Rapport de stage

Guillaume Laurens

Utilisation de bornes théoriques pour la sélection de modèles en apprentissage supervisé

SOMMAIRE

1. PRESENTATION DU STAGE

A. INTRODUCTION

Au cours de l'année 2004-2005, j'ai réalisé ma première année de master au sein de l'Institut Universitaire Professionnalisé en Génie Mathématiques et Informatique mention Systèmes Intelligents à l'université Paul Sabatier de Toulouse.

Dans le cadre de ma formation, un stage de fin d'année de quatre mois est nécessaire. J'ai décidé de réaliser mon stage à l'étranger pour vivre, en plus d'une expérience professionnelle, une nouvelle expérience personnelle. C'est à l'université Laval de Québec que j'ai trouvé un sujet intéressant en intelligence artificielle. Ainsi j'ai pu découvrir le milieu de la recherche lors de mon stage qui s'est déroulé au Québec du 4 Avril au 19 Août 2005.

B. L'UNIVERSITE ET MON GROUPE DE TRAVAIL

L'Université Laval se trouve au centre de Québec et compte parmi les grandes universités canadiennes. En 2004, elle accueillait plus de 38 000 étudiants dans tous les domaines d'études. En matière de recherche, elle fait partie des dix plus importantes universités canadiennes.

Durant cinq mois, j'ai travaillé avec mon maître de stage : François Laviolette, enseignant chercheur à l'Université Laval en mathématiques pour l'informatique, théorie des graphes, vérification automatisée et apprentissage par renforcement. J'ai également travaillé avec Mario Marchand enseignant chercheur, Philipe Choquette étudiant en maîtrise, et Stéphane Simard qui a fini son stage un mois après le début du mien. Nos travaux portent sur l'apprentissage supervisé qui est une branche de l'intelligence artificielle.

C. MON SUJET DE STAGE

L'apprentissage automatique est l'une des branches de l'intelligence artificielle qui prend de plus en plus d'importance au niveau de la recherche tant ses domaines d'applications dans notre société sont nombreux et diversifiés. En effet, ce problème s'applique dans de nombreux domaines au-delà de l'informatique tels que le diagnostic médical en fonction des symptômes ou de l'expression des gènes, la reconnaissance de formes, etc.

L'intitulé de mon sujet est « Utilisation de bornes théoriques pour la sélection de modèles en apprentissage supervisé ». Nous travaillons sur un algorithme d'apprentissage supervisé inventé et implémenté par Mario Marchand en 2002 : les Set Covering Machines. La mise en œuvre la plus efficace connue à ce jour est une sélection de modèles par une borne théorique.

Mon travail consiste dans un premier temps à valider l'utilisation d'une borne dans cet algorithme par des tests rigoureux. Cette étape a pour second objectif de m'initier à cet

algorithme et aux différentes notions utilisées dans ce projet. Dans un second temps, je dois reprendre l'étude de l'approche PAC bayes suivant l'avancement du travail de Stéphane Simard à la fin de son stage. Finalement, la plus importante partie de mon stage porte sur l'étude des Reduced Coulomb Energy. J'ai implémenté et analysé cette nouvelle approche qui présente des résultats très intéressants.

Étant donné l'aspect très théorique et technique de mon sujet de stage et suite à la demande de mon maître de stage, certaines parties de ce rapport rentrent dans des détails qui sont destinés aux personnes qui continuent le projet. Cependant toutes les notions sont reprises du début pour que les études présentées soient accessibles et intéressantes pour toute autre personne.

Dans la première partie du rapport, je présente toutes les notions théoriques nécessaires à la compréhension du rapport. Par la suite je développe les différentes études que nous avons menées pendant mon stage. Finalement, le rapport conclut sur l'aboutissement de nos recherches, les nombreuses possibilités de poursuites et ce que m'a apporté ce stage (sur le plan personnel et professionnel ainsi que les connaissances acquises).

2. PRISE EN MAIN

Le coté théorique de mon sujet le rend un peu difficile d'accès pour un public non spécialisé. Ce chapitre a donc pour objectif de décrire simplement les différentes notions qui seront manipulées dans la suite du rapport. Pour ce faire, ces différentes notions sont définies puis explicitées par des exemples simples en deux dimensions permettant de plus vite comprendre les idées proposées.

A. IMAGINONS

Imaginons des données simplifiées sur la grippe. Pour chaque patient, on note deux informations : des données continues concernant deux symptômes de la grippe (ces données seront par la suite nommées attributs). On dispose de 100 patients pour lesquels on a relevé ces deux informations et dont on a appris par la suite si ils avaient eu la grippe ou non :

	Donnée 1	Donnée 2	grippe
Patient 1 :	15	30	oui
Patient 2 :	3	20	non
Patient 3 :	30	4	oui
Patient 4 :	4	9	oui
…			
Patient 100 :	15	16	non

Maintenant de nouveaux patients viennent nous voir. On note les deux informations concernant leurs symptômes. On désire leur prédire si ils vont tomber malade dans les jours à venir ou si ils peuvent aller skier sans craintes… Pour ce faire, on observe les cents premiers

patients (dont la classe est connue) pour apprendre à classifier les nouveaux patients. Ici il y a deux classes : les gens qui vont avoir la grippe et ceux qui ne l'auront pas.

Ce problème soulève la question suivante : comment un algorithme peut il apprendre à effectuer une tache en observant des données ?

On parle d'apprentissage automatique lorsqu'un algorithme apprend à effectuer une tache en étudiant des données. L'apprentissage automatique dédié à la classification est plus spécifiquement appelé apprentissage supervisé.

B. QU'EST-CE QUE LA CLASSIFICATION ?

La **classification** consiste à envisager une façon de séparer des ensembles d'exemples (appelés classes). Une **classe** est constituée d'une certaine quantité d'exemples. La classe est en général une caractéristique commune aux exemples qu'elle contient (avoir la grippe, ne pas avoir la grippe, etc.). Les **attributs** sont des caractéristiques (numériques) qui définissent les exemples. Un **exemple** est défini avec une valeur par attribut et la classe à laquelle il appartient. L'objectif d'un **classificateur** est donc de différentier (classifier) des exemples en fonction de leurs attributs. On cherche à retrouver leurs classes.

Il parait évident que pour chaque problème, on doit utiliser un classificateur différent. On est obligé d'observer des exemples déjà classifiés pour construire le classificateur. C'est la **phase d'apprentissage**. Après cette phase, l'algorithme est en mesure de classifier les nouveaux exemples proposés.

Les attributs peuvent être perçus comme des coordonnées qui permettent de positionner les exemples dans un espace qui a pour dimension le nombre d'attributs. Par exemple, avec deux attributs, les exemples sont placés dans un plan. L'algorithme de la phase d'apprentissage a pour objectif de créer un classificateur. Le but est donc de déterminer la zone de l'espace qui correspond à chaque classe.

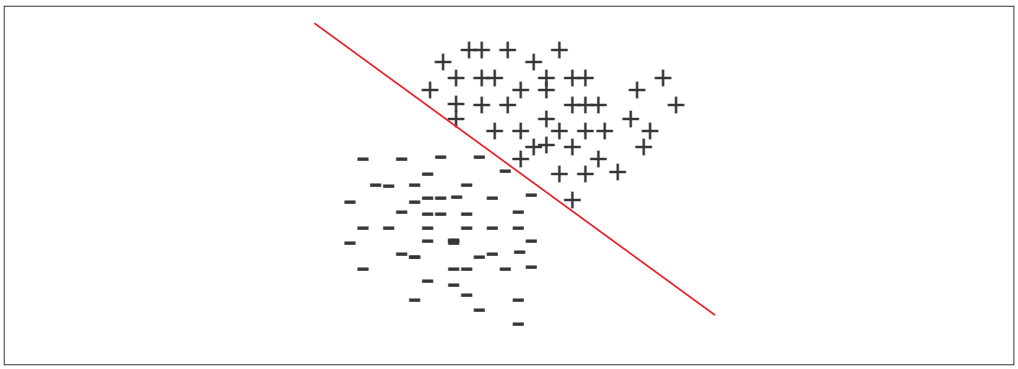

Figure 1 : exemple de classification

Sur la figure 1 on peut voir un exemple simple de classification en deux dimensions (deux attributs) avec deux classes (les exemples + et les exemples -).

Un algorithme de classification se déroule donc en deux étapes : l'apprentissage (dit aussi **train**) et la classification (qui sera souvent une **phase de test** pour évaluer l'efficacité du classificateur obtenu).

On commence par la phase d'apprentissage qui consiste à prendre un certain nombre d'exemples dont on connaît la classe (**training set** ou **échantillon d'apprentissage**). L'objectif de cette étape est d'observer ces exemples déjà classifiés pour générer une fonction, appelée **hypothèse** (c'est le classificateur), capable de classifier de nouveaux exemples (déterminer leurs classes) en fonction de leurs attributs.

Une fois que nous avons construit un classificateur, nous sommes capables de prédire si une personne va avoir la grippe ou non… C'est la deuxième étape de l'algorithme : classifier de nouveaux exemples proposés à l'aide de l'hypothèse préalablement générée. L'enjeu est maintenant de ne pas faire trop d'erreurs dans nos prédictions. L'objectif de nos travaux se situent donc à ce niveau : essayons de trouver et d'étudier un bon classificateur.

Ici nous n'étudierons que des problèmes de classification binaire (il n'y a que deux classes à séparer, par exemple + et -). En effet il est prouvé qu'un algorithme de classification binaire peut être appliqué à des problèmes contenant plusieurs classes (pour N classes, on utilisera N classificateurs binaires).

Il est évident que de tels algorithmes, pour avoir une chance d'être opérationnels, se basent sur l'hypothèse que les exemples sont donnés par des distributions de probabilités. Bien sûr le bruit est naturel et il est à l'origine des erreurs. L'algorithme ne connaît pas la distribution de probabilité utilisée mais en supposant qu'il en existe une, on peut espérer obtenir un bon classificateur si on dispose d'un échantillon d'apprentissage qui représente bien les deux classes.

C. LES SUPPORT VECTOR MACHINES

Il existe différents algorithmes de classification. Par exemple, l'un des algorithmes connus est constitué d'un réseau de neurones. Il y également l'algorithme basique du plus proche voisin. Mais les classificateurs les plus performants actuellement sont les Support Vector Machines. Il est intéressant de comprendre le fonctionnement de cet algorithme pour interpréter ses résultats. De plus il est important de pouvoir comparer nos résultats pour évaluer leur pertinence.

L'algorithme consiste à définir une droite, un plan ou un hyper plan (suivant la dimension de l'espace considéré) qui sépare au mieux les deux classes (c'est un algorithme de classification binaire).

