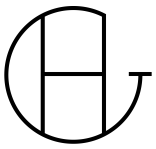


HoGent

Faculteit Bedrijf en Organisatie

Vergelijken voorspellingsalgoritmen met games

Matthias Seghers

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Nathalie Declercq

Instelling: ToThePoint

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Vergelijken voorspellingsalgoritmen met games

Matthias Seghers

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Nathalie Declercq

Instelling: ToThePoint

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

Voorwoord

Inhoudsopgave

1. Inleiding

1.1 Stand van zaken

De onderzoeken en technieken om de noden van gebruikers te voorspellen kent de laatste jaren een grote opmars. Google en Facebook zijn dan ook volop bezig met eigen onderzoekscentra en technologieën te ontwikkelen om aan Artificiële Intelligentie te doen. FAIR is de afkorting voor Facebook Artificial Intelligence Research. Er zijn wereldwijd 3 labo's die constant opzoek zijn naar nieuwe mogelijkheden binnen AI (Artificiële Intelligentie). RankBrain is een algoritme van Google die a.d.h.v. artificiële intelligentie de ranking van een bepaalde site op zoekpagina's bepaalt. De persoonlijke advertenties die Google toont zijn veelal gegenereerd met algoritmes. Maar ook gerelateerde producten of 'wat jou ook kan interesseren'-lijsten worden dikwijls door machine learning opgemaakt. Tensorflow („Tensorflow framework", g.d.) is een open source API ontwikkelt door Google die je uiteraard gratis kan gebruiken om zelf toepassingen met AI te maken. De API is ontwikkeld in Python, deze taal is dan ook een van de veel gebruikte in de data science(„The Most Popular Languages for Data Science", g.d.).

In deze bachelorproef zullen er voorspellingen gemaakt worden op een arcademachine (1.1). ToThePoint zal het eerste bedrijf zijn die AI in een arcademachine zal verwerken. Aangezien er nog geen andere gelijkaardige voorbeelden te vinden zijn zal ervan bij het begin onderzoek gedaan moeten worden. Hoe dit best wordt aangepakt en verklaren waarom. De spelletjes worden bestuurd door zes knoppen en een joystick. Aan de hand van de snelheid van een spel, aantal keer een knop ingedrukt wordt of joystickbeweging is gedaan zullen we kunnen onderscheiden welk spel de gebruiker aan het spelen is. Er bestaan al veel verschillende soorten algoritmen maar deze zijn niet allemaal geschikt voor deze casus en daar zal dus aan gewerkt moeten worden.

Figuur 1.1: De arcade machine van ToThePoint

Artificiële intelligentie is hot en zo worden er allerlei beginnende frameworks ontwikkeld die reeds geïmplementeerde algoritmen bevatten of het eenvoudiger maken om ermee te starten. Tensorflow is een populair framework ontwikkelt door Google gemaakt in Python. Dit wordt gebruikt in verschillende producten van Google zoals Google's stemherkenning, Google vertaler, het zoeken op afbeeldingen, etc. TensorFlow maakt gebruik van deep learning-algortimen of neurale netwerken. De voorbeelden van Tensorflow gaan bijna uitsluitend over image recognition maar het is mogelijk om andere applicaties ermee te ontwikkelen. Dit is dan ook de reden dat Google dit framework open source heeft gemaakt zodat ze kunnen zien waar er nog verbeteringen aangebracht kunnen worden en wat er nog allemaal mogelijk is. (Muoio, g.d.) Een ander interessant framework is het accord-framework („Accord framework", g.d.) die volledig ontwikkeld is in C#. Hierin zitten veel geïmplementeerde algoritmen die dan kan gebruikt kunnen worden door derden om een eigen applicatie met artificiële intelligentie te ontwikkelen.

Er zijn al enkele vergelijkingen over algoritmen geweest. Eén daarvan is een vergelijkende studie van verschillende gesuperviseerde algoritmen (Niculescu-Mizil, g.d.) Daarin was te vinden dat het support vector machine-algoritme tot de beste gesuperviseerde algoritmen behoort. Doorheen deze bachelorproef zullen we dit algoritme bespreken en vergelijken met logistische regressie. Er is wel vermeld dat de resultaten afhankelijk is van welke dataset gebruikt wordt dus misschien zal in deze casus logistische regressie de voorkeur krijgen.

1.2 Probleemstelling en Onderzoeksvragen

Machine learning is momenteel aan het boomen. Dit wordt nu zeer veel toegepast voor online advertising met Google en Facebook als de leiders. Maar ook meer en meer bedrijven beginnen zich te verdiepen in artificiële intelligentie. ToThePoint is zich hierop ook aan het voorbereiden. Doormiddel van een funproject willen ze zich zoveel als mogelijk verdiepen in allerlei gebieden binnen de informatica.

Men heeft een arcade machine gekocht die ze volledig gaan customizen met verschillende technologieën. Daar kunnen ze al hun kennis op loslaten. Inclusief de kennis die ze zullen opdoen via deze bachelorproef. Hierdoor zullen ze dus iets bijleren over artificiële intelligentie en dit dan ook kunnen toepassen. Verder zal de machine geplaatst worden op allerlei jobbeurzen om gegevens van studenten op te slaan bijvoorbeeld. Het nut van de machine is niet alleen om bij te leren maar hij zal ook dienen als referentie naar klanten toe. Zo kan ToThePoint aantonen tot wat ze instaat zijn.

In deze bachelorproef wordt er een vergelijkende studie gemaakt over twee verschillende algoritmen. Het beste algoritme zal dan uiteindelijk geïmplementeerd worden in de arcade machine. Je kan hier heel ver in gaan. De voorspellingen kunnen bijvoorbeeld gemaakt worden door wat de meest gebruikte knop is of joystick beweging. Maar ook door de snelheid dat knoppen bediend worden en de frequentie van dezelfde knop, is het spel eerder een multiplayer spel is of niet,... aan de hand van deze verschillende factoren is het mogelijk om voorspellingen te doen. Als het nog verder uitgewerkt wordt kan er naar

toetsencombinaties gekeken worden maar dit is te vergaand voor deze bachelorproef. Er zal vooral gefocust worden op hoe een goed algoritme gekozen wordt. Het stappenplan die uitgewerkt zal worden kan dan ook toegepast worden op toekomstige projecten. Uiteindelijk zullen we een antwoord hebben op de vraag welk algoritme het meest geschikt is om voorspellingen te doen op een arcademachine.

1.3 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Methodologie

2.1 Keuze algoritmen

Om goede resultaten te behalen is het uiterst belangrijk dat de juiste algoritmen gebruikt worden. Zo zijn er bepaalde algoritmen die helemaal niet bruikbaar zouden kunnen zijn voor deze casus. Er zal hieronder verder uitgelegd worden hoe we een mogelijk algoritme kunnen vinden.

2.1.1 Superviseerd vs Ongesuperviseerd leren

Machine learning kan onderverdeelt worden in verschillende types van algoritmen. Deze zijn gesuperviseerd leren, ongesuperviseed leren en reinforcement leren. Dit laatste type is niet van toepassing voor deze casus. Met dit type wordt er geleerd op basis van positieve signalen (beloningen). Er wordt ook niet gebaseerd op een dataset en aangezien wij in bezit zijn van datasets is dit type overbodig om verder te onderzoeken.

Superviseerd leren

Dit type heeft als doel om een hypothese te bekomen die dan zal kunnen gebruikt worden om nieuwe ongekende input toe te wijzen aan een label die voorkwam in de eerdere trainingsdataset. De trainingsdataset bestaat uit verschillende parameters en een label. Onder dit type kan je algoritmen vinden die voor zowat alle cases gebruikt kunnen worden. Deze zijn echter wel nog opgedeeld in drie verschillende categorieën. Zo kan je een nieuwe waarde voorspellen op basis van vroegere resultaten, dit wordt regressie genoemd. Verder heb je classificatiealgoritmen hiermee kan een input toegewezen worden aan een

bepaalde klasse met een label. En als laatste bestaan er clusteringsalgoritmen hiermee kan je ook onderverdelingen maken in klassen maar deze hebben geen label. Clustering en classificatie lijken op elkaar maar met classificatie weten we ook precies wat de data voorstelt.

