Table des matières

Introduction **6**

1 Ce qu'il faut savoir sur l'apprentissage artificiel **7**
1.1 Deux grandes familles : supervisé et non-supervisé 7
1.2 L'évaluation de l'apprentissage de fonctions d'un algorithme d'apprentissage supervisé 8
1.3 Hypothèse de non-hostitilité 9
1.4 L'apprentissage artificiel en ligne et le réapprentissage 10

2 Les Machines à Vecteurs Supports (SVMs) **12**
2.1 Principes de fonctionnement 12
2.2 Les SVMs à marges molles 13
2.3 Les SVMs incrémentaux 14

3 Formalisation des problèmes de sécurité des processus d'apprentissage artificiel **16**
3.1 Terminologie 16
3.2 Analyse de la sécurité 17
3.3 Une classification des attaques 17
3.4 Quelques exemples d'application 18
3.4.1 L'attaque par dictionnaire : un exemple d'attaque de type *Causative* et *Availability* 19
3.4.2 Discussion 20

4 Poisoning SVM : L'article à l'origine de cette étude **21**
4.1 Présentation de l'article 21
4.2 Le contexte de l'article 22
4.3 Les stratégies d'attaque mises en place dans cet article 22
4.4 Notion de carte d'influence 24
4.5 Quelques critiques sur l'approche proposée par *Biggio et al.* 24

5 Les approfondissements étudiés 26
5.1 Conditions expérimentales 26
5.2 La sélection d'un bon candidat comme point initial d'attaque 31
5.3 Le passage à l'échelle 33
5.4 La sélection de la classe à attaquer 36
5.4.1 dans le cas de classes équilibrées 36
5.4.2 La sélection de la classe à attaquer dans le cas déséquilibré 38
5.5 Comparaison de stratégies d'attaques 40
6 Quelques discussions ouvertes à la vue de ce qui a été étudié 42
6.1 La fonction à maximiser 42
6.2 La fonction considérée mise en relation avec le passage en grandes dimensions 42
6.3 Le passage aux cas multi-classes 43
6.4 L'utilisation d'ensemble d'algorithmes de classification 43
Conclusion 45

Introduction

L'apprentissage artificiel peut être vu comme une branche de l'intelligence artificielle. L'objectif est de construire et d'étudier un système qui peut inférer des informations à partir de données dans le but final de prendre des décisions de manière automatique sur ce type de données.

Ces décisions peuvent prendre la forme de labels (ou étiquettes) associés aux données.

Il existe deux notions importantes autour des principes de l'apprentissage artificiel : la représentation des données et la capacité à généraliser le processus de prise de décision à de nouvelles données.

De manière générale, les algorithmes d'apprentissage artificiel sont capables de prendre des décisions sur la nature des données grâce à des fonctions mathématiques. Ces fonctions mathématiques sont appelées fonctions hypothèses ou fonctions séparatrices.

Aujourd'hui, les algorithmes d'apprentissage artificiel sont utilisés dans différents domaines pour des applications très variées. Depuis quelques temps, ces algorithmes sont utilisés dans des systèmes de prise de décision comme les filtres spams, la détection de virus ou encore dans des systèmes de détection d'intrusion dans des réseaux. Ils y sont très appréciés puisqu'ils peuvent être entraînés pour détecter des comportements anormaux.

Ces comportements anormaux proviennent de deux types de données majoritairement : des données dont la nature n'avait pas encore été découverte jusqu'alors et les données provenant de comportements d'utilisateurs malveillants. Ce dernier cas est ce que nous appelons des attaques contre un système.

Nous proposons, dans ce rapport, d'essayer de comprendre l'interaction qu'il existe entre des attaques et les fonctions séparatrices. Pour cela, nous nous sommes basés sur un article présenté en section 3 qui se concentre sur l'étude de telles interactions pour des fonctions séparatrices à vastes marges (SVMs).

La première partie de ce document rappelle des notions générales sur les algorithmes d'apprentissage artificiel et sur le contexte dans lequel ils sont utilisés. La seconde partie se focalise sur les SVMs. La troisième partie met en place le contexte dans lequel se situe notre étude. La partie suivante présente l'article sur lequel s'est basée notre étude et en discute certaines limitations. Ensuite, nous détaillerons quelques points qu'il nous a semblé important d'étudier faisant parti des limitations de l'article et enfin, une dernière partie abordera quelques ouvertures de ce travail auxquelles nous avons pensé sans avoir eu le temps de les étudier en profondeur.

Chapitre 1

Ce qu'il faut savoir sur l'apprentissage artificiel

Dans cette première partie, nous allons énoncer quelques généralités sur les algorithmes d'apprentissage artificiel. Puis nous évoquerons un point important dans la dernière section qui constitue la base des raisonnements de ce rapport.

Les algorithmes d'apprentissage artificiel se basent sur un apprentissage statistique dans le but de définir des fonctions séparatrices qui séparent au mieux les données que l'algorithme va rencontrer.

1.1 Deux grandes familles : supervisé et non-supervisé

Les algorithmes d'apprentissage artificiel peuvent être divisés en deux grandes familles d'algorithmes. La première est composée des algorithmes supervisés alors que la seconde contient les algorithmes non-supervisés. Pourtant, ces deux familles suivent le même procédé à 2 phases :

- Pendant la première phase, appelée la phase d'apprentissage, les utilisateurs donnent des données en entrée de l'algorithme.
  Un algorithme d'apprentissage artificiel va entrainer les paramètres de ses fonctions séparatrices sur ces données dans le but de séparer ces données au mieux. Dans le cas où l'algorithme est un algorithme supervisé, des labels (ou étiquettes) sont pré-assignés aux données et sont ainsi utilisés comme une base pour séparer les données d'entrée.
  Dans le cas non-supervisé, cette phase d'apprentissage peut être itérative. Les fonctions séparatrices sont alors appliquées à plusieurs jeux de données pour permettre à l'algorithme d'ajuster ses paramètres au mieux.
- Une fois cette première phase terminée, il devient indispensable de savoir comment les fonctions mathématiques, qui viennent d'être apprises, se comportent avec des données que l'algorithme n'a encore jamais vu. Dit autrement, il faut déterminer si l'algorithme peut généraliser à de nouvelles données ce qu'il vient d'apprendre à partir des données d'apprentissage. On donne habituellement un nouveau jeu de données en entrée de l'algorithme. Ce jeu est appelé ensemble de test. L'algorithme va alors calculer la classe la plus vraisemblable à attribuer à chaque donnée. L'éva-

luation des résultats sur les données de tests permettent de caractériser les capacités de généralisation de l'algorithme d'apprentissage.

1.2 L'évaluation de l'apprentissage de fonctions d'un algorithme d'apprentissage supervisé

Le but final du processus précédent est donc de générer des fonctions séparatrices permettant à l'algorithme de prendre des décisions sur la nature des données en faisant le moins d'erreurs possibles. Ces erreurs se définissent relativement à une vérité terrain qui est le résultat attendu pour une donnée. Ce résultat attendu est défini par un expert. Il existe deux types d'erreurs : les faux positifs et les faux négatifs.
Dans le cas de la classification, ces erreurs sont définies par rapport à une classe donnée C. Ainsi :

- Les **faux négatifs** sont des données auxquelles la vérité terrain attribue le label C alors que l'algorithme d'apprentissage artificiel leur attribue le label d'une autre classe.
- Les **faux positifs** sont des données dont la vérité terrain leur attribue le label d'une autre classe alors que l'algorithme leur attribue le label C.

La matrice de confusion suivante résume toutes ces informations. Les colonnes représentent la classe attribuée par l'algorithme d'apprentissage artificiel pour une donnée précise. Les lignes représentent l'étiquette de cette donnée suivant la vérité terrain. Si les deux étiquettes sont en adéquations, il n'y a pas d'erreur et l'on se retrouve sur la diagonale principale du tableau. Si les deux labels sont différents, on se retrouve alors dans le cas d'une erreur qui se positionne sur l'autre diagonale, dont le contenu des cellules est écrit en rouge et italique.

Label calculé par l'algorithme

Etiquette de la vérité terrain		C	$\bar{C}$
	C	Vrai Positif	*Faux Négatif*
	$\bar{C}$	*Faux Positif*	Vrai Négatif

FIG. 1.1 – Une matrice de confusion présentant les erreurs de classification pouvant être rencontrées relativement à une classe

Ces erreurs ne sont pas forcément très gênantes à condition de bien définir, pour une application donnée, le type d'erreurs à éviter et le taux d'erreur acceptable.
Par exemple, on peut imaginer que dans le cas d'une application de diagnostic médical, il est important d'avoir un système où il n'y ait aucun cas d'erreur de type faux négatifs. En effet, un faux négatif dans ce type d'application se traduirait par le fait de ne pas détecter une maladie présente chez un patient. On comprend alors qu'il est peut-être mieux d'avoir beaucoup de faux positifs (le système détecte une maladie alors qu'il n'y en a pas) qui nécessiteront un examen de contrôle afin d'établir un traitement si besoin plutôt que le contraire.

1.3 Hypothèse de non-hostitilité

Habituellement, les algorithmes d'apprentissage artificiel entrainent leurs fonctions séparatrices sur des données qui sont supposées correctes, représentatives du monde des données à étudier et sûres. A priori, tout est fait, de la phase d'apprentissage à la généralisation qui en découle, pour que l'algorithme puisse retourner les meilleurs résultats possibles.

De plus, pour le cas d'apprentissage supervisé, les labels fournis avec les données d'apprentissage doivent être représentatifs des données « réelles » qu'un utilisateur peut s'attendre à traiter par la suite.
Bien sûr, tout le monde est supposé bien intentionné. Autrement dit, personne n'a de comportements qui essayeraient de faire augmenter de manière intentionnelle le nombre d'erreurs faites par l'algorithme.

Depuis à peu près 20 ans maintenant, cette hypothèse de non-hostilité a été mise de côté et a ainsi permis la mise en place d'un nouveau domaine d'application dans l'apprentissage artificiel qui paraît prometteur puisqu'il est plus proche des conditions réelles d'utilisation. Ainsi, il est supposé que des personnes mal-intentionnées ont des connaissances à priori sur les algorithmes de prises de décision, sur leurs fonctions séparatrices ainsi que leurs paramètres et également sur les données d'apprentissage utilisées.
À partir de ces connaissances, ces personnes essayent de faire augmenter le nombre d'erreurs de décision faites par l'algorithme. Dans un contexte où l'algorithme tente de décider si des messages électroniques sont des spams ou non, si des personnes malveillantes arrivent à bien utiliser leurs connaissances, alors des messages spams qu'ils auraient envoyés devraient se faire détecter comme des messages normaux.

Les personnes mal-intentionnées peuvent agir sur l'algorithme d'apprentissage artificiel de deux manières différentes :

- lors de la phase de test de l'algorithme, ces personnes ont mis au point des données qui seront mal classées puisque les fonctions séparatrices ont été ajustées aux données rencontrées lors de la phase d'apprentissage. Ainsi, la fonction séparatrice ne bouge pas et les classifications faites précédemment sur des données « normales » restent correctes. Par contre, des données corrompues ressemblant aux données mal classées ont des chances d'être également mal classées.
- lors de la phase de test, l'algorithme peut prendre en compte les nouvelles données entrantes et ainsi réapprendre une fonction séparatrice et ses paramètres pour essayer de faire moins d'erreurs. La définition des nouvelles valeurs de paramètres, guidée par les personnes mal-intentionnées, ne garantissent plus que l'algorithme va continuer de classer les données rencontrées précédemment de la même manière.

Il faut remarquer alors que les données générées par des personnes mal-intentionnées ont un label qui fait que les fonctions séparatrices apprises lors de la phase d'apprentissage doivent se distordre pour éviter de faire des erreurs. On peut ainsi dire que la distribution de ces données est très différente de celles des données d'apprentissage.

[1, 3, 10, 4, 7, 12, 13, 6, 11] ont montré que, dans de telles conditions, les algorithmes d'apprentissage artificiel ne se débrouillent plus aussi bien qu'avec les hypothèses initiales. Ils font effectivement plus d'erreurs lorsque les utilisateurs adoptent un comportement hostile. Cela peut donc mener à des problèmes de sécurité.

1.4 L'apprentissage artificiel en ligne et le réapprentissage

Ainsi, il existe plusieurs facteurs qui peuvent faire que le système de prise de décision voit son taux d'erreurs augmenter.
Tout d'abord, comme dit précédemment, parce que l'environnement dans lequel sont utilisés de tels systèmes est devenu hostile. Mais également, du fait que la nature des données change. Il suffit d'imaginer un système qui apprend les paramètres de ses fonctions séparatrices sur un jeu de données puis qui serait utilisé dans un contexte différent (par exemple, un système classant des images prises en extérieur suivant les couleurs présentes, typiquement par un histogramme de couleur, et à un moment donné, dans son utilisation, on lui demande de classer une image correspondant aux photos utilisées pour les cartes d'identité).
L'hypothèse spécifiant que les données d'apprentissage doivent être représentatives des données à traiter n'est donc plus vérifiée.

Une communauté s'est penchée sur le fait de faire prendre en compte à l'algorithme les changements de nature des données afin d'en limiter l'impact sur ses performances. Ainsi, il a été pensé qu'il fallait ajouter la connaissance apportée par ces nouvelles informations aux fonctions séparatrices déjà établies pour les parfaire et au final qu'elles se rapprochent plus des fonctions séparatrices souhaitées qui sépareraient parfaitement les données à traiter sans faire aucune erreur.