Imaginons un certain nombre de points disposés sur un plan. Chacun appartient à l'une des deux classes (classe + ou – par exemple). Un Support Vector Machine définit la séparation la plus efficace possible entre les deux classes comme le montre la figure 2. Bien sûr, l'algorithme doit parfois accepter de faire des erreurs mais il cherche à les minimiser autant que possible. En effet, certains jeux d'exemples contiennent beaucoup de bruit…

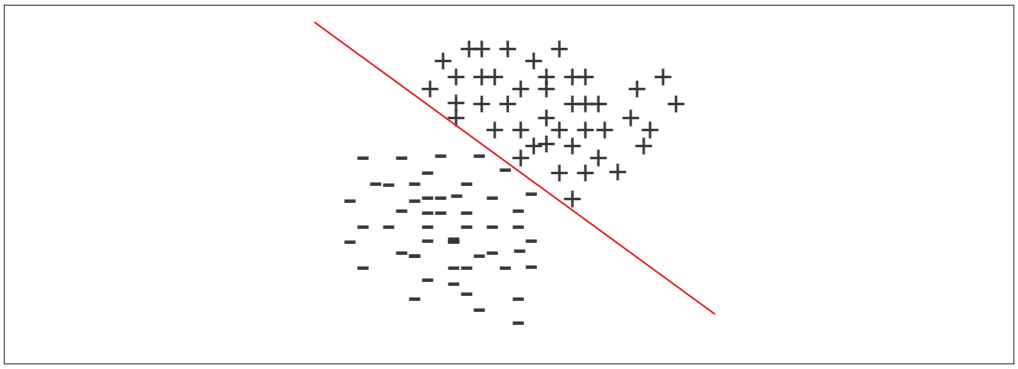

Figure 2 : exemple simple des Support Vector Machines

On peut se demander pourquoi un algorithme aussi simple obtient les meilleurs résultats. En effet, si on regarde en deux dimensions, une droite ne sera pas toujours l'outil idéal pour séparer un plan de façon efficace (une courbe serait mieux appropriée). L'astuce consiste donc à augmenter la dimension pour isoler au mieux les 2 classes (on replace ainsi les exemples dans l'espace pour qu'ils puissent être mieux séparés).

D. Les Set Covering Machines

Les Set Covering Machines ont été inventés en 2002 par Mario Marchand suivant l'idée qu'il est plus facile et plus efficace de couvrir une classe plutôt que d'en séparer deux. Les exemples d'une même classe ayant tendance à être regroupés dans l'espace, on couvre une classe en créant un certain nombre de **boules**[1] avec des **centres** et des **rayons** différents.

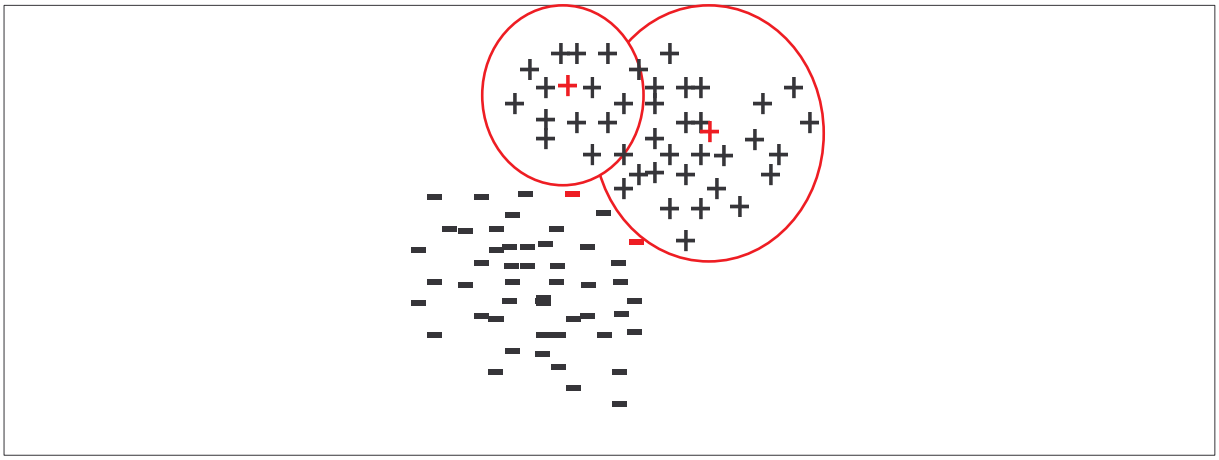

Figure 3 : exemple simple des Set Covering Machines

Nous travaillons sur une version de l'algorithme qui utilise des exemples de l'échantillon d'apprentissage comme centres et bordures pour les boules. Il existe cependant des versions qui permettent de définir des rayons quelconques.

[1] Une boule est un cercle en dimension 2, une sphère en dimension 3 et une hyper sphère au-delà.

Tout comme les Support Vector Machines, c'est un algorithme de classification binaire.

La phase d'apprentissage consiste donc ici à déterminer les boules qui couvrent au mieux l'une des deux classes. La liste des boules obtenue constitue ainsi le classificateur que l'on cherche à construire (appelé hypothèse).

On peut donc classifier de nouveaux exemples dont on ne connaît pas la classe. En plaçant l'exemple dans l'espace, on regarde s'il est dans l'une des boules ou pas. Ainsi s'il est dans au moins une boule, il sera classifié selon la classe couverte par les boules (classe + dans l'exemple de la figure 3). Inversement, s'il n'est dans aucune boule, l'autre classe lui sera attribuée (classe - sur la figure 3).

Beaucoup d'options sont disponibles dans cet algorithme. Nous allons donc étudier les plus importantes. Par exemple, la possibilité de commettre des erreurs : on crée une boule avec beaucoup d'exemples positifs et on se permet éventuellement quelques erreurs en incluant aussi des exemples négatifs. On introduit donc un paramètre de **pénalité** (une pénalité infinie consiste à ne pas faire d'erreurs lors de la création d'une boule). On peut ainsi éviter d'être « gêné » par le bruit.

Il est également possible de créer des **boules creuses** (des trous). C'est-à-dire que les exemples à l'intérieur de la boule n'appartiendront pas à la classe que l'on cherche à couvrir. Ainsi on peut corriger des erreurs précédemment autorisées.

E. LE RISQUE ET LA VALIDATION

Pour travailler sur de tels algorithmes, il est indispensable de définir un critère d'efficacité. Ce critère est bien sûr basé sur le nombre d'erreurs commises lors de la classification de nouveaux exemples. Ainsi lorsque l'on teste l'efficacité d'un algorithme, on prend des données, réelles ou synthétiques dont on connaît la classe, et on coupe nos exemples en deux parties : celle qui va servir pour l'apprentissage et celle qui va servir pour tester le classificateur obtenu. Le test est simple, on regarde si le classificateur classe chaque exemple dans la bonne classe. Dans le cas contraire on considère que c'est une erreur. La figure 4 ci-dessous illustre ce procédé.

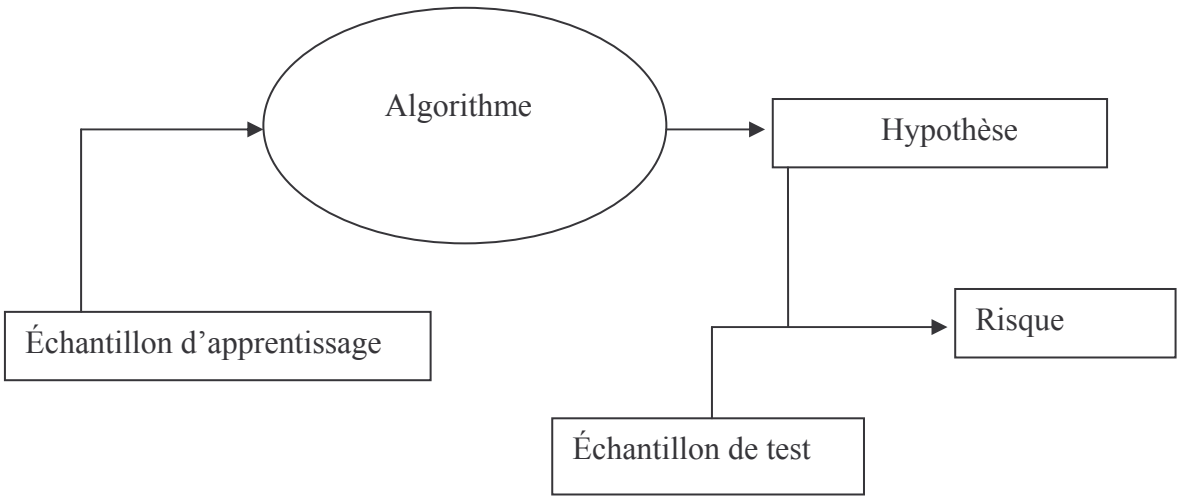

Figure 4 : Description du mode opératoire

i. Risque Empirique et risque réel

Le risque empirique est le nombre d'erreurs commises lors de la phase d'apprentissage sur l'échantillon d'apprentissage. Ce risque est une information intéressante mais ce n'est pas un bon critère pour évaluer l'efficacité du classificateur que l'algorithme propose.

C'est pour cette raison que l'on garde une partie des exemples pour une phase de test. Le pourcentage d'erreurs permet ainsi d'attribuer à l'algorithme une efficacité relative :

$$risque = \frac{100 * nombre\_d'erreurs}{nombre\_d'exemples\_testés}$$

Il faut se méfier des tests réalisés sur un petit nombre d'exemples (moins de 1000). Effectivement, lorsque l'on travaille sur peu d'exemples, le risque obtenu lors des tests n'est pas forcement significatif… On ne peut parler de vrai risque que lorsque le nombre d'exemples pour la phase de test est très grand (plus de 10000) et qu'il est largement supérieur au nombre d'exemples utilisés lors de la phase d'apprentissage.

ii. Les différents tests pour valider un algorithme

La méthode décrite précédemment (apprentissage puis test) est efficace lorsque l'on peut utiliser beaucoup d'exemples (avec des données synthétiques par exemple). Malheureusement, la majorité des résultats qui sont utilisés en apprentissage supervisé pour comparer les différences entre les algorithmes sont des données réelles avec moins de 500 exemples.

Le risque est alors de trop configurer son algorithme pour qu'il s'adapte au jeu de test que l'on utilise (problème plus couramment connu sous le nom d'« overfitting »). Par exemple pour définir un paramètre numérique continu, on décide de trouver le meilleur risque possible. On essaye ainsi plusieurs valeurs lors de l'apprentissage et on garde celle qui donne le meilleur risque sur le jeu de test. Mais on a utilisé plusieurs fois le jeu de test. Ce n'est donc plus un vrai jeu de test mais une partie des exemples ayant servi à la phase d'apprentissage. Le risque obtenu n'est pas significatif.

Lorsque ce problème se présente, on peut y remédier en utilisant la validation croisée. Cette méthode consiste à déterminer des paramètres de l'algorithme par une succession d'apprentissages et de tests sur des sous parties de l'échantillon d'apprentissage. Ainsi, on partage ce dernier en plusieurs parties (en cinq par exemple). Ensuite on réalise l'apprentissage sur 4/5 des exemples puis on teste sur le cinquième restant. On peut ainsi réaliser cette opération autant de fois qu'on le désire en permutant à chaque fois les sous parties (les cinquièmes dans mon exemple). Une fois que l'on a déterminé les valeurs de nos paramètres, on utilise tout l'échantillon d'apprentissage pour créer le classificateur final et on utilise enfin les exemples du jeu de test pour déterminer le risque.

