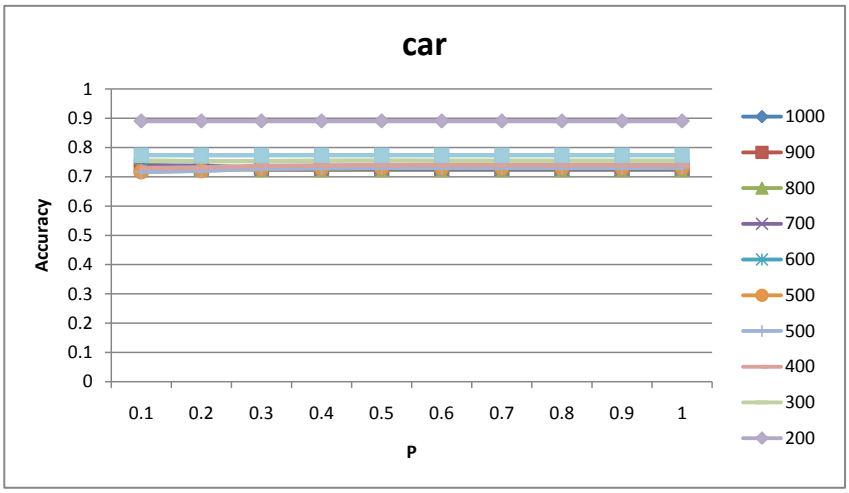


Figuur B.19: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'mammographic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

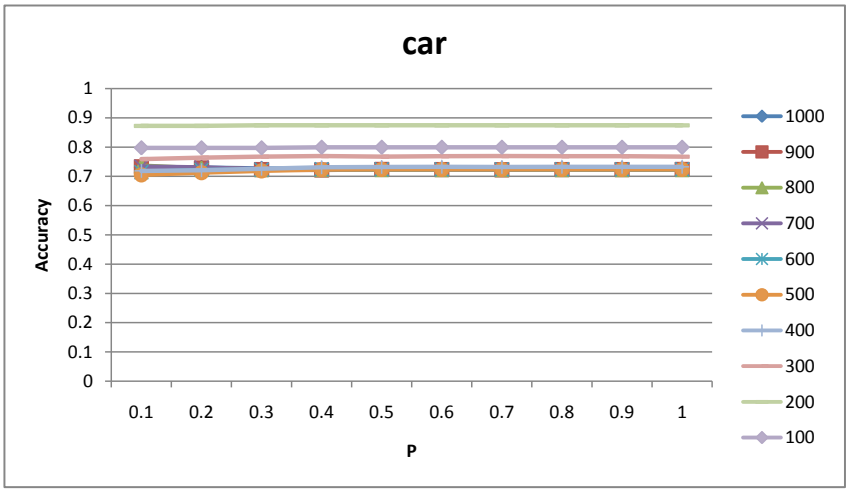


Figuur B.20: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'mammographic_masses' waarbij bij elke *fold* getraind en getest werd op de trainingsset, het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van van 0.1 tot 1 met interval 0.1.

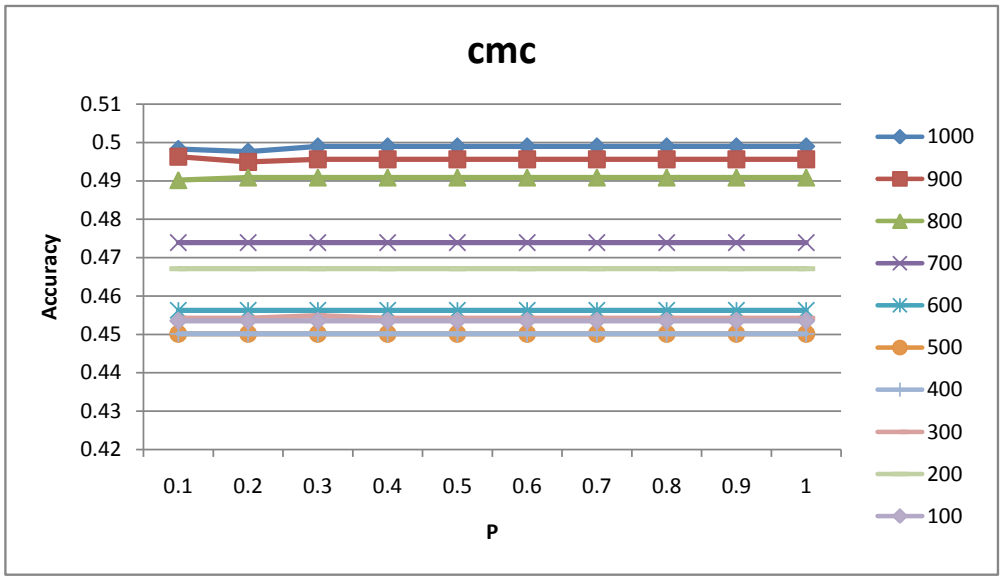
B.2 Optimalisatie - Methode 1



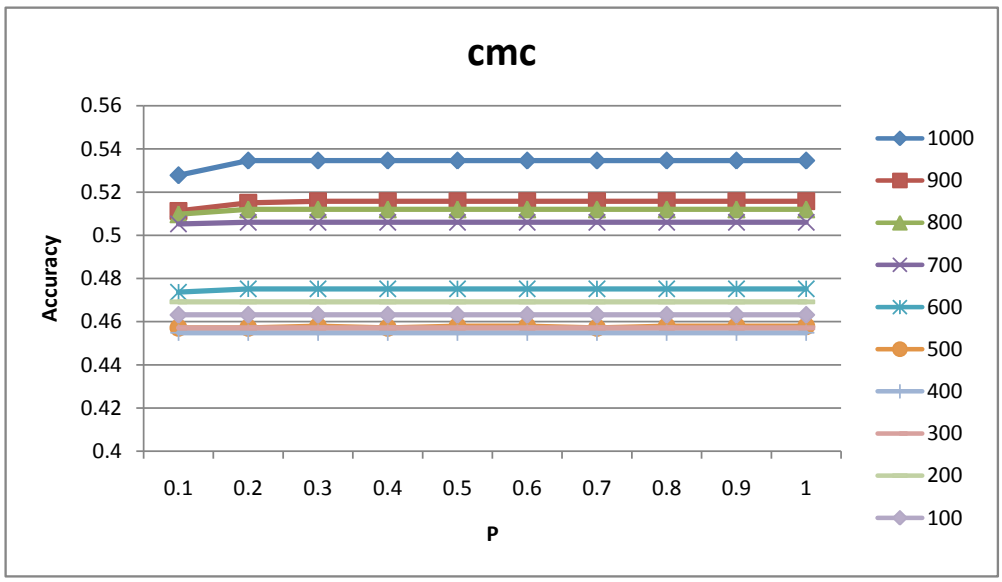
Figuur B.21: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'car', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