Cette prise en compte se fait par l'intégration des nouvelles données à l'ensemble d'apprentissage. En recommençant le processus à 2 phases détaillé précédemment permettant de fixer la valeur des paramètres des fonctions séparatrices, de nouvelles fonctions séparatrices sont créées en prenant en compte les données précédentes mais également les nouvelles que l'algorithme ne connaissait pas. Ce « réapprentissage » se fait de manière périodique, et l'application détermine la durée des périodes.
Si l'on applique ce principe de manière « naïve » comme décrit ci-dessus, il est facile d'imaginer que le fait d'agglomérer de plus en plus de données dans l'ensemble d'apprentissage augmente le temps passé à établir des fonctions séparatrices et à ajuster leurs paramètres lors de la phase d'apprentissage. Ceci n'est pas satisfaisant lorsque l'algorithme se situe dans un cadre où il doit être réactif. Par exemple, le cas où un nouveau virus est découvert. Il n'est pas souhaitable qu'un logiciel de protection contre les virus prenne plusieurs jours pour se mettre à jour et ainsi commencer à détecter de nouveaux logiciels malveillants. De plus, ajouter toutes les données rencontrées dans l'ensemble d'apprentissage, compte tenu de ce qui a été dit sur l'environnement hostile dans lequel sont utilisés ces algorithmes, ne paraît pas satisfaisant. Il existe donc deux leviers sur lesquels jouer afin d'améliorer ou, dans le pire des cas, de conserver les performances de l'algorithme. Ce sont des stratégies de mise à jour concernant d'une part l'utilisation

de la fonction séparatrice précédente associée aux nouvelles données et d'autre part un ajout « intelligent » de ces nouvelles données pour éviter de faire que des personnes malveillantes puissent créer facilement des données qui génèreront des erreurs, permettant à ces personnes d'atteindre leurs buts.

Au niveau de l'ajout de nouvelles données dans l'ensemble d'apprentissage de manière « intelligente », il existe des manières différentes de voir le problème. Il existe, par exemple, la prise en compte des données les plus « utiles ». Cela nécessite un calcul entropique entre les différentes données. L'idée étant d'ajouter d'abord un maximum de variété dans les données puis d'ajouter dans un second temps celles qui en apportent moins. Une autre technique est présentée à la fin de la troisième partie de ce document. Elle est fondée sur le fait d'ajouter une à une les données dans l'ensemble d'apprentissage et de mesurer l'impact de cet ajout sur les performances de l'algorithme.

Malgré cela, il est difficile de garantir un bon fonctionnement des algorithmes de prises de décision. C'est pourquoi il a été tenté de comprendre comment des personnes malveillantes pouvaient influencer ces algorithmes.

Chapitre 2

Les Machines à Vecteurs Supports (SVMs)

Nous venons d'énoncer le principe général des algorithmes d'apprentissage artificiel. Il existe différentes implémentations de ces algorithmes. Notre étude s'est portée sur une famille de ces implémentations que nous allons présenter dans cette partie : les machines à vecteurs supports ou SVMs. Dans cette partie du document, les données à classer seront divisées en deux classes.

2.1 Principes de fonctionnement

Les SVMs construisent des fonctions séparatrices pour séparer des données. Pour construire ces fonctions, il faut utiliser le procédé à deux phases décrit en début de document. Les SVMs font partie des algorithmes supervisés et également de la famille des classifieurs linéaires. Ainsi, les fonctions séparatrices construites sont des fonctions séparatrices linéaires, c'est pourquoi nous appellerons les fonctions construites par les SVMs des hyperplans séparateurs. Ces hyperplans sont de la forme suivante : $h(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x} + w_0$ où $\mathbf{w}$ est un vecteur de poids, $\mathbf{x}$ est un vecteur représentant les données et w_0 est une constante représentant le biais.

Dans le cas où les données sont linéairement séparables, il existe une infinité d'hyperplans séparateurs qui peuvent être construits pour séparer ces données. Pour déterminer un hyperplan unique, parmi tous les hyperplans possibles séparant correctement les données, les SVMs utilisent la notion de marges.
Les marges sont les distances qui existent entre l'hyperplan séparateur et les données de l'ensemble d'apprentissage les plus proches de ce dernier. Ces données de l'ensemble d'apprentissage sont alors appelées vecteurs supports.
L'équation de maximisation des marges s'écrit :

$$\arg\max_{w,w0} \min_{k} \{ \ ||\mathbf{x} - \mathbf{x}_k|| : x \in R^n, \mathbf{w}^\intercal \mathbf{x} + w_0 \ = 0\}$$

Pour trouver cet hyperplan optimal, une technique d'optimisation par les multiplicateurs de Lagrange peut être utilisée. L'équation de l'hyperplan s'écrit alors :

$$h(x) = \sum_{p=1}^{P} \alpha_p y_p (\mathbf{x}.\mathbf{x}_p) + w_0$$

où P est le nombre de données dans l'ensemble d'apprentissage, α_p sont les multiplicateurs de Lagrange ayant la propriété $\alpha_p \geq 0$ et en particulier $\alpha_p > 0$ pour les vecteurs supports, y_p est la classe de la donnée représentée par un vecteur $\mathbf{x}_p$ et (.) est le produit scalaire entre deux données.
On peut donc voir que l'équation de l'hyperplan ne dépend que des vecteurs supports et en particulier du produit scalaire entre ceux-ci.

Pour l'instant, nous nous sommes placés dans le cas où les données sont linéairement séparables. Ce n'est pas tout le temps le cas. Pour pallier cela, les SVMs réexpriment le problème de classification dans un espace de plus grandes dimensions. L'intérêt étant d'augmenter les chances de trouver une solution dans ce nouvel espace. Il peut être de dimension infinie.
Pour cela, une transformation non-linéaire ϕ est appliquée aux vecteurs de données $\mathbf{x}$. On appelle alors $\phi(\mathbf{x})$ l'espace de redescription. L'équation de l'hyperplan s'écrit alors :

$$h(x) = \mathbf{w}^\intercal \phi(\mathbf{x}) + w_0$$

et l'équation de l'hyperplan dans l'espace de redescription s'écrit donc :

$$h(x) = \sum_{n=1}^{N} \alpha_n y_n (\phi(\mathbf{x}).\phi(\mathbf{x}_n)) + w_0$$

Le produit scalaire se fait donc entre deux vecteurs qui peuvent être de très grandes dimensions, demandant alors un temps de calcul plus long.
Pour éviter cela, les SVMs utilisent une fonction noyau $K(\mathbf{x}_i, \mathbf{x}_j)$ qui correspond à : $\phi(\mathbf{x}_i)$. $\phi(\mathbf{x}_j)$. Cette fonction noyau a un double intérêt, les calculs restent dans l'espace de description initiale des données et il n'est pas obligatoire de connaître explicitement la fonction ϕ. L'équation de l'hyperplan s'écrit maintenant :

$$h(x) = \sum_{n=1}^{N} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + w_0$$

2.2 Les SVMs à marges molles

Eventuellement, il peut être impossible de trouver un hyperplan séparateur. Par exemple à cause de mauvais étiquettage ou lorsque les données ne sont pas séparables. Quand cela se produit, l'algorithme tel qu'il a été décrit ci-dessus peut ne pas réussir à converger vers une solution.
Il faut donc que l'algorithme des SVMs soit plus tolérant aux erreurs de classification. Il existe une version de l'algorithme des SVMs qui permet cela, ce type de SVMs est appelé « SVMs à marges molles » contrastant avec les SVMs présentés dans la section précédente qui sont appelées « SVMs à marges dures ».

La prise en compte d'erreurs se fait par l'introduction de variables dites « ressort » notées ξ. Elles correspondent à la distance entre les exemples $\mathbf{x}_n$ mal classés et les marges. Pour les autres données, ξ_n vaut zéro. Ces variables ressort viennent modifier l'inéquation

permettant de donner un label à une donnée qui n'a pas encore été vue. Cette dernière s'écrit alors :

$$y_p(\mathbf{w}^\intercal \mathbf{x}_p + w_0) \geq 1 - \xi_p$$

avec la contrainte que $\xi_p \geq 0$. On a alors également le problème d'optimisation qui est modifié où il faut définir un compromis entre l'établissement des marges tel qu'énoncé précédemment et le nombre d'erreurs faites. Pour cela, l'utilisateur définit une constante, appelée C, qui va venir pénaliser la permission de faire des erreurs de classification. Il n'existe aujourd'hui aucune technique qui permette de définir de manière automatique la valeur que doit prendre cette constante C. L'utilisateur doit alors essayer d'approcher cette valeur par une recherche expérimentale sur un intervalle qu'il aura défini avec un pas d'itération donné.

Du fait que des erreurs soient autorisées, il est possible que le positionnement de la séparatrice et des marges ne soient pas les mêmes qu'avec les marges dures. En particulier, alors qu'avec les marges dures, les vecteurs supports se trouvent **sur** les marges, dans le cas des marges molles ce n'est pas forcément le cas, ce qui aura un impact sur l'algorithme présenté dans [4].

2.3 Les SVMs incrémentaux

Il est important de rappeler que, dans ce document, nous avons fait l'hypothèse que l'algorithme d'apprentissage artificiel doit réentrainer sa fonction séparatrice de manière périodique. **?** dans [**?**] introduisent les SVMs incrémentaux qui permettent l'ajout de points dans l'ensemble d'apprentissage et le réapprentissage de la fonction séparatrice sans pour autant tout recommencer depuis le début. En effet, on peut remarquer qu'il existe, de manière implicite trois zones lors de l'utilisation d'un algorithme de SVMs. Ces zones implicites émergent vraiment avec l'utilisation des SVMs à marges molles. En effet, nous avons dit que des points de l'espace de description peuvent se retrouver entre les marges et la fonction séparatrice. D'autres peuvent se retrouver sur les marges. Et une dernière catégorie sont les points en dehors de ces catégories, c'est-à-dire, qui sont positionnés au-delà des marges.
D'après **?** dans [**?**] qui sera repris par *Biggio et al.* dans [4], ces catégories ont des influences différentes sur les fonctions séparatrices et leurs marges :

- Tout d'abord, considérons les points se situant entre les marges et la séparatrice. Ceux-ci sont appelés vecteurs « erreurs » dans [**?**] et [4]. Il s'agit en effet des points qui sont mal positionnés par rapport à l'hyperplan séparateur, se retrouvant de l'autre côté et étant ainsi mal-classés. Mais ce sous-ensemble de l'ensemble d'apprentissage comprend également les points qui sont du bon côté de l'hyperplan mais qui se retrouvent dans une zone délicate, puisque ce sont les points les plus proches de l'hyperplan et donc les plus susceptibles de passer de l'autre côté.

- Ensuite les points se situant sur les marges. Comme indiqué précédemment, ils sont appelés vecteurs « supports » puisque c'est par rapport à eux que l'hyperplan séparateur est défini.

- Enfin, les points qui sont positionnés au-delà des marges. Ils sont appelés vecteurs « réserves ». Ils ne risquent pas de changer de classe puisqu'étant les plus éloignés de la fonction séparatrice et ils n'y contribuent pas non plus.

La figure 2.1 présente un schéma pour mieux visualiser ces trois sous-ensembles.

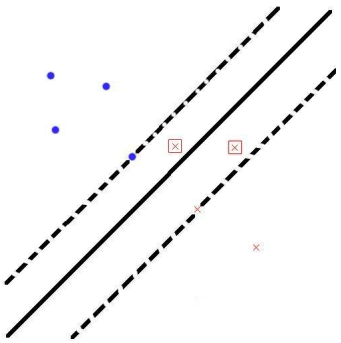

FIG. 2.1 – Une schématisation des trois sous-ensembles définis dans les SVMs incrémentaux. La ligne en trait continu représente l'hyperplan séparateur d'une SVM, les points et croix sont les données dans l'ensemble d'apprentissage. Les lignes discontinues sont les marges. Les points sur les marges sont les vecteurs supports. Les deux croix encadrées montrent les deux cas où une donnée peut être considérée comme vecteur erreur. Le reste sont les vecteurs réserves.

On peut donc remarquer que seules les deux premières catégories sont nécessaires pour établir la fonction séparatrice.
Ainsi, lors de l'ajout d'un point dans l'ensemble d'apprentissage, il faut connaître le sous-ensemble auquel il appartient. S'il fait partie des vecteurs réserves, alors il n'y a rien à recalculer puisque ce point n'a aucune influence sur la fonction séparatrice.
S'il fait partie des vecteurs supports, alors il faut le prendre en compte dans la position de la séparatrice, c'est-à-dire la recalculer, ainsi que les marges. On peut alors voir des points changer de sous-ensemble et il faut alors prendre en compte tous ces changements. Les équations de calculs de l'hyperplan séparateur peuvent alors rentrer dans un processus itératif.
Dans le dernier cas, où le point appartient au sous-ensemble des vecteurs erreurs, il faut là aussi le prendre en compte, pour les marges premièrement, ce qui pourrait faire apparaître de nouveaux vecteurs supports et ce qui amènera dans un second temps à redéfinir la fonction séparatrice. Là encore, l'algorithme peut rentrer dans un processus itératif.

Chapitre 3

Formalisation des problèmes de sécurité des processus d'apprentissage artificiel

Nous allons dans ce chapitre, tout d'abord, présenter la terminologie que nous utiliserons. Puis une section sera dédiée à une analyse de la sécurité permettant d'établir les possibilités en terme de buts et de capacités des personnes mal-intentionnées. Ensuite, une classification des types d'attaque sera développée pour terminer sur le déroulement de quelques scénarios d'attaques et de défenses qui seront mis en relation avec cette classification.

3.1 Terminologie

Tout d'abord, un **attaquant**, une **personne mal-intentionnée** ou un **utilisateur mal-intentionné** sera quelqu'un qui essaye de faire augmenter de manière intentionnelle le nombre d'erreurs fait par l'algorithme d'apprentissage.

Ainsi, des **données** dites **normales** seront supposées comme n'ayant pas été générées par des attaquants. De la même manière, des données **corrompues, mal-intentionnées, suspicieuses, anormales** ou tout simplement des **attaques** sont des données générées par des utilisateurs mal-intentionnés. Même si [13] fait la différence entre des données corrompues et des données anormales, pour notre étude, nous pouvons faire l'amalgame. En effet, [13] précise que, souvent, l'hypothèse est faite que les données suspicieuses sont différentes des données qui devraient être rencontrées dans l'espace étudié et que, de ce fait, elles se trouvent dans la catégorie des données anormales. On peut noter une différence entre ces données anormales et des données suspicieuses : les données suspicieuses sont générées pour atteindre un certain but. Ici, nous supposerons que la nature des données est très bien maîtrisée et qu'ainsi, les données étudiées lors de la phase d'apprentissage sont très représentatives du monde des données à étudier. Ainsi aucun changement ne peut intervenir dans la distribution de ces données.