Il faut savoir que certains résultats fournis dans ce domaine utilisent une variation de la validation croisée. En effet, les exemples ne sont pas séparés en deux parties (apprentissage et test) et on effectue une validation croisée sur l'ensemble des exemples. Il y a autant de sous parties que de cycles d'apprentissage avec test. On calcule à chaque fois un risque, et le risque final est donné par la moyenne de tous les risques obtenus lors de la validation croisée. Il n'y

a pas d'« overfitting » étant donné que chaque exemple n'est utilisé qu'une seule fois pour une phase de test. Cependant, les résultats sont améliorés (donc moins faciles à comparer avec les autres) car les phases d'apprentissages disposent de plus d'exemples qu'une validation classique (une moitié pour l'apprentissage et une autre pour le test). C'est pour cette raison que nous n'utilisons pas cette méthode pour présenter nos résultats.

F. LES BORNES THEORIQUES

Les Set Covering Machines sont basés sur l'idée suivante : si on peut trouver un classificateur simple qui fait peu d'erreurs lors de l'apprentissage, alors il est probable qu'il fasse peu d'erreurs sur les futurs exemples à classifier. Cette idée simple s'appuie sur des garanties théoriques qui nous donnent une borne supérieure du taux d'erreur à venir (soit le risque réel).

En plus de donner une garantie théorique d'efficacité, une telle borne sur l'erreur de généralisation est dans certaines situations tellement bien reliée à la réalité qu'elle peut être utilisée pour sélectionner le classificateur le plus performant. Une telle sélection (nommé sélection par la borne) consiste à retenir au cours du processus d'apprentissage, le classificateur qui aura la meilleur borne supérieure théorique. Cette borne permet ainsi de sélectionner un bon classificateur parmi tous les Set Covering Machines possibles.

La borne que nous utilisons est la borne de Marchand-Sokolova. Elle utilise le risque empirique, un critère de simplicité du classificateur (qui est le nombre de boules) et la compression de données.

Un algorithme de compression de données fournit un résultat en fonction d'un nombre limité d'exemples de la phase d'apprentissage (on appelle ces exemples l'ensemble de compression). Les Support Vector Machines et les Set Covering Machines sont des algorithmes de compression de données mais les Set Covering Machines ont pour objectif de l'optimiser en minimisant la taille de l'ensemble de compression en prenant ce paramètre en compte dans la définition de la borne.

Il existe également la borne de Langford qui varie un peu de la borne de Marchand. Dans la suite du rapport une petite étude comparative est menée entre les deux bornes.

3. DEBUT DU STAGE

La première étape de mon stage a été la mise à niveau théorique dans le domaine de l'apprentissage supervisé. J'ai donc lu le cours de Mario Marchand sur l'apprentissage automatique (« Machine Learning »). J'ai ainsi découvert les différentes notions que nous avons utilisées tout au long du stage. Ce cours d'intelligence artificielle utilise plusieurs démonstrations mathématiques qui m'ont poussé à revoir plusieurs bases en statistiques.

La première partie de mon stage a été l'étude des Set Covering Machines. Il se pose en effet plusieurs questions intéressantes au départ de ce projet :

- Cet algorithme est-il capable d'apporter de bons résultats ? Faire aussi peu d'erreurs que les Support Vector Machines par exemple ?
- La borne Marchand-Sokolova est-elle assez pertinente pour nous permettre de choisir un bon classificateur ? (sélection de modèles basée sur une borne théorique)

En effet, il y a trop de Set Covering Machines pour tous les tester. Un Set Covering Machine est une combinaison de boules ; donc plus il y a de boules possibles, plus on a de combinaisons pour générer différents classificateurs. Bien sûr, il y a énormément de boules possibles : autant de centres que d'exemples et pour chaque centre autant de bordures possibles que d'exemples.

A. L'ALGORITHME EXISTANT

Lors de mon arrivée au sein du groupe, j'ai découvert l'avancement des travaux. Un premier programme de Mario Marchand en C++ est à l'origine des Set Covering Machines. Ce programme d'environ 7000 lignes propose un algorithme avec une multitude d'options à définir dans un fichier texte. Le programme s'exécute en ligne de commande. Il faut principalement lui fournir un fichier d'apprentissage, un fichier de test et un fichier de configuration.

Voici une version très simplifiée de l'algorithme :

Tant que tous les exemples de la classe à couvrir ne sont pas couverts,
- Trouver la meilleure boule possible (couvrant le plus grand nombre d'exemples non couverts en faisant le moins d'erreurs possibles)
- Conserver cette boule dans la pile qui conserve les boules retenues

Fin de Tant que.
Tant que la pile n'est pas vide,
- Calculer la borne de Marchand et retenir le résultat
- Enlever la dernière boule de la pile

Fin de Tant que.
- Remettre toutes les boules dans la pile.
- Trouver la meilleure borne obtenue, repérer le nombre de boules correspondant à cette borne et tronquer les boules restantes.

Le classificateur obtenu est donc le résultat de la sélection d'un modèle parmi plusieurs disponibles et cela en fonction d'une borne théorique. L'objectif de mon stage est donc de poursuivre le travail de Stéphane Simard sur l'étude de cet algorithme. Les Set Covering Machines ayant apportés de bons résultats, il est prometteur de chercher à les étudier et les améliorer.

B. L'UTILISATION DU R

Au début de mon stage, j'ai découvert le logiciel R qui permet d'écrire des scripts pour différentes analyses statistiques. R est un logiciel libre sous licence GNU GPL disponible pour Unix, Linux, MacOS et Microsoft Windows (http://www.r-project.org).

Pendant son stage, Stéphane Simard a intégré le programme à l'environnement R par l'intermédiaire d'un « package ». Il a pu ainsi produire des scripts pour générer des données synthétiques variées. Ainsi il est possible d'exécuter l'algorithme automatiquement un grand nombre de fois et d'afficher des résultats sous forme de graphiques pour mieux les interpréter. Cet outil a donc permis de comparer la borne de Marchand et la borne de Langford. Il a aussi permis de vérifier la corrélation entre la borne et le risque. En effet lorsque l'on applique l'algorithme précédemment décrit, il est important que la courbe du risque et celle de la borne aient la même allure (courbe selon plusieurs classificateurs). Sans cela, la fin de l'algorithme, qui se fit à la borne, n'aurait pas de sens.

J'ai donc compris et appris à manipuler le programme en effectuant des tests. J'avais ensuite pour objectif de mettre en évidence des cas de bon fonctionnement mais aussi de dysfonctionnement des bornes (Marchand et Langford). Pour ce faire, j'ai commencé par étudier les possibilités offertes par le package R et les différents scripts déjà écrits.

J'ai beaucoup utilisé ces scripts puis j'en ai écrit de nouveaux pour réaliser des tests différents. Je me suis servi de l'algorithme existant pour générer des données synthétiques. Celui-ci était au point, utilisant des distributions de probabilités centrées en différents points de l'espace considéré. J'ai cependant était surpris de voir que l'algorithme utilisé ne permette pas d'obtenir des données avec du bruit (malgré l'utilisation de distributions de probabilités qui produisent par elles même du bruit). Ma première réaction a donc été de modifier ce problème pour travailler sur des données plus intéressantes.

C. LES RESULTATS

En utilisant les scripts avec différentes options, j'ai obtenu des cas d'études intéressants. J'ai mis en évidence des cas où les deux bornes se trompent (le minimum de la borne n'indique pas le classificateur qui a le meilleur risque), des cas où la borne de Marchand est plus efficace, des cas où la borne de Langford est la meilleure et des cas où les deux sont performantes.

Jusqu'à présent, beaucoup de tests ont été fait en créant des classificateurs obtenus à l'aide de l'algorithme proposé précédemment. J'ai eu l'idée d'étudier plus en détail cet algorithme. La première boucle définie une liste de boules qui couvre l'intégralité des exemples de la classe à couvrir. La fin de l'algorithme consiste à retirer les dernières boules de la liste suivant le calcul de la borne. J'ai donc eu l'idée de réaliser les mêmes tests que ceux déjà fait en gardant tous les classificateurs possibles à la fin de la première boucle. Voici l'algorithme :

Tant que tous les exemples de la classe à couvrir ne sont pas couverts,
- Trouver la meilleure boule possible (couvrant le plus grand nombre d'exemples non couverts en faisant le moins d'erreurs possibles)
- Conserver cette boule dans la pile qui conserve les boules retenues pour pouvoir créer les classificateurs suivants
- Créer un classificateur avec pour hypothèse l'état actuel de la pile

Fin de Tant que.

Chaque nouveau classificateur est donc le même que celui précédemment obtenu mais avec une boule de plus dans l'hypothèse. J'obtiens donc autant de classificateurs que le dernier classificateur contient de boules. Pour chaque classificateur je détermine les deux bornes, le risque empirique et le risque réel sur un échantillon de test.

La figure ci-dessous montre différents résultats obtenus. La courbe bleu clair (en bas) représente le risque empirique. Celle juste au dessus (bleu) est le risque réel. La courbe tout verte (en haut) représente la borne de Marchand tandis que la courbe juste au dessous (rouge) est la borne de Langford. Comme nous le savions déjà, cette dernière est plus « serrée » que la borne de Marchand (plus basse). Cependant on observe dans beaucoup de cas que son allure ne suit pas bien la courbe du risque là où la borne de Marchand reste un bon indicateur.