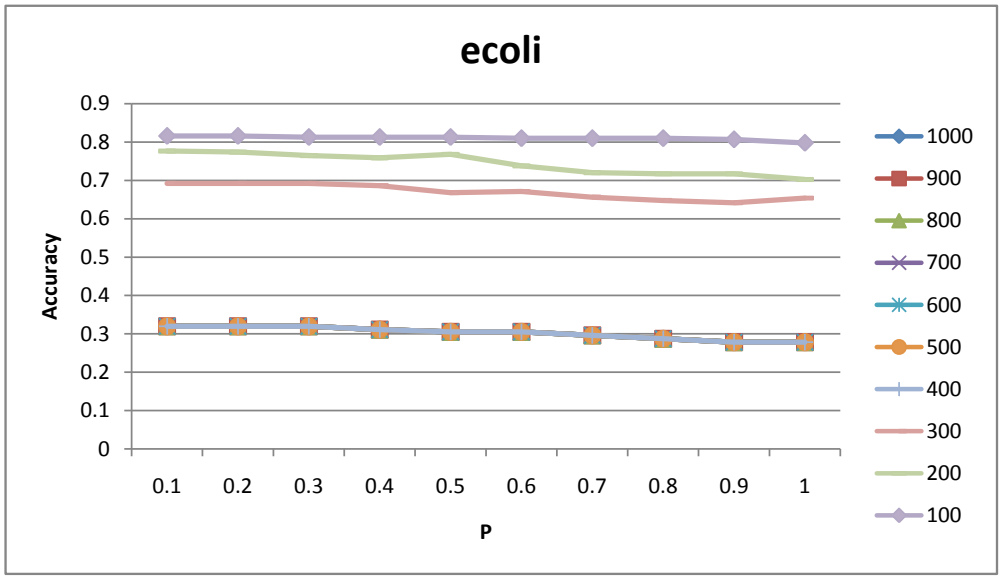
Figuur B.22: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'car', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



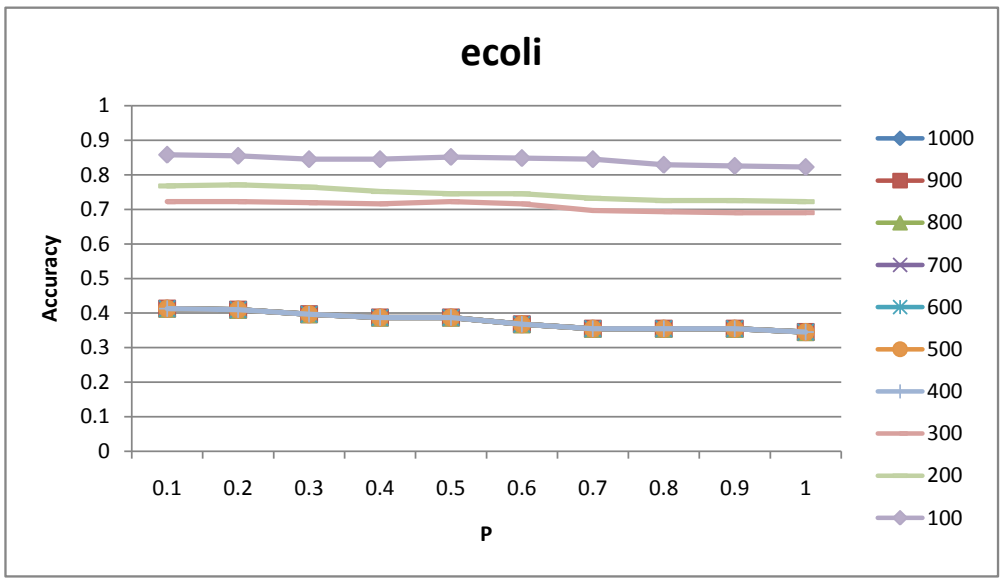
Figuur B.23: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'cmc', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



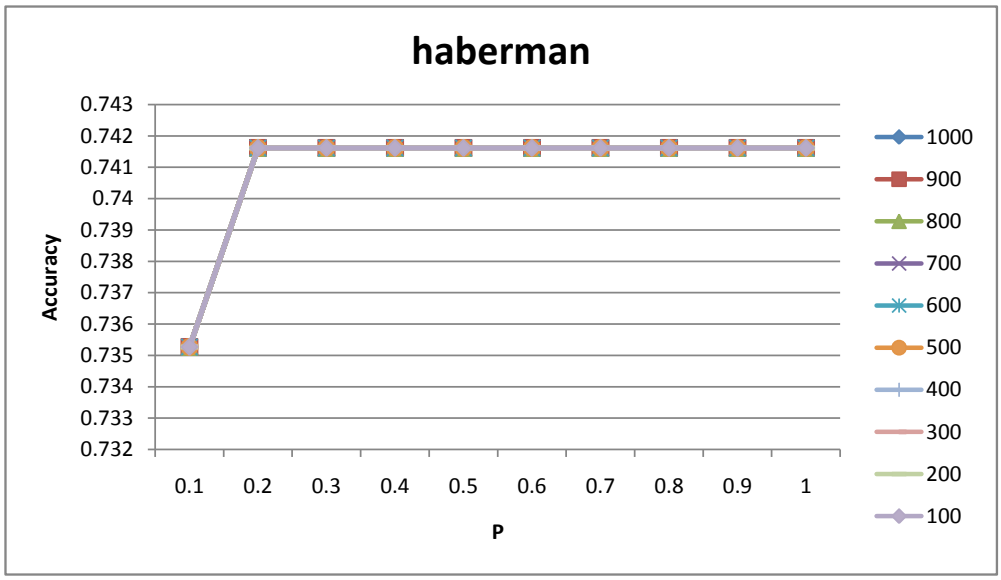
Figuur B.24: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'cmc', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



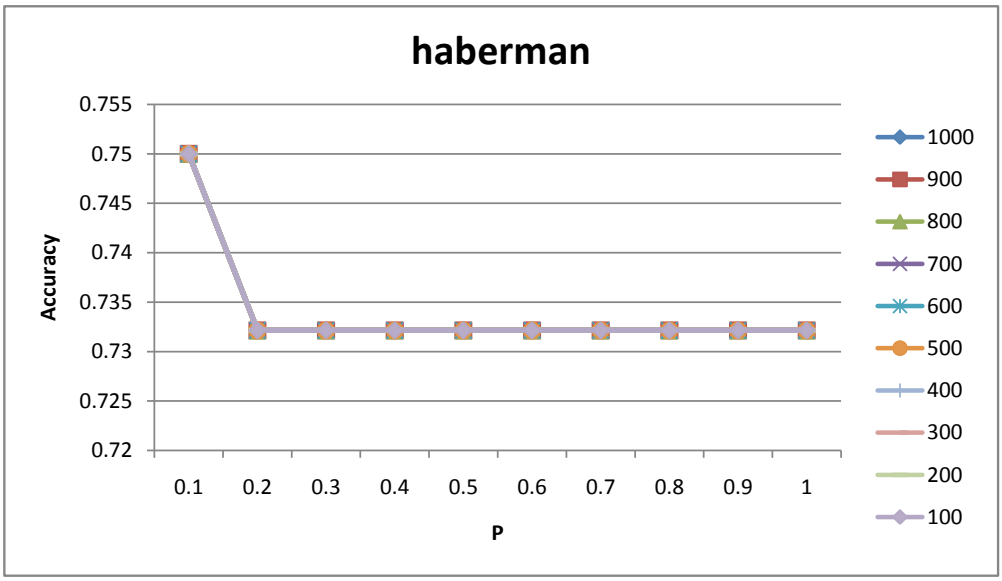
Figuur B.25: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'ecoli', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