Ongesuperviseerd leren

Verschillend met gesuperviseerd leren beschikt een ongesuperviseerd algoritme over een ongelabelde dataset. Er zijn verschillende inputs/parameters maar die behoren niet tot een specifieke klasse. Met dit type is het dan ook enkel maar mogelijk om aan clustering te doen. Als we dit willen toepassen voor deze casus is dit geen optimale manier. We kunnen wel bepaalde besturingsevents clusteren waardoor je tot een x aantal clusters kan komen maar we hebben geen idee of de ene cluster Pacman of Mortal Kombat voorstelt.

2.2 Gesuperviseerde classificatie algoritmen

In deze casus beschikken we over een gelabelde dataset. Het doel is om met een gegeven input een concreet spel te krijgen als output. Doordat we een gelabelde dataset hebben en er verdelingen moeten gemaakt worden op basis van labels is een gesuperviseerd classificatiealgoritme de meest geschikte voor dit onderzoek. Op deze manier kunnen we ongekende input plaatsen bij één bepaald spel.

2.2.1 Logistische regressie

Logistische regressie is het eerste algoritme die we zullen bespreken in deze bachelorproef. Ondanks de naam doet vermoeden, valt dit algoritme niet onder de categorie 'regressie' zoals we in sectie 2.1.1 superviseerd leren hebben gezien. Dit is wel degelijk een classificatiealgoritme die gebruik maakt van de logistische functie. Aan de hand van de hypothese kunnen er voorspellingen gemaakt worden. Een hypothese is een functie met x-aantal onbekenden waar de inputvectors ingevuld zullen worden. Gedurende het trainingsproces wordt de hypothese geoptimaliseerd. Er zullen meerdere spelletjes voorzien zijn op de arcademachine dus ook meerdere klassen. We starten binaire logistische regressie die later zal gebruikt worden bij twee mogelijke methoden om voorspellingen te maken met meerdere klassen.

Binaire logistische regressie

Er moeten enkele puntjes uitgelegd worden voordat we aan de effectieve uitleg kunnen beginnen. Eerst en vooral moeten we weten wat we willen voorspellen. Aangezien we beginnen met binaire logistische regressie maken we een voorspelling tussen twee klassen. Onze vraag kan dus als volgt luiden "Speelt de gebruiker Mortal Kombat of niet?". De waarde y zal ons het resultaat geven. Y kan slechts twee waarden aannemen $y \in \{0,1\}$

met 1 als de positieve klasse, Mortal Kombat. Als y 0 is dan duidt het op de negatieve klasse, in onze data stelt dit Pacman voor.

Logistische regressie start vanuit de hypothese van lineaire regressie die als volgt is:

$$h(x_i) = \theta^T x_i = \theta_0 + \theta_1 x_{i1} + ... + \theta_n x_{in}$$

θ^T is de getransponeerde vector van parameters die door het algoritme gegenereerd worden. Als de uitkomst van deze vergelijking ≥ 0 dan zal de positieve klasse voorspelt worden, de andere klasse zal dan gegeven worden wanneer $h(x) < 0$.

In figuur 2.1 op pagina 16zien we de sigmoïdfunctie getekend. Daaruit kunnen we afleiden dat wanneer $\theta^T x_i = 0$ de sigmoïd functie gelijk zal zijn aan 0.5. Beide klassen hebben dus even veel kans om gekozen te worden. Naar mate $\theta^T x_i$ van waarde verhoogt zal de waarschijnlijkheid voor de positieve klasse ook verhogen.

De logistische regressie is niet meer dan de sigmoïd functie van die lineaire hypothese. Zo krijgen we $h(x) = g(\theta^T x)$. De sigmoïd functie wordt ook wel logistische functie genoemd. Vandaar de naam logistische regressie.

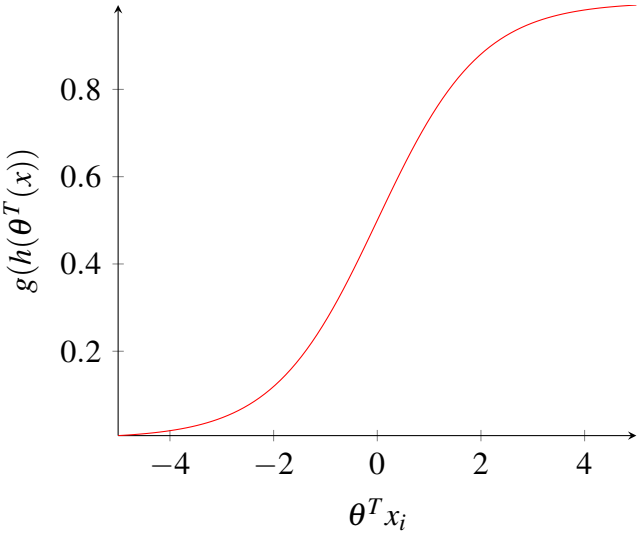
$$g(x) = \frac{1}{1+e^{-x}} => h(x) = \frac{1}{1+e^{-\theta^T x}}$$

De eigenschap van een sigmoïd functie is dat het resultaat altijd zal voldoen aan volgende voorwaarde: $0 \leq h(x) \leq 1$. Nu weten we nog niet wat de waarde van $h(x)$ voor logistische regressie precies uitdrukt. Dit kunnen we wiskundig uitdrukken als $P(y = 1|x;\theta)$. M.a.w. de kans dat $y = 1$ voor de gegeven vector x met de parameters θ. Wanneer je de kans op de negatieve klasse wenst te weten doet u eenvoudig weg $1 - P(y = 1|x;\theta)$. X kan nu gemakkelijk geclassificeerd worden. Wanneer $h(x) \geq 0.5$ heeft x de meeste kans om tot de positieve klasse te behoren in dit geval Mortal Kombat. Anderzijds als $h(x) < 0.5$ zal x behoren tot de negatieve klasse.

Multiklasse logistische regressie

Wat u uit de naam al kan afleiden is dat dit een algoritme is om een classificatie te maken over meerdere klassen. Dit kan met twee methoden gedaan worden. De one-vs-one methode of de one-vs-rest(/all) methode.

One-vs-all Dit is de gemakkelijkste van de twee methoden. Zoals de naam al doet vermoeden vergelijken we één klasse tegenover alle andere klassen. Als er n aantal klassen zijn dan zullen er n hypothesen $h^{(k)}$ met $k \in \{Pacman, Mortal\,Kombat, Tetris, ..\}$ gemaakt worden. k stelt de positieve klasse voor en alle andere klassen samen is dan één negatieve klasse.
Voor een nieuwe inputvector x die moet geclassificeerd worden gaat de methode als volgt te werk. $h^{(k)}(x)$ geeft een waarde terug die de waarschijnlijkheid uitdrukt dat x tot de klasse k behoort. Dit wordt voor alle n hypothesen gedaan. De klasse met de hoogste waarschijnlijkheid wordt dan logischerwijs gekozen als de voorspelde klasse waar x toe behoort.



Figuur 2.1: Sigmoïd functie

One-vs-one Met deze techniek vergelijken we twee klassen met elkaar zoals we gezien hebben in sectie 2.2.1 binaire logistische regressie. Er worden dus opnieuw meerdere hypothesen gemaakt maar anders dan in one-vs-all worden er nu combinaties van twee klassen gebruikt wat er zo uitziet $h^{(k,m)}$. k is in dit geval de positieve klasse en m de negatieve. In totaal zullen er $n(n-1)/2$ hypothesen gemaakt worden. Om te bepalen tot welke klasse een nieuwe inputvector x behoort berekenen we $h^{(k,m)}(x)$. Wanneer $h^{(k,m)}(x) \geq 0.5$ krijgt de positieve klasse (k) een "punt". Dit gebeurt voor alle hypothesen en de klasse met het meest aantal punten zal uiteindelijk het resultaat zijn van het algoritme.