Un **espace de description** est un espace abstrait. Cet espace est de dimension N où le nombre de dimensions est déterminé par le nombre de caractéristiques décrivant les données. Dans cet espace, les données sont représentées par des points. Une distance de similarité peut y être définie (souvent c'est la distance Euclidienne qui est utilisée). Elle

permet de décider si les points sont proches ou non entre eux.

Une **frontière de décision** est un **hyperplan séparateur** ou une **fonction séparatrice** qui définit les frontières entre les classes des données dans l'espace de description.

On appelle **feedback** ou **retour** d'un algorithme d'apprentissage artificiel, les informations données à l'utilisateur sur la décision d'appartenance à une classe prise par l'algorithme sur des données.

3.2 Analyse de la sécurité

Il s'agit dans cette section de définir quels sont les objectifs et les possibilités des personnes mal-intentionnées, comme l'ont fait *Barreno et al., Barreno et al., Huang et al.* et *Biggio et al.* dans [1, 3, 10] et [4].

D'un point de vue général, les attaquants essayent d'augmenter le nombre d'erreurs de décision prises par l'algorithme d'apprentissage artificiel. Comme cela a été dit auparavant, ces erreurs peuvent être de deux types : les faux positifs et les faux négatifs.

De manière assez réaliste, on peut supposer que les attaquants ont des connaissances sur l'algorithme d'apprentissage artificiel utilisé par le système ciblé. Par exemple, le noyau utilisé. Il est aussi réaliste de supposer qu'ils ont également des connaissances sur les données d'apprentissage. Ces connaissances ne sont pas nécessairement totales, il peut s'agir uniquement de la distribution des données utilisées. Nous supposons également que les personnes mal-intentionnées ont un certain contrôle sur les données d'apprentissage. Ce contrôle se justifie par le fait que ces personnes peuvent générer et envoyer des données au système qui peuvent ensuite être prises en compte en étant incorporées à l'ensemble d'apprentissage. [1]

3.3 Une classification des attaques

D'après [1] et [10], il existe 3 axes qui définissent les possibilités permettant de générer des attaques contre un système :

- **Influence**, divisé en *Causative* et *Exploratory*, cela définit l'influence que peut avoir une personne malveillante sur l'ensemble d'apprentissage. Les attaques *Causative* vont altérer le processus d'apprentissage en modifiant l'ensemble d'apprentissage. Les attaques *Exploratory* essayent seulement de récupérer des informations sur l'ensemble d'apprentissage sans pour autant avoir d'influence dessus. Une attaque *Exploratory* peut naturellement précéder une attaque de type *Causative*.
- **Specificity**, divisé en *Targeted* et *Indiscriminate*, définit la concentration des attaques sur quelques points dans l'espace de représentation. Les attaques *Targeted* se concentrent sur quelques points spécifiques dans cet espace tandis que les attaques *Indiscriminate* ont un but plus large.

[1] On se place dans un cas où l'algorithme d'apprentissage réajuste son hyperplan séparateur de manière périodique, tel qu'expliqué dans la section 1.4

<table>
<tr><th rowspan="2">Influence</th><th rowspan="2">Specificity</th><th colspan="3">Security violation</th></tr>
<tr><td>Integrity</td><td>Availability</td><td>Privacy</td></tr>
<tr><td rowspan="2">Causative</td><td>Targeted</td><td>Permet une attaque ciblée</td><td>Crée suffisamment d'erreurs pour faire que le système de prise de décision soit inutilisable pour un utilisateur ou un service</td><td rowspan="3">Récupère des informations sur l'algorithme d'apprentissage artificiel et sa fonction séparatrice</td></tr>
<tr><td>Indiscriminate</td><td>Permet au moins une intrusion dans le système</td><td>Crée suffisamment d'erreurs pour que l'algorithme d'apprentissage artificiel ne soit plus utilisable</td></tr>
<tr><td rowspan="2">Exploratory</td><td>Targeted</td><td>Trouve une intrusion dans un sous-ensemble restreint de possibilités</td><td rowspan="2">Trouve un ensemble de points qui sera mal classé par l'algorithme d'apprentissage</td></tr>
<tr><td>Indiscriminate</td><td>Trouve une intrusion</td></tr>
</table>

TAB. 3.1 – Résumé des 3 axes d'attaques

- **Security violation**, divisé en *Integrity*, *Availability* et *Privacy* définit la stratégie d'attaque. Les attaques *Integrity* se concentrent sur un type d'erreur à faire augmenter (typiquement des faux négatifs). Les attaques *Availability* augmentent de manière significative le nombre d'erreurs, quel que soit leur type, rendant inutilisable le système de prise de décision. Les attaques *Privacy* tentent d'inférer des informations sur la fonction séparatrice ou la fonction permettant de passer de l'espace de description des données à l'espace de redescription dans lequel travaille l'algorithme de prise de décision.

La table 3.1 résume cette classification des attaques selon les trois axes.[2]

3.4 Quelques exemples d'application

Un exemple va maintenant être présenté afin de comprendre comment il s'inscrit dans la taxonomie décrite ci-dessus.
Pour cet exemple, le contexte applicatif, le but de l'attaquant, une implémentation d'at-

[2]Elle peut également être trouvée en partie dans [1].

taque et un moyen de se défendre sont précisés.

Pour ces exemples, nous nous positionnons dans le cadre de la détection de spams. Un algorithme de prise de décision décide alors si un message est un spam ou non en attribuant un poids à chaque mot apparaissant dans un message. Pour un mot précis, son poids correspond à la relation : *mot* → *spam* (ce mot apparaît dans un spam). Ainsi, si ce mot est souvent présent dans un ensemble de spams avérés, utilisés comme base de l'apprentissage, le poids qui lui est associé sera élevé. Ainsi, un nouveau message qui doit être classifié par le système et qui contient ce mot aura de grandes chances d'être considéré comme spam.

3.4.1 L'attaque par dictionnaire : un exemple d'attaque de type *Causative* et *Availability*

Ici, l'attaquant essaye de réaliser des intrusions dans un système. Une intrusion signifie qu'une attaque, ici un spam, ne sera pas reconnue comme tel par l'algorithme d'apprentissage artificiel.
Il est supposé que l'attaquant peut contrôler une partie de l'ensemble d'apprentissage qui est donné à l'algorithme. L'attaque consiste à copier le vocabulaire qui pourrait être utilisé par un utilisateur normal.

Si l'attaquant connaît parfaitement le vocabulaire utilisé par un utilisateur normal, alors l'attaque est de type *Targeted* (elle cible un utilisateur particulier). Dans le cas contraire, le vocabulaire est plus large et l'attaque est de type *Indiscriminated* (l'attaquant essaye de ne pas se faire détecter).

Un scénario typique pour ce genre d'attaque est le cas où un commercial malveillant voudrait contrer ses concurrents lors de prospections en faisant que leurs e-mails soient détectés comme spams mais pas les siens. Dans ce cas, le commercial a des connaissances sur le type de vocabulaire qu'utilise ses concurrents. En prenant cela en compte, il peut alors créer des messages électroniques qui vont utiliser ce vocabulaire et les noyer dans du contenu typique des spams. Un premier but est de faire que l'algorithme d'apprentissage artificiel classe systématiquement ces messages comme étant du spam. Ainsi, par le mécanisme de poids expliqué précédemment, le vocabulaire des concurrents dans sa globalité sera mis en relation avec l'étiquette des spams et au final, plus aucun message de concurrents ne pourra arriver dans la boîte mail de potentiels clients.
Un second but est de réaliser des intrusions dans le système.
Finalement, le but global est d'arriver à inverser les labels pour que les messages des concurrents soient considérés comme spams et les messages du commercial malveillant ne le soient pas.

Une technique pour contrer cette attaque est d'essayer de mesurer l'impact de chaque mot dans la phase d'apprentissage. Pour cela, l'algorithme se réentraine de manière itérative, en prenant en compte un mot à la fois parmi tous ceux rencontrés depuis le début de la mise en place du système de prise de décision. Un nouvel ensemble d'apprentissage est ainsi créé et une fonction séparatrice calculée. Cette dernière est ensuite testée sur un ensemble de test et ses performances sont comparées avec la fonction séparatrice sans

ce mot ajouté. Si les performances sont meilleures ou égales, alors le mot est conservé dans l'ensemble d'apprentissage sinon il est rejeté. Est alors ajouté à ce nouvel ensemble d'apprentissage le mot suivant et le processus de sélection est ensuite répété.
Au final, cette procédure cherche à faire la différence entre le vocabulaire normal et le vocabulaire relatif à une attaque.

3.4.2 Discussion

Comme le montrent [1, 3, 10, 13, 6, 11, 5], le fait de remettre en cause l'hypothèse de non-hostilité lors de l'utilisation d'algorithmes d'apprentissage artificiel est encore quelque chose d'assez nouveau et qui soulève donc de nouveaux problèmes. Par exemple, [13, 6, 11] proposent de remettre en cause des hypothèses souvent implicites faites lors de l'utilisation de ces algorithmes. Parmi ces hypothèses : le fait que l'on puisse trouver des jeux de données qui ne contiennent pas d'attaques ou d'anomalies pour ainsi être sûr que l'algorithme apprenne une première fonction séparatrice sur des bases saines. Or, du fait, que l'on essaye de comprendre ces anomalies, de plus en plus de corpus de données sont créés en incorporant des données corrompues qui viennent perturber les performances de l'algorithme de classification. De plus, les attaques se renouvellent continuellement, il est donc difficile de trouver des corpus qui soient à jour avec des attaques qui n'ont pas encore été prises en compte par d'autres corpus. Une autre hypothèse est le fait que les attaques soient considérées comme rares comparées aux données normales. Le nombre d'attaques continue d'augmenter et prend toujours une nouvelle forme.

Ainsi, il y a un vrai problème d'entente au niveau des bases théoriques entre les différents auteurs où ils n'arrivent pas à bien définir ce qui doit être pris en compte ou non et comment. Beaucoup d'hypothèses sont discutées ce qui semble mener vers la construction d'une nouvelle base théorique autour de laquelle tout le monde serait d'accord. C'est ce que Kuhn appelle une révolution scientifique [**?**].

Chapitre 4

Poisoning SVM : L'article à l'origine de cette étude

L'équipe TexMex travaille avec des données provenant du multimédia. Elle souhaite appliquer les techniques d'attaques d'algorithme d'apprentissage artificiel dans un contexte différent de celui des messages électroniques ou des surveillance de trafic réseau. La littérature étudiée précédemment lors du Master Recherche en Informatique a montré que ce sont ces domaines qui ont été les plus largement étudiés et qu'il n'existe que très peu de travaux concernant d'autres domaines.

De ce fait, il n'existe pas beaucoup d'articles scientifiques proposant un algorithme d'attaque ou de défense testé sur des données autres que celles évoquées précédemment. *Biggio et al.* dans [4] présentent des résultats sur des images tirées du corpus MNIST. Cet article semble donc être un bon point d'entrée pour l'équipe et c'est pourquoi l'étude menée s'est basée dessus.

La suite de ce chapitre se compose d'abord d'une présentation générale de l'article, puis le contexte de son étude sera abordé. Par la suite, les stratégies mises en place dans l'article seront traitées pour finalement, dans un dernier temps, discuter de quelques limites.

4.1 Présentation de l'article

L'article se place du point de vue des attaquants. Ces attaquants essayent d'augmenter, de manière maximale, le nombre d'erreurs effectué par l'algorithme d'apprentissage artificiel en ajoutant des points avec des coordonnées spécifiques dans l'ensemble d'apprentissage. Cela fait donc référence aux attaques de type *Causative* de la taxonomie d'attaques présentée précédemment.
L'algorithme d'apprentissage artificiel utilisé est l'algorithme des SVMs. Cet algorithme permet de prendre en compte des distributions plus ou moins complexes grâce à l'utilisation de noyaux et ne nécessite, le plus souvent, l'ajustement que de quelques paramètres.

Il est important de noter que cette technique propose d'ajouter des points d'attaques dans l'ensemble d'apprentissage dans l'espace de description. Ainsi, les attaquants n'ont

pas à essayer de découvrir quel est l'espace de redescription qui est utilisé par l'algorithme d'apprentissage artificiel. Le fait d'ajouter des points directement dans l'espace de description est une contribution très forte au domaine puisque les autres méthodes utilisaient l'espace de redescription.

L'article place son attaque dans un problème de classification à deux classes.

4.2 Le contexte de l'article

Dans l'article, certaines hypothèses, sur les connaissances de l'attaquant vis-à-vis de l'algorithme d'apprentissage artificiel, sont faites. En effet, l'article propose que les classes des points d'attaques soient données par les attaquants eux-mêmes alors qu'en réalité cela n'est pas possible. De la même façon, il est supposé que l'attaquant force l'algorithme à prendre en compte le point d'attaque en l'incluant directement dans l'ensemble d'apprentissage. Habituellement, il n'est possible pour personne d'avoir accès directement à cet ensemble. Il est également fait l'hypothèse que l'attaquant connait exactement l'ensemble d'apprentissage initial, lui permettant de préparer très consciencieusement son attaque. Ces hypothèses semblent très irréalistes mais les auteurs les justifient en disant que cela mène à une analyse du pire cas. À la vue des pistes que nous avons exploré (voir chapitre 5), nous avons décidé que nous nous placerions dans un contexte d'apprentissage artificiel pur, quitte à oublier de temps en temps le côté applicatif, mais nous permettant d'avoir une meilleure compréhension du problème qui se pose à nous.