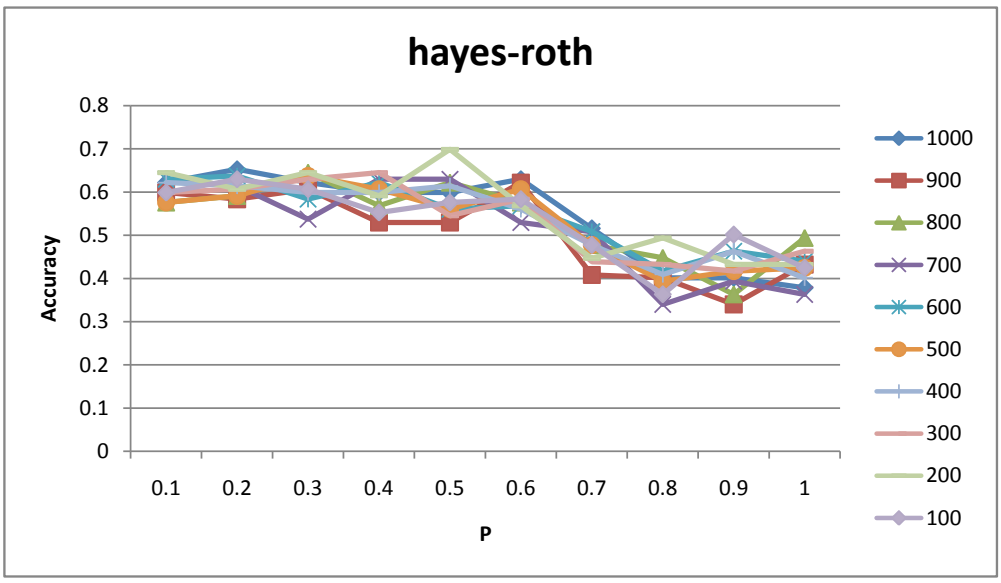
Figuur B.26: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'ecoli', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



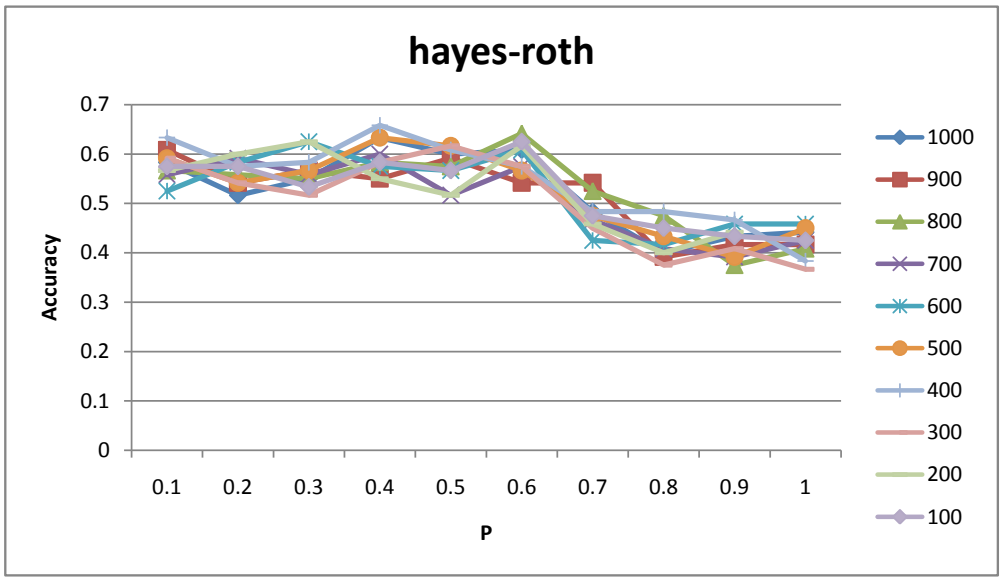
Figuur B.27: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



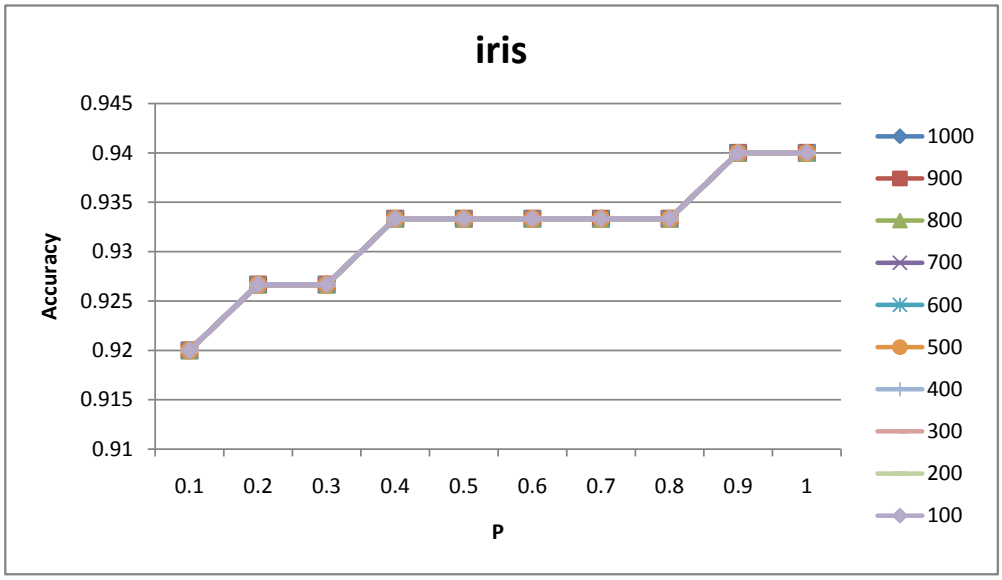
Figuur B.28: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