Optimaliseren van algoritme

Om een zo'n correct mogelijke hypothese te verkrijgen moet er gebruik gemaakt worden van optimalisatietechnieken. Gradiënt descent is zo'n techniek die de parameters θ in de hypothese optimaliseert. Voordat gradiënt descent uitgelegd kan worden moet eerst de kostfunctie besproken worden.

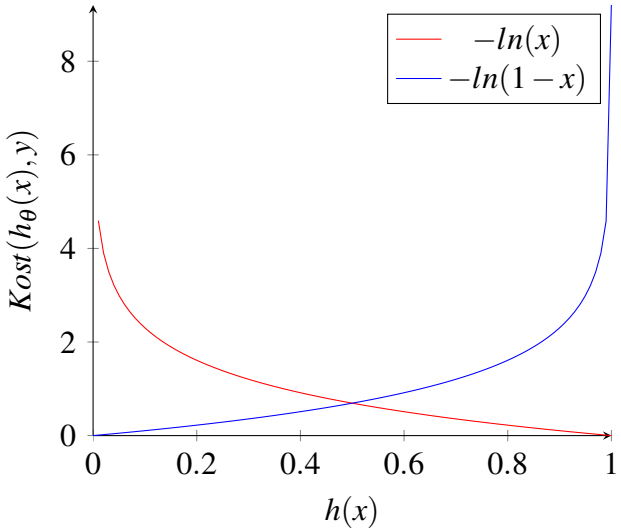
Kostfunctie logistische regressie

Een kostfunctie wordt genoteerd als $J(\theta)$. Die functie drukt de gemiddelde kost van de trainingsset met parameters θ uit. Aan de hand van gradiënt descent is het dan mogelijk om de parameters te optimaliseren zodat de kostfunctie geminimaliseerd wordt. Hoe lager de kostfunctie is hoe beter de hypothese. De kostfunctie voor logistische regressie ziet er als volgt uit:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Kost(h_\theta(x^{(i)}), y^{(i)})$$

De kost wordt als volgt berekent:

$$Kost(h_\theta(x), y) = -y\ln(h_\theta(x)) - (1-y)\ln(1-h_\theta(x)) \quad met\ y \in \{0,1\}$$



Figuur 2.2: Visualisatie functies

We stellen y gelijk aan 1. Dan zien we dat eigenlijk enkel het eerste deel van de functie $(-y\ln(h_\theta(x)))$ van belang zal zijn want het tweede deel $(-(1-y)\ln(1-h_\theta(x))$ zal nul zijn omdat 1-y dan nul zal zijn en dus het product ook nul is. Zo krijgen we:
$Kost(h_\theta(x),y) = -\ln(h_\theta(x))$. Op deze manier komen we uit bij de rode kostfunctie die u ziet in figuur 2.2 p17.

$h_\theta(x)$ is de hypothese die uitgelegd is in 2.2.1 binaire logistische regressie. Dus die drukt de waarschijnlijkheid uit dat een vector x positief of negatief is. Wanneer we een perfect voorspelling willen dan zou de kost 0 moeten zijn enkel zo ben je volledig zeker dat het label die aan x wordt toegekend 100% positief of negatief is. Als de hypothese zo goed als zeker is dat het vector positief is zal de kost nog dicht bij nul zijn. Eens $h(x) > 0.5$ zal de kost sneller stijgen. Ditzelfde principe geldt voor de negatieve voorbeelden. De bedoeling is om de kost zo laag mogelijk te krijgen.

De kostfunctie in één uitdrukking is:

$$min \ -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\ln(h_\theta(x^{(i)})) - (1-y^{(i)})\ln(1-h_\theta(x^{(i)}))$$

Er zijn nog andere kostfuncties die gebruikt kunnen worden maar deze wordt in het algemeen altijd gebruikt. De functie kan afgeleid worden door een principe binnen statistiek namelijk het 'maximum likelihood estimation'-principe Dit is een manier om parameters te zoeken voor verschillende modellen zoals logistische regressie. Deze functie is ook convex. Convex is een eigenschap die gebruikt wordt voor optimalisatiefuncties. Dit houdt onder meer in dat de functie het globaal minimum bereikt.

Gradient descent

Gradient descent is een optimalisatiealgoritme die veel gebruikt wordt voor logistische regressie. Op deze manier is het mogelijk om de parameters θ te minimaliseren zodat de

hypothese goede voorspellingen kan doen. Gradient descent vereist een continue afleidbare functie, de kostfunctie die we zonet besproken hebben voldoet aan die vereisten. Er is ook een functie g nodig die een vector teruggeeft als gradiënt en een stapgrootte α. De stapgrootte zorgt ervoor dat het algoritme naar een lokaal of globaal minimum kan convergeren. Wanneer het algoritme stopt in een lokaal minimum heb je niet altijd de beste oplossing tenzij het lokaal minimum ook het globaal minimum is. Indien dit niet het geval is kan gradient descent een aantal keer herhaald worden zodat het globaal minimum kan gevonden worden. Het belang van een goede stapgrootte α is belangrijk. Wanneer die te groot is dan zou het kunnen dat het algoritme in een oneindige lus geraakt. Of als α te klein is dan kan het zeer lang duren tegen dat er een lokaal minimum gevonden is.

Voor iedere parameter θ_n wordt volgende uitdrukking berekent en terug toegekend aan zichzelf. Tot de parameter amper nog verbetert. In implementaties is het mogelijk om te zeggen dat gradient descent moet stoppen wanneer het verschil van de nieuwe en oude parameter onder een bepaalde grens ligt.

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i}(J(\theta_i, ..., \theta n))$$

Deze functie kan afgeleid worden naar

$$\theta_i = \theta_i - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(Misschien nog voorbeeldje uitwerken en lokaal/globaal minimum uitleggen)

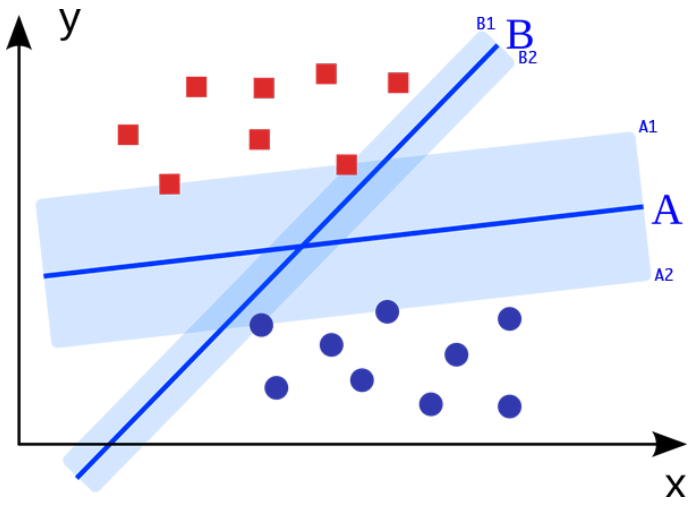
2.2.2 Support Vector Machine

Support Vector Machine is een volgend gesuperviseerd leeralgoritme die zal besproken worden. Andrew Ng is professor aan de Stanford University begon het deel support vectormachine in zijn cursus (Ng, g.d.) met volgende zin.

> SVMs are among the best (and many believe are indeed the best) 'off-the-shelf' supervised learning algorithms.

SVMs scoren over het algemeen beter dan logistische regressie of neurale netwerken (Niculescu-Mizil, g.d.). Het is uiteraard mogelijk dat een van die twee beter scoort maar dit is allemaal afhankelijk van de dataset. Vandaar dat het een logische keuze is om dit algoritme te vergelijken met ons vorige, logistische regressie.

