Bachelorscriptie

Intrusion Detection met Core Vector Machines

Jelle Schühmacher
s0314013
J.Schuhmacher@student.science.ru.nl

Juli 2009

Begeleider: Tom Heskes

Inhoudsopgave

1. Inleiding

In 1999 was er een wedstrijd KDD99 [1] waarbij deelnemers werd gevraagd een classificatiesysteem te maken dat onderscheid kon maken tussen normale en verdachte verbindingen binnen een computernetwerk. Dit is inmiddels 10 jaar geleden. In dit onderzoek wil ik evalueren of een populaire methode, de Support Vector Machine (SVM), geschikt is voor deze taak. Verder, is het interessant om te zien hoe de resultaten zich, in termen van precisie, verhouden tot de resultaten van de toenmalige winnaar [12].

De dataset die gebruikt werd voor KDD99, bevat records van bijna vijf miljoen connecties, verdeeld over vier typen aanvallen. Elke connectie bestaat uit 41 features. Het grote aantal connecties levert een probleem op voor het gebruik van SVM's, omdat het trainen van een SVM een tijdscomplexiteit heeft van $O(n \cdot n_{sv} + n_{sv}^3)$ [8] in het aantal datapunten n en het aantal Support Vectors (SV's) n_{sv}. Het aantal benodigde SV's is typisch een lineaire functie van het aantal datapunten. Dit maakt het leren van een model met een SVM uit een grote dataset een tijdrovende bezigheid.

1.1. Onderzoeksvraag

Is de SVM een geschikte methode om als classificatiesysteem te gebruiken voor een intrusion detection systeem? Daarbij moet een antwoord worden gezocht voor de volgende deelvragen:

1. Hoe kan de SVM binnen acceptabele tijd uit de dataset geleerd worden?
2. Is het systeem precies genoeg? Er mogen bijvoorbeeld niet teveel "false positives" optreden.
3. Is het classificeren van connecties snel genoeg voor realtime gebruik?
4. Kunnen nieuwe voorbeelden van aanvallen worden bijgeleerd?

In verband met de beperkte tijd zal dit onderzoek zich richten op het oplossen van de eerste deelvraag door middel van een implementatie van een nieuw classificatiealgoritme dat is gebaseerd op de SVM.

1.2. Verantwoording

Het lijkt een goed idee om een classificatiesysteem te gebruiken als onderdeel van een intrusion detection systeem (IDS), want een dergelijk systeem kan volgens sommige experts

nieuwe aanvallen herkennen. Zij stellen dat nieuwe aanvallen vaak varianten zijn van bekende aanvallen, daardoor zou er met behulp van een classificatiesysteem veel sneller gereageerd kunnen worden. Een snellere reactie resulteert in minder tijd voor een aanvaller om iets met zijn doelwit te doen. Met behulp van een classificatiesysteem zou het detecteren van aanvallen bijna real-time kunnen gebeuren.

Echter, het kan ook zijn dat een classificatiesysteem te vaak een foute beslissing neemt, waardoor het te beveiligen systeem niet meer efficiënt kan werken. Het is daarom belangrijk om een systeem te gebruiken met een grote precisie en een kleine *false positive rate*.

1.3. Methode

In het eerste gedeelte van deze bachelorscriptie zal ik literatuuronderzoek doen naar SVM's. Daaruit zal duidelijk worden wat de theorie achter de SVM is en welke eigenschappen deze heeft. Vervolgens zal ik met Python een implementatie van een SVM maken, gebaseerd op onderzoek dat suggereert dat trainen ook in lineaire tijd mogelijk is [7, 8, 17]. Echter, in het onderzoek van Joachims [7] is het niet duidelijk of de methode ook werkt voor niet lineair te classificeren data. In het onderzoek van Keerthi et. al. [8] wordt een methode beschreven waarbij sneller trainen ten koste gaat van precisie. Met name het onderzoek in [17] is interessant, de auteur claimt voor zijn variant op de SVM, de *Core Vector Machine*, dat trainen in lineaire tijd gaat en bovendien dat de benodigde geheugenruimte constant is en onafhankelijk van het aantal datapunten.

In dit onderzoek zal de CVM geïmplementeerd worden, omdat deze de meest gunstige eigenschappen heeft. Met deze implementatie en de KDD99 dataset zal ik een prototype classificatiesysteem voor een IDS maken. Daarvan wil ik de prestaties meten in termen van precisie.

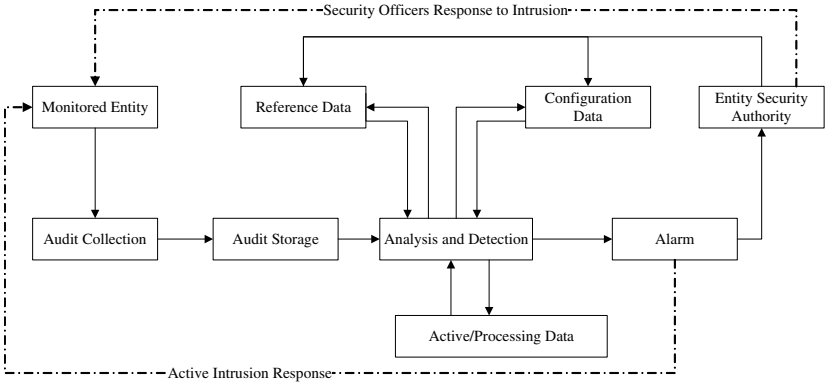
2. Intrusion Detection

Een *intrusion detection systeem* (IDS) is een computerprogramma waarmee ongeautoriseerde toegang tot een computersysteem of computernetwerk gedetecteerd kan worden. Het ideaal is dat een IDS elke vorm van ongeoorloofd netwerkverkeer en ongeoorloofd computergebruik moet kunnen detecteren.

IDS kunnen worden onderverdeeld in *host-based* en *network-based* systemen. Het eerste type IDS verzamelt zoveel mogelijk gebruiksdata van één enkel systeem. Deze data wordt door het programma geanalyseerd, waarbij geprobeerd wordt daarin aanwijzingen te vinden van misbruik. Er wordt typisch naar log-bestanden, netwerkverkeer en de staat van het bestandssysteem van één computer gekeken. Als er vervolgens misbruik wordt gesignaleerd, kan er een waarschuwing worden gestuurd naar een administrator, of er kan automatisch actie worden ondernomen. Een mogelijke actie zou kunnen zijn om een gebruiker de toegang ontzeggen tot een bepaald bestand of programma.

Het tweede type, een network-based IDS, verzamelt gebruiksdata van een compleet netwerk van computers. Hier wordt typisch gekeken naar netwerkverkeer, de log-bestanden van alle systemen in het netwerk en de staat de bestandssystemen van alle systemen in het netwerk. Typische reacties in het geval van misbruik zijn het verbreken van een netwerkverbinding, of het ontzeggen van toegang tot een bepaalde service.

In een onderzoek van Axelsson [2] naar de verschillende aanpakken van IDS wordt een generiek model gegeven van alle componenten die een dergelijk systeem bevat. Zijn model is te zien in Figuur 2.1.



Figuur 2.1.: Generiek model van een intrusion detection systeem

Door de omvang van een compleet intrusion detection systeem zal deze bachelorscriptie zich beperken tot het component van een host-based IDS dat onderscheid moet gaan maken tussen normaal en ongeoorloofd netwerkverkeer. Oftewel, het gedeelte *Analysis and Detection* in Figuur 2.1. Deze module wordt ontwikkeld op basis van de Core Vector Machine (CVM). Over de theorie en de implementatie is meer te vinden in Sectie 3.2 en Hoofdstuk 4. Daarbij wordt als *Reference Data* de dataset voor de KDDCup-99 gebruikt.

2.1. Reference Data

In 1998 is door MIT Lincoln Labs het DARPA Intrusion Detection Evaluation Program gecreëerd. Een onderdeel van dit programma was een grote verzameling netwerkverkeer. Om deze data te verzamelen werd er 9 weken lang netwerkverkeer uit een *local-area network* (LAN) opgeslagen. Dit LAN simuleerde een typisch netwerk van de Amerikaanse luchtmacht, waarbij er, naast normaal gebruik, kunstmatig verschillende aanvallen werden uitgevoerd.

Dataverkeer van 7 weken was bestemd om te worden gebruikt als trainingsdata. De data werd verwerkt tot ongeveer 5 miljoen verbindings-records. De overige 2 weken waren bestemd om te worden gebruikt als testdata. Deze data werd verwerkt tot ongeveer 2 miljoen verbindings-records.

De trainingsdata bevat 24 verschillende typen aanvallen en normaal verkeer. De testdata bevat nog 14 extra typen aanvallen die niet in de trainingsdata voorkomen. De verschillende aanvallen in de dataset kunnen worden onderverdeeld in 4 categorieën, *denial-of-service* (DOS), *remote-to-local* (R2L), *user-to-root* (U2R) en *probing*. Deze worden in de volgende secties beschreven.

2.1.1. Denial of service

De aanvallen in de categorie DoS, zijn een verzameling aanvallen waarbij een aanvaller ervoor zorgt dat zijn doelwit, een computer of service, zoveel verbindingen te verwerken krijgt dat het systeem niet meer voor legitieme doeleinden te gebruiken is [9]. Dit kan inhouden dat de complete machine niet meer bereikbaar is via een netwerk, maar ook dat een gebruiker niet meer kan inloggen. Een aantal van de aanvallen in deze categorie misbruiken legitieme functionaliteit van de doel-machine, anderen gebruiken fouten in de implementatie van services en weer anderen creëren misvormde netwerkpakketten om de TCP/IP implementatie in de war te brengen. In Figuur 2.2 staat een samenvatting van de aanvallen in de dataset en het effect dat ze hebben op hun doel.

Name	**Service**	**Vulnerable Platforms**	**Mechanism**	**Time to Implement**	**Effect**
Apache2	http	Any Apache	Abuse	Short	Crash httpd
Back	http	Any Apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailbomb	smtp	All	Abuse	Short	Annoyance
SYN Flood	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None	Bug	Short	None
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate/ Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/ chargen	All	Abuse	Short	Network Slowdown

Figuur 2.2.: Verschillende DoS aanvallen in KDD99

2.1.2. Remote to Local

De aanvallen in de categorie R2L beginnen met een aanvaller die geen toegang heeft tot een systeem. De aanvaller kan via een implementatiefout in een netwerkservice als gebruiker inloggen, of via *social engineering* toegang krijgen op het systeem. In Figuur 2.3 staat een samenvatting van de aanvallen in de dataset en het effect dat ze hebben op hun doel.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Dictionary	telnet, rlogin, pop, imap, ftp	All	Abuse of Feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Imap	imap	Linux	Bug	Short	Root Shell
Named	dns	Linux	Bug	Short	Root Shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	smtp	Linux	Bug	Long	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor Keystrokes remotely

Figuur 2.3.: Verschillende R2L aanvallen in KDD99

2.1.3. User to Root

De categorie U2R aanvallen, zijn aanvallen waarbij de aanvaller al als normale gebruiker toegang heeft tot een bepaald systeem. De aanvaller kan, door misbruik te maken van een implementatiefout of door op een ongeoorloofde manier een wachtwoord te verkrijgen, zijn gebruiksprivileges kan vergroten tot systeembeheerder. Als de aanvaller eenmaal deze rechten heeft, kan hij van alles met het systeem doen. In Figuur 2.4 staat een samenvatting van de aanvallen in de dataset en het effect dat ze hebben op hun doel.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Eject	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Ffbconfig	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Fdformat	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Loadmodule	Any user session	SunOS	Poor Environment Sanitation	Short	Root Shell
Perl	Any user session	Linux	Poor Environment Sanitation	Short	Root Shell
Ps	Any user session	Solaris	Poor Temp File Management	Short	Root Shell
Xterm	Any user session	Linux	Buffer Overflow	Short	Root Shell

Figuur 2.4.: Verschillende U2R aanvallen in KDD99

2.1.4. Probing

Probing is zelf geen aanval, maar over het algemeen wel hier wel een voorloper van. Probing is het automatisch op afstand onderzoeken van een computer of netwerk, op zoek naar netwerk-programma's waar een fout in zit, configuratiefouten en andere zwaktes. In Figuur 2.5 staat een samenvatting van de verschillende probing technieken in de dataset en datgene waarnaar ze zoeken.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Ipsweep	ICMP	All	Abuse of Feature	Short	Finds active machines
Mscan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Nmap	many	All	Abuse of Feature	Short	Finds active ports on a machine
Saint	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Satan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities

Figuur 2.5.: Verschillende probing technieken in KDD99

3. Theorie

Het classificatieonderdeel voor het IDS in deze bachelorscriptie wordt gebaseerd op een variatie op de *Support Vector Machine* (SVM), namelijk de *Core Vector Machine* (CVM) [17]. In Sectie 3.1 wordt de theorie achter de standaard SVM beschreven. In Sectie 3.2.1 wordt een alternatieve manier om het classificatieprobleem te zien beschreven. Verder beschrijft Sectie 3.2 een algoritme dat deze alternatieve probleemformulering gebruikt om data te classificeren.