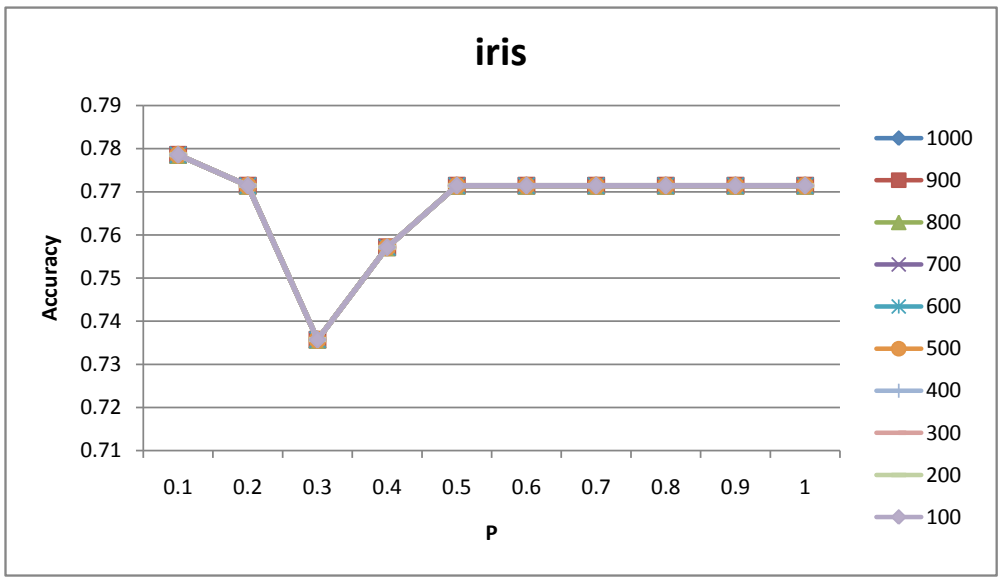
Figuur B.29: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



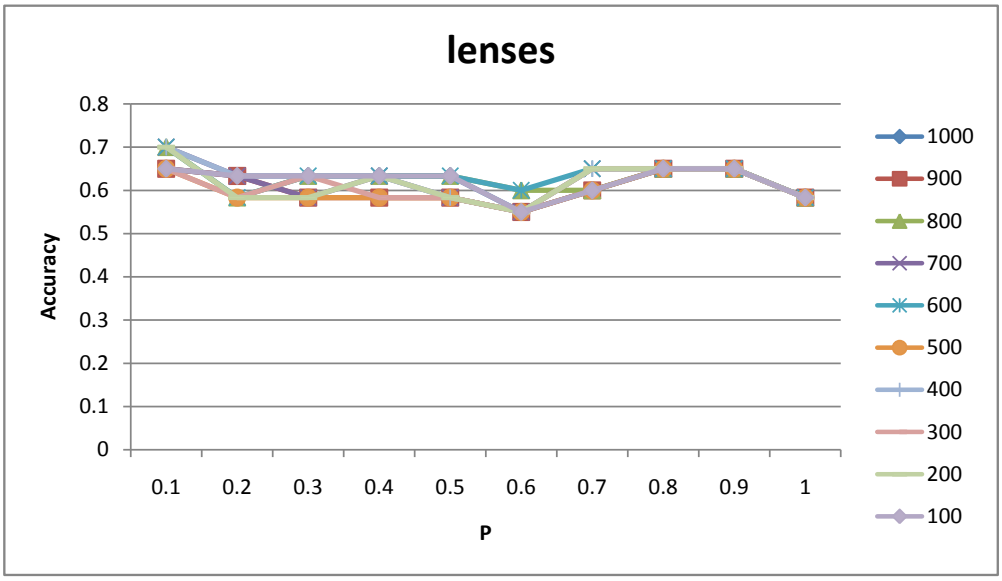
Figuur B.30: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'hayes-roth', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



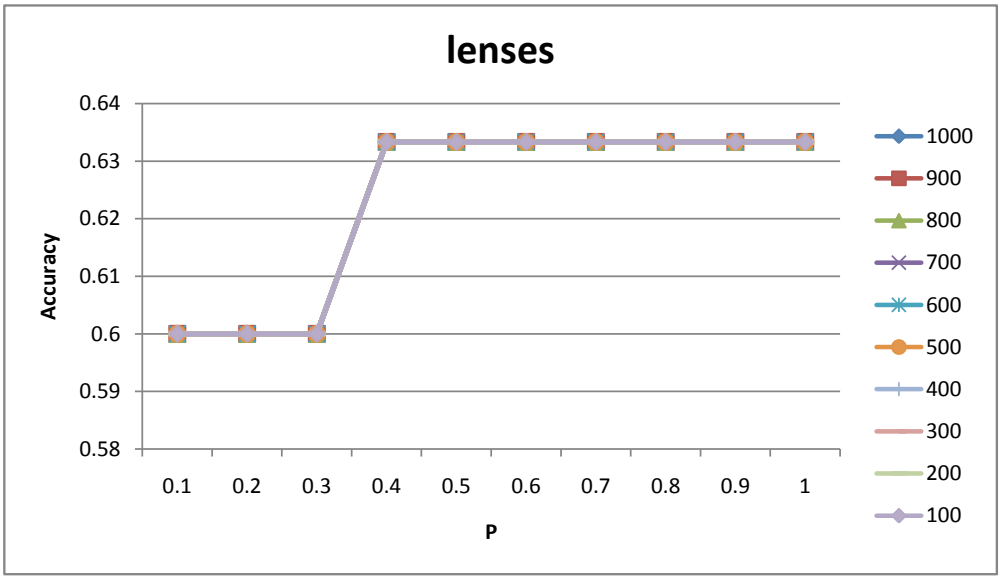
Figuur B.31: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



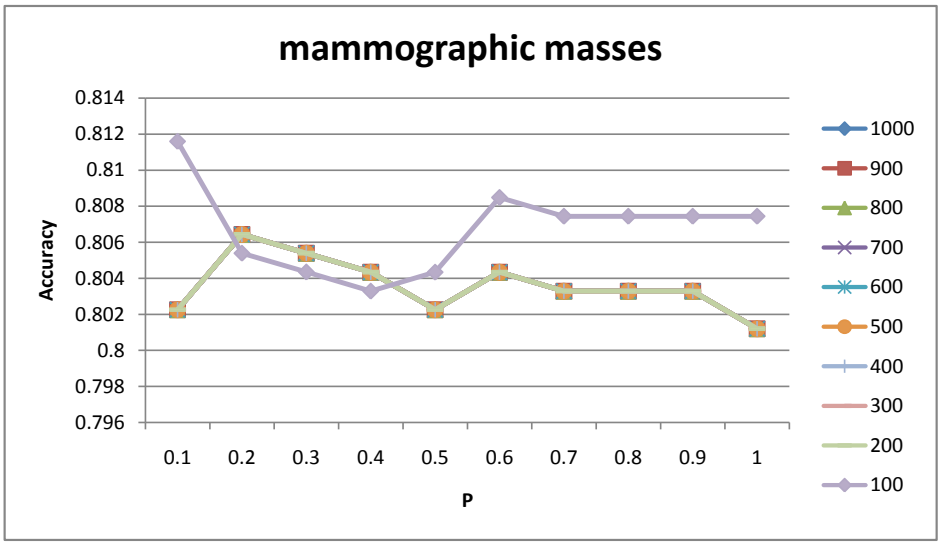
Figuur B.32: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