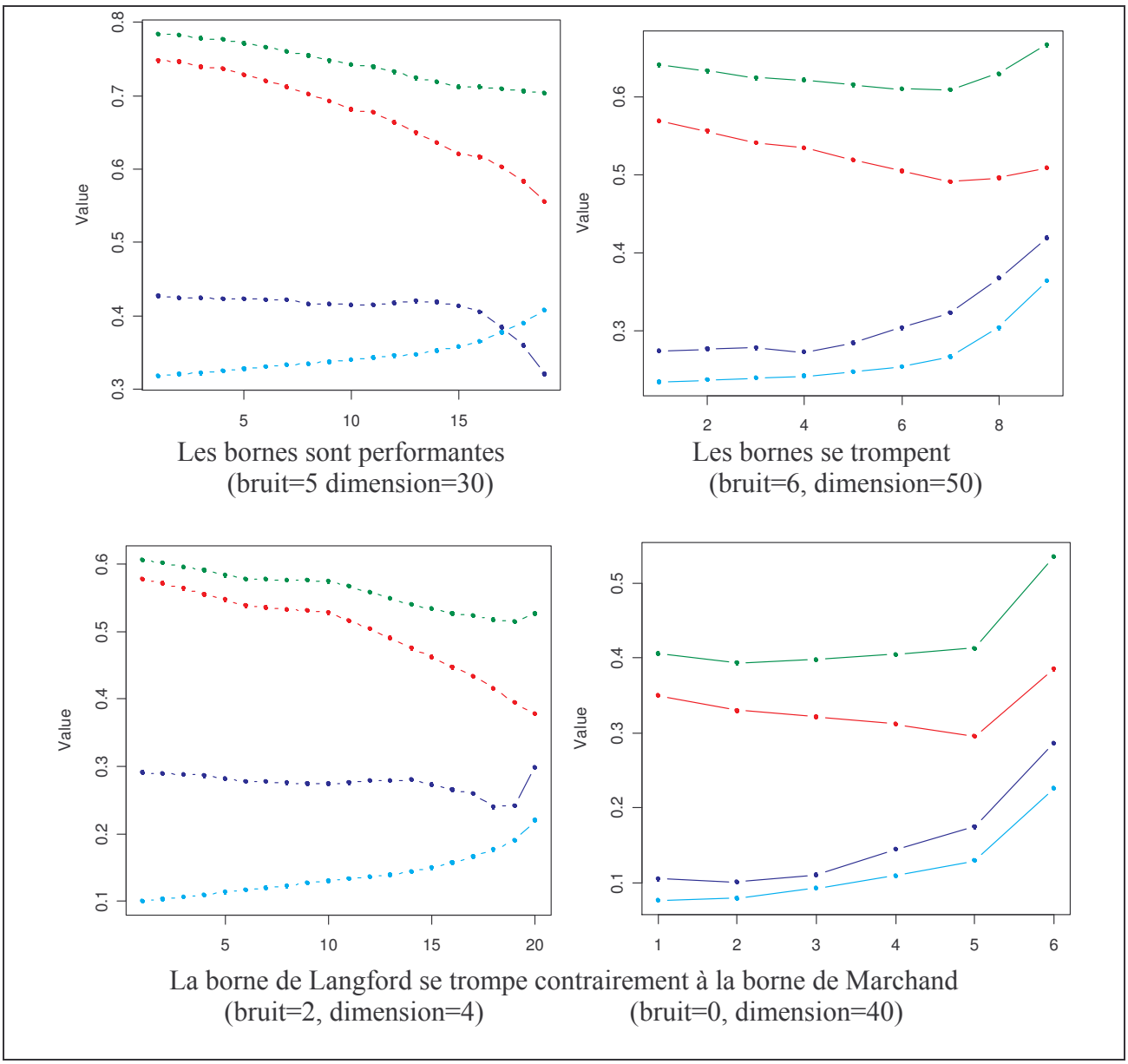

Figure 5 : Cas d'études des Set Covering Machines

À la fin de cette première étude, il semble que la corrélation entre la borne et le risque soit suffisante pour justifier l'efficacité de l'algorithme. Cependant, plusieurs cas de dysfonctionnement mis en évidence me poussent à penser que la borne est efficace mais pas

toujours suffisante. La réflexion sur ce sujet s'est prolongé au delà de cette étude et de nouveaux résultats intéressants sont présentés dans les chapitres suivants.

Au cours de mon utilisation du R, j'ai rencontré un certain nombre de limitations algorithmiques. De plus l'utilisation de scripts ralentit considérablement l'exécution des taches demandées en comparaison à l'utilisation d'un fichier binaire. Stéphane Simard m'ayant confirmé ces deux problèmes, qu'il avait également rencontré, nous sommes revenus à une implémentation en C++ pour la suite du projet tout en conservant le R pour tracer les graphiques de nos résultats.

4. PAC BAYES

L'une des premières idées pour améliorer les résultats des Set Covering Machines a été de développer une approche PAC Bayes. Pour implémenter cette idée, le stagiaire Stéphane Simard a utilisé un article publié par François Laviolette et Mario Marchand sur ce thème. Pour reprendre la fin de cette étude, j'ai eu également à lire cet article.

A. LA THEORIE

Envisageons un certain nombre de bons classificateurs obtenus par un algorithme. On peut se fier à une borne et sélectionner l'un de ces classificateurs mais on prend le risque de tomber dans un cas où celui-ci ne sera pas forcement le meilleur car la borne se trompe.

L'idée développée ici consiste à conserver beaucoup de bons classificateurs et de donner à chacun un droit de vote. On obtient ainsi un vote de majorité. Chaque Set Covering Machine a un droit de vote pondéré par une probabilité qui représente la confiance qu'on lui accorde. Cette confiance est déterminée en fonction de la borne de Marchand.

La méthode proposée comporte beaucoup de petits réglages à définir. Il y a notamment un paramètre appelé température, qui est une variable numérique continue difficile à fixer et qui varie d'un jeu de données à un autre. Il existe une borne théorique pour déterminer ce paramètre : la borne de Gibbs. Les résultats obtenus ont été difficiles à interpréter étant donné la superposition de deux bornes.

L'algorithme se partage donc en trois parties : la création d'un grand nombre de bons classificateurs avec un jeu d'apprentissage ; l'attribution d'un coefficient de confiance à chaque classificateur ; et le vote de majorité sur un jeu de test.

B. LE TRAVAIL DE STEPHANE SIMARD

Stéphane Simard a repris le programme en C++ de Mario Marchand de 7000 lignes ayant été retouché par Philipe Choquette, un étudiant en maîtrise. La seule modification apportée étant le découpage du code en une vingtaine de fichiers, séparant les différentes classes pour une meilleure reprise du programme.

La construction des Set Covering Machines peut être vue comme un arbre. À chaque niveau de l'arbre, une liste de boules propose tous les choix possibles pour la $N^{ième}$ boule du classificateur. Au premier niveau de l'arbre on trouve donc tous les Set Covering Machines à une boule. Au deuxième niveau, pour chaque classificateur du premier niveau, il y une liste de boules qui peuvent devenir la deuxième boule du classificateur. Le deuxième niveau de l'arbre représente toutes les deuxièmes boules possibles et donc tous les classificateurs à 2 boules. Et ainsi de suite, on rajoute autant de niveaux dans l'arbre que de boules à notre hypothèse.

Évidement, cet arbre ne peut pas être exploré entièrement étant donnée sa dimension. C'est pour cette raison que l'algorithme de Mario Marchand utilise une fonction de score qui essaye de couvrir le plus d'exemples de la bonne classe en faisant le moins d'erreurs possibles (suivant la pénalité définie pour une erreur commise).

La première chose à faire pour cette étude est de construire beaucoup de classificateurs pour faire un vote de majorité. Stéphane Simard a eu l'idée de parcourir cet arbre en limitant le nombre de boules à chaque niveau de l'arbre. Ainsi il sélectionne les N meilleures boules (toujours en fonction du score) puis pour chaque boule il sélectionne les N meilleures suivantes et ainsi de suite. Il descend ainsi dans l'arbre jusqu'à un niveau M. Il peut alors former « N à la puissance le niveau » classificateurs à chaque niveau de l'arbre. L'algorithme est limité par ce grandissement exponentiel du nombre de nœuds.

Une fois cet arbre construit, on applique la deuxième boucle de l'algorithme classique de Mario Marchand : pour chaque nœud tout en bas de l'arbre (feuille) on remonte toutes les boules du classificateur pour sélectionner le Set Covering Machine indiqué par la borne de Marchand (on supprime les boules non utilisées en bas de l'arbre). On obtient donc « N à la puissance M » classificateurs basés sur le même algorithme que les Set Covering Machines classiques.

La figure 6, ci-dessous, présente l'arbre de construction des Set Covering Machines. Le premier nœud représente le classificateur vide. Chaque nouveau nœud consiste à l'ajout d'une nouvelle boule proposant ainsi un nouveau classificateur.

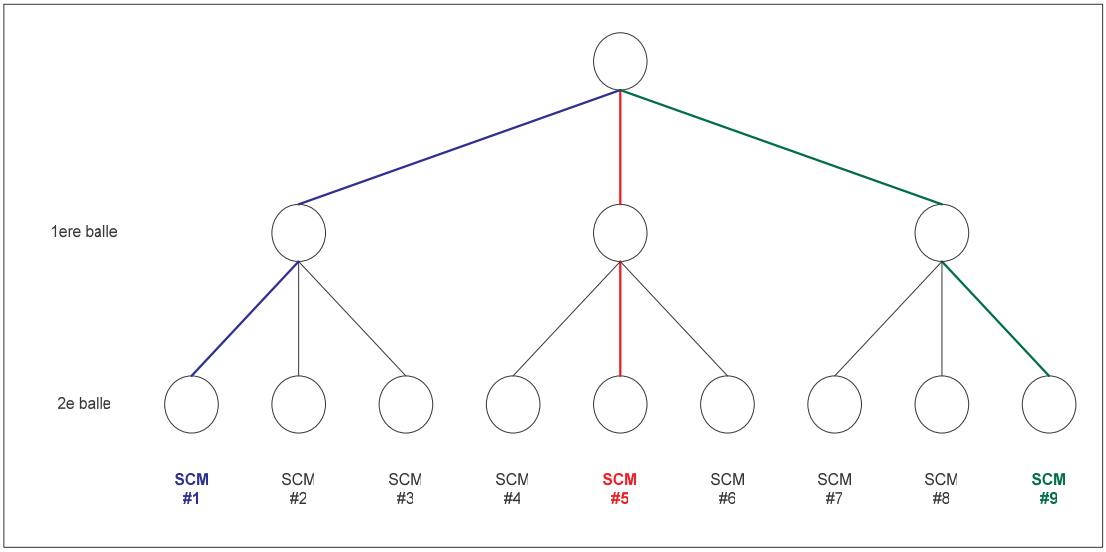

Figure 6 : L'arbre des Set Covering Machines

Après avoir enlevé les doublons, on attribue à chacun un droit de vote en fonction de la borne de Marchand. La classification des nouveaux exemples se fait par vote de majorité : chaque classificateur classe à sa manière, on y applique le droit de vote précédemment déterminé (coefficient de confiance du classificateur) et on regarde quelle classe obtient le plus « de voix ».

Les résultats obtenus montrent l'évolution de la borne de Gibbs et du risque en fonction de la variation du paramètre de température (bêta). En sélectionnant le minimum de la borne, on obtient le plus petit risque possible. La figure 7, ci-dessous montre un cas d'étude de cette borne. La courbe du haut (en rouge) est la borne de Gibbs tandis que les deux courbes du bas représentent le risque obtenu (vert) après le vote de majorité (la courbe bleu représente le risque sur le vote).

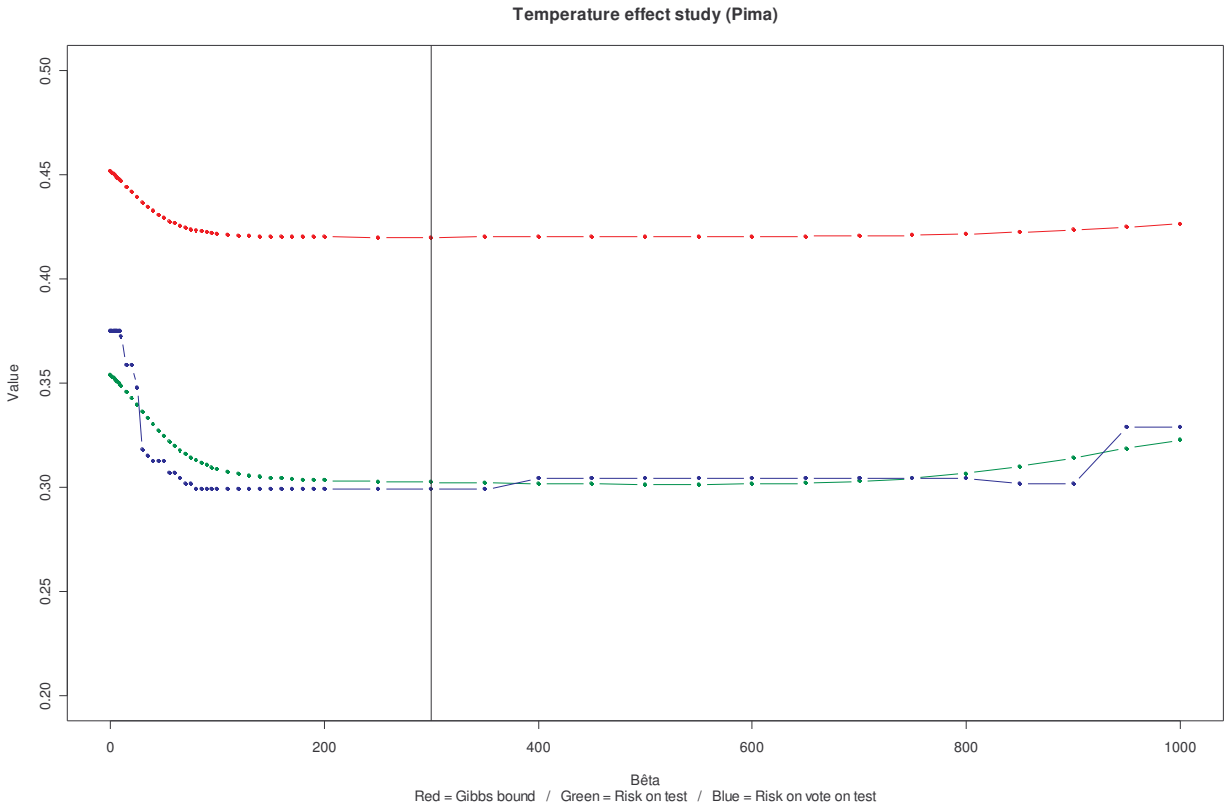

Figure 7 : Effet de la température dans l'étude PAC Bayes

C. MON TRAVAIL

Stéphane Simard a implémenté l'approche PAC Bayes avant la fin de son stage sur la base du programme des Set Covering Machines de Mario Marchand. Pendant ses dernières semaines de stage, j'avais donc pour objectif de travailler avec lui pour être ensuite capable de reprendre le programme et finir l'étude.