3.1. Support Vector Machine

De *Support Vector Machine* (SVM) [5] is een populaire methode bij veel onderzoekers, omdat deze op dit moment tot de meest precieze classificatiemethoden behoort. In zijn originele vorm is de SVM een binair classificatiealgoritme, alhoewel er al vele varianten bij zijn gekomen. De beschrijving hieronder zal echter beperkt blijven tot het binaire algoritme.

3.1.1. Het lineaire classificatieprobleem

De SVM biedt een oplossing voor het *lineaire classificatieprobleem*. Lineaire classificatie is het in twee klassen verdelen van een groep objecten in een ruimte, door middel van een lijn of vlak.

Dit kan preciezer geformuleerd worden als: Gegeven een lineair separeerbare dataset S met m punten, $S = \{(\mathbf{x}_i, t_i)\}_{i=1}^{m}$, waar t_i de klasse is van datapunt $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ en $t_i \in \{+1, -1\}$. Probeer nu een hypervlak te vinden dat een scheiding aanbrengt in de dataset zodat alle datapunten met klasse $+1$ zich aan één kant van het vlak bevinden en alle punten met klasse -1 zich aan de andere kant van het vlak bevinden. Een formele definitie van het hypervlak is:

$$y(\mathbf{x}_i) = \mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b, \tag{3.1}$$

waarin $\phi(\mathbf{x})$ een vooraf vastgelegde afbeelding is[1], b is de bias en $\mathbf{w}$ is een set gewichten. Zoek nu die gewichten $\mathbf{w} = (w_1, \ldots, w_n)$ en bias b zodat $t_i y(\mathbf{x}_i) > 0$, oftewel:

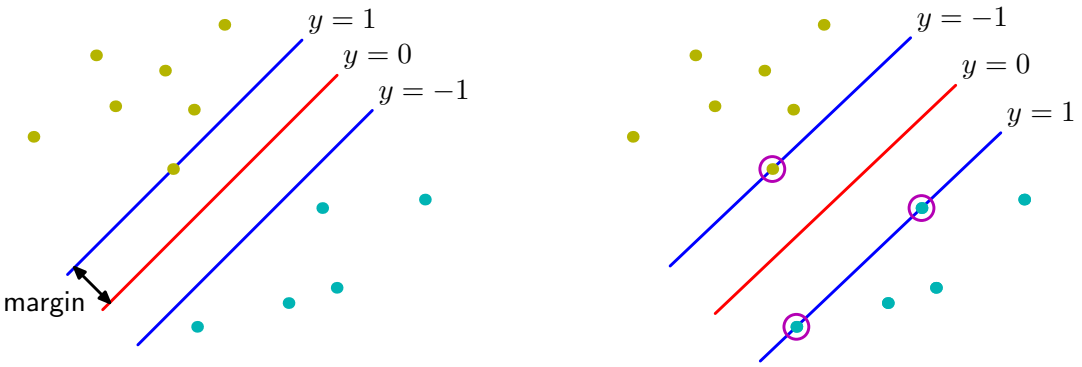
$$\text{sign}(y(\mathbf{x}_i)) = \begin{cases} +1 & \text{als } t_i = +1, \\ -1 & \text{als } t_i = -1. \end{cases} \tag{3.2}$$

Zodra het hypervlak $y(\mathbf{x})$ gevonden is kan deze gebruikt worden om nieuwe datapunten mee te classificeren.

[1] Zie ook Sectie 3.1.3

3.1.2. De maximale marge

In het algemeen kunnen er meerdere hypervlakken gevonden worden die een dataset verdelen. Deze classificeren niet allemaal even precies, dus is het interessant om de meest optimale scheiding te zoeken. In het geval van de SVM kiest men het hypervlak met de maximale marge. Een voorbeeld van de maximale marge is te zien in Figuur 3.1[2]. Een verantwoording van deze



Figuur 3.1.: Links: een hypervlak dat de data scheidt. Rechts: het hypervlak met de maximale marge

keuze wordt gegeven in de *statistical learning theory* [18]. Dit kan worden samengevat als: indien er geen extra informatie is over ongeziene punten, dan maximaliseert het hypervlak de verwachtte generalisatie. De classificatiefout wordt hiermee geminimaliseerd [5].

Indien alle datapunten correct zijn geclassificeerd, $t_i y(\mathbf{x}_i) > 0$, dan is het scheidende hypervlak gedefinieerd door $y(\mathbf{x}) = 0$. De afstand van een datapunt tot dit vlak is dan [3]:

$$\frac{1}{||\mathbf{w}||} \cdot t_i \cdot y(\mathbf{x}_i) = \frac{1}{||\mathbf{w}||} \cdot t_i(\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b). \tag{3.3}$$

Voor een maximale marge moet de afstand tussen het gevonden hypervlak en de set van dichtstbijzijnde datapunten zo groot mogelijk zijn. Oftewel, optimaliseer de parameters $\mathbf{w}, b$ zo dat de afstand van de dichtstbijzijnde punten tot het hypervlak maximaal is. Formeel kan dit geschreven worden als:

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||} \cdot \min_i \{ t_i(\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) \} \right\}. \tag{3.4}$$

Een directe oplossing voor dit probleem is erg complex, maar door het om te schrijven in een equivalent probleem is er toch een oplossing te vinden. De afstand van een punt $\mathbf{x}$ tot

[2]Figuren uit [3]

het hypervlak verandert niet als de gewichten en de bias met dezelfde constante, k, worden geschaald:

$$\frac{1}{||\mathbf{w}||} \cdot t_i \cdot y(\mathbf{x}_i) = \frac{1}{k \cdot ||\mathbf{w}||} \cdot t_i(k \cdot \mathbf{w}^T \cdot \phi(\mathbf{x}_i) + k \cdot b). \tag{3.5}$$

Dit maakt het mogelijk om die k zo te kiezen dat het punt dat het dichtst bij het hypervlak ligt een afstand van 1 tot het hypervlak heeft. Als de hele dataset correct geclassificeerd kan worden geldt hierdoor voor alle datapunten:

$$t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 \tag{3.6}$$

Per definitie is er altijd minstens één datapunt het dichtst bij het hypervlak. Indien de marge van het hypervlak maximaal is, moeten er minimaal twee punten zijn met afstand 1, één voor elke klasse. In Vergelijking 3.4 wordt het rechtergedeelte dan gelijk aan 1, dus hoeft alleen $||\mathbf{w}||^{-1}$ gemaximaliseerd te worden. Dit is hetzelfde als $||\mathbf{w}||^2$ minimaliseren:

$$\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2. \tag{3.7}$$

Traditioneel wordt Vergelijking 3.7 opgelost door deze te transformeren naar een duale formulering, al blijkt uit [4] dat dat strikt genomen niet nodig is. Vergelijking 3.7 samen met de restricties die worden gedefinieerd door Vergelijking 3.6 zijn een optimalisatieprobleem. Het dualiteitsprincipe uit de optimalisatietheorie stelt dat een dergelijk probleem bekeken kan worden vanuit twee gelijke standpunten, primaal en duaal. In de primale formulering moeten de gewichten $\mathbf{w}$ en de bias b expliciet berekend worden. In een duale formulering is de oplossing een lineaire combinatie van punten in de dataset.

Vergelijking 3.7 kan worden omgevormd tot een duaal probleem met behulp van Lagrange multiplicatoren. Met Lagrange multiplicatoren kunnen de restricties in Vergelijking 3.6 en het probleem zelf in één vergelijking worden geplaatst. Dit ziet er als volgt uit:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i \{t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) - 1\}. \tag{3.8}$$

In deze (primale) vergelijking is $\alpha = (\alpha_0, \ldots, \alpha_m)$ de vector van Lagrange multiplicatoren, hier geldt dat $\alpha_i \geq 0$ en $\alpha_i \{t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) - 1\} = 0$. Deze vergelijking kan worden omgevormd tot een duaal probleem door de partiële afgeleide naar $\mathbf{w}$ gelijk te stellen aan 0:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} t_i \alpha_i \mathbf{x}_i = 0, \tag{3.9}$$

en ditzelfde te doen voor b:

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \mathbf{w} - \sum_{i=1}^{m} t_i \alpha_i = 0. \tag{3.10}$$

Dit geeft de volgende twee relaties:

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^{m} t_i \alpha_i \mathbf{x}_i, \\ 0 &= \sum_{i=1}^{m} t_i \alpha_i. \end{aligned} \tag{3.11}$$

Deze kunnen dan ingevuld worden in Vergelijking 3.8 zodat het volgende duale probleem ontstaat:

$$\begin{aligned} L(\mathbf{w}, b, \alpha) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \{ t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) - 1 \} \\ &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} t_i t_j \alpha_i \alpha_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \end{aligned} \tag{3.12}$$

Hier gelden nog steeds de voorwaarden:

$$\begin{aligned} \alpha_i &\geq 0, \\ \alpha_i \{ t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) - 1 \} &= 0, \end{aligned} \tag{3.13}$$

deze worden de *Karush-Kuhn-Tucker* (KKT) voorwaarden genoemd.

In de duale formulering 3.12 is de oplossing alleen nog afhankelijk van de Lagrange multiplicatoren en de dataset, $\mathbf{w}$ en b kunnen gevonden worden met behulp van de vergelijkingen in 3.11. Alleen de Lagrange multiplicatoren die groter zijn dan 0 voegen iets toe aan de oplossing. Deze komen overeen met die datapunten die op de rand van de marge liggen, de omcirkelde punten in Figuur 3.1. De punten dragen als het ware het gevonden hypervlak, ze heten daarom dan ook *Support Vectors* (SV). Er geldt:

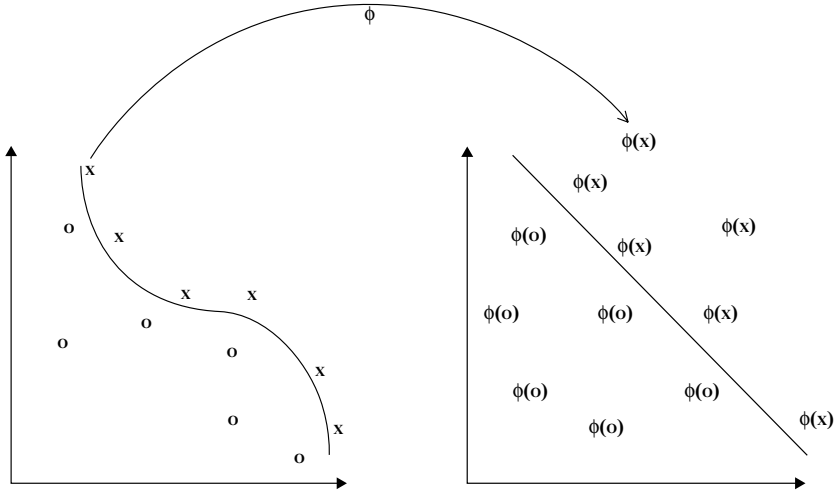
$$\begin{aligned} y(\mathbf{x}, \alpha, b) &= \sum_{i=1}^{m} t_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b \\ &= \sum_{i \in SV} t_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b. \end{aligned} \tag{3.14}$$

Oftewel alle andere punten geven geen extra informatie, ze dragen niets bij aan de oplossing. In theorie zouden deze punten kunnen worden weggelaten, zonder dat de oplossing zou veranderen.