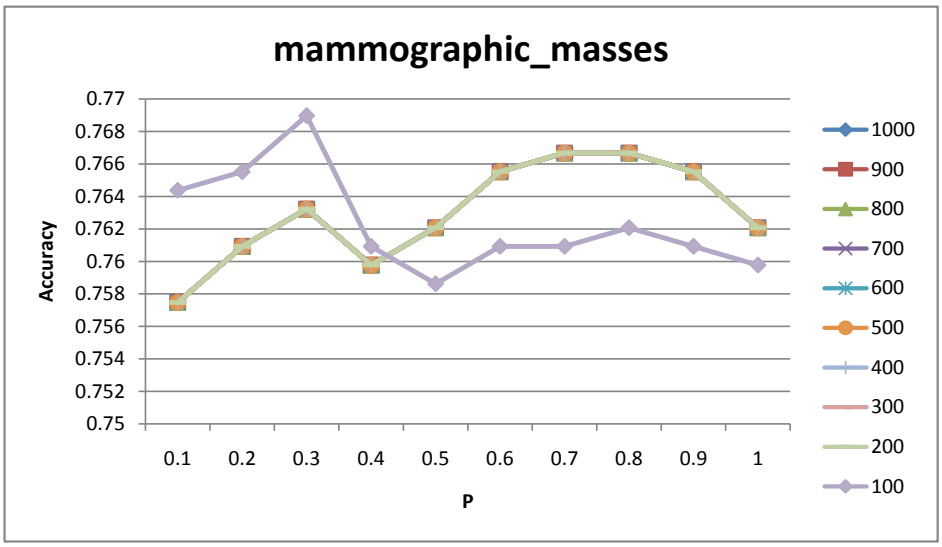
Figuur B.33: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



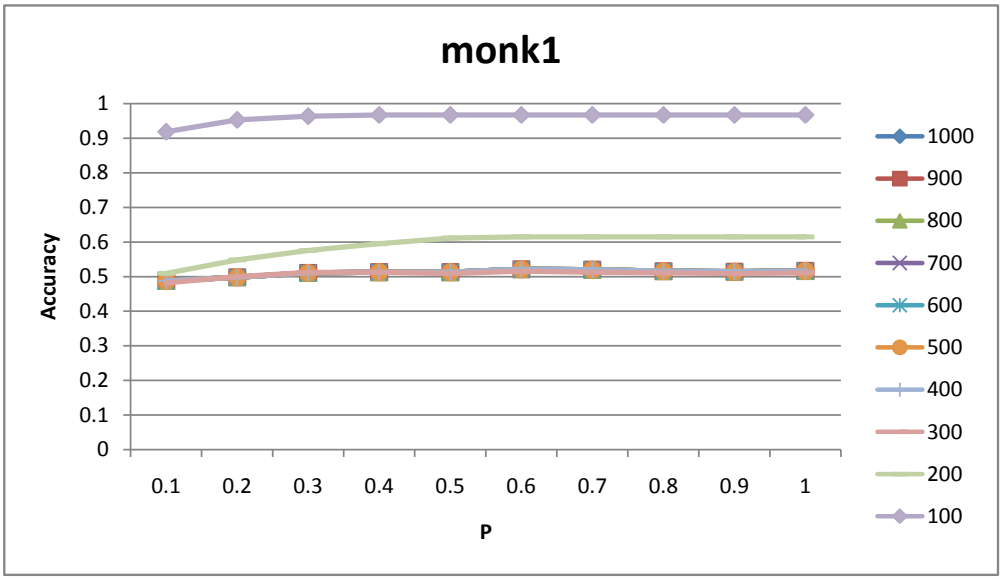
Figuur B.34: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



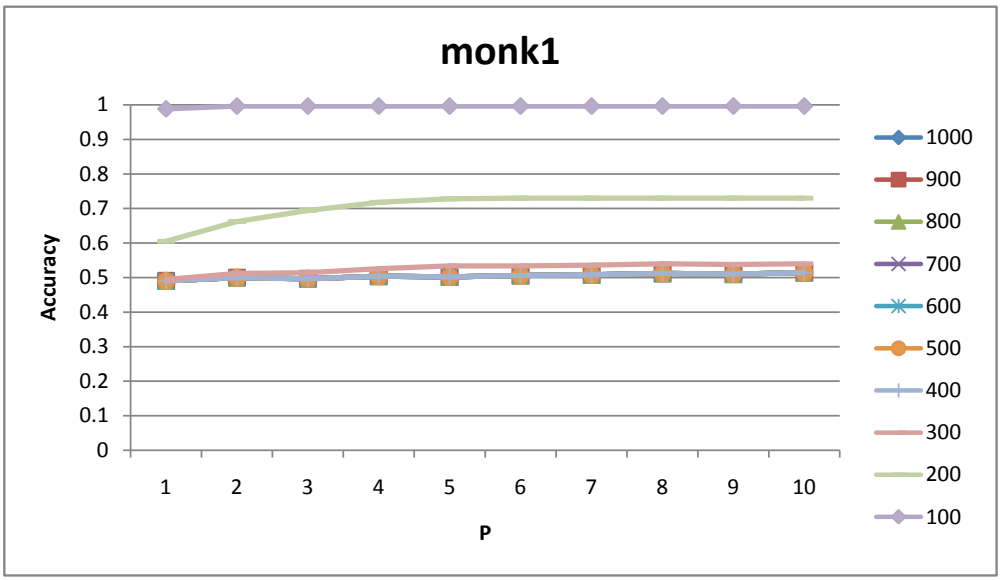
Figuur B.35: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'mammographic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



Figuur B.36: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'mammographic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

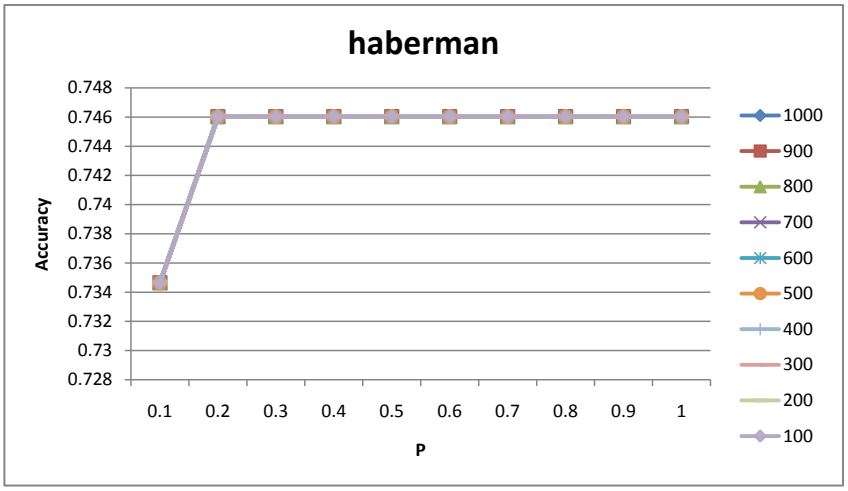


Figuur B.37: *Accuracies* verkregen met *tenfold cross validation* toegepast op UCI dataset 'monk1', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