Le but de l'attaque présentée dans cet article est d'ajouter des points dans l'ensemble d'apprentissage qui, une fois pris en compte dans le calcul de la fonction séparatrice et des marges, va distordre la fonction séparatrice par rapport à sa position initiale, entrainant ainsi des erreurs de classification lors de la phase de test de l'algorithme sur de nouvelles données. Cette attaque se place donc dans le second cas de la section 1.3 qui décrit deux types d'influence qu'un utilisateur mal-intentionné peut avoir sur un algorithme d'apprentissage artificiel, à savoir le fait que l'algorithme va tenter de réapprendre une nouvelle séparatrice en prenant en compte les données corrompues.

La section suivante décrit comment s'effectue l'attaque.

4.3 Les stratégies d'attaque mises en place dans cet article

La méthode d'attaque décrite dans [4] propose de trouver un point dans l'espace de description qui permette, après réapprentissage, d'avoir une influence maximale sur les fonctions séparatrices, se traduisant par une augmentation du nombre d'erreurs dans l'ensemble de test.

L'article propose donc une attaque à deux phases :

- tout d'abord, on suppose qu'un hyperplan séparateur a été construit pour un ensemble d'apprentissage donné. A partir de là, l'attaquant définit une classe attaquante et une classe attaquée. Il faut maintenant définir un point d'attaque. Puisque les auteurs ne font pas d'à priori sur la construction de points d'attaques qui ont de bonnes chances d'avoir une forte influence sur la solution, ils décident de partir d'un point existant dans l'ensemble d'apprentissage. Ce point est pris au hasard de la classe attaquée. Le point d'attaque est alors créé. Pour cela, il suffit de copier les coordonnées du point choisi et de changer son label. Deux points se superposent alors dans l'espace de description mais ils ont des labels différents. Il faut, ensuite, le faire prendre en compte par l'algorithme d'apprentissage artificiel. Comme le nombre d'éléments dans l'ensemble d'apprentissage peut être très grand, tout comme le nombre de dimensions de l'espace de descritption et que le noyau utilisé par les SVMs peut être complexe, augmentant ainsi le temps de calcul des nouvelles fonctions séparatrices après avoir inclus ce nouveau point dans l'ensemble d'apprentissage, les auteurs décident d'utiliser les SVMs incrémentaux décrits dans la seconde partie de ce rapport.
- une fois cette donnée corrompue introduite dans l'ensemble d'apprentissage, il faut déterminer où elle doit se positionner pour avoir une forte influence sur les fonctions séparatrices. Cette influence correspond à la fonction de coût donnée par le nombre d'erreur de classification réalisé par l'algorithme. Cette fonction de coût est la « hinge loss ». L'attaque décrite dans [4] utilise une technique d'ascension de gradient sur cette fonction. Une technique à base de gradient fonctionne avec des itérations. Dans cet article, le fait d'avoir des itérations permet d'avoir des déplacements relativement petits permettant de corriger la trajectoire de la donnée corrompue. Cette correction de trajectoire vise à faire se déplacer vers le maximum de la fonction de « hinge loss ». Un pas d'itération est donc choisi par l'attaquant et une direction de déplacement est calculée. Cette direction correspond à la direction où le gradient est le plus fort. Le processus de déplacement (calcul du gradient et direction décidée à partir de la valeur maximale de ce dernier) s'arrête lorsque le maximum est atteint ou lorsque le point se trouve trop proche d'un maximum et que l'itération suivante le fait redescendre.
  Comme la fonction représente l'influence qu'à un point d'attaque sur les fonctions séparatrices, et que les fonctions séparatrices sont définies par les vecteurs supports et les vecteurs erreurs pour un noyau donné, le gradient se compose d'une dérivation des multiplicateurs de Lagrange, d'une dérivation des biais des fonctions séparatrices ainsi que d'une dérivation du noyau entre les vecteurs supports et le point d'attaque tout cela suivant un vecteur de direction. Ce sont bien ces trois éléments qui définissent les fonctions séparatrices. Dériver la fonction noyau entre les vecteurs supports et le point d'attaque et dériver les multiplicateurs de Lagrange en fonction du point d'attaque, montrent bien l'interaction qu'il existe entre ces trois éléments.

  L'étude menée dans [4] a utilisé différents noyaux : le noyau linéaire et le noyau à bases radiales (RBF). L'article donne les dérivations de ces deux noyaux ainsi que du noyau polynomial.

4.4 Notion de carte d'influence

L'article précise que certaines zones de l'espace de description peuvent avoir plus d'influence que d'autres sur les fonctions séparatrices. Ces zones sont observables en créant une carte qui est calculée sur l'espace de description. Cette carte est créée en réalisant une grille régulière sur cet espace. Pour chaque point de cette grille, une simulation d'ajout d'un point de la classe attaquante est faite, de nouvelles séparatrices sont ainsi calculées et finalement c'est le nombre d'erreurs de classification sur l'ensemble de test[1] qui définit si l'influence est forte ou non. Plus le nombre d'erreurs de classification est important dans une zone, plus l'attaquant a envie que la donnée corrompue se positionne dans cette zone. Pour avoir une carte continue, une interpolation est faite entre les points de la grille. L'attaquant doit donc gérer un compromis entre précision de la grille, lui donnant des zones précises de forte influence, et temps de calcul.

4.5 Quelques critiques sur l'approche proposée par *Biggio et al.*

Même si à la vue des résultats, cette technique semble bien fonctionner, quelques critiques peuvent être faites sur la manière dont ont été menées les expériences.

Il n'existe aucune version disponible des SVMs incrémentaux qui soit performante à ce jour. De plus, l'utilisation de ce type de SVMs est justifiée lorsque les utilisateurs souhaitent insérer un ensemble de points dans l'ensemble d'apprentissage. Ici, il n'est question que d'un seul point. On peut donc se contenter d'utiliser une version traditionnelle des SVMs et de reprendre le calcul de l'hyperplan séparateur depuis le début lorsqu'un point doit être ajouté à l'ensemble d'apprentissage.

L'article ne présente que peu d'expériences et résultats associés pour valider l'approche suivie pour les attaques. En effet, l'article présente 2 types d'expériences : une première effectuée sur des données en deux dimensions générées de manière artificielle et une seconde prenant un corpus de données (MNIST) comportant des images en niveau de gris de 28x28 pixels. Chacun de ses pixels représentant une dimension de l'espace de description. On a donc un espace à 28*28 dimensions définit sur l'intervalle des entiers compris entre 0 et 255.

Les résultats et la présentation de la première expérience ne présentent que l'utilisation d'une unique distribution avec des paramètres bien établis. Cette distribution est la distribution normale qui est très utilisée et censée expliquer beaucoup de phénomènes. Mais on peut également se demander si ces résultats ne marchent pas aussi bien avec d'autres distributions.

Il est important de noter que pour cette expérience, l'ensemble d'apprentissage n'est composé que de 50 points au total pour 1000 points dans l'ensemble de test. Pour qu'un

[1] Les ensembles de test et de validation sont construits suivant la même distribution que l'ensemble d'apprentissage.

algorithme d'apprentissage artificiel fonctionne correctement, il faut que les données considérées soient représentatives des données à traiter. Avec 25 points d'apprentissage pour chaque classe générés suivant une loi normale comme celle utilisée dans l'article, il est peu probable que ces points forment une densité suffisante pour mettre en avant l'existence de différents comportements de points d'attaque.

On s'intéresse à la seconde expérience maintenant. Il a déjà été précisé que les auteurs ont utilisé un jeu de données différent, comportant des données de plus grandes dimensions. C'est le corpus MNIST qui comprend des images représentant des chiffres.
Alors que ce corpus comprend un nombre important d'exemples par classe (une classe étant un chiffre), les auteurs ont décidé de ne prendre en compte que 100 points par classe pour l'ensemble d'apprentissage et 500 points par classe pour l'ensemble de test. Là encore, le faible nombre d'exemples présents dans l'ensemble d'apprentissage est surprenant. En effet, avec autant d'exemples, les auteurs auraient pu utiliser des ensembles d'apprentissage avec des densités de points plus ou moins élevée ou encore avec plus ou moins de diversité dans les données. Avec une telle diversité, l'hypothèse de représentativité des données pourrait être validée.
De plus, l'utilisation d'ensembles comportant plus de données aurait appuyé l'utilisation de SVMs incrémentaux puisque le temps de convergence du calcul de la fonction séparatrice augmente avec le nombre d'exemples présents dans l'ensemble d'apprentissage.

Une dernière remarque sera une limite connue de cet article. En effet, sur cette deuxième expérience, les auteurs ont décidé d'utiliser la valeur de chaque pixel des images (de dimension 28x28) comme une dimension du vecteur de description des données. Habituellement, ce n'est pas ce genre de valeur qui est utilisée pour construire cet espace à partir d'images. Par exemple, le descripteur SIFT, qui est un descripteur très utilisé, se base sur des zones de fort gradient en considérant également l'orientation de ces gradients et leur voisinage.
Dans l'article, le passage de l'image à l'espace de description se fait donc de façon très simple. Il est donc facile de comprendre quelles modifications ont été apportées à une image et ainsi, avec un échantillon de points d'attaques, peut-être réussir à trouver une routine permettant de générer une nouvelle image ayant un fort impact sur les performances de l'algorithme de classification.
Dans le cas de descripteurs SIFT, la mise en correspondance entre modifications apportées et but visé se fait de manière plus difficile voire ne pas être intuitive du tout.

Chapitre 5

Les approfondissements étudiés

Cette partie a pour but d'expliquer et de discuter le travail mené durant ce stage afin d'approfondir les pistes dégagées dans la partie précédente, à savoir :

- le comportement de l'algorithme change-t-il si le nombre de données d'apprentissage augmente ?
- peut-on facilement choisir un point d'attaque initial qui arrivera à se déplacer rapidement vers le maximum de la fonction de « hinge loss » ?
- comment mener une attaque avec plusieurs points d'attaques différents, permettant d'avoir une influence plus forte qu'avec un unique point d'attaque ?
- est-il plus simple d'avoir une influence sur l'une ou l'autre des classes ?

Nous détaillerons d'abord le protocole expérimental mis en œuvre puis, chaque section reprendra une des pistes citées précédemment. Chaque section sera alors organisée de la façon suivante :

- le but de l'expérience sera d'abord précisé ;
- les conditions expérimentales seront brièvement rappelées ;
- des résultats représentatifs seront présentés ;
- quelques commentaires viendront finalement appuyer ces résultats.

5.1 Conditions expérimentales

Pour cette étude, nous nous sommes limités aux problèmes de classifications à deux classes en utilisant l'algorithme des SVMs à marges molles dans un espace de description à deux dimensions. Chaque dimension est défini dans $[-5; 5]$.

Pour générer des données, nous avons choisi différentes distributions : la distribution gaussienne, la distribution uniforme, une distribution bi-gaussienne et une distribution dite « spéciale » nous permettant d'utiliser pleinement les noyaux à bases radiales (RBFs). La distribution gaussienne est générée comme décrit dans [4]. La classe positive a donc une moyenne de 1.5 et un écart-type de 0.6 pour la première dimension et une moyenne de 0 et un écart-type de 0.6 pour la seconde. La classe négative a une moyenne de -1.5 et un écart-type de 0.6 pour la première dimension et une moyenne de 0 et un écart-type de 0.6 pour la seconde. De part les paramètres qui décrivent cette distribution, les classes peuvent être partiellement mélangées.
La distribution uniforme elle est générée de manière à ce que les deux nuages soient

parfaitement séparés. Typiquement, la classe positive à été générée sur un intervalle allant de 1 à 3 sur la première dimension et sur un intervalle de -3 à 3 sur la seconde dimension. La classe négative a été générée avec un premier intervalle de -1 à -3 sur la première dimension et sur un intervalle de -3 à 3 sur la seconde.
La classe bi-gaussienne consiste à diviser, pour une classe, le nombre de points à générer en deux gaussiennes. 66% des points se trouvent dans un premier nuage de point dont la moyenne est de -2 sur la première coordonnée avec un écart type de 0.4 et une moyenne de 0 sur la seconde dimension avec un écart-type de 0.4. Les 33% restants se trouvent dans un autre nuage dont la moyenne est -2 et l'écart-type est 0.4 sur la première dimension et sur la seconde 3 et 0.4.
La classe « spéciale » consiste à générer des points suivant une distribution gaussienne, centrée en 0 avec un écart-type de 0.4 sur les deux dimensions. Puis à ne garder que les points se trouvant entre 2 cercles. Typiquement, les rayons utilisés ont été 1 pour le cercle intérieur et 2 pour le cercle extérieur. Les points restants dans l'anneau sont alors divisés en deux classes (suivant l'axe vertical = 0 dans le cas de classes équilibrées, pour le cas déséquilibré, l'axe vertical = constante est utilisé comme séparation). Finalement, un des deux groupes est déplacé pour arriver au résultat suivant :

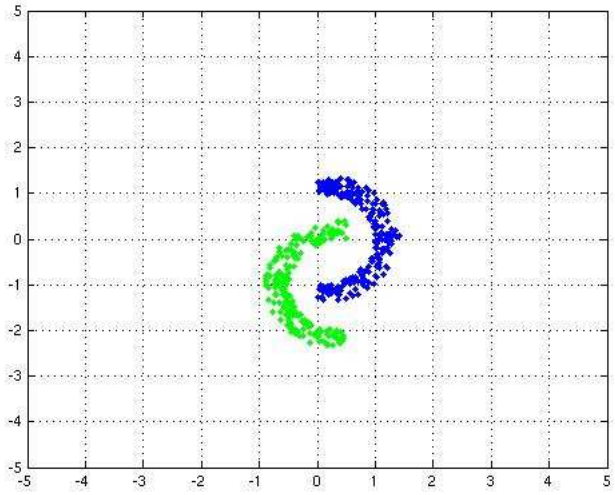

Fig. 5.1 – Un exemple de résultat de génération de la distribution « spéciale RBF »

Avoir une telle séparation des classes obligent la fonction séparatrice a être sinueuse. Cela permet de voir l'influence des paramètres C et Gamma sur l'hyperplan séparateur.