3.1.3. Gebruik van Kernels

In de beschrijving tot nu toe is alleen gekeken naar lineair separeerbare datasets, dat wil zeggen datasets waarvan de klassen gescheiden kunnen worden met een hypervlak. In de praktijk

komen echter ook vaak datasets voor die niet lineair separeerbaar zijn. Er bestaat echter een truuk die dit toch mogelijk maakt. In de vergelijkingen tot nu toe komt steeds de afbeelding $\phi(\mathbf{x})$ voor, dit is een transformatie van de datapunten $\phi(\mathbf{x}) : \mathscr{R}^n \rightarrow \mathscr{R}^q$ tot een dimensie q waarin een hypervlak de datapunten wel kan scheiden. Een illustratie van dit principe is te zien in Figuur 3.2



Figuur 3.2.: $\phi(\mathbf{x})$ maakt de dataset lineair separeerbaar

Als gevolg van deze transformatie komt er een computationeel duur inproduct in Vergelijking 3.12 voor, duur omdat de dimensie van $\phi((x))$ mogelijk oneindig zou kunnen zijn. Echter door $\phi(\mathbf{x})$ slim te kiezen is het mogelijk om de transformatie niet expliciet uit te voeren en impliciet in de hoog dimensionale ruimte te rekenen. Een voorbeeld uit [13]; neem de transformatie $\phi : \mathbf{x} = (x_1, x_2) \rightarrow \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$ van $\mathscr{R}^2$ naar $\mathscr{R}^3$. Dan geldt dat:

$$\begin{aligned}\phi(\mathbf{x})^T\phi(\mathbf{z}) =& (x_1^2, x_2^2, \sqrt{2}x_1x_2)^T(z_1^2, z_2^2, \sqrt{2}z_1z_2) \\ =& x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ =& (x_1z_1 + x_2z_2)^2 = (\mathbf{x}^T\mathbf{z})^2\end{aligned} \tag{3.15}$$

Het inproduct van de getransformeerde punten is nu gelijk aan het inproduct tussen twee datapunten in het kwadraat, de transformatie zelf hoeft niet meer expliciet gedaan te worden. Het resultaat $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^2$ wordt een *Kernel*-functie genoemd, overal waar $\phi(\mathbf{x})^T\phi(\mathbf{z})$ staat kan nu $K(\mathbf{x},\mathbf{z})$ worden ingevuld. Deze truc, de *Kernel trick*, is mogelijk zolang de transformatie $\phi(\mathbf{x})$ voldoet aan [15]:

Mercer's Theorema. *Een Kernel-functie K kan worden uitgedrukt als*

$$K(\mathbf{u},\mathbf{v}) = \phi(\mathbf{u})^T\phi(\mathbf{v})$$

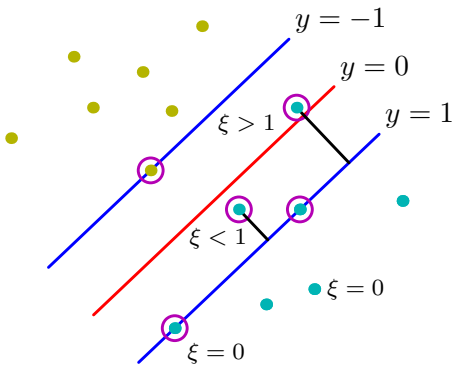
dan en slechts dan als, voor elke functie $g(x)$ waarvoor geldt dat $\int g(x)^2 dx$ eindig is, geldt dat:

$$\int K(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0.$$

In [13] staat een lijst met tot nu toe bekende kernels en manieren om nieuwe kernels te maken.

3.1.4. Omgaan met ruis

Afhankelijk van de gebruikte kernel is het nu soms mogelijk om altijd een oplossing te vinden in een hogere dimensie. Als een dataset ruis bevat, kan het voorkomen dat er een hypervlak in een hoge dimensie gevonden wordt die de data perfect separeert, terwijl er misschien een oplossing in een lagere dimensie bestaat met een grotere marge (en dus waarschijnlijk een betere generalisatie geeft). Dit gebeurt als er ruis-data in de marge van de ideale oplossing ligt en als de klassen in de dataset overlappen. Een oplossing hiervoor, is het gebruik van *slack*-variabelen $\xi_i \geq 0$, deze zorgen ervoor dat een gedeelte van de dataset fout geclassificeerd mag worden, zoals te zien in Figuur 3.3.



Figuur 3.3.: ξ_i maakt beperkt foute classificaties mogelijk

Er wordt één variabele per datapunt ingevoerd, waarvoor geldt dat:

$$\xi_i = \begin{cases} 0 & \text{als } sign(y(\mathbf{x}_i)) = t_i, \\ |t_i - y(\mathbf{x}_i)| & \text{als } sign(y(\mathbf{x}_i)) \neq t_i. \end{cases} \tag{3.16}$$

Verder wordt er een door de gebruiker te kiezen constante C ingevoerd, de complexiteits-constante. Deze constante wordt gebruikt om een evenwicht aan te brengen tussen de slack-

variabelen en normale classificatie. Het nieuwe primale optimalisatieprobleem wordt nu:

$$\arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i. \tag{3.17}$$

Het blijkt echter dat dit in de duale formulering alleen leidt tot een verandering in de voorwaarden, namelijk [6]:

$$\begin{aligned} \alpha_i \geq 0, \\ \alpha_i \leq C, \\ \alpha_i\{t_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)-1\} = 0, \end{aligned} \tag{3.18}$$

voor de rest blijven de vergelijkingen hetzelfde.

3.2. Core Vector Machine

Uit Vergelijking 3.14 wordt duidelijk dat alleen die punten die binnen de verzameling SV's vallen bijdragen aan een oplossing. Dit is één van de achterliggende gedachten waarop Tsang et al. [17] de *Core Vector Machine* (CVM) hebben gebaseerd. De CVM verdeelt de datapunten eerst in potentiële support vectors en nutteloze punten. De nuttige punten worden vervolgens gebruikt om $\mathbf{w}$ en b te vinden.

3.2.1. Minimum Enclosing Ball

Een belangrijk concept in de theorie van de CVM is de *Minimum Enclosing Ball* (MEB). Hiermee wordt de scheiding tussen potentieel nuttige en nutteloze datapunten berekend. De MEB wordt gedefinieerd als: gegeven een verzameling met m punten $S = \{\mathbf{x}_i\}_{i=1}^m$, dan is $MEB(S)$ de kleinste bal die alle punten uit S bevat. Exacte oplossingen voor het MEB probleem zijn in het algemeen niet efficiënt te berekenen zodra de dimensie van de punten groter wordt dan 30. Echter er bestaan snellere algoritmes die een benadering van de MEB met een kleine positieve foutmarge ϵ kunnen uitrekenen. Dit type benadering zal gebruikt worden om een efficiënter leeralgoritme te maken. Het idee is dat het deze benadering de uiteindelijke precisie niet significant negatief zal beïnvloeden, omdat deze in de meeste situaties toch afhangt van de gekozen complexiteits-constante [17].

Hiervoor is het eerst nodig om de definitie voor de benadering van de MEB formeler te maken. Definieer $B(\mathbf{c}, R)$ als een bal met radius R en middelpunt $\mathbf{c}$. Verder is er een fout ϵ, deze is per definitie altijd ≥ 0. De bal $B(\mathbf{c}, (1+\epsilon)R)$ is dan een $(1+\epsilon)$-benadering van $MEB(S)$ als geldt dat de radius kleiner of gelijk is aan de radius van de MEB:

$$R \leq r_{MEB(S)}, \tag{3.19}$$

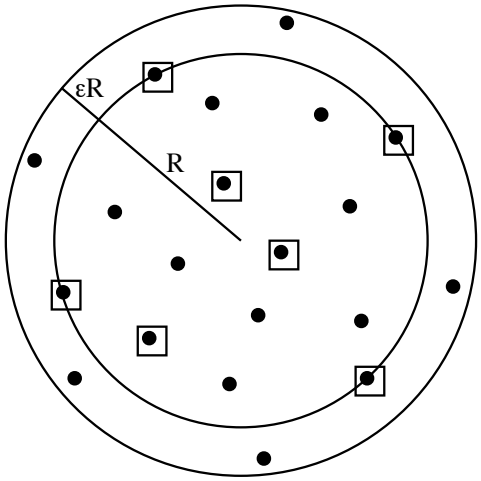
en verder geldt dat alle punten binnen de benaderde bal vallen:

$$S \subset B(\mathbf{c}, (1+\epsilon)R). \tag{3.20}$$

In deze vorm is het natuurlijk alleen interessant als ϵ zo klein mogelijk is. Het blijkt echter dat het ook mogelijk is om een goede benadering te vinden voor $MEB(S)$ zonder alle punten in de verzameling S te gebruiken. Dit kan met een deelverzameling, de *Core Set* van S. Een verzameling Q is een Core Set van S als de bal B met radius $(1+\epsilon)r_{MEB(Q)}$ alle punten uit S bevat, oftewel:

$$S \subset B(\mathbf{c}, (1+\epsilon)r), \tag{3.21}$$

waarbij $B(\mathbf{c}, r) = MEB(Q)$, dit is te zien in Figuur 3.4.



Figuur 3.4.: Punten met een vierkantje behoren tot de Core Set

3.2.2. Classificatie als MEB probleem

Stel dat er een afbeelding ϕ is waardoor alle punten in de dataset worden afgebeeld in een bal in de getransformeerde ruimte. De kleinst mogelijke bal die alle punten bevat kan dan worden gevonden door de kleinste radius te zoeken, oftewel:

$$\arg\min_{R^2, \mathbf{c}} \|\mathbf{c} - \phi(\mathbf{x}_i)\|^2 \leq R^2. \tag{3.22}$$

De kleinste radius geeft ook automatisch het middelpunt $\mathbf{c}$ van de bal. Deze vergelijking kan ook weer worden omgevormd tot een duaal probleem:

$$\arg\max_{\alpha_i} \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j), \tag{3.23}$$

waarbij de oplossing aan de voorwaarden $\alpha_i \geq 0$ en $\sum_{i=1}^{m} \alpha_i = 1$ moet voldoen. Als de gekozen Kernel-functie nu ook de eigenschap heeft dat $K(\mathbf{x},\mathbf{x})$ constant is, kan de gevonden vergelijking versimpeld worden tot:

$$\arg\max_{\alpha_i} - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j). \tag{3.24}$$

Uit het artikel van Tsang et. al. [17] blijkt dat dit optimimalisatieprobleem in feite dezelfde is als die van de L2-SVM. Maar dit geldt alleen als er een geschikte afbeelding ϕ is gekozen, dat wil zeggen een afbeelding die een kernel matrix met een constante diagonaal oplevert. De L2-SVM is een kleine variant op de L1-SVM zoals beschreven in Sectie 3.1. De L2-SVM wordt gevonden door het volgende optimalisatieprobleem op te lossen:

$$\arg\min_{\mathbf{w},b,\rho,\xi_i} ||\mathbf{w}||^2 + b^2 - 2\rho + C\sum_{i=1}^{m} \xi_i^2, \tag{3.25}$$

met de voorwaarden:

$$t_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b \geq \rho - \xi_i. \tag{3.26}$$

Deze vergelijking kan ook weer worden omgevormd tot een duaal probleem:

$$\arg\max_{\alpha} - \sum_{i,j=1}^{m} \alpha_i \alpha_j (t_i t_j k(\mathbf{x}_i, \mathbf{x}_j) + t_i t_j + \frac{\delta_{ij}}{C}). \tag{3.27}$$

Het enige verschil is dat de te minimaliseren fout wordt gekwadrateerd, hier zitten enkele voor- en nadelen aan, deze zijn na te lezen in [10]. Een gevolg hiervan is dat de bias niet wegvalt en dat het te vinden hypervlak ρ nu expliciet in het optimalisatieprobleem staat. Het idee is nu om te proberen een afbeelding $\phi(\mathbf{x})$ te zoeken zodat het hypervlak ρ altijd de vorm heeft van Vergelijking 3.24. Als dat lukt dan is het hypervlak een MEB en is het mogelijk om een oplossing voor de L2-SVM te vinden met behulp van een $(1+\epsilon)$-benadering van de MEB.

Met de afbeelding $\phi(\mathbf{x})$ kan een nieuwe afbeelding $\tilde{\phi}(\mathbf{x})$ worden gedefinieerd:

$$\tilde{\phi} : \phi(\mathbf{x}_i) \rightarrow \tilde{\phi}(\mathbf{x}) = (t_i \phi(\mathbf{x}_i), t_i, \frac{1}{\sqrt{C}} \mathbf{e}_i), \tag{3.28}$$

waarbij $\mathbf{e}_i$ een m-dimensionele vector is, zodat alleen op plaats i een 1 staat en de rest van de vector 0 is. Dan is de uiteindelijke kernel-functie door deze afbeelding veranderd in:

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = t_i t_j k(\mathbf{x}_i, \mathbf{x}_j) + t_i t_j + \frac{\delta_{ij}}{C}. \tag{3.29}$$

Deze functie kan gebruikt worden om Vergelijking 3.27 te vereenvoudigen tot:

$$\arg\max_{\alpha} - \sum_{i,j=1}^{m} \alpha_i \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_j). \tag{3.30}$$

Dit is hetzelfde als Vergelijking 3.24, dus door een $(1+\epsilon)$-benadering van de MEB van de dataset te vinden is het nu mogelijk om een hypervlak te vinden dat even precies is als de oplossing die door de L2-SVM gevonden zou worden.