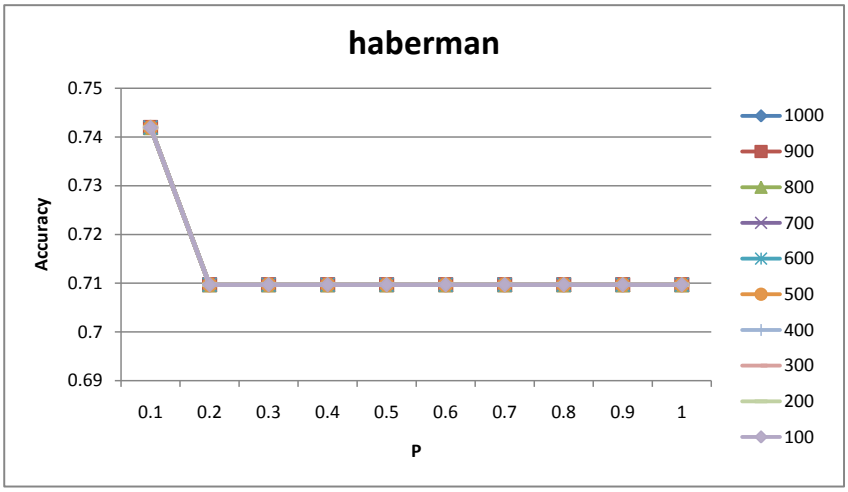


Figuur B.38: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 1 (zie Figuur 5.4 pagina 48) toegepast op UCI dataset 'monk1', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

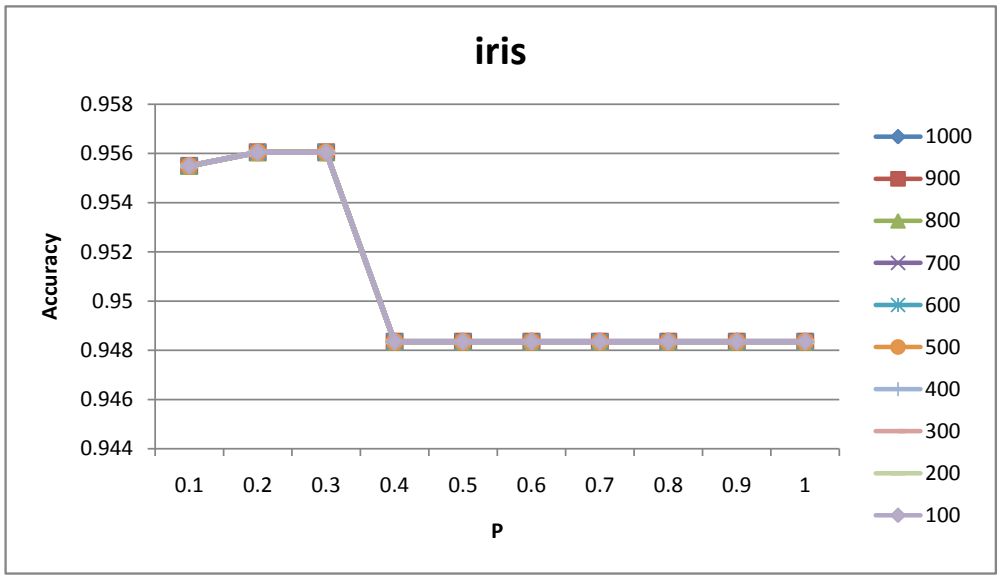
B.3 Optimalisatie - Methode 2



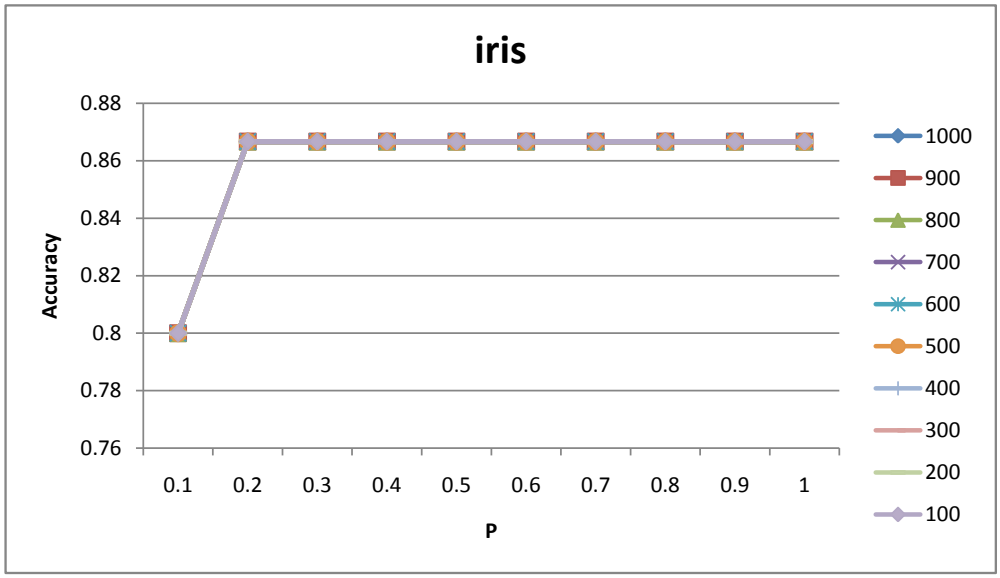
Figuur B.39: *Accuracies* verkregen op de classificatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



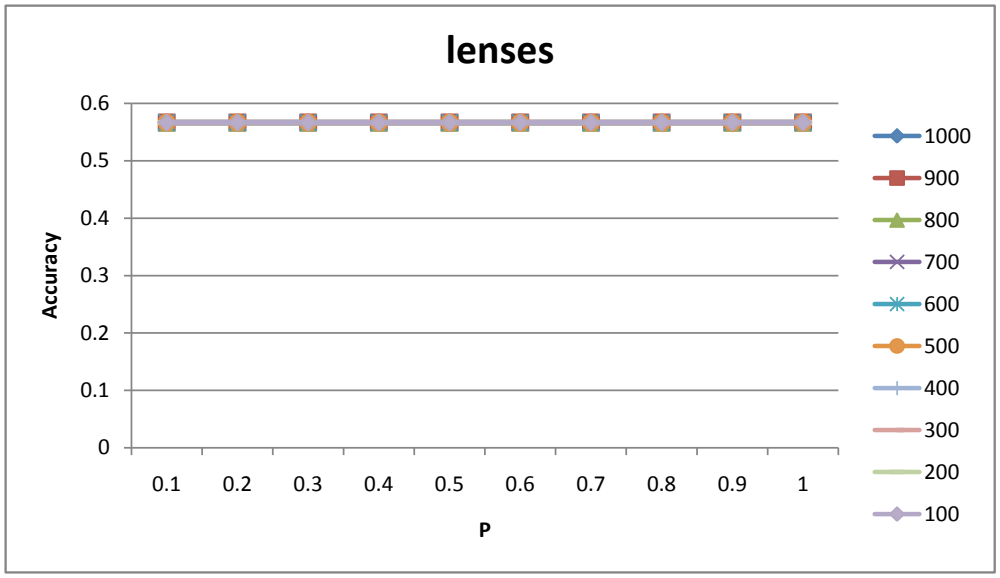
Figuur B.40: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'haberman', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