Nous avons également utilisé deux types de noyaux : les noyaux linéaires et les noyaux à bases radiales (RBFs). Nous n'avons utilisé que les paramètres C, permettant de pénaliser les erreurs commises par l'algorithme des SVMs, pour les deux distributions et le paramètre Gamma pour les noyaux RBFs permettant aux fonctions séparatrices de suivre plus ou moins bien la forme des distributions des données. Dans la seconde partie du document, nous avions dis que les utilisateurs devaient réglés ces paramètres par eux-mêmes en faisant une recherche sur un intervalle qu'ils auraient défini. Pour C, nous avons choisi un intervalle allant de 2^{-5} à 2^{20} en incrémentant d'un en un la puissance de

deux. Pour Gamma, l'intervalle considéré va de 2^{-10} à 2^{15} en incrémentant de 0.25 en 0.25 la puissance de deux. Lorsque les deux classes sont équitablement représentées dans l'ensemble d'apprentissage, il n'est pas nécessaire de forcer l'algorithme à faire ou à éviter de faire des erreurs. C est donc fixé à 1 et n'est pas réajusté dans ces cas là.

Les expériences ont montré qu'avec de tels intervalles, les paramètres C et Gamma pouvaient contraindre les fonctions séparatrice à « coller » aux distributions. Cela peut poser problème après chaque déplacement de points d'attaques puisqu'il est possible que l'ensemble des vecteurs supports devient vide, ce qui ne permet pas à l'algorithme de s'exécuter. Pour éviter ce problème, l'algorithme a la possibilité de réentrainer ces paramètres C et Gamma après chaque déplacement de points si l'ensemble des vecteurs supports est vide. Les intervalles et les pas d'incrémentation restent les mêmes, ce qui amènent quelques fois à retomber sur les mêmes jeux de paramètres faisant que l'algorithme ne peut pas continuer, ne retrouvant pas de nouveaux vecteurs supports.

Plusieurs expériences ont été menées. Nous avons ainsi confronté :
- deux classes générées à partir d'une distribution gaussienne et un noyau linéaire pour les séparer,
- deux classes générées avec une distribution uniforme un noyau linéaire pour les séparer,
- une classé générée à partir d'une bi-gaussienne contre une classe générée avec une distribution gaussienne et un noyau linéaire pour les séparer,
- une classé générée à partir d'une bi-gaussienne contre une classe générée avec une distribution gaussienne et un noyau RBF pour les séparer,
- deux classes générées à partir de la distribution « spéciale » et un noyau RBF pour les séparer.

Les ensembles d'apprentissage comportent soit 25, 250 ou 1000 points. Les ensembles de tests sont générés suivant la même loi avec les mêmes paramètres mais avec 20 fois plus de points que dans l'ensemble d'apprentissage. Malgré cela, d'après la description de la génération des points suivant une distribution « spéciale RBF », il est difficile de contrôler exactement le nombre de points présents dans ces deux ensembles. Il aurait été peut-être plus judicieux d'utiliser une autre distribution que la distribution gaussienne pour créer cette distribution « spéciale » afin de mieux contrôler le nombre de points gardés dans les ensembles.

Un ensemble de test de 50000 points par classe est également créé. Il nous permet de noter les erreurs de classification pour faites par les différentes séparatrices. Cela est utile pour comparer les attaques dans le cas où le déplacement de la fonction séparatrice n'est pas clairement visible.

Nos expériences ont vu deux types d'attaques se dérouler :
- soit un point de l'ensemble d'apprentissage est choisi, copié et son label changé, l'algorithme de l'article [4] est alors exécuté et l'attaque s'arrête,
- soit le processus précédent est itéré 20 fois suivant différente modalité de choix de point à copier et à changer de label.

La condition d'arrêt que nous avons utilisé pour arrêter les itérations de déplacement

du point d'attaque se compose de :

- d'un minimum d'itération à effectuer, fixé à 10 itérations de manière arbitraire, pour permettre aux points de sortir d'une zone dans laquelle le nombre d'erreur de classification n'évolue pas.
- d'un maximum d'itération à atteindre, fixé à 100 itérations de manière arbitraire, pour être sûr que l'algorithme va se terminer dans le cas où le pas d'ascension serait trop grand ne permettant pas d'atteindre un maximum (local ou global).
- de la condition suivante : « tant que l'erreur de classification ne baisse pas » encourageant ainsi le point à aller au-delà d'une zone où le nombre d'erreur de classification n'évoluerait pas.

Le pas d'itération a été fixé après plusieurs expérimentations à t=0.15.

Une carte de la fonction de « hinge loss » est calculée après chaque ajout de points d'attaques et après chaque fin de déplacement de ce point. Ce calcul se fait en établissant une grille régulière et en calculant le nombre d'erreurs de classification fait en chaque point de cette grille. Pour que celle-ci soit continue, une interpolation est faite entre les points de la grille.
Puisque certaines expériences s'appuient sur la proximité avec des zones de fortes influence pour choisir le prochain point d'attaque, la grille régulière a une résolution de 100x100.
Cette carte est normalisée entre 0 et 1 sur l'ensemble de l'exécution de l'attaque. Ainsi, à chaque fois qu'une carte est générée, il faut garder en mémoire la valeur maximale et la valeur minimale rencontrée jusqu'alors et à la fin de l'attaque, chaque valeur présente sur la carte est ramenée entre 0 et 1 en effectuant le calcul suivant : $\frac{valeur\ courante\ -\ valeur\ minimale}{valeur\ maximale\ -\ valeur\ minimale}$

Le calcul d'une carte peut facilement se paralléliser. Pour gagner encore en temps de calcul, le programme Matlab a été compilé et exécuté sur une machine à huit cœurs de la grille de calcul Igrida. C'est la librairie libSVM qui a été choisie pour pouvoir être utilisée à la fois par le code compilé et par le code Matlab. Et en utilisant une machine à huit cœurs, le temps de calcul d'une attaque s'élève à entre une et deux heures lorsque tout se passe bien[1] contre quatre à huit heures sur la machine que j'avais à disposition qui ne comportait que deux cœurs.

Finalement, chaque carte qui sera présentée par la suite est composée de 4 sous-images. Pour chacune des 4 sous-images sera présentée :

- la position des points de l'ensemble d'apprentissage,
- la fonction d'influence sur le fond de l'image,
- la fonction séparatrice en trait continu noir,
- les marges de part et d'autres de la séparatrice en ligne discontinue,
- les vecteurs supports seront les points qui sont entourés par un cercle noir,

[1]Les dernières expériences menées ont montré qu'il peut y avoir des problèmes au niveau de l'établissement des fonctions séparatrices lorsque le paramètre C devient très grand, ne permettant ainsi plus aucune erreur. La convergence vers des fonctions séparatrice à marges maximales ne devient plus possible et il faut attendre en chaque point d'une zone à forte influence que l'algorithme de libSVM arrive à un nombre maximal d'itérations avant d'avoir un résultat. Il faut plus de cinq heures à la machine que nous avons utilisé pour pouvoir créer une seule carte.

- la trajectoire des points d'attaque sera montrée par une succession de points noirs reliés entre eux
- le cadre en traits discontinus montrent une borne pour éviter que le point d'attaque ne s'éloigne trop en dehors de l'image. S'ils essayent de sortir de ce cadre, leurs coordonnées sont alors reprojetées sur la bordure du cadre.

Voici un exemple de ces cartes :

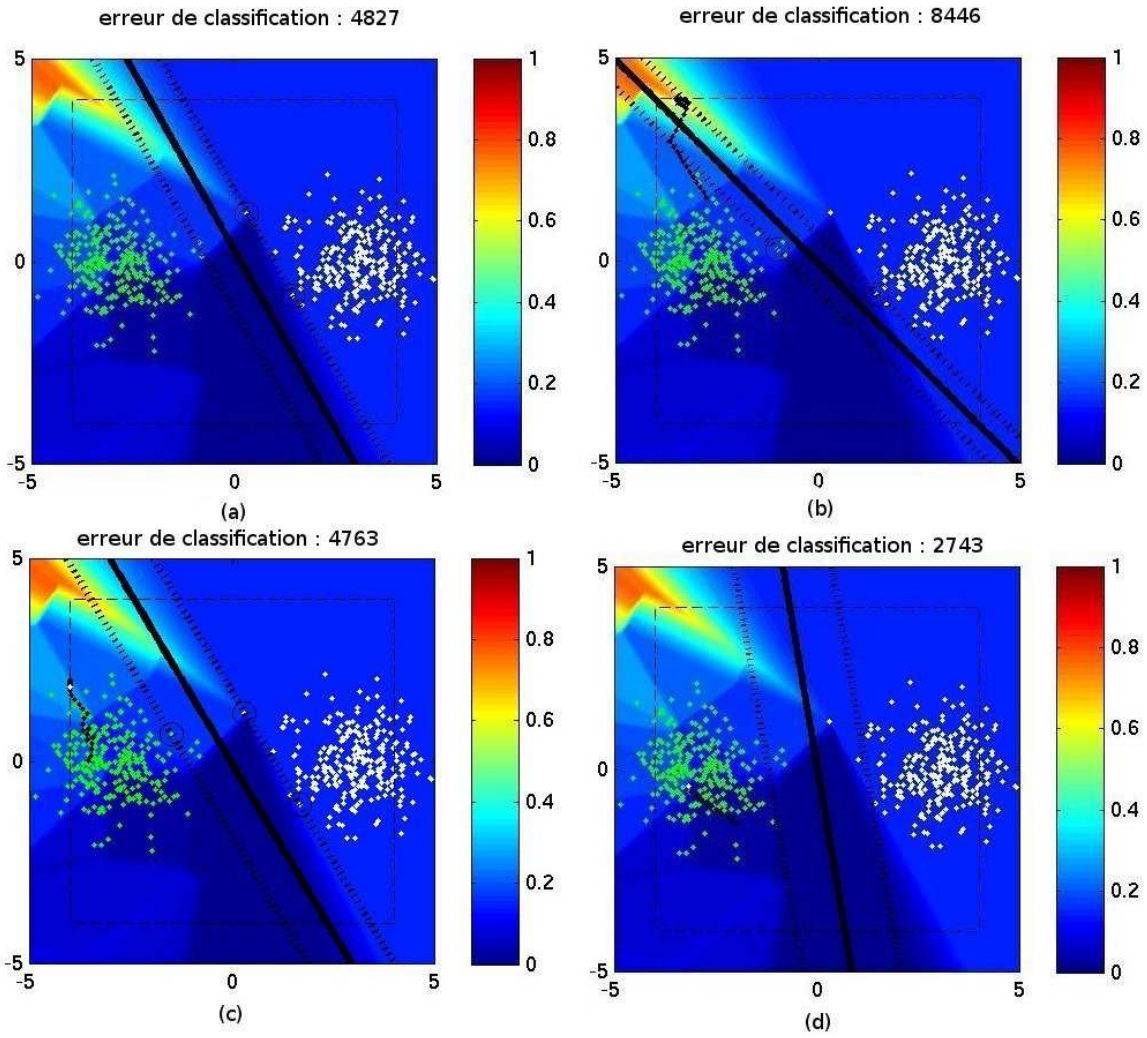

FIG. 5.2 – Un exemple de figure qui viendra appuyer les résultats de nos études

La figure (a) montre la fonction d'influence sur la séparatrice initiale, sans avoir ajouter de points d'attaques. La carte présente en arrière-plan représente la fonction de « hinge loss ». Cette fonction d'influence est normalisée entre 0 et 1 comme le suggère le ruban associant la couleur à une valeur sur la gauche de chaque imagette. Dans cet exemple, une

zone de forte influence est présente en haut à gauche des imagettes. L'attaquant souhaite donc que sa donnée corrompue arrive sur cette zone à la fin de son déplacement.
Les deux classes sont représentées par les deux nuages de points présents sur la moitié gauche et droite des imagettes. On remarque qu'elles sont parfaitement séparées.
La figure (b) présente la fonction séparatrice après avoir créé un point d'attaque à partir d'un point de l'ensemble d'apprentissage qui est proche de la zone de forte influence. Le déplacement de ce point d'attaque est observable par la ligne reliant le point de l'ensemble d'apprentissage à partir duquel le point d'attaque a été créé à la partie supérieure du cadre. Les deux autres figures (c) et (d) montrent le déplacement d'autre point d'attaque ainsi que l'hyperplan séparateur résultant de cette attaque.
Au dessus de chaque figure se trouve le nombre d'erreurs de classfication qui a été observé.

Par soucis de place et du fait du nombre important des expériences, nous nous permettons de ne pas présenter tous les résultats mais seulement ceux qui nous semblent importants et représentatifs.

5.2 La sélection d'un bon candidat comme point initial d'attaque

Dans cette première section, nous essayons de déterminer si il est possible de choisir un point qui donnera naissance à un point d'attaque et dont le nombre d'itérations qui le mène à une zone de forte influence soit faible.

Pour cela, nous avons utilisé des classes équilibrées ce qui implique que la variable C est fixée à 1. Toutes les expériences utilisent un noyau linéaire pour séparer les données. La variable Gamma n'est donc pas utilisée. Plusieurs distributions ont été utilisée :

- deux distributions gaussiennes avec 250 points dans l'ensemble d'apprentissage par classe et 5000 pour l'ensemble de test par classe,
- deux distributions gaussiennes avec 1000 points dans l'ensemble d'apprentissage par classe et 20000 pour l'ensemble de test par classe,
- deux distributions uniformes avec 250 points dans l'ensemble d'apprentissage par classe et 5000 pour l'ensemble de test par classe,

Chaque point présent dans l'ensemble d'apprentissage devient un point d'attaque tour à tour et on observe leurs déplacements.
La classe attaquée se trouve à gauche de l'axe vertical=0 et la classe attaquante à droite.