We maken gebruik van figuur 2.3 op pagina 19 om de werking van een SVM uit te leggen. Daarna zal de kostfunctie uitgelegd worden die gebruikt wordt om het algoritme te optimaliseren. Dit alles is vereenvoudigd zodat het al mens makkelijker in te beelden is. Het concept van SVM klinkt eenvoudig. De bedoeling is om een hyperplane te vinden die tussen de dichtstbijzijnde vector van een klasse en de hyperplane een zo groot mogelijke



Figuur 2.3: Begeleidende afbeelding SVM („What is a support vector machine?”, 2016)

marge heeft. U ziet in de figuur dat het mogelijk is om meerdere hyperplanes te tekenen maar niet iedere hyperplane heeft een even grote marge. Om de marge beter te begrijpen gaan we eventjes terug naar logistische regressie. Daar kregen we een waarschijnlijkheid dat vector x van een bepaalde klasse is. Als $\theta^T x \geq 0$ was $y = 1$ en vice versa. Het is logisch dat hoe verder $\theta^T x$ van 0 ligt hoe zekerder de klasse kan toegekend worden. Dit is net wat SVM bijbrengt, een marge zodat je vanaf een bepaalde zekerheid, marge, pas een klasse toekent. Een marge wiskundige noteren is als volgt: $\theta^T x^i \ll 0$. '$\ll$' betekent 'veel minder'. Als deze uitdrukking waar is, stelt y^i de negatieve klasse voor.

Eerst en vooral wordt er een hyperplane gezocht die twee klassen van elkaar onderscheid. Zoals je in de figuur ziet is het mogelijk om meerdere hyperplanes (A,B) te maken. Die hyperplanes zijn van de vorm $h_{w,b}(x) = w^T x + b$ w stelt nu θ_n^T voor en b stelt θ_0 voor. y is nu geen element meer van 0,1 maar wel van 1,-1. Als $h_{w,b}(x) > 1$ dan kunnen we stellen dat voor vector x een positieve klasse kan voorspelt worden. We nemen nu hyperplane A om verder op te bouwen. We weten ook dat $A1$ het functie voorschrijft $wx + b = 1$ heeft en $A2$ dan $wx + b = -1$. Op deze manier is het mogelijk om de marge tussen $A1$ en $A2$ te berekenen. En die dus te optimaliseren.

Kostfunctie Support Vector Machine

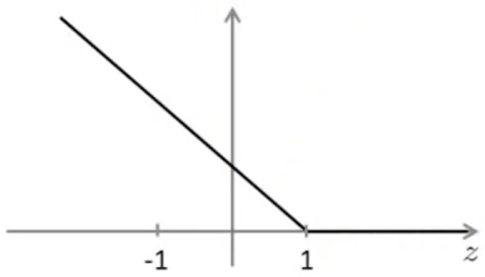
De kostfunctie van een SVM is gelijkaardig als de kostfunctie van logistische regressie. Er zijn enkele verschillen. Hier ziet u nog eens deze van logistische regressie:

$$min \ -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \ln(h_\theta(x^{(i)})) \ - \ (1-y^{(i)}) \ln(1-h_\theta(x^{(i)}))$$

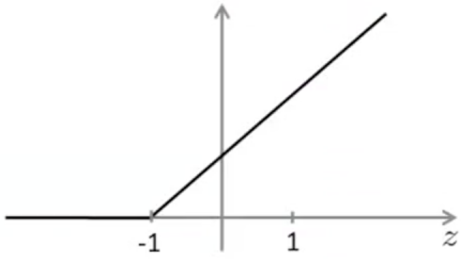
en de kostfunctie van een support vector machine is:

$$min \ C\sum_{i=1}^{m} \left[y^{(i)} \ cost_1(\theta^T(x^{(i)})) \ - \ (1-y^{(i)}) \ cost_0(\theta^T(x^{(i)})) \right]$$

Er zijn 2 grote wijzigingen. Het gedeelte $\ln(h_\theta(x^{(i)}))$ en $\ln(1-h_\theta(x^{(i)}))$ zijn vervangen



Figuur 2.4: $cost_1\left(\theta^T(x^{(i)})\right)$



Figuur 2.5: $cost_0\left(\theta^T(x^{(i)})\right)$

door respectievelijk $cost_1\left(\theta^T\left(x^{(i)}\right)\right)$ en $cost_0\left(\theta^T\left(x^{(i)}\right)\right)$ Die kost functies kan u zien in figuren 2.4 en 2.5. Als u de functies bekijkt zie u dat er eerst een plat stuk is en dan pas vanaf 1 of -1 begint te stijgen. Dit komt door de marge die we eerder besproken hebben. Wanneer $\theta^T\left(x^{(i)}\right) \geq 1$ dan kunnen we ervan uit gaan dat $y = 1$.

2.2.3 Selectiecriteria

Hoe weten we nu welk algoritme het best is om voorspellingen te doen op de arcade machine? We bespreken enkele selectiecriteria zodat de 2 algoritmen die we zonet besproken hebben kunnen vergelijken. Er zijn twee belangrijke factoren die beslissen of dat het ene algoritme beter is dan het andere voor een bepaalde dataset. Als we de algoritmen gaan implementeren dan zal altijd exact dezelfde dataset en testset gebruikt worden per fase. De algoritmen worden beiden geïmplementeerd in Visual Studio 2015 en worden uitgevoerd zonder dat er een ander programma openstaat of draait in de achtergrond. Tussen de test wordt er ook nooit een ander programma geopend want het is altijd mogelijk dat die daarna nog in de achtergrond processen uitvoert.

Snelheid

Snelheid is uiteraard een van de belangrijkste voorwaarden om een goed algoritme te zijn. Je wilt bijvoorbeeld geen 2 minuten wachten tegen dat je krijgt te weten wat de computer dacht dat je aan het spelen was. Een programma bestaat uit verschillende delen. Niet alle onderdelen van een programma zijn relevant voor deze bachelorproef. Zo heb je bijvoorbeeld het inlezen van de data. Dit proces zal even veel tijd in beslag nemen voor logistische regressie als voor vector support machine doordat dezelfde data gebruikt wordt voor beiden. Een tweede deel is natuurlijk de declaratie van variabelen en objecten dit gebeurt zodanig snel dat dit irrelevant is.
Dan heb je het leerproces of optimalisatieproces van een algoritme. Dit is wel belangrijk want hiermee kan het verschil gemaakt worden. Daarvoor gebruiken we de klasse Stopwatch die in het .NET framework voorzien is. Die wordt gestart net voor de start van het leerproces en gestopt net erna. Op deze manier kunnen we precies meten hoelang het algoritme nodig heeft om een hypothese op te stellen.
We hebben ook nog een gedeelte waar we een nieuwe vector laten voorspellen maar ook dit is enkel maar een wiskundige berekening wat voor de computer praktisch niets inhoudt.

F-score

Om een idee te hebben hoe goed een algoritme voorspellingen doet kan je een simpele test doen en kijken hoeveel procent van de testdataset correct werd voorspelt. Er is echter een manier om nog correcter weer te geven hoe goed een algoritme effectief is. Dit kan aan de hand van de F-score.
Wanneer een algoritme een voorspelling doet zijn er vier mogelijke uitkomsten onderverdeelt. Hieronder ziet u de correcte en foute resultaten:

- Correcte voorspellingen
  - Positieve voorspelling voor een positieve uitkomst (A)
  - Negatieve voorspelling voor een negatieve uitkomst (B)
- Foutieve voorspellingen
  - Positieve voorspelling terwijl de uitkomst negatief had moeten zijn (C)
  - Negatieve voorspelling terwijl de uitkomst positief had moeten zijn (D)

Om de F-score te begrijpen moeten er nog 2 begrippen uitgelegd worden. Als eerste zullen we beginnen met precisie.
Als je de effectieve juiste positieve voorspelling (A) deelt door het totaal aantal positieve voorspelling (dus inclusief positieve voorspellingen van de hypothese maar die eigenlijk negatief had moeten zijn). Dan bekom je een percentage die de precisie uitdrukt.

$$precisie = \frac{A}{A + C}$$

Rappel is een tweede begrip die eveneens een percentage uitdrukt. Dan wil je weten van alle positieve voorbeelden in de dataset (A + D) hoeveel de hypothese ook als positief heeft teruggegeven (A).

$$rappel = \frac{A}{A + D}$$

Precisie en rappel zullen dus altijd gelijk aan 1 zijn wanneer het algoritme alle testvoorbeelden het correcte label heeft gegeven. Stel dat een algoritme alle voorbeelden negatief labelt dan zal de rappel gelijk zijn aan 0 ($\frac{0}{0+D} = 0$). Voor precisie geldt het omgekeerde. De uiteindelijke formule voor de F-score is als volgt:

$$F - score = \frac{2 * precisie * rappel}{precisie + rappel}$$

De F-score zal altijd tussen 0 en 1 liggen. Hoe hoger de score hoe beter het algoritme is.