Het algoritme om een dergelijke benadering te vinden maakt in stap 4 nog steeds gebruik van een SVM-implementatie, de snelheidswinst bij het trainen is grotendeels te danken aan het feit dat deze SVM-implementatie alleen de potentiële SVs hoeft te bekijken. Het aantal punten in de Core Set is vele malen kleiner dan het aantal punten in de totale dataset. Het algoritme werkt in grote lijnen als volgt:

1. Initialisatie:
   a) Kies een willekeurig punt $\mathbf{x}_0$ en zoek daarbij een ander punt dat er zo ver mogelijk vandaan ligt: $\mathbf{x}_1$. Zoek daar weer een punt bij dat er zo ver mogelijk vandaan ligt: $\mathbf{x}_2$. Dit zijn de eerste 2 punten in de Core Set SC: $SC_0 = \mathbf{x}_1, \mathbf{x}_2$.
   b) Kies als voorlopig middelpunt het midden van de Core Set: $\mathbf{c}_0 = \frac{1}{2}(\tilde{\phi}(\mathbf{x}_1) + \tilde{\phi}(\mathbf{x}_2))$.
   c) Kies als voorlopige radius de helft van de afstand in de Core Set: $R_0 = \frac{1}{2}||\tilde{\phi}(\mathbf{x}_1) - \tilde{\phi}(\mathbf{x}_2)||$.
2. Stop indien er geen punt in de rest van de dataset zit die buiten de $(1+\epsilon)$-benadering van de MEB valt.
3. Neem het verste punt van het midden van de huidige MEB en voeg deze toe aan de Core Set: SC_t. Het verste punt kan worden gevonden met de vergelijking:
$$||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_l)||^2 = \sum_{\mathbf{z}_i, \mathbf{z}_j \in SC_t} \alpha_i \alpha_j \tilde{k}(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{\mathbf{z}_i \in SC_t} \alpha_i \tilde{k}(\mathbf{z}_i, \mathbf{z}_l) + \tilde{k}(\mathbf{z}_l, \mathbf{z}_l).$$
4. Zoek een nieuwe MEB, de oplossing voor het optimalisatieprobleem in Vergelijking 3.24.
5. Reken hiermee de nieuwe radius uit via:
$$R_{t+1} = \sqrt{\sum_{i \in SC_{t+1}} (k(\mathbf{x}_i, \mathbf{x}_i) + 1 + \frac{1}{C}) - \sum_{j \in SC_{t+1}} \alpha_i \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_j)}.$$
6. Update $t = t+1$ en ga naar stap 2.

Per iteratie wordt maar 1 punt toegevoegd aan de Core Set, dus de oplossing in stap 4 zal niet ver liggen van de oplossing uit de vorige iteratie. Dus als tussen de iteraties door steeds de vector van Lagrange multiplicatoren wordt onthouden, kan hiermee de oplossing van de volgende iteratie veel sneller worden gevonden. De auteurs noemen dit een *warm start*.

Het zoeken van het verste punt in stap 3 kan erg veel tijd kosten, om het verste punt te zoeken moeten alle punten in de dataset elke iteratie vergeleken worden met punten in de Core Set. Echter in [14] staat voor dit probleem een snellere oplossing beschreven, de *probabilistic*

speedup. Als er een voldoende grote willekeurige subset van de datapunten wordt genomen, dan is de kans dat daar een punt in zit die bij de 5% verste punten hoort 95%. Door dit te gebruiken hoeven er elke iteratie maar 2 kleine verzamelingen punten met elkaar te worden vergeleken.

4. Implementatie

Bij de implementatie van het prototype is gebruikt gemaakt van de taal *Python*, versie 2.6 [1]. Deze taal leent zich uitstekend om snel prototypes in te bouwen, maar is op zichzelf niet bijzonder geschikt voor rekenwerk. Daarom is er veel gebruik gemaakt van de Python bibliotheken *Numpy* en *Scipy* [2]. Deze bibliotheken maken efficiënte operaties op matrices mogelijk. Het prototype werkt in fases, deze zullen in de volgende secties één voor één aan bod komen.

4.1. Data verzamelen

Zodra het programma wordt gestart, zal het beginnen met het inlezen van de dataset. Voor dit prototype is het geen probleem dat alle data in het werkgeheugen wordt opgeslagen, echter voor een echt onderdeel van een IDS zou dit niet kunnen. Verbindingen moeten dan getest worden tijdens, of vlak na het moment dat ze gemaakt worden.

Het classificatiealgoritme kan alleen omgaan met reële getallen, echter de dataset bevat ook symbolische informatie, zoals het protocol dat wordt gebruikt voor de verbinding. Alle mogelijke symbolen worden in deze fase verzameld, zodat ze kunnen worden omgezet in getallen. Voor elk uniek symbool in de dataset wordt een extra feature gecreëerd. Deze zal de waarde 0 hebben als een symbool niet in de beschrijving van een bepaalde verbinding voorkomt en de waarde 1 als het symbool wel voorkomt. Dit worden indicator variabelen genoemd.

Na deze bewerkingen zal de gehele dataset uit reële getallen bestaan. Het programma zal de dataset daarna omzetten naar een *sparse matrix* formaat. Dit levert een grote geheugenwinst op, doordat een groot gedeelte van de features van de datapunten de waarde 0 hebben. Deze representatie wordt opgeslagen op schijf, voor als de dataset later nog eens gebruikt wordt.

4.2. Trainingsfase

De dataset bevat meer dan twee klassen, maar we zijn alleen geïnteresseerd in of een verbinding normaal is. Daarom wordt er alleen onderscheid gemaakt tussen normale verbindingen en foute verbindingen. In de trainingsfase wordt het model geleerd uit de data. In principe is het ook mogelijk om per type aanvalscategorie een model te trainen. Echter, in de gebruikte dataset zijn dergelijke labels niet beschikbaar in de testset.

[1] Meer informatie op `http://python.org/`
[2] beiden zijn te vinden op `http://scipy.org/`

Voor het oplossen van het optimalisatieprobleem in Vergelijking 3.24 wordt de bibliotheek CVXOPT [3] gebruikt. Om hiervan gebruik te maken moet het optimalisatieprobleem worden omgevormd naar een geschikt formaat. CVXOPT lost het volgende probleem op:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x}+\mathbf{q}^T\mathbf{x}, \tag{4.1}$$

waarbij moet worden voldaan aan de voorwaarden:

$$\mathbf{G}\mathbf{x} \leq h, \tag{4.2}$$

en

$$\mathbf{A}\mathbf{x} = b, \tag{4.3}$$

en geeft dan x terug. Dit kan gebruikt worden om Vergelijking 3.24 met de voorwaarden op te lossen, door de juiste waarden voor $\mathbf{G}$, $\mathbf{A}$, h, $\mathbf{q}$ en b te kiezen.

4.3. Testfase

In de testfase wordt elk datapunt door het model van een score voorzien, met Vergelijking 3.1. Verder wordt de *precisie*, de *true positive rate* en de *false positive rate* bijgehouden. De precisie is de fractie van verbindingen die goed worden geclassificeerd, echter dit alleen is niet genoeg. Neem bijvoorbeeld een dataset met twee klassen die niet uniform verdeeld zijn, de precisie zou hoog kunnen zijn als er voor elk willekeurig punt steeds dezelfde klasse zou worden aangewezen. De *true positive rate* wordt gegeven door:

$$TPR = \frac{TP}{TP+FP}, \tag{4.4}$$

dit is het gedeelte van de punten die in een klasse zijn geplaatst en die daar ook daadwerkelijk behoren. Analoog hieraan, bestaat er de *false positive rate*:

$$FPR = \frac{TN}{TN+FN}. \tag{4.5}$$

Voor een intrusion detection systeem is het belangrijk dat de false positive rate zo laag mogelijk is. Als er teveel false positives zijn, kunnen die de normale werkzaamheden zo vaak onderbreken, dat detectie niet meer nuttig is.

[3] meer informatie op `http://abel.ee.ucla.edu/cvxopt/`

5. Resultaten

5.1. Experimentele opzet

De KDDCup-99 dataset bestaat uit een trainingsset met daarin gegevens over 4.898.431 netwerkverbindingen en een testset met daarin gegevens over nog eens 311.029 netwerkverbindingen. In de testset komen ook klassen voor die niet in de trainingsset zitten.

De CVM heeft 2 parameters die gevarieerd kunnen worden. Als eerste is er de complexiteitsconstante C, die een weging geeft tussen de hoeveelheid punten die fout geclassificeerd mogen worden en de uiteindelijke generalisatie van de CVM. Ten tweede is er ϵ, de precisie waarmee een benadering wordt gezocht voor de MEB. Een ϵ van 0 houdt in dat informatie in de trainingsset exact wordt overgenomen. Er is geprobeerd om een optimale waarde voor deze twee parameters te vinden voor deze specifieke dataset, door middel van experimenten op een kleine subset van de dataset.

5.2. Parameters

We beginnen met het uitzoeken van welke parameters het beste werken voor de CVM. We nemen hiervoor steeds een willekeurige subset van 1000 verbindingen, waarmee een classificatiemodel word gemaakt. Verder worden er 500 andere verbindingen genomen om het verkregen model op te testen. In de literatuur wordt hiervoor vaak de waarde 1000 of 1000000 gekozen. De resultaten voor $C = 1, 2, 5, 10, 100, 1000, 1000000$ zijn te vinden in Tabel 5.1. In deze tabel blijkt $C = 10$ de beste keuze te zijn, met een minimale *false positive rate* van 0.3% en een maximale *true positive rate* van 99.0%.

	ACC	TPR	FPR
normal $C = 1$	0.916	0.596	0.000
normal $C = 2$	0.924	0.990	0.093
normal $C = 5$	0.782	0.990	0.273
normal $C = 10$	0.996	0.990	0.003
normal $C = 100$	0.970	0.865	0.003
normal $C = 1000$	0.948	0.750	0.000
normal $C = 1000000$	0.958	0.808	0.003

Tabel 5.1.: Resultaten met $\epsilon = 10^{-6}$

	ACC	TPR	FPR
normal $\epsilon = 0.1$	0.930	0.990	0.086
normal $\epsilon = 0.01$	0.992	0.990	0.008
normal $\epsilon = 0.001$	0.992	0.990	0.008
normal $\epsilon = 0.000001$	0.992	0.990	0.008
normal $\epsilon = 0.000000001$	0.992	0.990	0.008

Tabel 5.2.: Resultaten met $C = 10$

	ACC	TPR	FPR
dos	0.835	0.778	0.000
probe	0.299	1.000	0.710
u2r	1.000	0.000	0.000
r2l	0.944	0.020	0.011
normal	**0.923**	**0.983**	**0.091**

Tabel 5.3.: Resultaten met $C = 10$ en $\epsilon = 0.01$ op de hele dataset

De tweede parameter is de ϵ, de precisie waarmee een benadering wordt gezocht voor de MEB. Het is belangrijk om deze waarde zo groot mogelijk te houden, want hoe preciezer de benadering is, hoe meer tijd het kost om de MEB uit te rekenen. Er zijn experimenten gedaan met de volgende waarden voor $\epsilon = 0.1, 0.01, 0.001, 0.000001, 0.000000001$, de resultaten daarvan zijn te zien in Tabel 5.1. Hier blijkt de waarde $\epsilon = 0.01$ de beste afweging tussen snelheid en precisie.

5.3. Resultaten

Met de gevonden optimale parameterwaarden is een model getraind op de gehele trainingsset met 5 miljoen verbindingsgegevens. Het verkregen model is vervolgens gebruikt om de verbindingsgegevens in de testset mee te classificeren. Daarbij werden de resultaten in Tabel 5.3 behaald.

Het model heeft een true positive rate van 98.3%, het merkt bijna alle normale verbindingen als normaal aan. Ook is te zien dat het model een false positive rate van 9.1% heeft. Dit is niet acceptabel, 9.1% van de aanvallen zouden door het filter als normaal worden aangemerkt. Echter, de resultaten doen ook vermoeden dat dit vooral ligt aan de moeilijkheden bij het herkennen van de probe categorie. Probe is op zichzelf geen aanval, dus dit hoeft geen probleem te zijn.

Dit model werd uit de volledige dataset geleerd in 59 minuten. Vergeleken met andere methoden is dit vrij snel. Opvallend is dat het testen vele malen langer duurt, de complete testfase op 311.029 verbindingen duurde 10 uur en 31 minuten. Dit ligt aan het gebruik van de probabilistische versnelling in de Core Vector Machine, maar een kleine fractie van de punten

worden tijdens de trainingsfase bekeken.

6. Conclusie

In deze bachelorscriptie is geprobeerd om een antwoord te vinden op de vraag: Is de SVM een geschikte methode om als classificatiesysteem te gebruiken voor een intrusion detection systeem? Daarbij heeft de nadruk gelegen op de deelvraag: Hoe kan de SVM binnen acceptabele tijd uit de dataset geleerd worden?