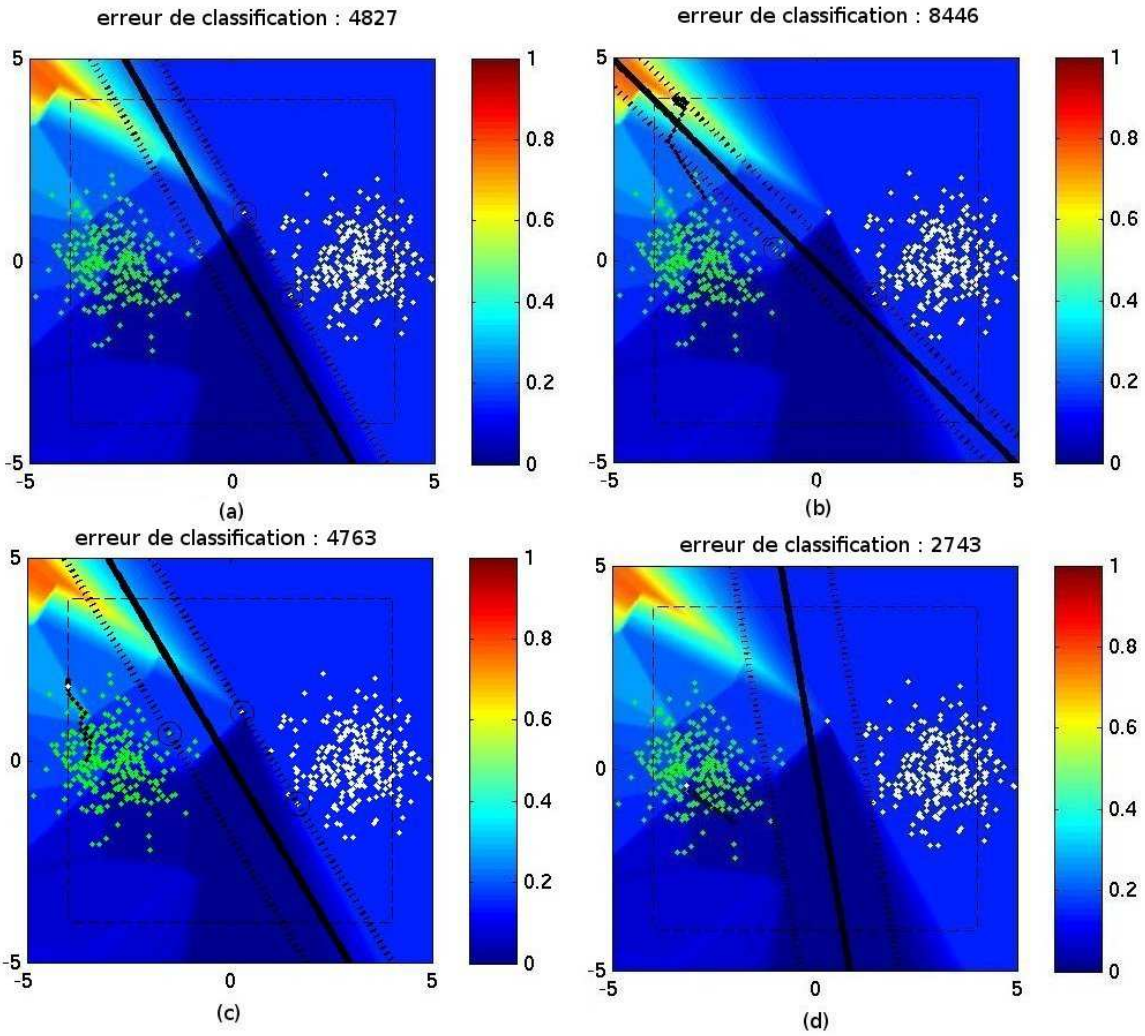

FIG. 5.3 – distribution gaussienne, 250 points par classe dans l'ensemble d'apprentissage

La figure (a), sur la partie gauche, montre que la zone de forte influence se trouve en haut à gauche. Un point proche (b) arrive à atteindre cette zone. Un point plus éloigné (c), se trouvant derrière le nuage de point et à proximité du cadre, arrive à contourner le nuage de point mais tombe dans un maximum local. Enfin, la dernière figure (d), montre qu'un point à l'opposé de cette zone ne donne pas de résultat intéressant.
Ces observations sont corroborés par le nombre d'erreurs de classification observé sur l'ensemble de test. En effet, la séparatrice en (b) fait à peu près deux fois plus d'erreurs de classfication que l'hyperplan initial de (a). Les erreurs faites dans (c) sont à peu près égales à celles de (a) et (d) en fait quasiment deux fois moins. On peut également remarquer que l'effet d'un point qui arrive sur cette zone d'influence fait que l'équation de la séparatrice est x=y ce qui lui donne une pente moins forte. Par contre, ses marges ont rétréci.

On peut donc en déduire, que les points étant les plus proches d'une zone d'influence ont de grandes chances de l'atteindre alors que ceux qui sont plus éloignés ont plus de chances de se faire piéger par des maxima locaux. Les expériences suivantes montreront que les noyaux RBFs sont des noyaux plus complexes mais pour lesquels cette hypothèse de sélection d'un point d'attaque est vérifiée également.

5.3 Le passage à l'échelle

Dans cette section, les expériences menées ont pour but d'observer le comportement de l'algorithme lorsque le nombre de points dans l'ensemble d'apprentissage augmente. Par exemple, les trajectoires des points, le nombre d'itération ou la zone dans laquelle se positionne un point à la fin de l'attaque peuvent-ils être affectés ?

Pour cela, nous avons utilisé des classes équilibrées ce qui implique que la variable C est fixée à 1. Toutes les expériences utilisent un noyau linéaire pour séparer les données. La variable Gamma n'a donc pas besoin d'être fixée. Plusieurs expériences ont été menées mettant en jeu différentes distribution de données :

- deux distributions gaussiennes avec 25 points dans l'ensemble d'apprentissage et 500 pour l'ensemble de test par classe,
- deux distributions gaussiennes avec 250 points dans l'ensemble d'apprentissage et 5000 pour l'ensemble de test par classe,
- deux distributions gaussiennes avec 1000 points dans l'ensemble d'apprentissage et 20000 pour l'ensemble de test par classe,
- deux distributions uniformes avec 25 points dans l'ensemble d'apprentissage et 500 pour l'ensemble de test par classe,
- deux distributions uniformes avec 250 points dans l'ensemble d'apprentissage et 5000 pour l'ensemble de test par classe,

Chaque point présent dans l'ensemble d'apprentissage devient un point d'attaque tour à tour et on observe leurs déplacements.
La classe attaquée se trouve à gauche de l'axe vertical=0 et la classe attaquante à droite.

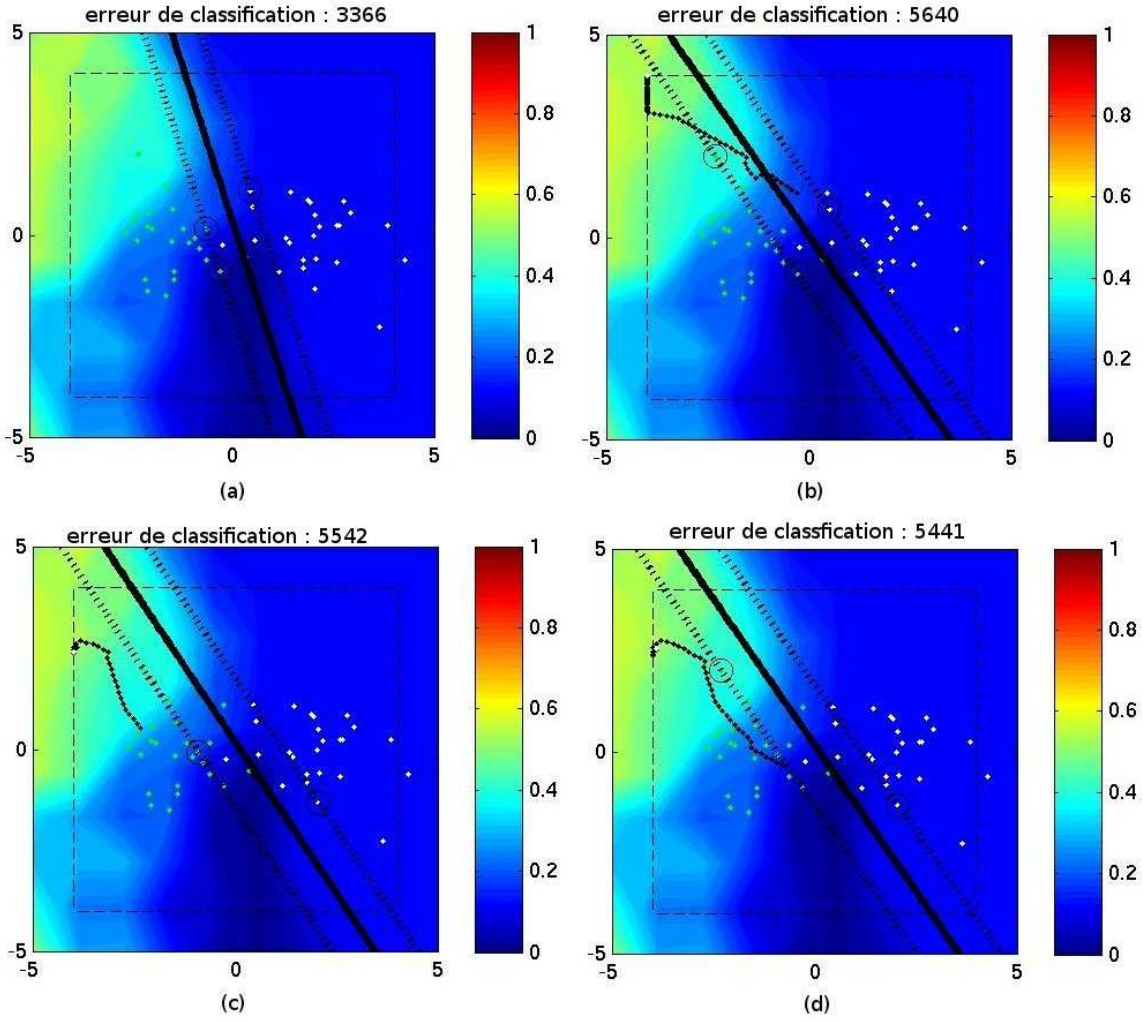

FIG. 5.4 – distribution gaussienne, 25 points par classe dans l'ensemble d'apprentissage

Comparé à l'expérience présente, on retrouve bien le fait qu'un point proche d'une zone d'influence a plus de chances d'arriver sur celle-ci alors que les autres points vont vraisemblablement tomber sur un maximum local. Malgré cela, les erreurs de classification ne changent pas énormément.

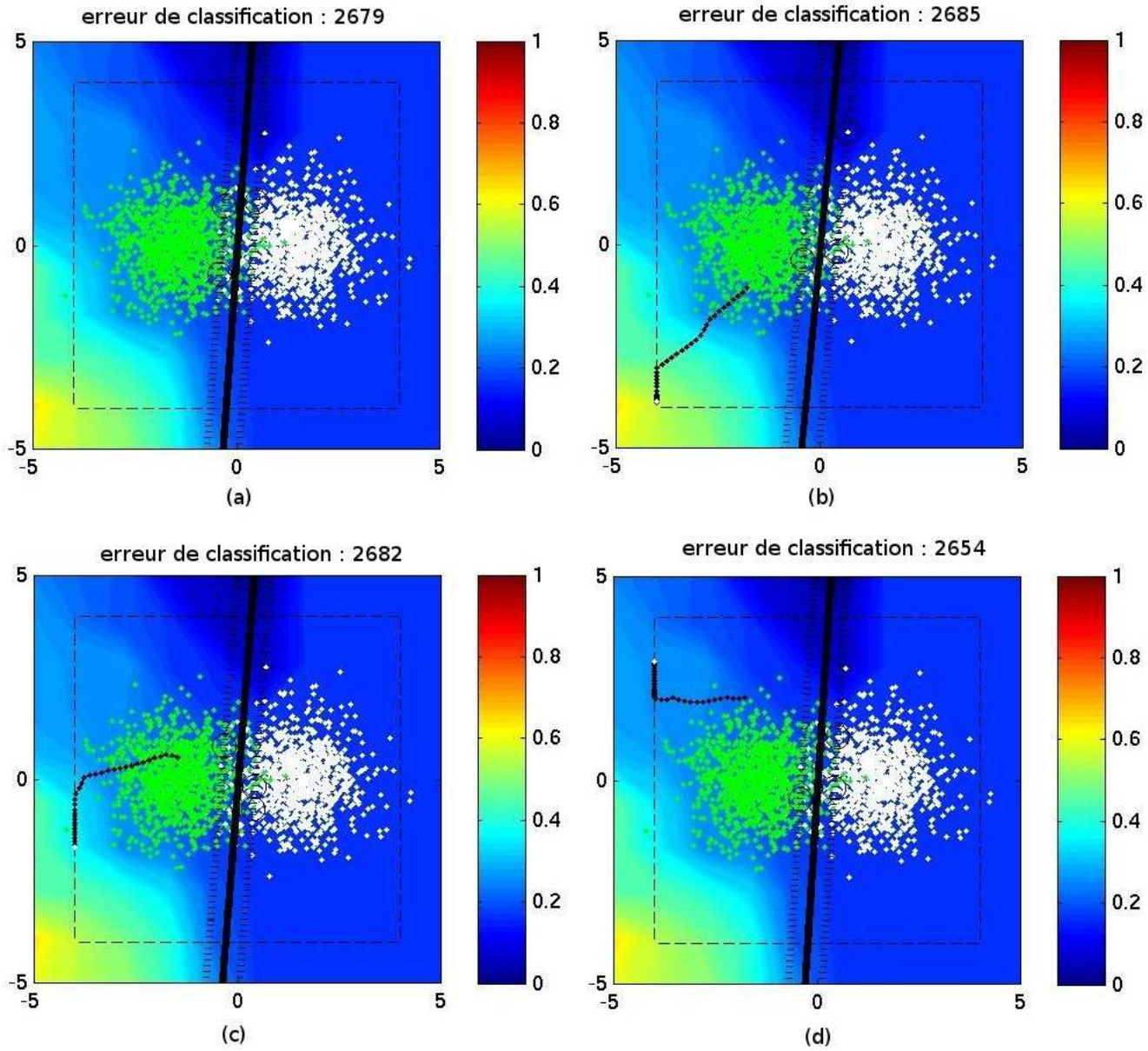

FIG. 5.5 – distribution gaussienne, 1000 points par classe dans l'ensemble d'apprentissage

Ici, la zone de forte influence se situe en bas à gauche des imges. Le mouvement de la fonction séparatrice est difficile à observer. Bien que le déplacement du point d'attaque en (b) semble meilleur, arrivant sur la zone de forte influence, les résultats au niveau des erreurs de classfication n'augmentent pas de manière drastique comparé aux images (a),(c) et (d).
On peut en conclure qu'avec un nombre de croissant de points dans l'ensemble d'apprentissage, l'influence d'un unique point d'attaque sur la fonction séparatrice diminue.