3. Uitwerking

3.1 Benodigdheden

3.1.1 Data

Een onmisbaar onderdeel om aan machine learning te doen is uiteraard de data. Tijdens de deze bachelorproef wordt er gebruik gemaakt van zelf gegenereerde data in Excel. Het systeem om data uit te lezen afkomstig van de arcademachine is niet tijdig klaar geraakt om te gebruiken. De data die uit de arcademachine ontvangen zal worden zal in CSV-formaat zijn. Doordat Excel-tabbladen opgeslagen kunnen worden in een CSV-file is dit dan ook een logische keuze. En zal ToThePoint geen extra moeilijkheden ondervinden als ze dit verder willen implementeren in de arcademachine. Er zullen zo'n 500 metingen beschikbaar zijn die kunnen ingelezen worden in het programma.

We willen uiteraard weten hoe goed het algoritme scoort dit kunnen we testen door testdata te voorzien. De dataset die we uit Excel halen zal opgesplitst worden in trainingsdata en testdata. Het algoritme zal met de trainingsdata een hypothese vormen die ervoor zal zorgen dat ook nieuwe of nog ongekende data een goede voorspelling krijgt. Daarom is het belangrijk om een dataset te hebben die data bevat die nog niet eerder gebruikt is door het algoritme. Enkel op deze manier kunnen we zeker zijn dat het algoritme een goede hypothese heeft gemaakt. Als testdata nemen we 20% van de hele dataset. Naar mate we verdere stappen doen in de ontwikkeling zal de dataset uitgebreid worden.

3.1.2 Frameworks

Zoals in de inleiding reeds besproken was zijn er verschillende soorten frameworks die reeds geïmplementeerde algoritmen hebben. Een eerste mogelijk framework was TensorFlow van Google. Dit is ontwikkeld in Python, de algoritmen die hierin voorzien zijn, zijn voornamelijk neurale netwerken of deep learning algoritmen. Het is mogelijk om ook eenvoudigere algoritmen te gebruiken maar daar ligt de specialiteit niet op. En dit in combinatie met een taal die ik niet machtig ben lijkt mij geen goede keuze. Er zijn nog een aantal andere frameworks die gemaakt zijn in andere programmeertalen zoals Javascript, Node.js, Java, etc.

Accord-framework is daar een van, dit is ontwikkeld in .NET. Alle algoritmen die nodig zijn om deze bachelorproef tot een succesvol einde te brengen zijn beschikbaar.
Door te werken met het Accord-framework bestaat er een mogelijk om in de toekomst een webapplicatie te ontwikkelen. Dit openend dus ook nog extra mogelijkheden om te experimenteren met Artificiële Intelligentie.
.NET is een nog niet zo'n populaire taal om aan artificiële intelligentie te doen maar er zit potentieel in. Doordat .NET vele gelijkenissen heeft met Java is er een nog groter publiek die hiermee aan de slag kan.
Door deze mogelijkheden lijkt dit dan ook het meest geschikte framework om aan het werk te gaan.

3.2 Fase 1

De eerste stap om een programma te maken om voorspellingen te doen is de data maken. Omdat dit ook nog maar de eerste fase is beginnen we gemakkelijk. Dit wil zeggen dat we de computer een eerste voorspelling laten doen tussen twee totaal verschillende spelletjes. We nemen Pacman en Mortal Kombat (schietspel) om te starten. Pacman kan gespeeld worden enkel d.m.v. joystickbewegingen. Om Mortal Kombat te spelen heb je veel de knoppen nodig maar ook de joystick. Nu als mens is het gemakkelijk om dit verschil te kunnen zien, als er knoppen gebruikt geweest zijn kunnen we concluderen dat de speler Mortal Kombat aan het spelen was.

3.2.1 Data

De eerste dataset is zelf gegenereerd zonder echte input van de arcademachine omdat het systeem die de data zal inlezen nog in ontwikkeling is bij ToThePoint. In Excel zijn er drie kolommen voorzien een kolom voor het aantal keer dat de knoppen ingedrukt zijn geweest gedurende twee minuten. Dan het aantal keer dat de joystick bewogen is in diezelfde tijdsspanne. De derde kolom staat het ID van het spel. In figuur 3.1 op pagina 25 ziet u random 10 voorbeelden uit de dataset. Het GameID 0 wijst op Pacman en 1 is dan Mortal Kombat. Als u de rijen van Pacman bekijkt dan valt op dat de ButtonPresses niet 0 zijn, dit komt omdat er bij het begin van een spel soms eens geprobeerd wordt wat de functies van de knoppen zijn.

	A	B	C
1	ButtonPresses	JoystickMovements	GameID
2	180	281	1,00
3	2	275	0,00
4	170	449	1,00
5	5	415	0,00
6	1	387	0,00
7	177	375	1,00
8	180	285	1,00
9	10	362	0,00
10	3	245	0,00

Figuur 3.1: Tien voorbeelden van games

3.2.2 Logistische regressie

Het eerste algoritme die we gaan testen is logistische regressie. In fase 1 gaan we slecht voorspellingen doen tussen twee verschillende spellen vandaar dat we binaire logistische regressie gaan toepassen die eerder uitgelegd is in sectie 2.2.1.

In de code 3.2 op pagina 26 ziet u de implementatie van de binaire logistische regressie. Er worden twee parameter meegegeven in de functie, input en output. De input is een dubbele array van *double* waarden. Daarin zitten rijen met twee kolommen die de ButtonPresses en JoystickMovements bevatten. De ouput parameter bevat een enkele array die het GameID bevat. Hoe deze data ingelezen wordt kan u zien in de code 3.3 op pagina 27. Met een ExcelReader krijgen we een DataTable die we met de methode ToJagged kunnen omvormen naar ons gewenste dubbele array met de kolomnamen kunnen we de kolommen selecteren. Deze methode is onderdeel van het Accord-framework.

Resultaten

Snelheid van het algoritme

Een eerste belangrijke metric om algortimen te kunnen vergelijken is de snelheid. Het zou niet correct zijn om een stopwatch te laten lopen bij het begin van het programma en te stoppen wanneer het programma klaar is. De tijd om de data in te laden in het programma, de omzetting naar arrays, de objecten die worden geïnitialiseerd, ... is allemaal niet zo belangrijk. Wat wel interessant is, is de tijd die het algoritme nodig heeft om tot een hypothese te komen. Dit gebeurt in de *Learn* methode. Net voor die lijn code wordt uitgevoerd starten we een stopwatch die direct erna wordt gestopt. Omdat de duur verschillend kan zijn nemen we het gemiddelde van 20 metingen. Doordat deze dataset slechts 500 voorbeelden bevat en de verschillen tussen de spelletjes Pacman en Mortal Kombat zeer duidelijk zijn heeft het algoritme slechts 4,5487 milliseconden nodig om een hypothese te maken. Dit resultaat is het gemiddelde van twintig testen.

```
public static void StartLogisticRegression(double[][]
   input, int[] output)
{
        //LogisticRegression object initialiseren met 2
           inputparameters (ButtonPresses &
           JoystickMovements)
        LogisticRegression logisticRegression = new
           LogisticRegression()
        {
                NumberOfInputs = 2
        };

       var learner = new
          LogisticGradientDescent(logisticRegression)
        {
             Tolerance=0.1
        };

      // De gradient descent begint met de logistische
         regressie te optimaliseren
      logisticRegression = learner.Learn(input, output);
}
```

Code 3.2: Implementatie binaire logistische regressie

F-score

Doordat deze dataset zo simpel is heeft het algoritme geen probleem om de testdata, die uiteraard ook even duidelijk is, te classificeren. De precisie en rappel, die we in sectie 2.2.3 hebben uitgelegd, zijn dan logischerwijs 1 Zo bekomen we een F-score van 1 wat wil zeggen dat het algoritme foutloos is op de testdata.

```
using (var table = new
   ExcelReader("../../datasetExcel.xls").GetWorksheet("Training"))
   {
        // Convert the DataTable to input and output
           vectors
        double[][] inputs =
           table.ToJagged<double>("ButtonPresses",
           "Joystickmovements");
        int[] outputs =
           table.Columns["Game"].ToArray<int>();

        ...
   }
```

Code 3.3: Inlezen Excel data

3.2.3 Support Vector Machine

Support Vector Machine is het tweede algoritme die we zullen gebruiken en vergelijken met logistische regressie. We zijn nog altijd in de eerste fase dus vergelijken we twee klassen met elkaar. De code voor Support Vector Machine vindt u in code 3.4 op pagina 28.