Dans un premier temps, Stéphane m'a proposé de chercher des pertes de mémoire dans son algorithme. Cette mission avait pour principal objectif de me faire découvrir le code. J'ai donc lu le programme et discuté avec lui des différents algorithmes.

La reprise de ce gros programme n'a pas été très facile mais m'a fait rapidement progresser en C++. Certaines fonctions utilisant de l'assembleur, le programme n'était pas forcement portable de Visual C++ 6 à d'autres compilateurs. J'ai cependant compilé le projet sous linux afin d'utiliser un logiciel efficace en matière de détection des fuites de mémoire. Le problème que j'ai trouvé étant plus au niveau de l'algorithme que de l'implémentation, j'en ai parlé avec Stéphane qui a pu le corriger.

Stéphane a fini son stage sur la fin de cette étude. J'ai donc repris son programme pour effectuer les dernières modifications et les tests restants. L'algorithme a apporté des résultats surprenants que nous n'avons pas réussi à expliquer avant son départ. Parfois, la courbe de la borne, pour la section de la température, était totalement opposée à la courbe du risque (comme le montre la figure 8). Ce résultat n'est pas logique car il sous entend que l'on donne le plus de confiance aux classificateurs les moins bons. Après plusieurs vérifications du code, aucun bogue n'a été trouvé malgré plusieurs modifications.

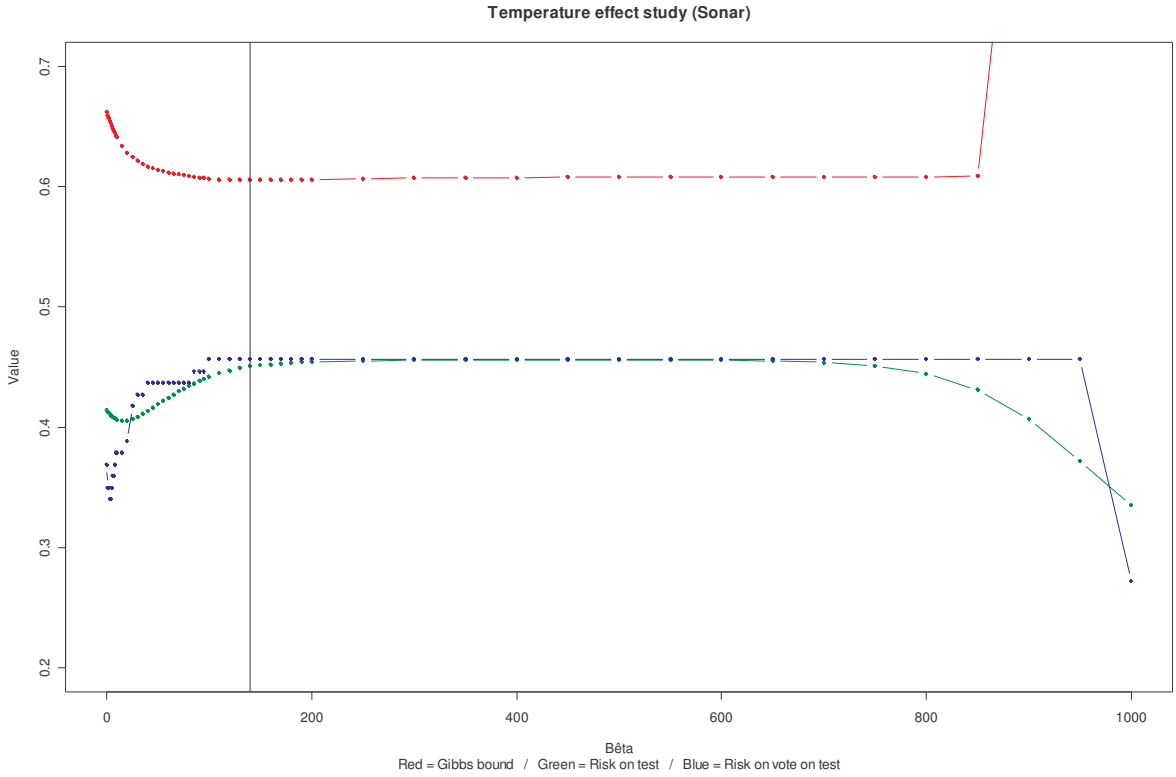

Figure 8 : Dysfonctionnement de la borne de Gibbs

Devant ces résultats que nous n'arrivions pas à interpréter, j'ai décidé de revenir en arrière pour étudier les classificateurs utilisés, au lieu de passer tout de suite au vote de majorité. J'ai donc calculé la borne de Marchand et le risque de chaque classificateur généré par l'algorithme. Puis j'ai trié et affiché ces classificateurs suivant l'ordre décroissant du risque (voir figure 9, en haut la borne de Marchand et en bas le risque obtenu par les classificateurs). J'ai ainsi pu montrer que l'on obtient notre problème lorsque la première

borne (celle de Marchand) se trompe[2]. Nous aurions pu appréhender ce phénomène déjà observé dans l'étude précédente sur les Set Covering Machines classiques. Le résultat est alors logique, si une première borne se trompe et indique les moins bons classificateurs, on leur attribue un droit de vote important à tort, et la deuxième borne (Gibbs) ne peut que se tromper.

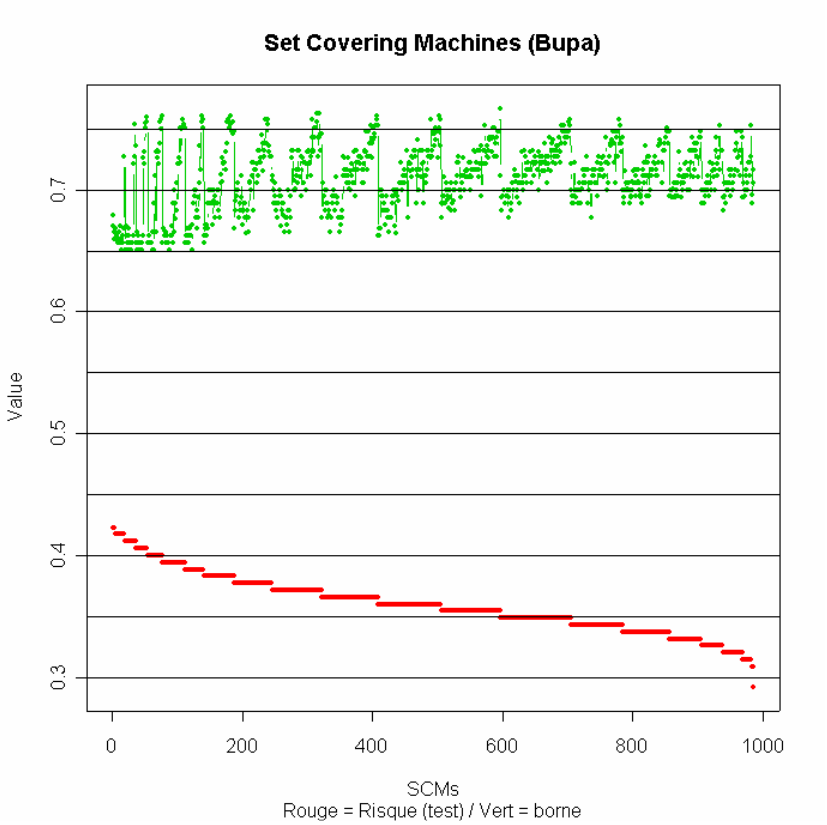

Figure 9 : Dysfonctionnement de la borne de Marchand

On remarque sur cette figure qu'un grand nombre de classificateurs peu performant (à gauche du graphique) ont une meilleure borne que des classificateurs bien plus efficaces (à droite).

Une de mes missions a également été d'utiliser le programme PAC Bayes avec tous les Set Covering Machines à une boule (tous les classificateurs du premier niveau de l'arbre). Cette étude n'a pas apporté de résultats marquants. L'objectif n'était pas d'obtenir un bon risque mais de travailler avec des bons et des mauvais classificateurs pour valider le bon fonctionnement de l'algorithme (borne de Gibbs et attribution des droits de vote).

D. CONCLUSIONS

La théorie PAC Bayes s'est avérée intéressante pour faire une moyenne des résultats d'un lot de bons classificateurs. On assure ainsi un risque stable. Les problèmes que nous

[2] La borne se trompe lorsqu'elle propose la sélection d'un classificateur moins efficace que d'autres qui sont à disposition.

avons rencontrés pour attribuer les droits de vote aux différents classificateurs sont dus aux imperfections de la première borne utilisée.

Je pense qu'il serait intéressant d'envisager une approche moins gourmande qui consisterait à effectuer un vote uniforme. On commencerait par effectuer une sélection très diversifiée de classificateurs mais en imposant qu'ils soient tous très bons. Puis on ferait en sorte qu'ils aient le même droit de vote. Une telle approche nous assure que l'on ne tient pas le meilleur résultat possible. Mais en contre partie, on est sûr de toujours avoir un risque convenable et de ne pas tomber sur des cas particuliers qui inversent la deuxième borne.

5. UNE VARIANTE DE L'ALGORITHME

A. LES OBJECTIFS

À cette étape du projet, nous avons répondu à la question suivante : la borne Marchand-Sokolova est-elle assez pertinente pour nous permettre de choisir un bon classificateur ? Lors de la première étude avec le logiciel R, nous avons dit oui en soulignant cependant des cas où la borne ne donne pas une indication exacte. Ce problème s'est retrouvé sur certains jeux de tests dans l'étude PAC Bayes.