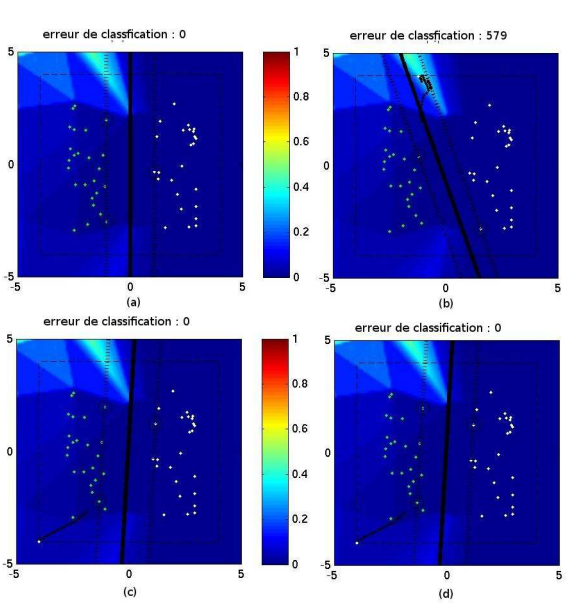

FIG. 5.6 – distribution uniforme, 25 points par classe dans l'ensemble d'apprentissage

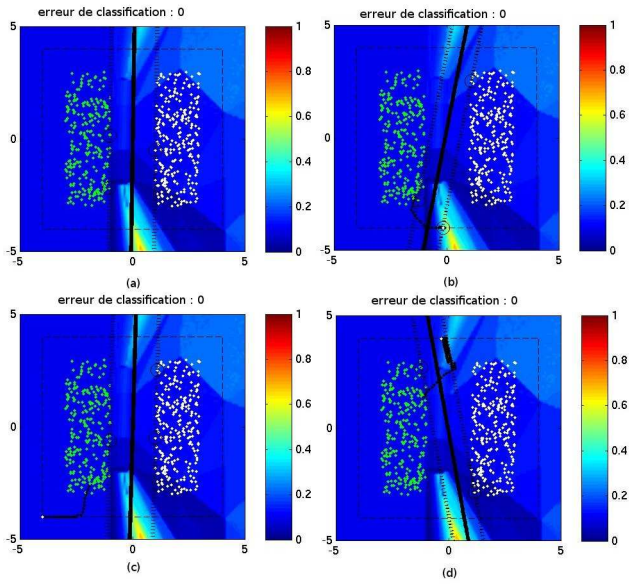

FIG. 5.7 – distribution uniforme, 250 points par classe dans l'ensemble d'apprentissage

C'est également le cas pour la distribution uniforme. Où, malgré les déplacements de l'hyperplan séparateur observables dans la figure 5.7. Le nombre d'erreur de classification est nul.

Dans le cas de l'expérience où deux gaussiennes sont représentées par 1000 exemples chacune dans l'ensemble d'apprentissage, le déplacement de la fonction séparatrice est à peine perceptible. Dans le cas des distributions uniformes, de par la construction des deux classes, les fonctions séparatrices se déplacent mais sans jamais commettre d'erreur de classification ce qui n'est pas intéressant non plus.

5.4 La sélection de la classe à attaquer

D'après les conditions expérimentales que nous avons evoqué, il apparaît que certaines expériences présentent des symétries spatiales dans les distributions des données. Alors que d'autres n'ont pas cette propriété.
Nous nous proposons alors d'étudier l'influence de cette symétrie spatiale sur le resultat des expériences. Cette étude se décompose en deux cas : le cas où le nombre de données pour chaque classe dans les ensembles d'apprentissage et de test est équilibré et le cas où il ne l'est pas.

5.4.1 dans le cas de classes équilibrées

Nous n'utiliserons ici que le cas où le nombre de points dans chaque classe est équilibré. On a alors un paramètre C qui est fixé à 1. Pour les expériences où un noyau

RBF est utilisé, le paramètre Gamma est ajusté. Dans le cas des noyaux linéaires, ce paramètre n'intervient pas. Gamma est ajusté à l'ensemble d'apprentissage initial avant même d'avoir ajouté un point d'attaque mais il peut également être ajusté au cours de l'attaque dans le cas où l'ensemble des vecteurs supports est vide.

Les distributions utilisées avec les noyaux sont :

- deux distributions gaussiennes avec 250 points dans l'ensemble d'apprentissage par classe et 5000 pour l'ensemble de test par classe et un noyau linéaire pour les séparer,
- deux distributions uniformes avec 250 points dans l'ensemble d'apprentissage par classe et 5000 pour l'ensemble de test par classe et un noyau linéaire pour les séparer,
- la distribution spéciale et un noyau RBF pour les séparer,
- une distribution bi-gaussienne contre une distribution gaussienne avec 250 points dans l'ensemble d'apprentissage par classe et 500 pour l'ensemble de test par classe avec un noyau linéaire pour les séparer,
- une distribution bi-gaussienne contre une distribution gaussienne avec 250 points dans l'ensemble d'apprentissage par classe et 500 pour l'ensemble de test par classe avec un noyau linéaire pour les séparer,

Dans le cas de classes équilibrées, on peut se convaincre assez facilement que le fait qu'il y ait une symétrie spatiale entre les représentations des classes ne va pas favoriser l'attaque contre l'une ou l'autre des classes.

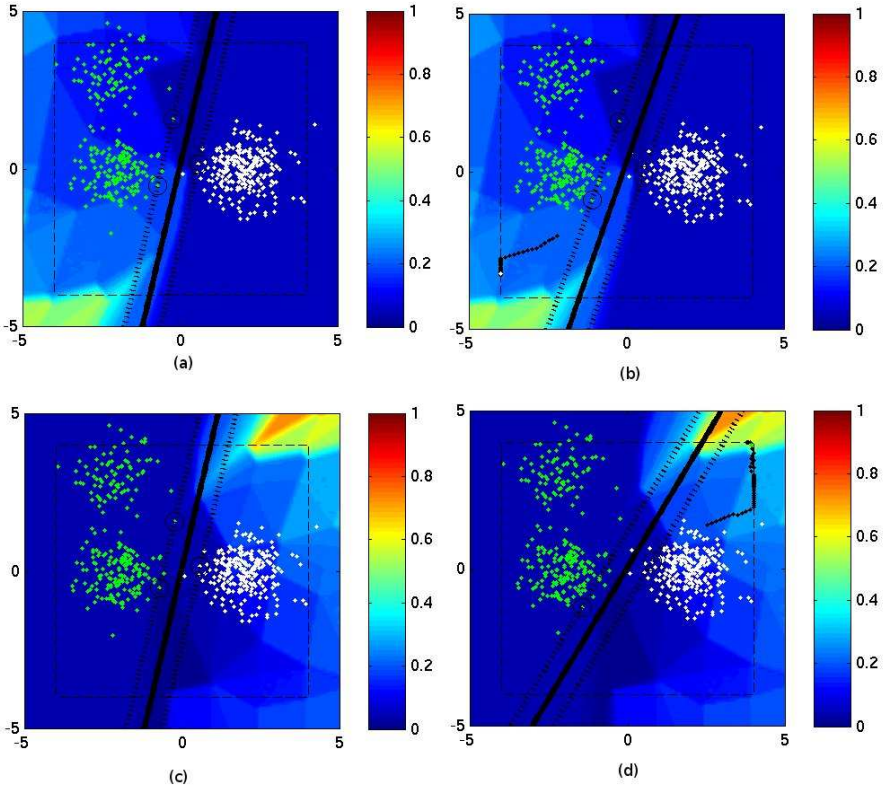

FIG. 5.8 – bi-gaussienne contre gaussienne, noyau linéaire, 250 points par classe dans l'ensemble d'apprentissage

La ligne du haut (a) et (b) présente une exécution d'une attaque où la classe attaquée est la classe négative et la ligne du bas (c) et (d) présente une attaque contre la classe positive. (a) et (c) présentent des ensembles d'apprentissage non-corrompus avec, en fond, une carte prévoyant l'attaque contre la classe négative (a) et l'attaque contre la classe positive (c).
En comparant l'image (d) et l'image (b), l'attaque contre la classe positive (d) semble avori plus d'impact sur la séparatrice que l'autre attaque (b). Mais, en fait, dans les deux cas, aucune n'erreur n'est faite en plus de l'erreur initiale par la fonction séparatrice initiale. En effet, un point blanc, masqué par l'épaisseur de la fonction séparatrice, est déjà mal-classé avant même d'introduire des points d'attaques.

5.4.2 La sélection de la classe à attaquer dans le cas déséquilibré

Que se passe-t-il maintenant dans le cas où le nombre d'exemple par classe est déséquilibré ?
Dans ce cas, les paramètres C et Gamma (pour les noyaux RBFs), sont ajustés pour l'ensemble d'apprentissage initial et également au cours de l'attaque lorsque l'ensemble des vecteurs supports est vide afin d'essayer d'en trouver de nouveaux.
C'est systématiquement la classe positif qui est sur-représentée par rapport à la classe négative et le facteur de représentation entre les deux est un facteur dix.

Les distributions et noyaux utilisés sont :

- deux distributions gaussiennes avec 25 et 250 points dans l'ensemble d'apprentissage par classe et 500 et 5000 pour l'ensemble de test par classe et un noyau linéaire pour les séparer,
- deux distributions uniformes avec 25 et 250 points dans l'ensemble d'apprentissage par classe et 500 et 5000 pour l'ensemble de test par classe et un noyau linéaire pour les séparer,
- la distribution spéciale et un noyau RBF pour les séparer,
- une distribution bi-gaussienne contre une distribution gaussienne avec 25 et 250 points dans l'ensemble d'apprentissage par classe et 500 et 5000 pour l'ensemble de test par classe avec un noyau linéaire pour les séparer. C'est la distribution bi-gaussienne qui est sous-représentée,
- une distribution bi-gaussienne contre une distribution gaussienne avec 25 et 250 points dans l'ensemble d'apprentissage par classe et 500 et 5000 pour l'ensemble de test par classe avec un noyau linéaire pour les séparer. C'est la distribution bi-gaussienne qui est sous-représentée.

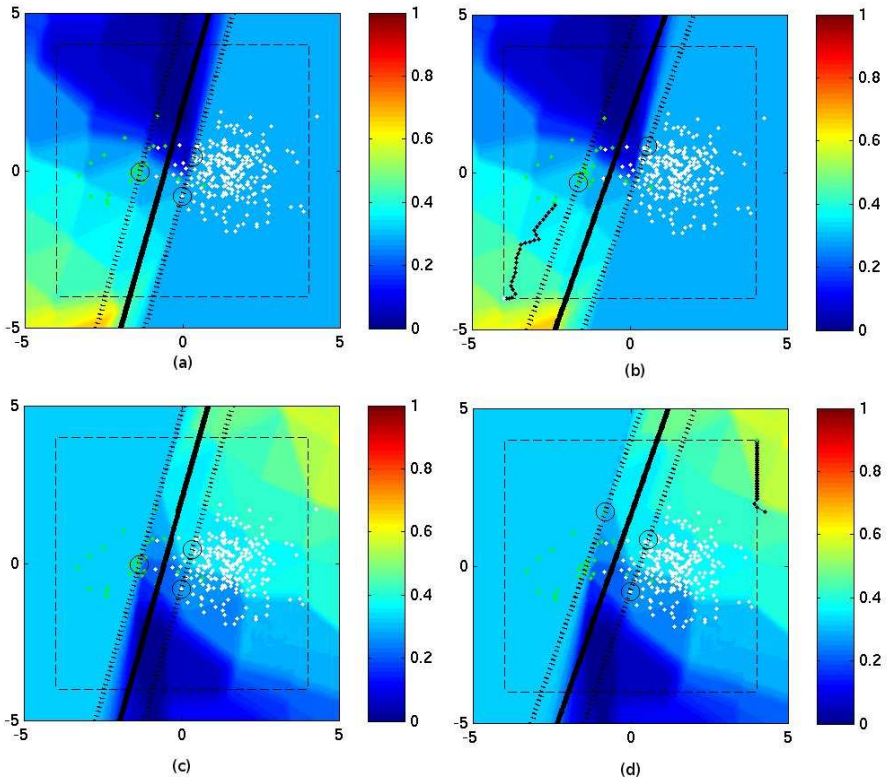

FIG. 5.9 – gaussienne contre gaussienne, noyau linéaire, 25 points pour la classe négative contre 250 pour la classe positive dans l'ensemble d'apprentissage

Ici encore, la fonction séparatrice ne semble pas être énormément influencer malgré le fait que les points d'attaques sur des zones qui semblent être les zones maximales se trouvant dans le cadre.

Avec plus de 250 points dans l'ensemble d'apprentissage, les dernières expériences ont montré que les déplacements des fonctions séparatrices n'étaient pas dû aux placements des points d'attaques mais plutôt à la marge de manœuvre qu'elles avaient avant de commettre une nouvelle erreur de classification.
Dans une véritable application, le nombre de données présent dans les ensembles d'apprentissage sont bien supérieurs à ce que nous avons utilisé ici.
Ainsi, nous avons pensé à construire une attaque en déplaçant plusieurs données corrompues.
Biggio et al. précisent dans [4] qu'essayer de mener une attaque en déplaçant simultanément différents points d'attaque de sorte que leur impact sur la séparatrice soit maximale n'est pas chose facile. Nous avons donc décidé de nous intéresser d'abord à les faire se déplacer de manière séquentielle.
A ce stade, nous avons établi trois stratégies de choix du prochain point d'attaque.