De parameters inputs en outputs worden opnieuw meegegeven nadat de data is opgehaald zoals we gezien hebben in code 3.3. Vervolgens initialiseren we het SupportVectorMachine object. We geven in de constructor het aantal features mee (in het Accord-framework gebruiken ze inputs wat verwarrend kan zijn). Het aantal features in dit geval is 2 namelijk de kolommen 'ButtonPresses' en 'Joystickmovements'. Vervolgens wordt het optimalisatie algoritme gecreëerd met de nodige attributen die voor zichzelf spreken. Tot slot kan de SVM geoptimaliseerd worden d.m.v de Learn methode.

Resultaten

Snelheid van het algoritme

Om de snelheid van dit algoritme te meten plaatsen we opnieuw een stopwatch voor net voor het de lijn *"svm = teacher.Learn(inputs, outputs)"*. En we stoppen die uiteraard net nadat deze lijn is uitgevoerd. Nadat het programma 20 keer uitgevoerd is geweest nemen we het gemiddelde van de resultaten. De Support Vector Machine heeft 34,4823 milliseconden nodig om een hypothese te ontwikkelen. En dit met een standaarddeviatie van slechts 1,3146.

```
public static void StartSupportVectorMachine(double[][]
   inputs, int[] outputs)
{
//Support Vector Machine initialiseren met 2 input
   features namelijk ButtonPresses en Joystickmovements
SupportVectorMachine svm = new
   SupportVectorMachine(inputs: 2);
        //Het optimalisatiealgoritme voor SVM van het
           Accordframework is als vlogt geinitialiseerd
        var teacher = new LinearDualCoordinateDescent()
        {
              Inputs = inputs,
              Outputs = outputs,
              Model = svm,
              Tolerance = 0.1
        };
        //SVM wordt geoptimaliseerd
        svm = teacher.Learn(inputs, outputs);
}
```

Code 3.4: Support Vector Machine implementatie

F-score

De F-score van SVM is dezelfde als logistische regressie omdat ook hier foutloos alle voorbeelden voorspelt worden.

3.2.4 Conclusie na fase 1

Aangezien dit een simpele start is met twee duidelijk verschillende spelletjes hebben beide algoritmen een perfecte F-score van 1. Het verschil wordt nu dus gemaakt op de snelheid. Logistische regressie had slechts 4,5487 milliseconden nodig om een foutloze hypothese te verkrijgen terwijl de SVM daar gemiddeld 34,4823 over gedaan heeft. Dit is $\pm$ 6 keer zo lang.
Op dit ogenblik krijgt logistische regressie logischerwijs nog de voorkeur. Uiteraard zal in de arcade machine meerdere spelletjes beschikbaar zijn. Ook zullen er meer dan twee metrics zijn waarop we spelletjes kunnen vergelijken.

3.3 Fase 2

In 3.2 fase 1 maakten we gebruik van ButtonPresses en JoystickMovements. We gaan nu nog een extra inputvariabele toevoegen. Er zijn spelletjes waar je soms eventjes niet kan spelen omdat er van level wordt veranderd of als je bijvoorbeeld Pacman speelt wordt er afgeteld van 3 naar 1 wanneer je dood geweest bent, dit zijn dan 3 seconden dat er niets gebeurt is. Terwijl er bij Mortal Kombat meer gespeeld wordt en minder lang moet wachten. Vandaar dat er een kolom 'TotalTimeOfNoUseOfControls' is toegevoegd aan de dataset met random getallen die voor Mortal Kombat tussen 800 ms en 2500 ms liggen. Terwijl er bij Pacman gestart wordt vanaf 3000 ms tot 5000 ms. Doordat er nu een extra variabele toegevoegd is gaan we ervan uit dat de algoritmen er beetje langer over zullen doen dan in de vorige fase. Nadat we deze extra inputvariabele besproken hebben gaan we de dataset nog beetje realistischer te maken. Er zullen nog twee extra inputvariabelen toegevoegd worden namelijk de frequentie van de buttons en joystick per 20 seconden.

3.3.1 Logistische regressie

Omdat we nog altijd met twee klassen werken moet er bijna niets aan de implementatie veranderd worden. Enkel bij de declaratie van het object moet 'NumberOfInputs' naar 3 veranderd worden en vervolgens naar 5 wanneer we de kolommen 'ButtonPressesPer20Seconds' en 'JoystickmovementsPer20Seconds' in het algoritme willen betrekken.

Resultaten

Snelheid van het algoritme

We beginnen bij onze eerste extra kolom, 'TotalTimeOfNoUseOfControls'. De snelheid van het algoritme werd opnieuw 20 keer gemeten en met een gemiddelde van 8,6405 ms is wat al dubbel zo lang is als de eerste implementatie. Dit komt doordat het verschil tussen Pacman en Mortal Kombat nu niet meer afhankelijk is van 2 duidelijk verschillende inputvariabelen maar nu één extra is waar de verschillen dichter bij elkaar liggen.

In de volgende test voegen we de kolommen 'ButtonPressesPer20Seconds' en 'JoystickmovementsPer20Seconds' toe. Verrassend genoeg is het gemiddelde met deze variabelen erbij slechts .7 ms gestegen. Dit kan verklaard worden doordat er een correlatie is tussen het aantal keer er op een knop gedrukt is in 1 minuut en de frequentie van de knoppen. Om de test te doen hebben we de correlatie tussen die twee kolommen berekent en die komt uit op 0.8010, wat wil zeggen dat er een sterk verband is. Doordat er zo'n sterk verband is brengen die extra inputvariabelen geen meerwaarde aan de dataset.

Een andere inputvariabele die we kunnen gebruiken is het aantal keer dat de speler 'dood' is geweest. In termen van Pacman is dit iedere keer wanneer hij opgegeten wordt. In geval van Mortal Kombat, het aantal keer de speler dood geschoten is geweest. In beide spelletjes is het mogelijk dat er geen enkele keer een 'dood' is geweest. Of als de speler een beginner is kan die vrij veel 'dood' gegaan zijn. Dit geldt voor beide spelletjes. Wanneer we de gemiddelden meten van met deze kolom erbij dan krijgen we een gemiddelde van 8 ms. De

```
public static void StartLogisticRegression(double[][]
   input, int[] output)
{
        //LogisticRegression object initialiseren met 5
           inputvariabelen
        LogisticRegression logisticRegression = new
           LogisticRegression()
        {
                NumberOfInputs = 5
        };

                ...
}
```

Code 3.5: Implementatie binaire logistische regressie met 5 inputvariabelen

verklaring waarom het algoritme niet veel meer vertraagt is omdat de waarden van de eerste twee kolommen veel zwaarder doorwegen dan de laatste twee die nu toegevoegd geweest zijn. Dit geldt ook voor een 5de kolom die toegevoegd zal worden om de moeilijkheid nog beetje te vergroten. 'Multiplayer' is een kolom met boolean waarden. Wanneer er 2 spelers tegen elkaar aan het spelen waren zal die waarde op true staan, anders op false. Pacman zal altijd alleen gespeeld worden dus die waarde zal altijd false zijn.