Om deze vraag op te lossen is een literatuuronderzoek gedaan naar nieuwe SVM varianten die met grote hoeveelheden data kunnen omgaan. Daaruit is de meest veelbelovende, de Core Vector Machine, geïmplementeerd. De CVM is inderdaad een methode die in zeer korte tijd een model uit een grote dataset kan leren. In veel gevallen is het leren van een model sneller klaar dan valideren van het model met een testset. Dit zou ook kunnen liggen aan de gebruikte implementatietaal Python, maar waarschijnlijk komt het door het gebruik van de probabilistische speedup bij het trainen. Bij de CVM blijft de Core Set zeer klein, er zijn meestal niet meer dan 40 Core Vectors, daarom worden er bij het trainen ook zeer weinig matrixmultiplicaties uitgevoerd. Zodra het model moet worden getest, wordt voor elke punt uit de testset het inproduct met alle punten uit de Core Set uitgerekend.

De klassendistributie is niet uniform verdeeld, het blijkt dat van sommige klassen aanvallen zeer weinig voorbeelden beschikbaar zijn in de dataset. Waarschijnlijk zouden de resultaten met de testset beter worden indien van alle klassen genoeg trainingsdata beschikbaar was.

Verder blijkt dat de Core Vector Machine in ieder geval snel genoeg is om in een acceptabele tijd een nieuw model voor een intrusion detection systeem te leren. De testfase kan nog vele malen versneld worden door gebruik te maken van een andere implementeringstaal.

A. Appendices

A.1. Prototype

Deze appendix bevat de broncode van het geproduceerde prototype.

A.1.1. cerberus.py

De hoofdmodule van het prototype.

```
#!/usr/bin/env python
# vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:

################################################################################
# File: cerberus.py
#
# Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
#
# This program is free software: you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation, either version 3 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
# details.
#
# You should have received a copy of the GNU General Public License along with
# this program.  If not, see <http://www.gnu.org/licenses/>.
################################################################################

'''
Cerberus is prototype analysis and detection module for an intrusion detection
system. It uses the Core Vector Machine (CVM) for analysis.
'''

# System
import sys
import traceback
import logging
import cPickle as pickle

```

```
34 # 3rd-party
35 import argparse
36
37 # Local
38 import common.data
39 import common.model
40 import train.trainer
41 import classify.predictor
42
43 def data_collection( header_file, data_file ):
44     ''' Collect data '''
45     logging.info ( 'Data collection phase...' )
46     dataset = common.data.Dataset()
47     dataset.read( header_file, data_file )
48     return dataset
49
50 def feature_selection( dataset ):
51     ''' In the future do some feature selection '''
52     logging.info ( 'Feature selection phase...' )
53     logging.info ( '  Not yet implemented' )
54     return dataset
55
56 def prepare_training( dataset, C, EPS, kernel, gamma ):
57     ''' Prepare for training '''
58     logging.info ( 'Prepare for training phase...' )
59     trainer = train.trainer.Trainer( dataset, C , EPS, kernel, gamma )
60     return trainer
61
62 def do_training( trainer ):
63     ''' Construct classifiers from the training set'''
64     logging.info ( 'Training phase...' )
65     models = trainer.train()
66     return models
67
68 def load_models( model_file ):
69     ''' Load trained models from file '''
70     logging.info ( 'Loading models...' )
71     models = pickle.load( model_file )
72     return models
73
74 def save_models( models, model_file ):
75     ''' Save trained models to file '''
76     logging.info ( 'Saving models...' )
77     pickle.dump( models, model_file, pickle.HIGHEST_PROTOCOL )
78
79 def prepare_testing( models, dataset ):
80     ''' Initialisation for the testing phase '''
81     logging.info ( 'Prepare for testing phase...' )
82     predictor = classify.predictor.Predictor( models, dataset )
83     return predictor
```

```
84
85 def do_testing( predictor ):
86     ''' Do the testing phase '''
87     logging.info ( 'Testing phase...' )
88     predictor.concurrent_predict()
89     #predictor.predict()
90
91 def initialise_cli ():
92     ''' Set up command line arguments '''
93     parser = argparse.ArgumentParser( description=__doc__ )
94
95     parser.add_argument( 'header', type=argparse.FileType( 'r' ),
96                          help='Read feature information from this file.' )
97
98     parser.add_argument( 'dataset', type=str,
99                          help='Read data from this file.' )
100
101    parser.add_argument( '--log', type=argparse.FileType( 'a' ),
102                         default=sys.stdout,
103                         help='Send log output to this file \
104                               (defaults to stdout)' )
105
106    parser.add_argument( '--C', type=float,
107                         default=1e6,
108                         help='Complexity constant (defaults to 1000000)' )
109
110    parser.add_argument( '--EPS', type=float,
111                         default=1e-6,
112                         help='Epsilon, defaults to 1e-6' )
113
114    parser.add_argument( '--gamma', type=float,
115                         default=0.1,
116                         help='Gamma for RBF kernel' )
117
118    parser.add_argument( '--kernel', type=int,
119                         default=1,
120                         help='Kernel type: 0 -> dot, 1 -> RBF' )
121
122    group = parser.add_mutually_exclusive_group( required=True )
123    group.add_argument( '--model_out', type=argparse.FileType( 'w' ),
124                         help='Store trained models in this file.' )
125
126    group.add_argument( '--model_in', type=argparse.FileType( 'r' ),
127                         help='Load trained models from this file.' )
128
129    return parser
130
131 def initialise_logging ( log_file ):
132     ''' Initialise the logging module '''
133     logging.basicConfig( level=logging.DEBUG,
```

```
134                             format='%(asctime)s - %(levelname)-8s - %(message)s',
135                             datefmt='%Y-%m-%d %H:%M:%S',
136                             stream=log_file, )
137     logging.info( 'Initialised Cerberus' )
138 
139 def main():
140     ''' Entrypoint '''
141     parser = initialise_cli()
142     arguments = parser.parse_args()
143     initialise_logging( arguments.log )
144     try:
145         if arguments.model_in == None and arguments.model_out != None:
146             trainset = data_collection( arguments.header, arguments.dataset )
147             features = feature_selection( trainset )
148             trainer = prepare_training( trainset, arguments.C, arguments.EPS,
149                                         arguments.kernel, arguments.gamma )
150             models = do_training( trainer )
151             save_models( models, arguments.model_out )
152             arguments.header.seek( 0 )
153         elif arguments.model_in != None and arguments.model_out == None:
154             models = load_models( arguments.model_in )
155             testset = data_collection( arguments.header, arguments.dataset )
156             predictor = prepare_testing( models, testset )
157             do_testing( predictor )
158     except:
159         exceptionType, exceptionValue, exceptionTraceback = sys.exc_info()
160         logging.error( traceback.format_exc() )
161 
162 if __name__ == '__main__':
163     sys.exit( main() )
```

A.1.2. common/data.py

De data-verzamel-fase van het systeem.

```
1  #!/usr/bin/env python
2  # vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:
3  
4  ##############################################################################
5  # Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
6  #
7  # This program is free software: you can redistribute it and/or modify it under
8  # the terms of the GNU General Public License as published by the Free Software
9  # Foundation, either version 3 of the License, or (at your option) any later
10 # version.
11 #
12 # This program is distributed in the hope that it will be useful, but WITHOUT
13 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
14 # FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
15 # details.
```

```
16 #
17 # You should have received a copy of the GNU General Public License along with
18 # this program.  If not, see <http://www.gnu.org/licenses/>.
19 ###############################################################################
20
21 '''
22 File: data.py
23
24 This is the "data" module. It contains data structures for labelled and
25 unlabelled sparse data sets, for use within a linear classifier. It should
26 eventually provide several convenience methods, e.g. for IO, splitting,
27 shuffling, etc.
28 '''
29
30 # System
31 import logging
32 import sys
33 import pprint
34 import cython
35
36 # 3rd-party
37 import numpy as np
38 from scipy import sparse
39
40 class Dataset( object ):
41     ''' Data structure representing a dataset.
42     '''
43
44     # To be more useful with other datasets eventually this should be removed,
45     # either the dataset itself will need to be changed, or some more general
46     # mechanism will need to be implemented.
47     TRAINING_ATTACK_TYPES = {
48         'back': 'dos',
49         'buffer_overflow': 'u2r',
50         'ftp_write': 'r2l',
51         'guess_passwd': 'r2l',
52         'imap': 'r2l',
53         'ipsweep': 'probe',
54         'land': 'dos',
55         'loadmodule': 'u2r',
56         'multihop': 'r2l',
57         'neptune': 'dos',
58         'nmap': 'probe',
59         'normal': 'normal',
60         'perl': 'u2r',
61         'phf': 'r2l',
62         'pod': 'dos',
63         'portsweep': 'probe',
64         'rootkit': 'u2r',
65         'satan': 'probe',
```

```
        'smurf': 'dos',
        'spy': 'r2l',
        'teardrop': 'dos',
        'warezclient': 'r2l',
        'warezmaster': 'r2l',
        'apache2':'dos',
        'back':'dos',
        'buffer_overflow':'u2r',
        'ftp_write':'r2l',
        'guess_passwd':'r2l',
        'httptunnel':'r2l',
        'httptunnel':'u2r',
        'imap':'r2l',
        'ipsweep':'probe',
        'land':'dos',
        'loadmodule':'u2r',
        'mailbomb':'dos',
        'mscan':'probe',
        'multihop':'r2l',
        'multihop':'u2r', # note that this is a duplicate
        'named':'r2l',
        'neptune':'dos',
        'nmap':'probe',
        'perl':'u2r',
        'phf':'r2l',
        'pod':'dos',
        'portsweep':'probe',
        'processtable':'dos',
        'ps':'u2r',
        'rootkit':'u2r',
        'saint':'probe',
        'satan':'probe',
        'sendmail':'r2l',
        'smurf':'dos',
        'snmpgetattack':'r2l',
        'snmpguess':'r2l',
        'sqlattack':'u2r',
        'teardrop':'dos',
        'udpstorm':'dos',
        'warezmaster':'dos',
        'worm':'r2l',
        'xlock':'r2l',
        'xsnoop':'r2l',
        'xterm':'u2r',
    }

    #    TRAINING_ATTACK_TYPES = {
    #    'Iris-setosa':'Iris-setosa',
    #    'Iris-versicolor':'Iris-versicolor',
    #    'Iris-virginica':'Iris-virginica',
```

```
116     #}
117
118     def __init__( self ):
119         ''' A list of classes '''
120         self.__classes = None
121
122         #print ",".join(sorted(self.TRAINING_ATTACK_TYPES.keys()))
123         #stop
124
125         ''' "Feature: type" pairs. Type is either continuous or symbolic. '''
126         self.__features = None
127
128         ''' Number of symbolic features '''
129         self.__nr_symbolic = 0
130
131         ''' Number of continuous features '''
132         self.__nr_continuous = 0
133
134         ''' List of lists of features '''
135         self.__data = None
136
137         ''' Class labels, if present '''
138         self.__labels = None
139
140         ''' Is this dataset labelled? '''
141         self.__has_labels = False
142
143         ''' Number of points in the dataset '''
144         self.__nr_points = 0
145
146     def read ( self, header_file, data_file ):
147         ''' Loads the data '''
148
149         self.__read_header( header_file )
150         self.__read_data( data_file )
151
152     def get_nr_features( self ):
153         return ( sum( map( len, self.__features[2] ) ) + self.__nr_continuous )
154
155     def get_nr_points( self ):
156         ''' Return the total number of points in this dataset '''
157         return self.__nr_points
158
159     def get_point( self, index ):
160         ''' Returns the data point at index '''
161         return self.__data[index].toarray().reshape( -1 )
162
163     def get_sparse_point( self, index ):
164         ''' Returns the data point at index '''
165         return self.__data[index, :]
```

```
166
167     def get_label( self, index ):
168         ''' Returns the label for the point at index '''
169         return self.__labels[index]
170
171     def get_classes ( self ):
172         ''' Returns a list of classes '''
173         return self.__classes
174
175     def __read_header( self, names ):
176         self.__classes = list()
177         self.__features = list()
178         self.__nr_continuous = 0
179         self.__nr_symbolic = 0
180
181         self.__classes = names.readline().strip( ".\n" ).split( ',' )
182         self.__classes = ['normal', 'r2l', 'u2r', 'probe', 'dos']
183         for line in names:
184             key_value = line.strip( ".\n" ).split( ':' )
185             if key_value[1] == 'continuous':
186                 self.__features.append( ( key_value[0], key_value[1] ) )
187                 self.__nr_continuous += 1
188             else:
189                 self.__features.append( ( key_value[0], key_value[1].strip(
190                                           ".\n" ).split( ',' ) ) )
191                 self.__nr_symbolic += 1
192
193         logging.info( '  Read {0} features from file'.
194                       format( self.__nr_continuous + self.__nr_symbolic ) )
195         logging.info( '    of which {0} are symbolic and {1} are continuous'.
196                       format( self.__nr_symbolic, self.__nr_continuous ) )
197
198     def __read_data( self, data_file ):
199         ''' Read the actual data '''
200
201         logging.info( '  Read data from file.' )
202         self.__data = np.loadtxt( data_file, delimiter=',' )
203         #self.__has_labels = ( self.__nr_continuous +
204         #                      self.__nr_symbolic ) < self.__data.shape[1]
205         self.__has_labels = True
206         self.__nr_points = self.__data.shape[0]
207
208         logging.info( '  Read {0} connections from file'.
209                       format( self.__nr_points ) )
210
211         if self.__has_labels:
212             logging.info( '    labels are present' )
213             self.__labels = list()
214             keys = self.TRAINING_ATTACK_TYPES.keys()
215             keys.sort()
```

```
216
217             for i in self.__data[:, -1]:
218                 self.__labels.append( self.TRAINING_ATTACK_TYPES[keys[int( i )]] )
219
220             # Strip labels
221             self.__data = self.__data[:, 0:-1]
222         else:
223             logging.info( '    labels are not present' )
224
225         logging.info( '  Converting to sparse format' )
226         self.__data = sparse.csr_matrix( self.__data, dtype=np.float )
227         self.__data.eliminate_zeros()
228
229 if __name__ == '__main__':
230     import doctest
231     doctest.testmod()
```