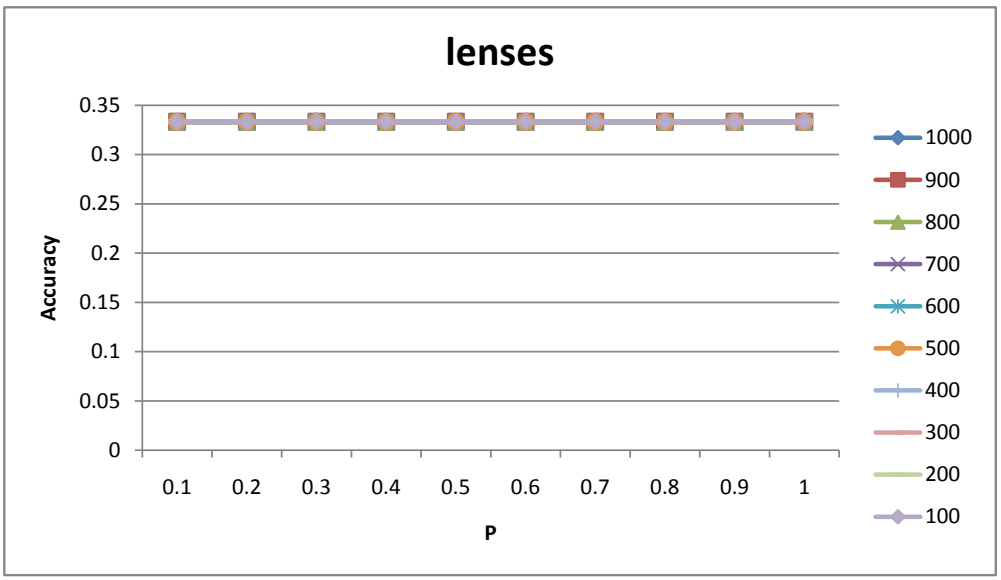
Figuur B.41: *Accuracies* verkregen op de classificatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



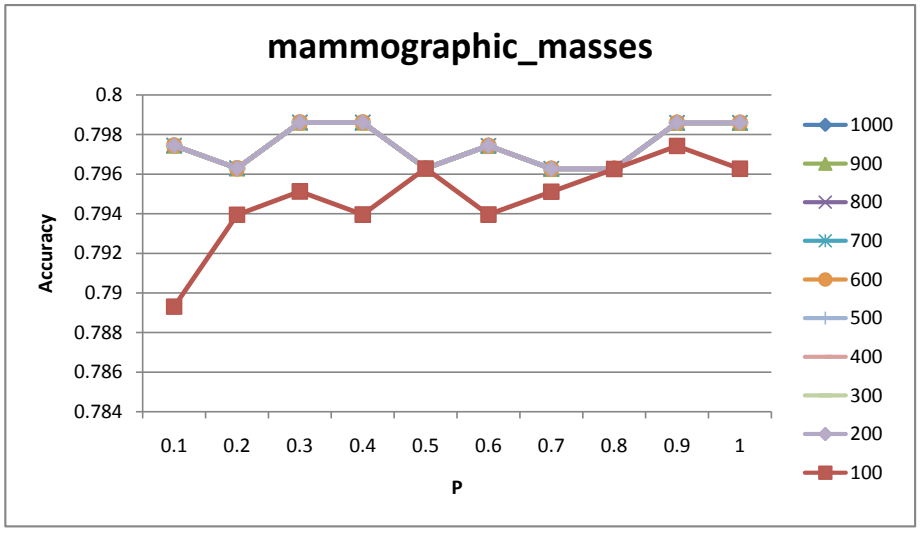
Figuur B.42: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'iris', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



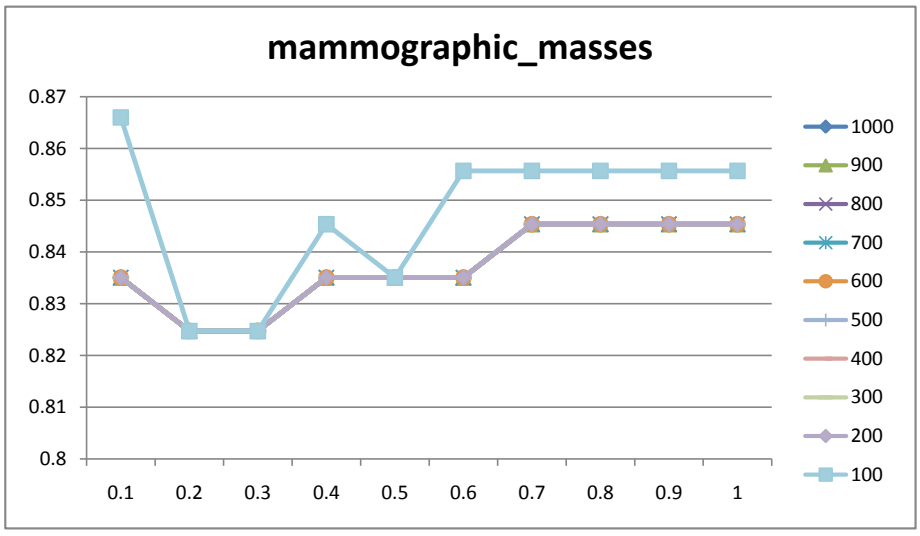
Figuur B.43: *Accuracies* verkregen op de classificatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.



Figuur B.44: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'lenses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

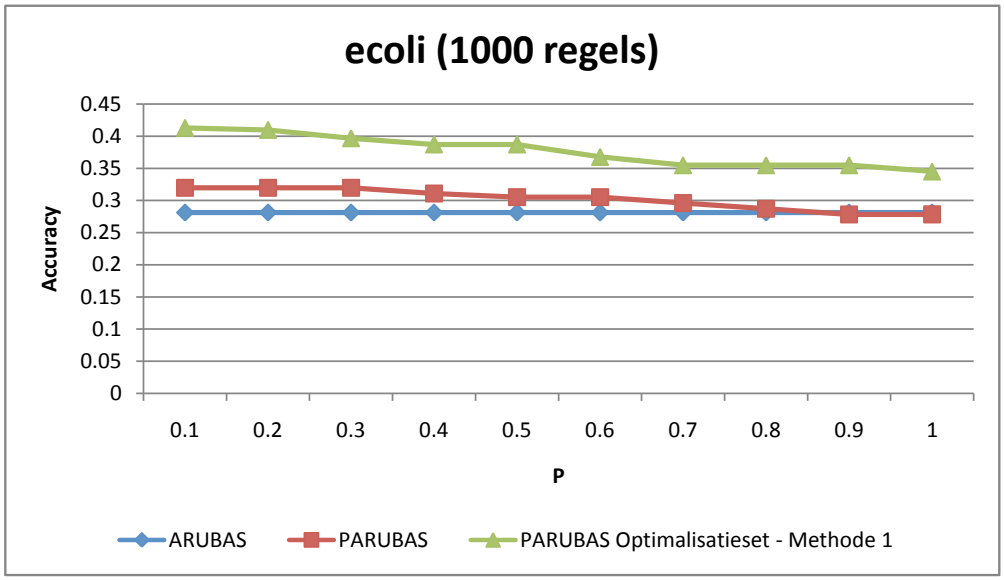


Figuur B.45: *Accuracies* verkregen op de classificatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'mammograhic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