Als we het programma debuggen kunnen we de coëfficiënten bekijken. Als we die uitschrijven voor de logistische regressie tot nu toe dan krijgen we volgend functievoorschrift. Daarin ziet u duidelijk dat de laatste 2 kolommen eigenlijk overbodig zijn.

$$y(x0,x1,x2,x3,x4) = 255,958 * x0 + 230,2784 * x1 + -37,0513 * x2 + 0,1878 * x3 + 0,7306 * x4 + 0,858$$

Zoals we in de sectie 2.2.1 binaire logistische regressie hebben gezien moet de waarde van de functie hierboven nog verwerkt worden in de sigmoïdfunctie.

F-score

De logistische regressie met een Tolerance van 0.1 is nog steeds foutloos dus blijft de F-score gelijk aan 1.

3.3.2 Support Vector Machine

Voor de support vector machine gaan we uiteraard dezelfde kolommen gebruiken als dat we voor logistische regressie hebben gedaan. Als het aantal inputvariabelen relatief klein zijn tegenover het aantal trainingsvoorbeelden dan wordt er aangeraden om logistische regressie te gebruiken of een SVM zonder kernel. („Lecture week 7, Using an SVM", g.d.). Met geen kernel wordt de standaard lineaire kernel bedoelt. Dit is dan ook de kernel die in deze bachelorproef gebruikt wordt. Ook voor dit algoritme moeten we niet veel aanpassen om meerdere inputvariabele in rekening te brengen.

Resultaten

Snelheid van het algoritme

We starten met 'TotalTimeOfNoUseOfControls' toe te voegen. Het algoritme heeft daarvoor gemiddeld 39,3622 ms voor nodig gehad om een hypothese te bekomen. Dit is slechts een vermeerdering van $\pm$ 5 milliseconden. Ook voor de andere twee extra inputvariabelen bedraagt de vermeerdering bijna niets. Dit komt door de Tolerance waarde in het LineaarDualCoordinateDescent object. Die staat standaard ingesteld op 0.1 wat een goede keuze is. Omdat we niet willen dat onze hypothese 'overfit' heeft. Bij SVM's kan de Tolerance waarde niet veranderd worden Overfit is een term die gegeven wordt aan hypothesen die te nauwkeurig zijn wat uiteraard niet goed is. Zo kan het zijn dat nieuwe voorbeelden slecht gegeneraliseerd worden.

F-score

Ook de support vector machine is nog altijd feilloos in het voorspellen van de testdata.

3.3.3 Conclusie na fase 2

Ondanks dat we ondertussen al gebruik maken van vijf inputvariabelen hebben de algoritmen het nog niet moeilijk om correcte voorspellingen te doen. De Tolerance staat voor beiden algoritmen op 0.1 wat aanduidt wanneer een leerproces mag stoppen. Als die waarde de laag staat ingesteld dan zal de hypothese overfitten. Als een hypothese overfitted is dan wil dit zeggen dat hij te nauwkeurig wil voorspellen. Dit kan leiden tot een slechte generalisatie en dus slechte scores op de testdata. Als de Tolerance waarde te hoog is zal de hypothese te algemeen zijn en dus ook niet goed scoren.

3.4 Fase 3

In fase 3 beginnen we terug opnieuw met slechts twee inputparameters maar nu wel met drie verschillende klassen. We bouwen dit opnieuw stap per stap op. Starten met 2 inputparameters dan 3 en tenslotte alle 5. Het derde spel die we in de dataset gaan toevoegen is Tetris. Tetris wordt enkel maar gespeeld met de knoppen. Op die manier hebben we 3 totaal verschillende spelletjes. Een waar bijna voornamelijk de joystick gebruikt wordt (Pacman), een waar enkel de knoppen gebruikt worden(Tetris) en nog een spel waar zowel knoppen als joystick gebruikt worden (Mortal Kombat). De 'TotalTimeOfNoUseOfControls' is ook veel lager bij Tetris omdat dit lang aan 1 stuk kan gespeeld worden. Ook 'AmountOfDeads' ligt veel lager dan andere spelletjes.

3.4.1 Logistische regressie

De implementatie van logistische regressie voor meerdere klassen is beetje anders. Maar wel eenvoudig gemaakt door het Accord framework. Met de klasse MultinomialLogisticRegressionAnalysis kunnen een hypothese laten genereren. Die dan de voorspellingen zal doen. Deze klasse bepaalt ook zelf hoeveel inputparameters in de trainingsset aanwezig zijn. Dit geldt ook voor het aantal klassen.

Resultaten

Snelheid van het algoritme

We beginnen de testen met twee inputparameters. Het algoritme blijft foutieve voorspellingen doen zolang er geen 25 iteraties gebeurt zijn door het leeralgoritme. Dit duurt 15,3824ms wat dus relatief gezien al een stuk langer duurt dan wanneer er slechts 2 klassen waren met 2 parameters.

Als we 3 inputparameters gebruiken ('ButtonPresses', 'JoystickMovements', 'TotelTimeOfNoUseOfControls') heeft het algoritme 95 iteraties nodig om een foutloze hypothese te maken. Dit duurt gemiddeld 22,67 ms.

Bij vijf inputparameters (+ 'AmountOfDeads' & 'Multiplayer') doet het algoritme er 28,9126 ms over. Als we dit vergelijken met ons eerste resultaat in fase 3 is dit al bijna een verdubbeling.

Foutratio

Nu we met meerdere klassen werken is het niet meer mogelijk om de F-score te berekenen omdat die enkel kan berekend worden tussen binaire classificaties. Onze testdataset telt nu 150 voorbeelden doordat er 50 extra voorbeelden van Tetris zijn toegevoegd. We laten het leeralgoritme stoppen wanneer een tolerance van 0.1 bereikt is. Bij eerste test met twee inputparameters beslist logistische regressie 143 van de 150 voorbeelden correct. Van de foutieve voorspellingen dacht het algoritme vijf maal dat Tetris gespeeld werd en een twee

```
public static void StartLogisticRegression(double[][]
   input, int[] output)
{
      MultinomialLogisticRegressionAnalysis
         multiLogisticRegression = new
         MultinomialLogisticRegressionAnalysis();

          //multiLogisticRegression.Iterations = 135
          multiLogisticRegression.Tolerance = 0.1;

      multiLogisticRegression.Learn(input, output);
}
```

Code 3.6: Implementatie multiklasse logistische regressie met een tolerance van 0.1 het Iterations property werd gebruikt om de snelheid te meten.

keer Pacman terwijl het eigenlijk zeven keer Mortal Kombat moest zijn.

Wanneer er drie inputvectors gebruikt worden dan voorspelt het algoritme even veel voorbeelden fout maar nu denkt hij vier keer dat Mortal Kombat gespeeld werd ipv Pacman. De andere drie fouten gebeuren bij het verschill tussen Tetris en Mortal Kombat waar hij telkens de voorkeur geeft aan Tetris.

De volgende test met vijf inputvectors maakt acht foutieve voorspellingen. Waarvan vier keer de voorspelling Mortal Kombat Pacman had moeten zijn. Twee keer kreeg Tetris de voorkeur op Mortal Kombat net als dat Pacman twee keer daar de voorkeur op kreeg.

```
var teacher = new
   MulticlassSupportVectorLearning<Linear>()
  {
    Learner = (p) => new LinearDualCoordinateDescent()
    {
        Tolerance = 0.1
    }
  };

    teacher.ParallelOptions.MaxDegreeOfParallelism = 1;

    var machine = teacher.Learn(inputs, outputs);
```

Code 3.7: Implementatie multiklasse logistische regressie met een tolerance van 0.01 het Iterations property werd gebruikt om de snelheid te meten.

3.4.2 Support Vector Machine

Ook voor de Support Vector Machine is de implementatie gewijzigd. U kan de code zien in figuur **??**. We maken gebruik van de lineaire kernel wat eigenlijk de standaard kernel is voor SVMs.