A.1.3. common/model.py

De modellen die geleerd worden.

```
1  #!/usr/bin/env python
2  # vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:
3
4  ##############################################################################
5  # Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
6  #
7  # This program is free software: you can redistribute it and/or modify it under
8  # the terms of the GNU General Public License as published by the Free Software
9  # Foundation, either version 3 of the License, or (at your option) any later
10 # version.
11 #
12 # This program is distributed in the hope that it will be useful, but WITHOUT
13 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
14 # FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
15 # details.
16 #
17 # You should have received a copy of the GNU General Public License along with
18 # this program.  If not, see <http://www.gnu.org/licenses/>.
19 ##############################################################################
20
21 '''
22 File: model.py
23
24 moduledocs
25 '''
26
27 import math
28 import numpy as np
29 from scipy import sparse
```

```
30 import common.kernels as kernels
31
32 class Model( object ):
33
34     def __init__( self, alphas, b, class_name, nr_svs, support_vector_labels,
35                   support_vectors, kernel_func, gamma, ro, C ):
36         self.__class_name = class_name
37         self.__support_vectors = support_vectors.copy()
38         self.__support_vector_labels = support_vector_labels.copy()
39         self.__nr_svs = nr_svs
40         self.__alphas = alphas.copy()
41         self.__b = b
42         self.__kernel_func = kernel_func
43         self.__gamma = gamma
44         self.__ro = ro
45         self.__C = C
46
47     def predict( self, x ):
48         ''' Predict class for point x. '''
49         score = 0.0
50         for i in range( self.__nr_svs ):
51             score += ( self.__alphas[i, 0] * \
52                        self.__support_vector_labels[i, 0] * \
53                        self.__kernel( i, x ) )
54
55         score += self.__b
56         return score
57
58     def __kernel( self, i, x ):
59         '''
60         The actual kernel function, only dot product for now
61         '''
62         return self.__kernel_func( x, self.__support_vectors[i], self.__gamma )
```

A.1.4. train/trainer.py

De trainingsfase van het prototype.

```
1 #!/usr/bin/env python
2 # vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:
3
4 ##############################################################################
5 # Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
6 #
7 # This program is free software: you can redistribute it and/or modify it under
8 # the terms of the GNU General Public License as published by the Free Software
9 # Foundation, either version 3 of the License, or (at your option) any later
10 # version.
11 #
12 # This program is distributed in the hope that it will be useful, but WITHOUT
```

```
13 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
14 # FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
15 # details.
16 #
17 # You should have received a copy of the GNU General Public License along with
18 # this program.  If not, see <http://www.gnu.org/licenses/>.
19 ##############################################################################
20
21 '''
22 File: trainer.py
23
24 moduledocs
25 '''
26
27 # System
28 import logging
29 import multiprocessing
30
31 # Local
32 import common.data
33 import common.model
34 import cvm
35
36 class Trainer( object ):
37     ''' classdocs
38     '''
39     dataset = None
40     C = 1e6
41     EPS = 1e-6
42     gamma = 2.0
43     sigma = 300.0
44     kernel = 0
45
46     def __init__( self, dataset, C, EPS, kernel, gamma ):
47         self.dataset = dataset
48         self.C = C
49         self.EPS = EPS
50         self.kernel = kernel
51         self.gamma = gamma
52
53     def train_tasklet ( self, label ):
54         ''' Train one classifier for one class '''
55         logging.info( "  Starting training job for class <{0}>".format( label ) )
56         classifier = cvm.CVM( label, self.dataset, self.EPS, self.kernel, self.C, self.gamma )
57         return ( label, classifier.train() )
58
59     def train( self ):
60         ''' Train a classifier for every class, tries to train the profiles
61             concurrently.
62         '''
```

```
63          classes = zip( [self] * len( self.dataset.get_classes() ), self.dataset.get_classes()
64          pool = multiprocessing.Pool( processes=len( classes ) )
65          # Workaround python bug: http://www.mail-archive.com/python-list@python.org/msg228648.
66          result = pool.map( train_tasklet_mp, classes )
67
68          #result = []
69          #for i in self.dataset.get_classes():
70          #    if i == 'normal':
71          #        result.append( self.train_tasklet( i ) )
72
73          models = {}
74          for model in result:
75              models[model[0]] = model[1]
76          return models
77
78  def train_tasklet_mp( ar, **kwar ):
79      return Trainer.train_tasklet( *ar, **kwar )
```