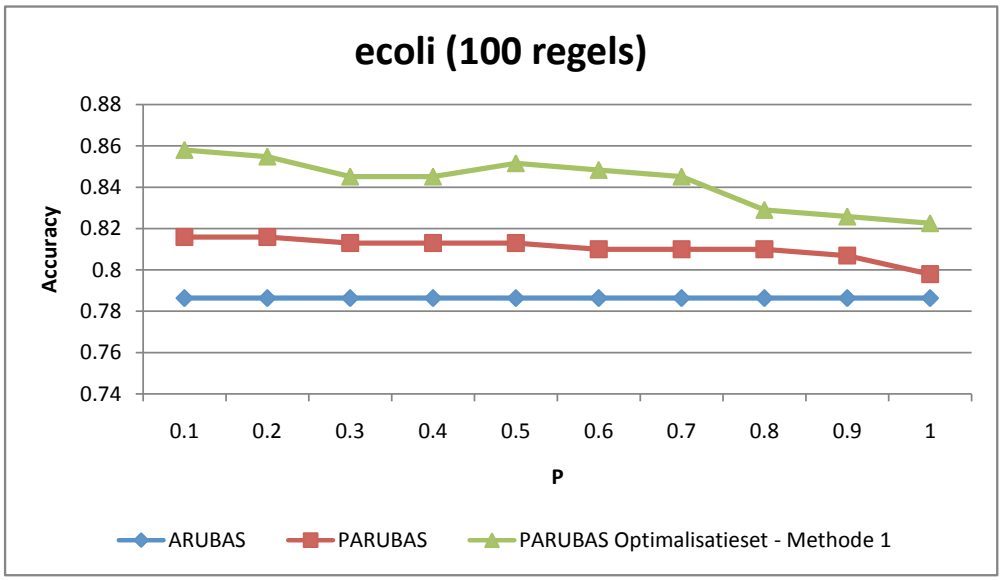


Figuur B.46: *Accuracies* verkregen op de optimalisatiegegevens, geproduceerd met Methode 2 (zie Figuur 5.5 pagina 50) toegepast op UCI dataset 'mammographic_masses', het aantal regels aangeleerd variërend van 100 tot 1000 met interval 100 en bij elk interval de fractie van instanties waarmee vergeleken wordt variërend van 0.1 tot 1 met interval 0.1.

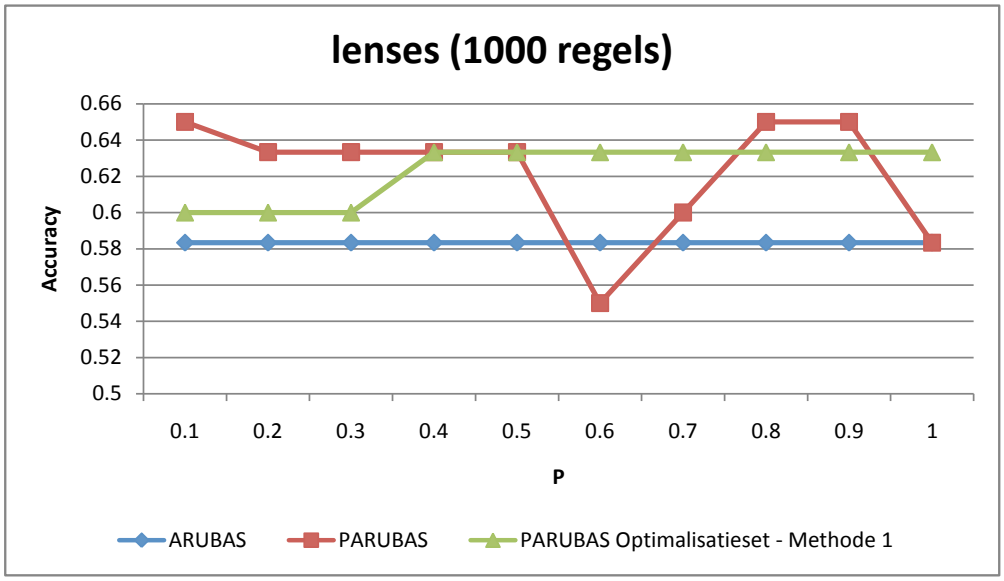
B.4 Grafische analyse



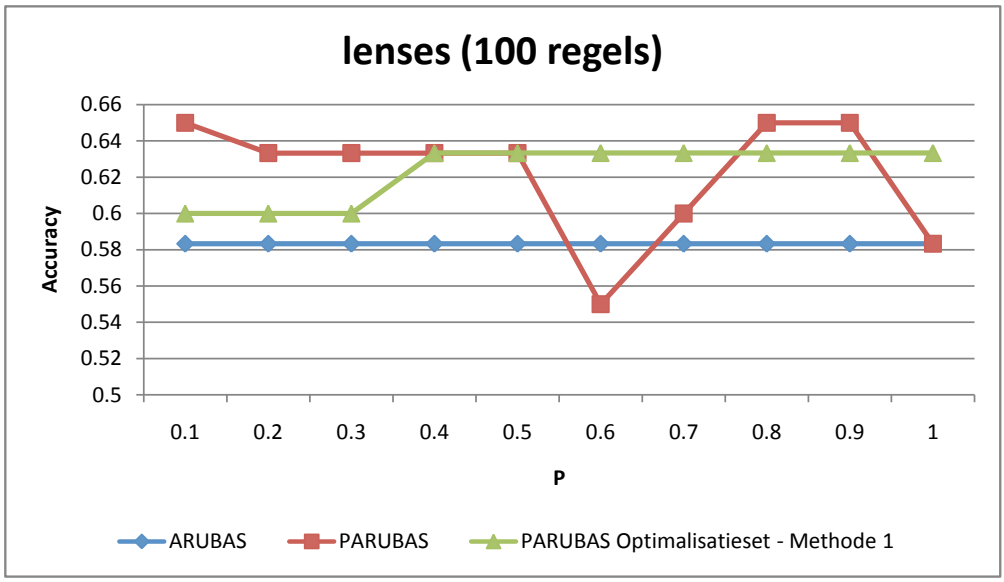
Figuur B.47: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset ecoli waarbij 1000 regels aangeleerd werden.



Figuur B.48: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset ecoli waarbij 100 regels aangeleerd werden.



Figuur B.49: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset lenses waarbij 1000 regels aangeleerd werden.



Figuur B.50: Grafische Analyse van ARUBAS en PARUBAS op UCI dataset ecoli waarbij 100 regels aangeleerd werden.