Avec un peu plus d'expérience et de recul sur le projet, j'ai essayé de me faire ma propre opinion sur l'algorithme. L'idée d'utiliser une borne, là où j'aurais utilisé un algorithme déterministe, ne m'a pas rassuré sur la performance des résultats. J'ai donc voulu savoir dans quelle proportion la borne peut se tromper. Et par la même occasion, je cherchais de meilleurs résultats atteignables par les Set Covering Machines.

B. LA REALISATION

J'ai décidé de modifier le programme de Stéphane Simard pour ne plus utiliser la borne et je me suis concentré sur un jeu de données réelles : sonar (ces données ayant toujours été le point noir des Set Covering Machines). Afin de pouvoir construire des classificateurs avec beaucoup de boules, j'ai fixé le nombre de meilleures boules par niveau à 2 (N = 2 dans l'algorithme précédemment décrit). Je peux ainsi descendre jusqu'à 16 niveaux dans l'arbre (soit « 2 exposant 16 » boules). Une fois l'arbre construit, j'ai supprimé la dernière étape de l'algorithme qui consistait à remonter du bas de l'arbre pour sélectionner les meilleurs classificateurs désignés par la borne. Et en revanche, je parcours tous les niveaux pour enregistrer tous les classificateurs possibles (il y en a « 2 à la puissance le niveau » à chaque niveau de l'arbre) : 2 + 4 + 8 + 16 + … + (2 puissance M) classificateurs.

Après avoir enlevé les doublons, je calcule pour chaque classificateur le risque et la borne, puis je trace ces 2 informations pour chaque classificateur, en les classant par ordre décroissant du risque. L'analyse du résultat donne deux informations :

- le dernier classificateur affiché (à droite) est le classificateur qui fait le moins d'erreurs parmi tous ceux que l'on a créé. Il est donc intéressant de regarder quel taux d'erreur il fait (15,5% d'erreurs pour sonar, ce qui est le meilleur résultat des Set Covering Machines) ;

- parmi tous les classificateurs, il est intéressant de repérer celui qui a la plus petite borne pour savoir le résultat qu'aurait fourni l'algorithme classique (17.5% pour sonar, ce qui bon aussi).

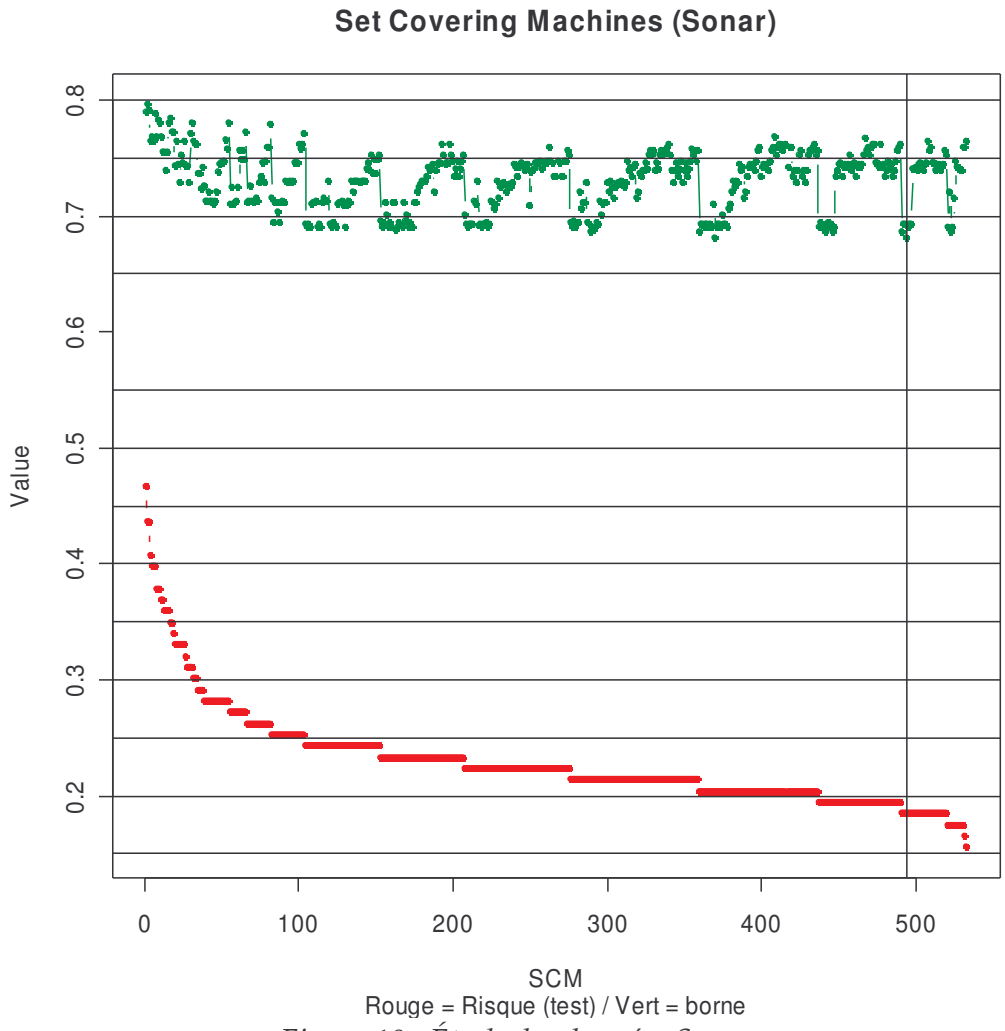

Figure 10 : Étude des données Sonar

Ces résultats sont intéressants : la borne propose des classificateurs performants. Mais entre les bons classificateurs, on observe du bruit sur la borne et celle-ci n'indique pas forcement le meilleur d'entre eux. Je pense que même si la borne joue bien son rôle, elle ne peut pas permettre d'obtenir le Set Covering Machine le plus performant. Il faut noter ici que la borne indique un classificateur performant mais qu'a peut de choses près, elle pourrait indiquer un classificateur bien moins efficace. Ce constat n'est pas forcement rassurant quand à l'utilisation de la borne. Avec tous les jeux de test courants, on remarque par le même procédé qu'il existe des classificateurs plus performants que la borne ne propose pas.

On peut donc dire que les Set Covering Machines sont capables de donner de bons résultats (pour sonar par exemple). Mais on ne sait toujours pas si on a obtenu les meilleurs résultats possibles car on n'a exploré qu'une petite partie de l'arbre.

Seule une méthode complète[3] peut nous assurer de trouver le meilleur classificateur. Il est bien difficile de mettre en œuvre une telle méthode étant donnée la quantité de Set Covering Machines à tester. De plus il est difficile de savoir quel algorithme sera le plus efficace car on sait maintenant que la borne n'offre pas une grande précision entre plusieurs classificateurs performants. L'étudiant Philippe Choquette essaye de résoudre ce problème par l'implémentation d'un « branch and bound ». Cette méthode est bien adaptée au parcours d'un arbre de ce type. L'objectif est de déterminer les branches de l'arbre à ne pas prendre en compte suite à un calcul prouvant que celle-ci ne sera pas intéressante. De plus, on retrouve presque toujours les mêmes boules dans cet arbre. Ainsi, une méthode efficace d'optimisation de graphe a beaucoup de chances d'aboutir. Malheureusement, nous ne connaîtrons pas les résultats de cette longue étude avant la fin de mon stage.

On peut alors se demander : mis à part l'algorithme existant, quels autres algorithmes permettraient d'aller chercher de bons classificateurs ?

Nous avons pensé à une approche différente de ce qui a déjà été vu et qui consiste à faire un « recuit simulé ». On n'utiliserait plus un arbre. On commence par générer un classificateur de départ, puis en changeant certaines boules suivant des critères, que nous n'avons pas encore défini, on essaye d'améliorer le résultat. Mais je crains que cette méthode n'améliore pas beaucoup les résultats si elle se base sur la borne pour affiner le résultat car entre plusieurs bons classificateurs, il y a trop de bruit sur la borne.

Bien sûr, une solution simple consisterait à sélectionner le classificateur par validation croisée en fonction des risques obtenus. Cette méthode apporterait sans doute de bons résultats mais ce n'est pas l'objectif des études menées ici.

L'étude des résultats que j'ai obtenu lors de ces travaux nous a permis de corriger deux erreurs dans l'implémentation de la borne théorique. L'une de ces corrections a énormément amélioré les taux d'erreurs de l'algorithme basé sur la borne (détails : données sonar, options CentersOfSameClass et pénalité infinie). Il est probable que ces corrections améliorent significativement les résultats de l'étude PAC Bayes et de tous les tests réalisés avec l'option CentersOfSameClass. Cette option a pour but d'interdire les boules creuses (dites « trous », voir chapitre de prise en main), ce qui implique un calcul de la borne différent.

6. LES REDUCED COULOMB ENERGY

La plus grosse partie de mon stage a porté sur l'implémentation, l'étude et l'amélioration de l'algorithme des Reduced Coulomb Energy. J'ai commencé ce travail début Juin 2005.

[3] Une méthode dite complète assure d'obtenir le résultat optimal. Ces méthodes ne sont pas toujours envisageables étant donné les limitations matérielles actuelles (calcul et mémoire).

A. L'IDEE

Les Set Covering Machines ont pour objectif de couvrir l'une des deux classes. Tous les exemples que le classificateur couvre appartiennent à cette classe et tous les autres sont pour l'autre classe. On peut faire la critique suivante : le fonctionnement de l'algorithme du point de vue géométrique est totalement asymétrique si on regarde d'une part la classe couverte et d'autre part la classe non couverte par le classificateur. Cela peut poser problème si une grande partie de l'espace n'est pas recouverte par le jeu d'apprentissage (cela se produit forcement avec des jeux d'exemples de grandes dimensions).

Les Reduced Coulomb Energy sont nés de cette réflexion pour donner une variante des Set Covering Machines. L'algorithme proposé cherche ainsi à créer deux listes de boules. Chacune essaye de couvrir l'une des deux classes. De plus, cet algorithme est totalement déterministe, et non pas basé sur une borne théorique.

Cette approche consiste donc à créer deux Set Covering Machines. Chacun a pour objectif de couvrir une classe. On partage ainsi l'espace en 3 zones :

- les zones couvertes par une seule sorte de boules (que nous appelons zones de sécurité ou de sûreté) ;
- les zones couvertes par des boules des deux classes (que nous appelons zones d'intersection) ;
- et les zones couvertes par aucune des deux listes de boules (que nous appelons le « vide »).