A.1.5. train/cvm.py

Implementatie van de Core Vector Machine.

```
1  #!/usr/bin/env python
2  # vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:
3
4  ##############################################################################
5  # Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
6  #
7  # This program is free software: you can redistribute it and/or modify it under
8  # the terms of the GNU General Public License as published by the Free Software
9  # Foundation, either version 3 of the License, or (at your option) any later
10 # version.
11 #
12 # This program is distributed in the hope that it will be useful, but WITHOUT
13 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
14 # FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
15 # details.
16 #
17 # You should have received a copy of the GNU General Public License along with
18 # this program.  If not, see <http://www.gnu.org/licenses/>.
19 ##############################################################################
20
21 '''
22 File: cvm.py
23
24 Implementation of the Core Vector Machine.
25 '''
26
27 # System
28 import logging
```

```
29 import random
30 import pprint
31 import math
32
33 # 3rd-party
34 import numpy as np
35 import scipy as sp
36 from scipy import sparse
37 import cvxopt
38 import cvxopt.solvers
39
40 # Local
41 import common.data
42 import common.model
43 import common.kernels as kernels
44 import classify.predictor
45
46 random.seed( 400 )
47
48
49 ''' Size of the initial set of samples taken '''
50 INITIAL_CS = 2
51
52 ''' Size of the initial epsilon '''
53 INITIAL_EPS = 0.1
54 EPS_SCALE = 0.5
55
56 ''' Size of the set of samples taken at each iteration '''
57 NR_SAMPLES = 60
58
59 ''' Maximum number of iterations to train '''
60 MAX_ITERS = 150
61
62 ''' Maximum number of iterations to train '''
63 MAX_TRIES = 7
64
65 class CVM( object ):
66     '''
67     Implementation of the Core Vector Machine described in the 2006 paper
68     by Tsang et al. "Core Vector Machines: Fast SVM Training on Very Large
69     Data Sets" published in The Journal of Machine Learning Research.
70     '''
71
72     # Frustrating fact: Python class attributes are global to all
73     # instances of that class
74
75     def __init__( self, class_name, dataset, EPSILON=1.0e-6, kernel=0,
76                   C=1e6, gamma=0.01 ):
77         self.__NR_SAMPLES = NR_SAMPLES
78
```

```
79          ''' Acceptable approximation for the MEB '''
80          self.__EPSILON = EPSILON
81
82          ''' Regularisation constant '''
83          self.__C = C
84
85          ''' RBF kernel '''
86          self.__gamma = gamma
87
88          ''' The dataset used for training. '''
89          self.__dataset = dataset
90
91          ''' The class this classifier will learn. '''
92          self.__class_name = class_name
93
94          if kernel == 0:
95              self.__kernel = kernels.norm_dot_kernel
96          elif kernel == 1:
97              self.__kernel = kernels.rbf_kernel
98          else:
99              logging.fatal( '  Unkown kernel type' )
100
101         '''
102         The core set: S
103         A list of pointers into the dataset
104         '''
105         self.__coreset = []
106
107         ''' The squared radius of the ball: R^2 '''
108         self.__radius2 = 0.0
109
110         ''' Class labels, translated to {-1, +1}, for the current class '''
111         self.__labels = []
112
113         ''' List of pointers to all positive records. '''
114         self.__pos_ids = []
115
116         ''' List of pointers to all negative records. '''
117         self.__neg_ids = []
118
119         ''' Data distribution for this class '''
120         self.__nr_pos = 0
121         self.__nr_neg = 0
122
123         ''' Should next sample be positive or negative? '''
124         self.__pos_turn = True
125
126         ''' Lagrange multipliers '''
127         self.__alphas = np.zeros( ( 1, 1 ), np.float )
128
```

```
129         ''' Small cache for the core vector kernel evaluations '''
130         self.__k_cache = np.zeros( ( 2, 2 ), np.float )
131 
132         self.__iteration = 0
133 
134         # Solver options
135         cvxopt.solvers.options['show_progress'] = False
136 
137         # Maximum number of iterations
138         cvxopt.solvers.options['maxiters'] = 100
139 
140         # Absolute accuracy
141         cvxopt.solvers.options['abstol'] = self.__EPSILON * self.__EPSILON
142 
143         # Relative accuracy
144         cvxopt.solvers.options['reltol'] = self.__EPSILON * self.__EPSILON
145 
146         # Tolerance for feasibility condition
147         cvxopt.solvers.options['feastol'] = 1e-12
148         cvxopt.solvers.options['refinement'] = 2
149 
150         logging.debug( ( "    Classifier for <{0}> starts with EPSILON: {1}, C:" +
151                          " {2}, NR_SAMPLES: {3}" ).format( self.__class_name,
152                                  self.__EPSILON,
153                                  self.__C,
154                                  self.__NR_SAMPLES ) )
155 
156         # Count class distribution
157         for i in range( self.__dataset.get_nr_points() ):
158             label = self.__dataset.get_label( i )
159             if label == self.__class_name:
160                 self.__labels.append( 1 )
161                 self.__pos_ids.append( i )
162                 self.__nr_pos += 1
163             else:
164                 self.__labels.append( -1 )
165                 self.__neg_ids.append( i )
166                 self.__nr_neg += 1
167 
168         logging.debug( "    Class distribution for class <{0}>:{1}, <other>:{2}.".
169                        format( self.__class_name, self.__nr_pos, self.__nr_neg ) )
170 
171     def train ( self ):
172         '''
173         Learns a linear model from training examples with 2 classes.
174         '''
175         self.__initialise()
176         self.__iteration = 0
177 
178         tries = 0
```

```
179         currentEpsilon = INITIAL_EPS
180         while( tries < MAX_TRIES and self.__iteration < MAX_ITERS ):
181 
182             if currentEpsilon < self.__EPSILON:
183                 currentEpsilon = self.__EPSILON
184 
185             eps_factor = ( 1. + currentEpsilon )
186             eps_factor *= eps_factor
187 
188             new_cs = self.__furthest_point( eps_factor * self.__radius2 )
189 
190             # No more points remain
191             if new_cs == None and tries >= MAX_TRIES:
192                 logging.info( "  Completed training for class <{0}> on max tries.".
193                               format( self.__class_name ) )
194                 break
195             elif new_cs == None:
196                 tries += 1
197             else:
198                 tries = 0
199                 self.__update_coreset( new_cs )
200                 self.__solve_qp()
201                 self.__update_radius2()
202                 self.__iteration += 1
203 
204         logging.info( "  Completed training for class <{0}> in {1} iterations".
205                       format( self.__class_name, self.__iteration ) )
206 
207         return self.__make_model()
208 
209     def __initialise( self ):
210         '''
211         Initialise S_0, c_0 and R_0
212         '''
213 
214         # Initialize the core set with a balanced sample
215         for i in range( INITIAL_CS ):
216             z_i = self.__next_sample()
217             if z_i != None:
218                 self.__update_coreset( z_i )
219                 self.__alphas[i, 0] = ( 1.0 / INITIAL_CS )
220 
221         # Construct a cache of kernel evaluations of the core vectors
222         n = len( self.__coreset )
223         for i in range( n ):
224             for j in range( n ):
225                 self.__k_cache[i, j] = self.__cvm_kernel( i, j )
226 
227         self.__solve_qp()
228 
```

```
229         # Compute a new radius
230         self.__update_radius2()
231 
232     def __next_sample( self ):
233         '''
234         Returns an id in the list of datapoints. Tries to return balanced
235         random samples when enough data is present.
236         '''
237         # No points left
238         if self.__nr_pos <= 0 and self.__nr_neg <= 0:
239             return None
240 
241         if self.__nr_pos <= 0:
242             self.__pos_turn = False
243         elif self.__nr_neg <= 0:
244             self.__pos_turn = True
245         else:
246             self.__pos_turn = not self.__pos_turn
247 
248         # Select a random point that has not yet been used
249         # if self.__pos_turn:
250         #r = random.randint( 1, 10 )
251         #if r <= 5 and self.__nr_pos > 0:
252         #    id = self.__pos_ids[random.randint( 0, self.__nr_pos - 1 )]
253         #elif r > 5 and self.__nr_neg > 0:
254         #    id = self.__neg_ids[random.randint( 0, self.__nr_neg - 1 )]
255         if self.__pos_turn:
256             id = self.__pos_ids[random.randint( 0, self.__nr_pos - 1 )]
257         else:
258             id = self.__neg_ids[random.randint( 0, self.__nr_neg - 1 )]
259 
260         return id
261 
262     def __solve_qp( self ):
263         '''
264         We need to solve the quadratic programme:
265 
266             maximise: - \alpha^T __cvm_kernel \alpha
267              \alpha
268 
269             s.t.: \alpha       >= 0
270                   \alpha^T 1    = 1
271 
272         using cvxopt. cvxopt solves:
273 
274             minimise: 0.5 x^T P x + q^T x
275              x
276 
277             s.t.: Gx    <= h
278                   Ax     = b
```

```
279
280         We transform the qp to:
281
282             minimise: 0.5 \alpha^T (__cvm_kernel) \alpha + 0^T \alpha
283              \alpha
284
285             s.t.: -1^T \alpha  <= 0
286                    1^T \alpha   = 1
287
288         to make it compatible with cvxopt.
289         '''
290         n = len( self.__coreset )
291
292         P = cvxopt.matrix( self.__k_cache )
293
294         q = cvxopt.matrix( 0.0, ( n, 1 ) )
295         G = cvxopt.matrix( 0.0, ( n, n ) )
296         G[:: n + 1] = -1.0
297         h = cvxopt.matrix( 0.0, ( n, 1 ) )
298         A = cvxopt.matrix( 1.0, ( 1, n ) )
299         b = cvxopt.matrix( 1.0 )
300         x = cvxopt.matrix( self.__alphas )
301
302         logging.info( "  {0} Solving QP for iteration {1}".format( self.__class_name,
303                                                                    self.__iteration ) )
304
305         solution = cvxopt.solvers.qp( P, q, G, h, A, b, None, {'x': x} )
306
307         if solution['status'] != 'optimal':
308             logging.debug( '    {0} Could not solve QP problem, results: {1}\n'.
309                            format( self.__class_name, pprint.pformat( solution ) ) )
310
311         # Update multipliers
312         self.__alphas = np.asarray( solution['x'] ).copy()
313         print self.__class_name, self.__iteration, pprint.pformat(solution)
314
315         #logging.debug( '  New Lagrange multipliers:\n' + pprint.pformat( self.__alphas ) )
316
317     def __update_radius2( self ):
318         '''
319         Update the MEB.
320
321         Calculate the new radius:
322             R^2 = \alpha^T diag(K) - \alpha^T K \alpha
323
324         as \alpha_i is 0 for non CVs, only entries corresponding to CVs need to
325         be evaluated.
326         '''
327         self.__radius2 = np.dot( self.__alphas.T, self.__k_cache.diagonal() )[0] - \
328                                  self.__get_ro()
```

```
329
330         logging.info( "  {0} New squared radius: {1:<10.11}".format( self.__class_name,
331                                                                       self.__radius2 ) )
332
333         # The radius is squared, it should be positive
334         assert( self.__radius2 >= 0.0 )
335
336     def __distance_to_centroid( self, z_l ):
337         '''
338         Computes the squared distance from a point to the center of
339         the coreset:
340
341             ||c_t - \phi(point)||^2
342             = \sum_{i,j} \alpha_i \alpha_j __cvm_kernel(z_i, z_j)
343             - 2 \sum_i \alpha_i __cvm_kernel(z_i, z_l) __cvm_kernel(z_l, z_l)
344
345         see Eq. 18 in the paper.
346         '''
347         cs_size = len( self.__coreset )
348
349         distance2 = 0.0
350         for i in range( cs_size ):
351             distance2 += self.__alphas[i, 0] * \
352                          self.__cvm_kernel( self.__coreset[i], z_l )
353
354         return ( self.__get_ro() - 2.0 * distance2 + self.__cvm_kernel( z_l, z_l ) )
355
356     def __furthest_point( self, prev_max_distance ):
357         '''
358         Take NR_SAMPLES samples and select the one point that is furthest away
359         from the current center c_t.
360         '''
361         max_distance_id = None
362         max_distance = prev_max_distance
363
364         for i in range( NR_SAMPLES ):
365             new_id = self.__next_sample()
366
367             # No more points
368             if new_id == None:
369                 break
370
371             new_distance = self.__distance_to_centroid( new_id )
372             if new_distance >= max_distance:
373                 max_distance = new_distance
374                 max_distance_id = new_id
375
376         return max_distance_id
377
378     def __update_coreset( self, max_dist_id ):
```

```
379         ''' Add new points to the coreset '''
380         assert( max_dist_id != None )
381         assert( ( len( self.__pos_ids ) +
382                   len( self.__neg_ids ) ) <= len( self.__labels ) )
383 
384         if self.__labels[max_dist_id] == 1:
385             self.__nr_pos -= 1
386             self.__pos_ids.remove( max_dist_id )
387         elif self.__labels[max_dist_id] == -1:
388             self.__nr_neg -= 1
389             self.__neg_ids.remove( max_dist_id )
390         else:
391             assert( self.__labels[max_dist_id] == 1 or
392                     self.__labels[max_dist_id] == -1 )
393 
394         self.__coreset.append( max_dist_id )
395         self.__alphas.resize( ( len( self.__coreset ), 1 ) )
396 
397         # Only do this for the newly added CV, the rest of the matrix can be
398         # copied from the previous iteration.
399         n = len( self.__coreset )
400         if n > 2:
401             new_row = np.zeros( ( 1, n ), np.float )
402             new_column = np.zeros( ( n - 1, 1 ), np.float )
403 
404             for i in range( n ):
405                 if i < ( n - 1 ):
406                     new_column[i, 0] = self.__cvm_kernel( i, n - 1 )
407                 new_row[0, i] = self.__cvm_kernel( n - 1, i )
408 
409             self.__k_cache = np.hstack( [self.__k_cache, new_column] )
410             self.__k_cache = np.vstack( [self.__k_cache, new_row] )
411         else:
412             for i in range( n ):
413                 for j in range( n ):
414                     self.__k_cache[i, j] = self.__cvm_kernel( i, j )
415 
416     def __make_model( self ):
417         '''
418         Store the lagrange multipliers, bias and core vectors. These will
419         be used for classifying unseen points.
420         '''
421         logging.info( '  Creating model' )
422         nr_cvs = len( self.__coreset )
423         alphas = sparse.lil_matrix( ( nr_cvs, 1 ) )
424         support_vectors = []
425         support_vector_labels = sparse.lil_matrix( ( nr_cvs, 1 ) )
426 
427         for i, cs in enumerate( self.__coreset ):
428             support_vectors.append( self.__dataset.get_point( cs ) )
```

```
429             support_vector_labels[i, 0] = self.__labels[cs]
430             alphas[i, 0] = self.__alphas[i, 0]
431 
432         # Storage in scipy sparse matrix format
433         support_vectors = np.asarray( support_vectors )
434 
435         support_vectors = sparse.csr_matrix( support_vectors, dtype=np.float )
436         support_vector_labels = support_vector_labels.tocsr()
437         alphas = alphas.tocsr()
438 
439         model = common.model.Model( alphas, self.__get_b(), self.__class_name,
440                                     nr_cvs, support_vector_labels,
441                                     support_vectors, self.__kernel, self.__gamma,
442                                     self.__get_ro(), self.__C )
443         return model
444 
445 
446     def __cvm_kernel ( self, i, j ):
447         '''
448         The CVM kernel function. A wrapper around a module compiled with
449         Cython.
450         '''
451         return kernels.cvm_kernel( self.__labels[i],
452                                    self.__labels[j],
453                                    self.__dataset.get_sparse_point( i ),
454                                    self.__dataset.get_sparse_point( j ),
455                                    i, j, self.__kernel, self.__gamma, self.__C )
456 
457     def __get_b( self ):
458         ''' Calculate the current bias '''
459         b = 0.0
460         n = len( self.__coreset )
461 
462         for i in range( n ):
463             b += ( self.__alphas[i, 0] * self.__labels[self.__coreset[i]] )
464 
465         return b
466 
467     def __get_ro( self ):
468         ''' Calculate the current Ro '''
469         return np.dot( np.dot( self.__alphas.T, self.__k_cache ),
470                        self.__alphas )[0, 0]
471 
472 if __name__ == '__main__':
473     import doctest
474     doctest.testmod()
```

A.1.6. classify/predictor.py

De testfase van het systeem.

```
#!/usr/bin/env python
# vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:

##############################################################################
# Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
#
# This program is free software: you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation, either version 3 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
# details.
#
# You should have received a copy of the GNU General Public License along with
# this program.  If not, see <http://www.gnu.org/licenses/>.
##############################################################################

'''
File: predictor.py

moduledocs
'''

# System
import pprint
import logging
import multiprocessing

# Local
import common.data
import common.model
import numpy as np


class Predictor( object ):
    '''
    classdocs
    '''

    def __init__( self, models, testset ):

        ''' Previous learned models '''
        self.__models = models