Resultaten

Snelheid van het algoritme

Een probleem die zich voordoet bij de Support Vector Machines is dat het aantal iteraties niet kunnen vastgezet worden. De Tolerance kunnen we wel wijzigen maar gelijk welke waarde we daar aan toekennen, de snelheid blijft hetzelfde. Dit zal waarschijnlijk een fout zijn in het Accord-framework. Of er nu 2, 3 of 5 inputvectoren zijn, de gemiddelde tijd van het leerproces is 135,0201 ms.

Foutratio

Ook de Support Vector Machine blijft niet foutloos in deze fase tot en met 3 inputparameters is het algoritme wel nog feilloos vanaf vier worden er 2 spelletjes foutief voorspelt.

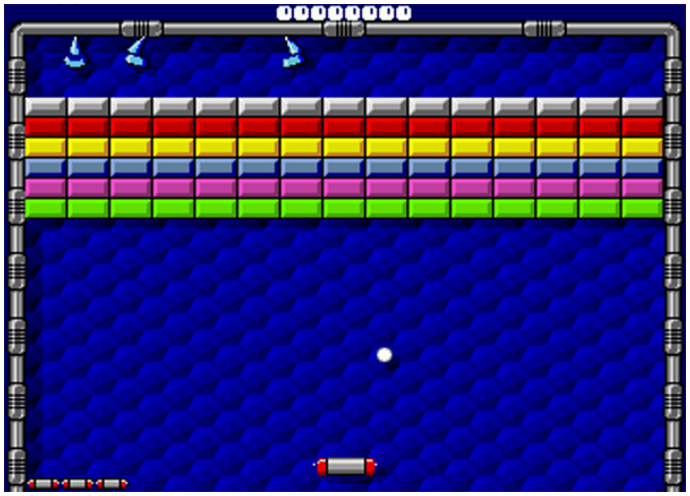
3.4.3 Conclusie na fase 3

Ondanks dat we de snelheid niet precies met elkaar kunnen vergelijken zien we toch wel dat een Support Vector Machine meer tijd nodig heeft dan logistische regressie. We kunnen dit ook staven. We weten dat de tijd van een SVM met vijf inputvectors er gemiddeld 135,0201 ms over doet maar dan zijn er wel 2 foute voorspellingen. Om een perfect algoritme te krijgen zal het dus nog beetje langer dan 135,0201 ms duren. De tijd van een

logistische regressie met vijf inputparameters bedraagt 66,7429 ms en dan is het algoritme foutloos. Dus op die manier weten we zeker dat het algoritme van SVM langer duurt dan logistische regressie.

Bij het foutratio zien we dan wel dat de SVM beter presteert dan logistische regressie met slecht 2 fouten tegenover 7.

Er komt nog een fase 4 waar we nog een 4de spel toevoegen. Dit is de laatste fase, dan zou dit moeten volstaan om een conclusie te maken



Figuur 3.8: Het 4de spel in de dataset

3.5 Fase 4

We voegen tijdens fase 4 nog een laatste spel toe. Arkanoid (figuur: 3.8) is een spel waar je enkel met de joystick speelt. De enige bewegingen die daarbij gemaakt worden is naar links en rechts. Doordat het kan zijn dat de bal boven de blokken eventjes blijft. De joystickmovements zullen dus een pak minder zijn dan andere spelletjes. Doordat er ook geen knoppen gebruikt worden zal het algoritme deze 2 spelletjes het moeilijkst uit elkaar kunnen gehouden worden want Pacman heeft gelijkaardige waarden.

3.5.1 Logistische regressie

Qua implementatie hoeft er niets te veranderen dus die blijft dezelfde als in code 3.6. We gaan nu ook direct door naar de vijf inputparameters omdat die het meest aanleunen bij de realiteit.

Resultaten

Snelheid van het algoritme

Het algoritme presteert na 29,8291 ms. Het is niet meer mogelijk om een foutloze hypothese te ontwikkelen dus kijken we wanneer het algoritme het minst fouten maakt. Dit is nadat er 54 iteraties zijn gebeurt. Vanaf 64 iteraties worden er weer meer fouten gemaakt. Dit komt doordat de hypothese begint te overfitten dus niet meer goed generaliseert en dus sneller fouten maakt. Wanneer het algoritme optimaal is, na 54 iteraties, maakt het algoritme slechts 3 fouten van de 200 voorbeelden.

Foutratio

Voor het foutratio meten we het aantal fouten wanneer de tolerance is ingesteld op 0,1. Dan behaalt het algoritme 181/200 wat nog altijd niet slecht is. Wat opvalt en ook wel te verwachten was is dat er 9 fouten van de 19 gebeuren tussen Pacman en Arkanoid. Als we kijken naar een foutieve voorspelling dan zagen we dat Arkanoid voorspelt was terwijl het Pacman moest zijn uiteraard zijn de ButtonPresses en JoystickMovements gelijkaardig voor de 2 spelletjes. Wanneer we kijken naar 'TotalTimeOfNoUseOfControls' zien we dat Pacman daar 4892 ms heeft wij weten dat het maximum 5000 ms is. Terwijl die 5000 ms bij Arkanoid ongeveer in de midden ligt. Zo kunnen we ons inbeelden waarom de computer Arkanoid gekozen heeft boven Pacman.

3.5.2 Support Vector Machine

De implementatie voor de Support Vector Machine blijft net als de logistische regressie hetzelfde als in fase 3. De code kan u terugvinden in figuur 3.7.

Resultaten

Snelheid van het algoritme

De support vector machine haalt een gemiddelde van 169,4067 ms wanneer de complexity 100 is. Hoe hoger de complexity hoe preciezer de hypothese maar dus hoe slechter die generaliseert. 100 is een matige waarde hiervoor. Er zijn gemiddeld 8 fouten met een standaard deviatie van 2. We praten nu over gemiddeld aantal fouten omdat het leeralgoritme van de support vector machine stopt in een lokaal minimum. Maar met deze complexity zijn er minst fouten.

Foutratio

We verwijderen de complexity nu en de Tolerance zetten we op 0,1. Op deze manier scoort de SVM 188/200. Wat opvalt is dat hier alle fouten gemaakt worden tussen Arkanoid en Pacman.

3.5.3 Conclusie na fase 4

Nadat we een vierde spel hebben toegevoegd aan de dataset kunnen we duidelijkere conclusies trekken. Een eerste duidelijk verschil bevindt zich in de snelheid. Wanneer we de algoritmen naar hun optimale resultaat dwingen is logistische regressie 5,6 keer sneller dan een support vector machine. Ook de resultaten van het geoptimaliseerde algoritme waren in het voordeel van logistische regressie.

De foutratio's van beide algoritmen verschillen niet zo super veel. We merken wel dat de support vector machine het toch beter doet dan logistische regressie. Wanneer de fouten onderzocht werden viel het op dat de SVM enkel fouten maakten tussen Pacman en Arkanoid. terwijl logistische regressie nog 10 fouten maakte op de andere klassen, Tetris en Mortal Kombat.

4. Conclusie

Bibliografie

Muoio, D. (g.d.). Google's new artificial intelligence tool has massive potential. *Business Insider UK*.

Ng, A. (g.d.). *CS 229: Machine Learning*. Stanford University.

Niculescu-Mizil, R. C. /. A. (g.d.). *An Empirical Comparison of Supervised Learning Algorithms* (paper, Department of Computer Science, Cornell University, Ithaca, NY 14853 USA).

Accord framework. (g.d.). Verkregen van www.accord-framework.net

Lecture week 7, Using an SVM. (g.d.). Verkregen van https://www.coursera.org/learn/machine-learning/resources/Es9Qo

Tensorflow framework. (g.d.). Verkregen van www.tensorflow.org

The Most Popular Languages for Data Science. (g.d.). Verkregen van https://dzone.com/articles/which-are-the-popular-languages-for-data-science

What is a support vector machine? (2016). Verkregen van https://www.quora.com/What-is-a-Support-Vector-Machine

Lijst van figuren

Lijst van tabellen