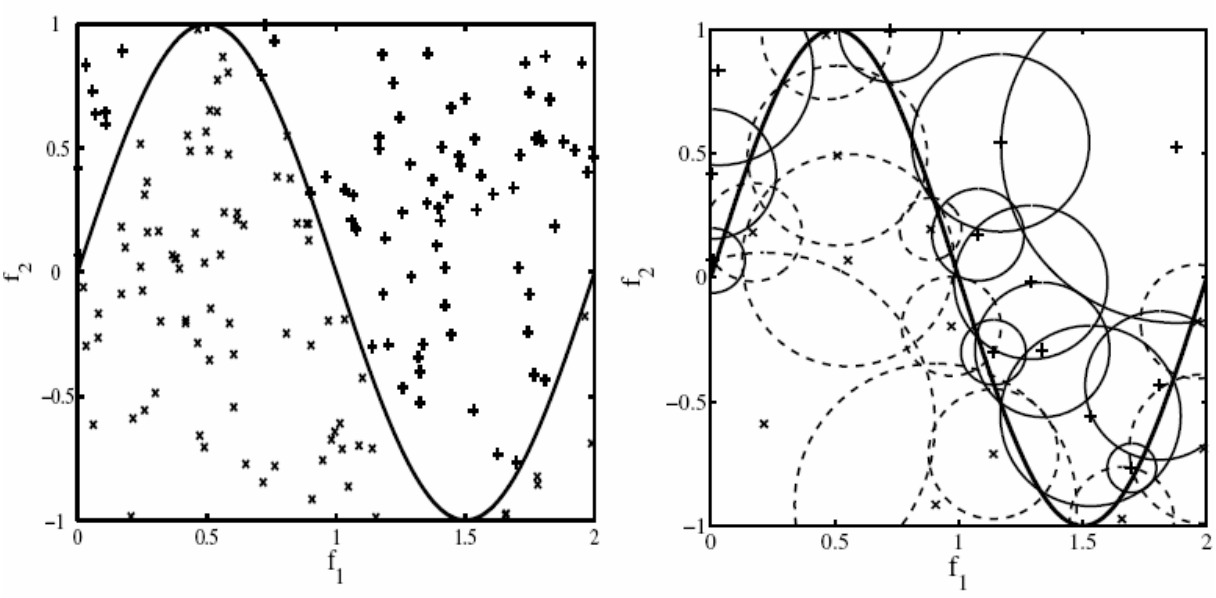

Figure 11 : Reduced Coulomb Energy

La figure 11 ci-dessus montre géométriquement l'idée des Reduced Coulomb Energy dans un espace à deux dimensions.

Cet algorithme a plusieurs avantages intéressants. Tout d'abord, cette capacité de séparer l'espace en plusieurs zones de confiances relatives, permet d'envisager beaucoup d'applications réelles. C'est-à-dire la possibilité de classifier un exemple avec un taux général d'erreur mais également avec un taux d'erreur lié à la zone où se trouve l'exemple. De plus, il

est intéressant de remarquer que l'algorithme est facilement généralisable à la classification avec N classes (N > 2) (au lieu d'utiliser N fois un classificateur binaire).

Les Reduced Coulomb Energy classifient donc les exemples de deux façons :

- l'exemple à classifier est dans une zone de sécurité (couverte par une seule classe) et dans ce cas la classe attribuée est évidente ;
- l'exemple à classifier est dans le vide ou en zone d'intersection (entre les deux classes) et dans ce cas on effectue une « discussion » pour déterminer la classe.

B. L'IMPLEMENTATION ET LES PREMIERS RESULTATS

Dans un premier temps, mon travail était de programmer l'algorithme de base des Reduced Coulomb Energy. Pour ce faire, j'ai décidé de partir du programme original de Mario Marchand et non pas de la version sur laquelle je travaillais depuis l'étude PAC Bayes. Il était en effet difficile de reprendre un programme plusieurs fois modifié alors qu'une version de base était suffisante. Le fait de revenir au programme d'origine m'a permis de mieux comprendre ce qui avait été programmé et c'est ainsi que j'ai pu faire mon travail plus proprement que si j'étais reparti d'une version retouchée par Philippe puis Stéphane. Cependant, mon travail précédemment effectué sur le programme de Stéphane, m'a beaucoup aidé à comprendre le programme ce qui m'a facilité pour l'implémentation des Reduced Coulomb Energy.

J'ai donc commencé par séparer le fichier en une dizaine de classes (avec les fichiers C++ et les headers). La figure 12, ci-dessous, montre la structure du programme. La classe « RCE » propose tous les outils nécessaires à l'algorithme et contient notamment deux objets de la classe « SetCoveringMachine » (pour couvrir chacune des deux classes à couvrir). Cette dernière a dû être entièrement réécrite. Son rôle reste le même (c'est-à-dire couvrir une classe avec une liste de boules), mais il s'agit ici d'utiliser un algorithme déterministe et non pas une borne théorique. J'ai du également refaire la classe « Gball ».

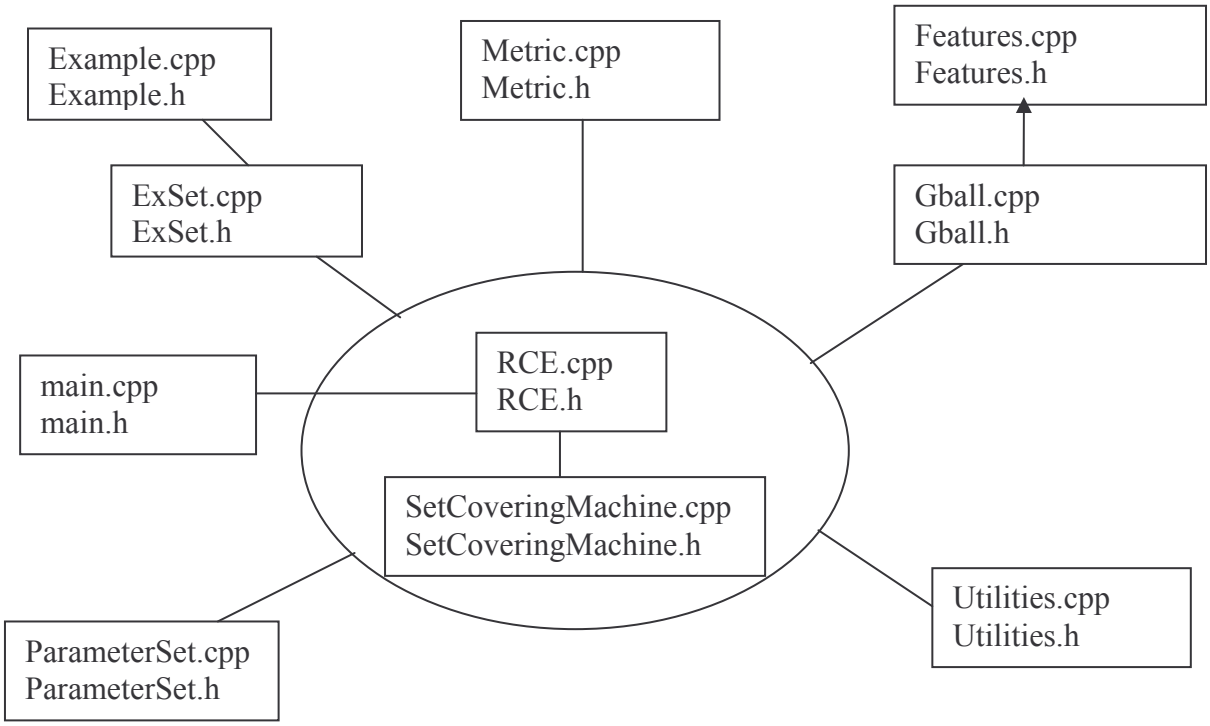

Figure 12 : présentation du programme

Pour chaque classe à couvrir, les Reduced Coulomb Energy utilisent un algorithme déterministe. La première étape de cet algorithme consiste à utiliser tous les exemples de l'échantillon d'apprentissage (de la classe à couvrir) pour créer pour chacun d'eux une boule aussi grande que possible (qui ne couvre que des exemple de la bonne classe : pas d'erreurs possibles).

La deuxième étape consiste à sélectionner les boules que l'on va garder. Pour cela il existe deux méthodes. Voici les deux algorithmes :

- Confident Sphere-Covering (CSC)

Pour chaque exemple,
    Sélectionner la boule contenant le plus d'exemples et qui contient cet exemple
Fin de pour.

- Greedy Sphere-Covering (GSC)

Tant que tous les exemples ne sont pas couverts,
    Sélectionner la boule contenant le plus d'exemples non couverts
Fin de tant que.

Nous avons rajouté à ces deux algorithmes un troisième mode de sélection des boules qui consiste simplement à conserver toutes les boules construites (nous avons nommé cette méthode « AllBall »). Ce sont donc les trois algorithmes déterministes qui remplacent la sélection de modèle par la borne des Set Covering Machines.

On peut remarquer que le Greedy Sphere-Covering revient à effectuer l'algorithme des Set Covering Machines sans pénalité, sans boule creuse et sans la deuxième partie de l'algorithme (sélection de modèle par la borne).

Dans le vide et dans les intersections, on fait une discussion pour savoir quelle classe est la plus probable. Pour le vide on considère toutes les boules et pour les intersections, on ne considère que les boules à l'origine de l'intersection. On calcule le critère suivant pour chaque boule et celle qui minimise le plus ce critère détermine la classe de l'exemple :

$$\frac{distance(centre, exemple)}{rayon}$$

L'annexe A montre un exemple d'exécution du programme avec les paramètres classiques des Reduced Coulomb Energy. On peut voir lors de la phase de tests, le nombre d'exemples qui sont dans les différentes zones ainsi que le nombre d'erreurs commises suivant les zones.

Après l'implémentation de l'algorithme, je suis arrivé à la phase incontournable des tests. Le tableau ci-dessous (figure 13) montre les résultats obtenus pour des jeux de tests les plus répandus dans le domaine de l'apprentissage supervisé. Il est important de comparer ces résultats aux Support Vector Machines et aux Set Covering Machines.

Dans les colonnes en gras on peut lire le nombre total d'erreurs commises sur le jeu de test. Les autres colonnes présentent le nombre d'erreurs de la zone par rapport au nombre d'exemples qui sont dans la zone.

	CSC	sécurité	inter	vide	**AllBall**	sécurité	inter	vide	**GSC**	sécurité	inter	vide	**SVM**	**SCM**	**SCM-PB**
breastw	14	10/328	2/4	2/8	11	7/327	3/12	1/1	13	9/324	3/7	1/9	15	11	10
bupa	69	32/99	20/38	17/38	63	22/86	31/65	10/24	66	34/99	17/39	15/37	66	71	69
credit	57	24/216	22/54	11/30	52	16/200	33/91	3/9	58	28/214	16/40	14/46	51	65	56
haberman	50	29/107	9/19	12/24	45	26/106	14/32	5/12	49	31/108	9/19	9/23	39	41	37
usvotes	18	10/178	2/9	6/13	17	8/173	6/20	3/7	18	9/162	3/11	6/27	13	26	12
sonar	22	9/71	7/19	6/13	16	9/62	5/31	2/10	20	8/69	5/18	7/16	9	18	18

Figure 13 : résultats, algorithme d'origine

Ce sont de bons résultats. On peut noter l'efficacité du mode « AllBall ». On remarque aussi que beaucoup d'exemples sont en zone de sécurité. Parmi ces exemples, certains sont mal classifiés. Il est évident qu'un faible taux d'erreurs dans cette zone serait très intéressant pour beaucoup d'applications pratiques. Ainsi, on pourrait quasiment garantir une bonne classification lorsque l'on classe un exemple dans cette zone. Les résultats dans cette zone ne sont cependant pas mauvais mais il est possible de les améliorer.

On peut également voir qu'en zone d'intersection et de vide, il y a un gros pourcentage d'erreurs. Ces résultats peuvent également être améliorés, ce qui est l'objectif des travaux qui ont suivi.

C. LES AMELIORATIONS

La première idée que j'ai eue consiste à utiliser un seuil lors de la création des boules. Ce seuil a pour objectif d'interdire les boules qui couvrent trop peu d'exemples. Le seuil optimal pour la majorité des cas est deux. Donc on ne conserve pas une boule qui ne couvre qu'un seul exemple (son centre). Dans certains jeux de données très bruitées, un seuil égal à trois est parfois meilleur. Le seuil a été conservé par la suite car il améliore systématiquement les résultats.