```

```
48          ''' A labelled testset '''
49          self.__testset = testset
50
51      def score_point ( self, model, point ):
52          ''' Calculate the score of one record '''
53          return model.predict( point )
54
55      def concurrent_score_point ( self, point ):
56          ''' Concurrently score points '''
57
58          if ( point % 1000 ) == 0 and point > 0:
59              logging.debug( '  Scored {0} records'.format(
60                              len( self.__models.keys() ) * 1000 ) )
61          true_label = self.__testset.get_label( point )
62          x = self.__testset.get_sparse_point( point )
63
64          scores = []
65          for label in self.__models.keys():
66              scores.append( ( label,
67                               self.__models[label].predict( x ),
68                               true_label ) )
69
70          return scores
71
72      def concurrent_predict( self ):
73          ''' Evaluate the models with the testset '''
74          nr_classes = len( self.__models.keys() )
75          classes = self.__models.keys()
76          nr_points = self.__testset.get_nr_points()
77
78          pool = multiprocessing.Pool( processes=(multiprocessing.cpu_count()-1) )
79          logging.debug( '  Scoring {0} records, using {1} CPUs'.format(
80                          nr_points, multiprocessing.cpu_count() )
81                      )
82
83          indices = zip( [self] * nr_points, range( nr_points ) )
84
85          scores = pool.map( concurrent_score_point_mp,
86                             indices, 100 )
87
88          class_statistics = {}
89          for label in self.__models.keys():
90              class_statistics[label] = {'total': 0.0,
91                                         'tp': 0.0,
92                                         'fp': 0.0,
93                                         'tn': 0.0,
94                                         'fn': 0.0 }
95
96          for score in scores:
97              for label, score, true_label in score:
```

```
98                  class_statistics[label]['total'] += 1.0
99                  #print "Pred:", label, "True: ", true_label, "Score:", score
100
101                 if score >= 0.0:
102                     if label == true_label:
103                         class_statistics[label]['tp'] += 1.0
104                     else:
105                         # label != true_label
106                         class_statistics[label]['fp'] += 1.0
107                 if score < 0.0:
108                     if label == true_label:
109                         class_statistics[label]['fn'] += 1.0
110                     else:
111                         # label != true_label:
112                         class_statistics[label]['tn'] += 1.0
113
114         self.print_class_statistics( class_statistics )
115
116         # Python dies without this print statement
117         print ''
118
119     def predict ( self ):
120         ''' Evaluate the models with the testset '''
121         nr_classes = len( self.__models.keys() )
122         classes = self.__models.keys()
123         nr_total = self.__testset.get_nr_points()
124         nr_scores = 0
125
126         # total, fp, fn, tp
127         class_statistics = {}
128
129         for label in self.__models.keys():
130             class_statistics[label] = {'total': 0.0,
131                                        'tp': 0.0,
132                                        'fp': 0.0,
133                                        'tn': 0.0,
134                                        'fn': 0.0 }
135
136         for point in range( nr_total ):
137             true_label = self.__testset.get_label( point )
138             x = self.__testset.get_sparse_point( point )
139
140             for index, label in enumerate( classes ):
141                 score = self.score_point( self.__models[label], x )
142                 nr_scores += 1
143
144                 class_statistics[label]['total'] += 1
145
146                 if score >= 0:
147                     if label == true_label:
```

```
148                         class_statistics[label]['tp'] += 1.0
149                     if label != true_label:
150                         class_statistics[label]['fp'] += 1.0
151                 if score < 0:
152                     if label == true_label:
153                         class_statistics[label]['fn'] += 1.0
154                     if label != true_label:
155                         class_statistics[label]['tn'] += 1.0
156 
157             if ( nr_scores % 1000 ) == 0 and nr_scores > 0:
158                 logging.debug( '  Calculated 1000 scores' )
159 
160         self.print_class_statistics( class_statistics )
161         #self.print_total_statistics( class_statistics )
162 
163     def print_total_statistics( self, class_statistics ):
164         ''' Print total statistics to the screen. '''
165         pass
166 
167     def print_class_statistics( self, class_statistics ):
168         ''' Print some statistics to the screen '''
169         print ''
170         print 'Class statistics:'
171         print ''
172         print '{0:<14}|{1:>6}{2:>6}{3:>6}{4:>6} |{5:>8}{6:>8}{7:>8}'.\
173               format( 'Pred. class', 'TP', 'TN', 'FP', 'FN', 'ACC', 'TPR',
174                       'FPR' )
175         print '-----------------------------------------------------------------'
176         for label in class_statistics.keys():
177             precision = 0.0
178             if ( class_statistics[label]['tp'] + class_statistics[label]['fp'] ) > 0:
179                 precision = ( class_statistics[label]['tp'] /
180                               ( class_statistics[label]['tp'] +
181                                 class_statistics[label]['fp'] ) )
182 
183             recall = 0.0
184             if ( class_statistics[label]['tp'] + class_statistics[label]['fn'] ) > 0:
185                 recall = ( class_statistics[label]['tp'] /
186                            ( class_statistics[label]['tp'] +
187                              class_statistics[label]['fn'] ) )
188 
189             fall_out = 0.0
190             if ( class_statistics[label]['fp'] + class_statistics[label]['tn'] ) > 0:
191                 fall_out = ( class_statistics[label]['fp'] /
192                              ( class_statistics[label]['fp'] +
193                                class_statistics[label]['tn'] ) )
194 
195             accuracy = 0.0
196             if class_statistics[label]['total'] > 0:
197                 accuracy = ( class_statistics[label]['tp'] +
```

```
198                             class_statistics[label]['tn'] ) / \
199                             class_statistics[label]['total']
200
201             f1 = 0.0
202             if ( precision + recall ) > 0:
203                 f1 = ( 2.0 * precision * recall ) / ( precision + recall )
204
205             print '{0:<14}|{1:>6}{2:>6}{3:>6}{4:>6} |{5:>8.4}{6:>8.4}{7:>8.4}'.\
206                    format( label,
207                            int( class_statistics[label]['tp'] ),
208                            int( class_statistics[label]['tn'] ),
209                            int( class_statistics[label]['fp'] ),
210                            int( class_statistics[label]['fn'] ),
211                            accuracy,
212                            recall,
213                            fall_out )
214
215     def print_confusion_matrix( self, classes, confusion ):
216         ''' Print a confusion matrix to the screen '''
217
218         print ''
219         print 'Confusion matrix:'
220         print ''
221         print '{0:>8} | '.format( ' ' ),
222         for actual, alabel in enumerate( classes ):
223             print '{0:>8}'.format( alabel ),
224         print ''
225         print '-----------------------------------------------------------'
226
227         for pred, plabel in enumerate( classes ):
228             print '{0:<8} | '.format( plabel ),
229             for actual, alabel in enumerate( classes ):
230                 print '{0:>8}'.format( int( confusion[pred, actual] ) ),
231             print ''
232
233         print ''
234         print '* Columns are truth, rows are predicted.'
235         print ''
236
237 if __name__ == '__main__':
238     import doctest
239     doctest.testmod()
240
241 def concurrent_score_point_mp ( ar, **kwarg ):
242     return Predictor.concurrent_score_point( *ar, **kwarg )
```

A.1.7. common/kernels.pyx

Aparte module voor de gebruikte kernelfuncties. Als optimalisatie geïmplementeerd in Cython.

```
#!/usr/bin/env python
# vim: set fileencoding=utf-8 ts=4 sw=4 expandtab:

##############################################################################
# Copyright (c) 2009, Jelle Sch hmacher <j.schuhmacher@student.science.ru.nl>
#
# This program is free software: you can redistribute it and/or modify it under
# the terms of the GNU General Public License as published by the Free Software
# Foundation, either version 3 of the License, or (at your option) any later
# version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
# FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
# details.
#
# You should have received a copy of the GNU General Public License along with
# this program.  If not, see <http://www.gnu.org/licenses/>.
##############################################################################
cimport numpy as np
cimport cython

import logging
import numpy as np
import scipy as sp
import scipy.sparse as sparse

@cython.boundscheck( False )
def norm_dot_kernel ( x_i, x_j ):
    assert( sparse.isspmatrix_csr( x_i ) and sparse.isspmatrix_csr( x_j ) )
    cdef double k_ij = ( x_i * x_j.T )[0, 0]
    cdef double k_ii = ( x_i * x_i.T )[0, 0]
    cdef double k_jj = ( x_j * x_j.T )[0, 0]

    cdef double res = ( k_ij / ( np.math.sqrt( k_ii ) * np.math.sqrt( k_jj ) ) )
    return res

@cython.boundscheck( False )
def dot_kernel ( x_i, x_j ):
    assert( sparse.isspmatrix_csr( x_i ) and sparse.isspmatrix_csr( x_j ) )
    cdef double res = ( x_i * x_j.T )[0, 0]
    return res

@cython.boundscheck( False )
def rbf_kernel ( x_i, x_j, double GAMMA ):
    assert( sparse.isspmatrix_csr( x_i ) and sparse.isspmatrix_csr( x_j ) )
    cdef double res = np.exp( -GAMMA * ( ( x_i * x_i.T )[0,0] - 2.0 * ( x_i * x_j.T )[0,0] + (
        .T )[0,0] ) )
    return res

```

```
@cython.boundscheck( False )
def cvm_kernel ( double label_i, double label_j, x_i, x_j, unsigned int i,
                 unsigned int j, kernel, GAMMA, C ):
    ''' The CVM kernel function
    '''
    assert( sparse.isspmatrix_csr( x_i ) and sparse.isspmatrix_csr( x_j ) )
    cdef double d_ij = 0.0
    if i == j:
        d_ij = 1.0

    cdef double t_ij = label_i * label_j
    cdef double res = t_ij * kernel( x_i, x_j, GAMMA ) + t_ij + ( d_ij / C )
    return  res
```

A.1.8. scripts/c45-to-numpy.py

Script om de data in het juiste formaat te krijgen.

```
#!/usr/bin/env python2.6

from numpy import *
import argparse

def read_names( filename ):
    classes = list()
    features = list()

    with open( filename + '.names', 'r' ) as names:
        classes = names.readline().strip( ".\n" ).split( ',' )
        for line in names:
            key_value = line.strip( ".\n" ).split( ':' )
            if key_value[1] == 'continuous':
                features.append( ( key_value[0], key_value[1] ) )
            else:
                features.append( ( key_value[0], key_value[1].strip( ".\n" ).split( ',' ) ) )

    return ( classes, features )

def read_data( filename, classes, features ):
    nr_features = len( features )
    counter = 0

    nr_total_features = 1
    for feature in features:
        if feature[1] == 'continuous':
            nr_total_features += 1
        else:
            nr_total_features += len( feature[1] )

#    data = None
```

```
33      data = []
34
35      with open( filename + '.data', 'r' ) as datafile:
36          for line in datafile:
37              record = line.strip( ".\n" ).split( ',' )
38              numeric_record = zeros( nr_total_features, dtype=float32 )
39
40              index1 = 0
41              for index2, feature in enumerate( record ):
42
43                  if index2 < nr_features and features[index2][1] == 'continuous':
44                      numeric_record[index1] = feature
45                      index1 += 1
46                  elif index2 == nr_features:
47                      if feature not in classes:
48                          classes.append(feature)
49                          print feature
50                      numeric_record[index1] = classes.index( feature )
51                      index1 += 1
52                  else:
53                      for indicator in features[index2][1]:
54                          if feature == indicator:
55                              numeric_record[index1] = 1.
56                          else:
57                              numeric_record[index1] = 0.
58                          index1 += 1
59                      func = None
60
61  #             if data == None:
62  #                 data = numeric_record
63  #             else:
64  #                 data = vstack( (data, numeric_record) )
65              data.append(numeric_record)
66              counter += 1
67              record = None
68              if ( counter % 10000 ) == 0:
69                  print "{0:>6.4} % of lines read...".format( ( counter / 4898431. ) * 100. )
70
71      return data
72
73  def normalise( data ):
74      maxima = amax(data, axis=0)
75      print maxima
76      maxima = maxima + select( [maxima == 0, True], [1., 0.] )
77      maxima[-1] = 1.
78      print maxima
79      data = data /  maxima
80      return data
81
82  if __name__ == '__main__':
```

```
83      parser = argparse.ArgumentParser( description=__doc__ )
84      parser.add_argument( 'dataset_in', type=str,
85                      help='Read data from this file.' )
86
87      parser.add_argument( 'dataset_out', type=str,
88                      help='Write data to this file.' )
89
90      arguments = parser.parse_args()
91      filename = arguments.dataset_in
92      filename_out = arguments.dataset_out
93
94      print "Read names.."
95      classes, features = read_names( filename )
96      print "Read data.. & convert to matrix"
97      data = read_data( filename, classes, features )
98      data = array(data)
99      print "Save array.."
100     savetxt( filename_out + '.txt.gz', data, '%.10g', ',' )
101     print "Normalise data..."
102     data = normalise( data )
103     print "Save normalised array.."
104     savetxt( filename_out + '-normalised.txt.gz', data, '%.10g', ',' )
```

Bibliography

[1] *ACM SIGKDD: 1999 KDD Cup competition*. http://www.sigkdd.org/kddcup/index.php?section=1999&method=info. 1999.

[2] Stefan Axelsson. *Research in Intrusion-Detection Systems: A Survey*. Tech. rep. University of Technology, Goteborg, Sweden, 1998.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006, pp. 291–339. ISBN: 0-38-731073-8.

[4] Olivier Chapelle. "Training a Support Vector Machine in the Primal". In: *Neural Comput.* 19.5 (2007), pp. 1155–1178. ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.5.1155.

[5] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[6] Nello Cristianini and John Shawe-Taylor. *An introduction to Support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, 2000, pp. 93–112. ISBN: 0-521-78019-5.

[7] Thorsten Joachims. "Training linear SVMs in linear time". In: *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 217–226. ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150429.

[8] S. Sathiya Keerthi, Olivier Chapelle, and Dennis DeCoste. "Building Support Vector Machines with Reduced Classifier Complexity". In: *Journal of Machine Learning Research* 7 (Dec. 2006), pp. 1493–1515. ISSN: 1533-7928.

[9] Kristopher Kendall. "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems". MA thesis. Massachusetts Institute of Technology, 1999.

[10] Yoshiaki Koshiba and Shigeo Abe. "Comparison of L1 and L2 Support Vector Machines". In: *Proceedings of the International Joint Conference on Neural Networks*. 3. IEEE Press, 2003, pp. 2054–2059. ISBN: 0-7803-7898-9. DOI: 10.1109/IJCNN.2003.1223724.

[11] Gaëlle Loosli and Stéphane Canu. "Comments on the "Core Vector Machines: Fast SVM Training on Very Large Data Sets"". In: *Journal of Machine Learning Research* 8 (2007), pp. 291–301. ISSN: 1533-7928.

[12] Bernhard Pfahringer. "Winning the KDD99 classification cup: bagged boosting". In: *ACM SIGKDD Explorations Newsletter* 1.2 (2000), pp. 65–66. ISSN: 1931-0145. DOI: 10.1145/846183.846200.

[13] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004, pp. 3–104. ISBN: 0-521-81397-2.

[14] Alex J. Smola and Bernhard Schökopf. "Sparse Greedy Matrix Approximation for Machine Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 911–918. ISBN: 1-55860-707-2.

[15] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 2006, pp. 256–276. ISBN: 0-321-32136-7.

[16] Ivor W. Tsang, Andras Kocsor, and James T. Kwok. "Simpler core vector machines with enclosing balls". In: *ICML '07: Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 911–918. ISBN: 978-1-59593-793-3. DOI: 10.1145/1273496.1273611.

[17] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. "Core Vector Machines: Fast SVM Training on Very Large Data Sets". In: *Journal of Machine Learning Research* 6 (Dec. 2005), pp. 363–392. ISSN: 1533-7928.

[18] Vladimir N. Vapnik. *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995. ISBN: 0-387-94559-8.