5.5 Comparaison de stratégies d'attaques

Voici une description des différentes stratégies que nous avons utilisées :

- la première solution consiste à choisir un point d'attaque, le déplacer. À la fin de cette première itération, la dernière position du point d'attaque est copiée et un nouveau point d'attaque est créé. Ainsi, deux points d'attaque se situent au même endroit dans l'espace de description. Le fait d'ajouter un nouveau point d'attaque au même endroit peut faire que la fonction d'influence va également changer permettant ainsi au nouveau point de se déplacer.
- la seconde solution est de prendre des points d'attaque uniquement dans l'ensemble d'apprentissage. Ainsi, une fois qu'un point est déplacé, il n'est pas possible de le redéplacer mais on peut avoir une chance d'aller choisir un nouveau point d'attaque à un autre endroit de l'espace qui est plus proche d'une zone d'influence.
- finalement, on peut combiner les deux premières solutions. Si une zone de forte influence se trouve à proximité, on peut alors créer un nouveau point d'attaque à partir de sa dernière position ou si une zone est plus éloignée, un autre point est choisi à la place.

Nous souhaitons alors savoir si une stratégie est meilleure qu'une autre.

Ces stratégies ne répondent cependant pas au même problème initial que les expériences précédentes. En effet, l'algorithme va maximiser l'influence d'un point d'attaque sur la fonction séparatrice mais au cours de l'attaque, cette maximisation se fait sur la fonction séparatrice de l'itération précédente et non plus sur l'hyperplan initial.
L'hyperplan peut ainsi osciller entre deux positions et faire que l'influence globale de l'attaque sur l'hyperplan séparateur initial soit moindre.

Ces stratégies correspondent à l'hypothèse de nous trouver dans un contexte où l'algorithme réapprenait les paramètres de sa fonction séparatrice de manière périodique. Cela voudrait dire qu'une attaque prévue sur plusieurs jours, ou plusieurs semaines ne se confronterait pas au même hyperplan séparateur.

De même, nous pourrions essayer de garantir une influence croissante de l'attaque sur l'hyperplan initial en venant comparer l'angle formé par l'hyperplan séparateur initial et la position de l'hyperplan après avoir déplacé un i-ème point d'attaque d'une part et les angles formés avec les itérations précédentes d'autre part.
Si ce critère n'est pas correcte, peut-être que l'aire définie en bornant l'espace d'une part et la fonction séparatrice d'autre part pourra être un bon indicateur.

De la même façon, des questions se posent sur ce qui doit être fait lorsque le critère n'est pas validé. Est-ce que l'attaque se termine ? Est-ce qu'il faut choisir un autre point d'attaque ?

Pour la suite de ce travail, Nous avons décidé que nous allions nous concentrer sur ces problèmes.

Voici une dernière remarque qui peut être faite sur les expériences que nous avons menées. Dans le cas des classes déséquilibrées, l'option proposée par libsvm -w (weight) permet de donner un poids différent aux 2 classes. Un problème bien connu des SVMs est

qu'en présence de déséquilibre dans la représentation des classes, l'hyperplan séparateur a tendance à venir se rapprocher de la classe la moins représentée, laissant ainsi plus de marges à la classe la plus représentée. Ainsi, à terme, sur une attaque de type *Causative*, le SVM va avoir tendance à tout classer dans la classe majoritaire. Cela aurait pu être intéressant d'utiliser cette option à la place de faire uniquement varier les paramètres C et Gamma.

Encore une fois, le domaine étant assez vaste et nouveau, il est très difficile de tout traiter. A peu près toutes les expériences ont mené à de nouveaux questionnements et ainsi de nouvelles pistes à explorer.

Chapitre 6

Quelques discussions ouvertes à la vue de ce qui a été étudié

Les expériences réalisées nous ont permi de comprendre certains points mais aussi de soulever d'autres problèmes qui n'ont pas pu être traités, par manque de temps.

6.1 La fonction à maximiser

Premièrement, les auteurs ont décidé de maximiser une fonction de coût pour l'algorithme. Cette fonction représente la priorité que va donner l'algorithme à cette erreur. Plus le coût de l'erreur est élevé, plus l'algorithme va essayer de minimiser cette erreur. On peut se demander si cette fonction est facile à obtenir dans toutes circonstances. *Biggio et al.* mettent cette fonction en relation avec le nombre d'erreur que l'algorithme fait dans l'article étudié. Une autre fonction qui serait peut-être plus visuelle et peut-être plus compréhensible pour l'attaquant serait de prendre en compte le déplacement de la fonction séparatrice. Cette nouvelle fonction peut très certainement également être mise en relation avec la fonction de coût initiale.
Une autre fonction encore plus simple serait de visualiser l'aire occupée par les 2 classes suivant l'hyperplan séparateur.

Ce ne sont que des suppositions, il n'a pas été étudié l'impact que ces fonctions auraient sur les objectifs des attaquants et sur les résultats obtenus. Pourrait-on mieux définir les objectifs visés ? Est-ce que ces fonctions seraient aussi permissives que l'initiale en permettant de formuler des objectifs qui n'auraient pu être atteints avant ?

6.2 La fonction considérée mise en relation avec le passage en grandes dimensions

La fonction de coût affichée en arrière-plan des figures est calculée par estimation. Il s'agit de simuler, en discrétisant l'espace de description, le fait d'ajouter un point de la classe attaquante sur une grille. Ce nouveau point est ensuite ajouter à l'ensemble d'apprentissage et une nouvelle fonction séparatrice est simulée. Elle est alors testée et c'est le nombre d'erreurs de classification total qui est sauvegardé en mémoire pour ce

point et qui est affiché par la suite.
En rappelant que l'équipe TexMex traite des données multimédia, il est habituel de passer dans des espaces de description de grandes dimensions avec un volume important de données à traiter. Il n'est pas rare de travailler avec des vecteurs de description de dimension 1 million. La carte de la fonction de coût devient alors beaucoup trop chère à calculer puisqu'il faut discrétiser l'espace de description avec une résolution importante pour avoir une bonne estimation de cette fonction.
Comment avoir une bonne intuition alors de cette fonction ? A cause de ce coup de calcul qui augmente considérablement avec le nombre de dimensions de l'espace de description, il est possible que cela justifie le fait que les auteurs aient choisi de prendre un point au hasard dans l'espace pour commencer l'attaque. Cela leur permettant d'avoir des chances que le point ne soit pas trop loin d'une zone où la fonction à une valeur forte et leur évitant ainsi d'avoir à la calculer.

6.3 Le passage aux cas multi-classes

Les expériences qui ont été présentées ne comportaient que deux classes. Que se passe-t-il quand il existe un nombre supérieur de classes ? Il existe des algorithmes de SVMs qui permettent de gérer ce cas avec une stratégie du « one-vs-all » où l'on simplifie le problème à une problème de classification à deux classes : la classe considérée et le reste. On passe alors en revue chaque classe et on établit une fonction séparatrice qui lui est propre.
Une autre stratégie est le « on-vs-one » où, ici encore, on considère un sous-problème à deux classes. Une fonction séparatrice va être défini pour une classe contre une autre puis tour à tour, une classe va changer, générant ainsi autant de fonctions séparatrices que de paires de classes.
Cela peut avoir un impact sur les objectifs de l'attaquant. En effet, dans le cas « one-vs-one », il peut définir une classe attaquante et une classe attaquée. Alors que dans une stratégie de « one-vs-all » il est plus difficile de connaitre l'impact d'une attaque sur une classe en particulière.
De même, il peut y avoir un problème de définition sur les notions de classes attaquantes et de classes attaquées. Dans les deux stratégies, rien n'empêche l'attaquant d'utiliser plusieurs classes pour en attaquer une (ou plusieurs) autre(s).

6.4 L'utilisation d'ensemble d'algorithmes de classification

Comme nous l'avons vu dans les résultats précédents, les noyaux RBF que nous avons utilisés, même si leurs paramètres n'étaient pas ajustés de manière optimale, semblaient avoir plus de possibilités à se mouvoir pour éviter de mal-classer les points d'attaques que les noyaux linéaires. Ce qui laisse entendre que les noyaux à bases radiales peuvent prendre en compte des distributions plus complexes. Dans notre cas, où l'on supposait connaitre parfaitement la distribution des données, nous avons remarqué que si l'utilisateur ne maitrise pas la méthode de réapprentissage de ces noyaux, cela générait des

erreurs (exemple avec avec les bornes limites utilisées). De plus en plus de techniques de défense se basent, à la manière de la technique de boosting, sur non pas un seul algorithme de prise de décision mais sur un ensemble d'algorithmes ayant des fonctions séparatrices plus simples. Chaque algorithme peut ne prendre en compte qu'une partie de l'ensemble d'apprentissage et avoir une fonction séparatrice avec un noyau linéaire par exemple. Ensuite, lorsqu'un nouveau point doit être classé, chacun des algorithmes donne sa réponse en fonction de ses connaissances et l'ensemble des classes donne typiquement lieu à un vote final pour savoir quelle classe ce système va attribuer à cette nouvelle donnée. Il a été montré dans différents travaux (comme dans [**?**]) que ce type de système de prise de décision avait l'air de bien se comporter lorsque l'hypothèse de non-hostilité n'était pas levée, mais qu'en est-il dans les conditions dans lesquelles nous nous sommes positionnés ?

Conclusion

Ce travail s'est donc porté sur la sécurité des processus d'apprentissage artificiel et notamment sur les algorithmes de Machines à Vecteurs Supports (SVMs).

Ce travail s'est appuyé sur l'étude de l'article [4] qui propose de venir perturber l'hyperplan séparateur qu'un SVM a construit pour séparer les données d'un ensemble d'apprentissage. Pour arriver à le perturber, les auteurs ont mis au point une technique qui tente de placer des points dans l'espace de description de telle sorte que l'influence de ces points d'attaque sur l'hyperplan séparateur soit maximale.
L'article propose de choisir ces points corrompus dans l'ensemble d'apprentissage. Un point d'attaque est alors un point de l'ensemble d'apprentissage qui est copié et dont le label a été modifié. Après l'ajout de ce nouveau point dans l'ensemble d'apprentissage, pour être pris en compte par l'algorithme, il est déplacé vers une zone de forte influence. Ce déplacement est basé sur une ascension de gradient d'une fonction représentant l'influence qu'à un point à un certain endroit de l'espace de description sur l'hyperplan séparateur.

Nous avons d'abord mis en avant quelques limites de l'étude menée dans ce papier. Puis nous avons défini, à partir de ces limites, des éléments d'étude à approfondir. Nous nous sommes portés dans un premier temps sur la définition de bons candidats dans l'ensemble d'apprentissage pour devenir des points d'attaques. Les points d'attaque créés à partir de points de l'ensemble d'apprentissage proches d'une zone de forte influence ont plus de chances d'arriver à cette zone que d'autre points plus éloignés qui risquent d' être piégés dans des maxima locaux.
Ensuite nous avons étudié si le nombre de points présents dans l'ensemble d'apprentissage avait un impact sur le comportement des points d'attaques. D'après nos résultats, l'observation faite dans l'étude précédente est aussi valable quand on augmente ou diminue le nombre de points présents dans l'ensemble d'apprentissage.
Puis, alors que l'article propose de créer et de déplacer uniquement un point d'attaque, nous nous sommes posés la question de savoir si un seul point d'attaque était suffisant. A priori non, mais dans ce cas, comment choisir les différents points d'attaque ? 3 stratégies ont alors été présentées. Finalement, il a été observé que le choix du noyau des SVMs définit la stratégie à utiliser.
Enfin, nous avons souhaité comprendre si la symétrie spatiale des distributions pouvaient avoir un impact sur le déroulement d'une attaque ainsi que la présence d'un déséquilibre dans la représentation d'une classe par rapport à une autre. La déséquilibre semble avoir un fort impact dans le sens où il vaut mieux toujours attaquer la classe la moins représentée. L'autre problème nous a montré, qu'apparemment, les classes ayant une forte densité de points sont plus difficiles à attaquer que les autres.

La dernière partie du document ouvre l'étude à des questions que nous n'avons pas eu le temps de traiter.

Bibliographie

[1] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure ? In *Proceedings of the ACM Symposium on Information, computer and communications security*, 2006.

[2] Marco Barreno, Peter L. Bartlett, Fuching Jack Chi, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, Udam Saini, and J. D. Tygar. Open problems in the security of learning. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, 2008.

[3] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *In Machine Learning Journal*, 2008.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the* 29^{th} *International Conference on Machine Learning*, 2012.

[5] Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems*, 2000.

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection : A survey. *ACM Computing Surveys*, 2009.

[7] Mark Dredze, Reuven Gevaryahu, and Ari Elias-bachrach. Learning Fast Classifiers for Image Spam. In *Conference on Email and Anti-Spam*, 2007.

[8] Honorius Gâlmeanu and Răzvan Andonie. Implementation issues of an incremental and decremental svm. In *Proceedings of the 18th international conference on Artificial Neural Networks, Part I*, 2008.

[9] Carrie Gates and Carol Taylor. Challenging the anomaly detection paradigm : a provocative discussion. In *Proceedings of the 15 th New Security Paradigms Workshop*, 2006.

[10] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011.

[11] Thomas Samuel Kuhn. *La structure des révolutions scientifiques*. 1983.

[12] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection : supervised or unsupervised ? In *Proceedings of 13th ICIAP conference*, 2005.

[13] Kwok Ho Law and Lam For Kwok. Ids false alarm filtering using knn classifier. In *WISA*, 2004.

[14] N. Oza and K. Tumer. Classifier ensembles : Select real-world applications. *Information Fusion*, 2008.

[15] Konrad Rieck. Computer security and machine learning : Worst enemies or best friends ? In *Proceedings of the 1st SysSec Workshop*, 2011.