L'idée suivante que nous avons eue consiste à supprimer les exemples couverts par aucune boule pour ensuite refaire l'apprentissage sur les exemples restant. Ainsi on espère se débarrasser de quelques exemples identifiables comme étant du bruit. Cette option a améliorée certains résultats sur des jeux de données qui contiennent beaucoup de bruit.

A ce moment de l'étude nous avons séparé le problème en deux parties. La première idée est de chercher à réduire le taux d'erreurs en zone de sécurité. La deuxième idée consiste à trouver un critère plus efficace pour classifier les exemples qui sont dans le vide ou dans la zone d'intersection.

i. La zone de sécurité

On veut donc diminuer le taux d'erreurs dans cette zone. Ma première réaction face à ce problème a été d'appliquer un coefficient fixe sur les boules pour les réduire et ainsi diminuer les exemples en zone de sûreté avec pour objectif que le taux d'erreur diminue également.

Cette étude a amené de très bons résultats sur les données de tests habituels (figure 14 ci-dessous). Les résultats ne sont cependant pas admissibles car j'ai volontairement déterminé

le coefficient des boules en fonction des résultats obtenus sur l'échantillon de test (ce qui provoque de l'« overfitting »). Cette petite astuce m'a ainsi permis de dire que les Reduced Coulomb Energy peuvent obtenir de bien meilleurs résultats.

	RCE, AllBall	RCE, Coef rayon	SVM	SCM	SCM-PB
breastw	11	10	15	11	10
bupa	63	51	66	71	69
credit	52	46	51	65	56
haberman	45	35	39	41	37
usvotes	17	12	13	26	12
sonar	16	11	9	18	18

Figure 14 : résultats, coefficient fixe sur le rayon

En réduisant le rayon des boules, on obtient donc une plus petite zone de sécurité, et donc plus de discussions dans le vide. Et de la même façon, en augmentant la taille des boules, la zone d'intersection grossit et ainsi on finit par réduire la zone de sûreté. On peut également trouver une troisième solution. En effet, c'est pour cette même raison que le mode « AllBall » apporte de meilleurs résultats que les deux autres algorithmes : en gardant toutes les boules, il y a plus d'intersections, donc la zone de sûreté est moins grande.

Ainsi nous avons eu l'idée de réduire le rayon de chaque boule pour qu'il n'y ait plus d'intersections. Cette méthode déterministe évite ainsi de « tricher » comme précédemment en utilisant l'échantillon de test (mais bien sur, les résultats sont moins bons). C'est ainsi que deux nouveaux algorithmes sont apparus dans le programme avec une nouvelle option. Soit on laisse les rayons tels quels, soit on applique l'un de ces deux algorithmes qui coupent les intersections (l'un diminuant plus les rayons que l'autre). Cette nouvelle option a bien amélioré les résultats des Reduced Coulomb Energy. Nous l'avons ainsi conservé par la suite.

Malgré nos efforts, le taux d'erreur dans la zone de sécurité n'a pas assez diminué. Nous avons envisagé de réduire considérablement plus le rayon des boules à l'aide d'une validation croisée. Mais après réflexion, il était impossible de définir un critère efficace.

Mon approche est la suivante : après réduction des boules pour ne pas avoir d'intersections, les boules représentent des zones de sécurité sans erreur possible mais du point de vu de l'échantillon d'apprentissage. Si des exemples de l'échantillon de test sont dans ces zones, on ne peut que les considérer comme du bruit et même si on réduit plus les boules, aucun algorithme logique ne classifiera bien ces exemples. Je pense donc qu'en utilisant la réduction du rayon des boules pour enlever les intersections, on peut définir un pourcentage de bruit dans le jeu de données utilisé (en regardant le nombre d'erreurs en zone de sécurité). Si ces exemples ne sont pas du bruit, c'est que l'échantillon d'apprentissage n'est pas assez représentatif des données.

ii. Les discussions

Dans le vide et en zone d'intersection, le risque atteint presque une chance sur deux. Si on considère que nos données ne sont pas trop bruitées, il est probable que l'on détermine un critère plus efficace. Les Reduced Coulomb Energy proposent le critère suivant (déjà défini précédemment) :

$$\frac{distance(centre, exemple)}{rayon}$$

Nous avons donc réfléchi à de nouveaux critères dans l'optique d'améliorer les résultats :

- plus proche voisin (distance à la plus proche boule) « ppv » ;
- plus proche centre (distance au plus proche centre) « ppc » ;
- vote de majorité avec une gaussienne sur le centre de chaque boule « ppcg »;
- vote de majorité avec une gaussienne sur la bordure de chaque boule « ppcv »;
- classe qui compte le plus grand nombre d'exemples dans l'échantillon d'apprentissage (ce critère peut aussi être utile dans le cas où un autre critère n'arrive pas à classifier l'exemple pour cause d'équidistance par exemple) ;
- classe qui compte le plus de boules dans l'intersection considérée (uniquement pour les discussions en zone d'intersection bien sûr).

Pour les critères qui utilisent une gaussienne, une variance égale à environ 0.22 est toujours efficace. Lors de nos tests, nous avons déterminé cette variance par validation croisée pour plus d'efficacité.

Ces critères peuvent être appliqués pour le vide et pour les intersections. Cependant nous avons réalisé les tests uniquement pour le vide (peut être à tort) en pensant que le critère d'origine était idéal pour les intersections.

Ces critères améliorent les résultats de façon importante. Cependant ce ne sera pas toujours le même critère qui sera le meilleur suivant les jeux de données utilisés. On pourrait envisager une validation croisée pour sélectionner le bon critère… Il faut cependant noter que la gaussienne sur les centres a toujours donné de bons résultats. Mais ce critère est potentiellement améliorable si on trouve un moyen de personnaliser la variance pour chaque boule (en fonction du rayon par exemple). Les tests que nous avons faits sur ce point ne sont pas mauvais mais toute fois moins bons qu'avec une variance fixe.

	bordure différente classe			**bordure même classe**							**gaussienne**	
	CSC	**AllBall**	**GSC**	**CSC**	**AllBall**	**GSC**	coef rayon	SVM	SCM	SCM-PB	**var = 0,22**	
breastw	10	12	10	11	11	11	10	15	11	10	14	ppv
bupa	62	56	62	63	59	63	51	66	71	69	58	ppc
credit	43	45	44	44	44	44	46	51	65	56	44	ppc
haberman	39	39	39	39	39	38	35	39	41	37	44	ppv, seuil=3
usvotes	18	19	18	14	15	15	12	13	26	12	17	ppv, seuil=3
sonar	17	18	16	17	18	16	11	9	18	18	12	ppc

Figure 15 : résultats avec différents critères, un seuil et une fonction pour réduire les boules

La dernière idée que nous ayons testée utilise l'idée de réduction et d'agrandissement des rayons. Pour chaque boule on définit deux rayons : le rayon de sûreté qui a été réduit et un rayon maximum qui va jusqu'au prochain rayon de sûreté rencontré (le rayon maximum choisi peut être très contestable). Ce deuxième rayon permet de classer du premier coup les exemples lorsqu'il n'y a pas d'intersections. En cas d'intersection nous avons défini un vote de majorité des boules concernées par une relation linéaire (plus on va du rayon minimum vers le rayon maximum et plus le droit de vote de la boule diminue).

En définitive, les Reduced Coulomb Energy se sont avérés être de bons classificateurs utilisant un algorithme déterministe simple. Les idées que nous avons apportées ont bien amélioré les résultats comme le montre ce dernier tableau (figure 16 ci-dessous).

	MinMax								validation croisée	
	CSC	**AllBall**	seuil	vide	coef rayon	SVM	SCM	SCM-PB	**gaussienne**	**Rayon**
breastw	9	11	2	ppv	10	15	11	10	10	11
bupa	61	58	2	ppc	51	66	71	69	60	58
credit	44	44	3	ppc	46	51	65	56	44	50
haberman	37	39	3	ppv	35	39	41	37	36	35
usvotes	14	15	2	ppv	12	13	26	12	13	12
sonar	17	18	2	ppc	11	9	18	18	15	15

Figure 16 : résultats avec l'option MinMax

L'annexe B propose l'algorithme très simplifié avec les options les plus importantes. L'annexe C montre un exemple d'exécution du programme avec les dernières options présentées.

7. ABOUTISSEMENT

A. AVANCEMENT DU PROJET

Pendant mon stage, plusieurs résultats d'études m'ont permis de mieux comprendre les Set Covering Machines. L'utilisation d'une borne théorique s'est révélée très efficace pour cet algorithme. La perte d'efficacité en haute dimension n'étant pas aussi importante que nous le pensions, il s'agit plus d'améliorer les réglages de l'algorithme que de mettre en doute son efficacité.

L'alternative proposée par les Reduced Coulomb Energy, que j'ai eu à implémenter, donne de bons résultats. De plus, nous avons trouvé un grand nombre d'optimisations. En effet, les idées que j'ai implémentées ont sensiblement amélioré les résultats. On peut noté cependant la difficulté d'évaluer chaque option séparément, l'algorithme formant un tout difficile à interpréter lorsque les résultats varient beaucoup d'un jeu de test à un autre.

Les données réelles utilisées dans le domaine de la classification comportent peu d'exemples. De plus, il y a parfois, beaucoup de bruit. Ainsi lorsque l'on obtient de bons résultats sur un échantillon, il est possible que les progressions suivantes ne soient qu'un peu d' « overfitting » de la part de l'algorithme. On prend ainsi le risque de fournir de bons résultats sur différents jeux de tests en ayant modifié à chaque fois les multiples options disponibles. On peut alors complexifier un algorithme à tort.

Ces jeux de tests atteignent donc possiblement leurs limites… Ce problème peut nous faire tourner en rond lors de l'évaluation de certains algorithmes performants. Il est bien sûr possible d'utiliser de nouveaux échantillons (synthétiques ou non) mais il devient ensuite difficile d'évaluer l'efficacité de l'algorithme lorsque l'on ne peut plus comparer ses résultats aux autres algorithmes.

En ce qui concerne la continuité des recherches sur ce thème, je pense que les Set Covering Machines sont très efficaces, et qu'il serait intéressant de tester des méthodes comme le recuit simulé. Je pense aussi qu'une méthode déterministe, et non pas basée sur une borne, pourrait apporter de bons résultats.

En ce qui concerne les Reduced Coulomb Energy qui ont occupé la plus grande partie de mon stage, je suis sûr que cette approche à beaucoup d'avenir notamment pour son évolution facile vers un traitement avec N classes. De plus, une approche géométrique et symétrique vis-à-vis de chaque classe me parait bien appropriée. Les recherches dans cette direction sont nombreuses :

- Peut on définir une nouvelle borne ?
- Trouver un algorithme déterministe qui soit systématiquement le plus efficace…
- Essayer de couvrir les régions à l'aide de l'algorithme de Mario Marchand basé sur une borne
- Utiliser la zone de sécurité de l'algorithme pour des applications pratiques…
- Tester l'efficacité avec plus de deux classes

B. BILAN DU STAGE

Mon stage s'est bien déroulé. Le sujet m'intéresse beaucoup et m'a permis d'étudier de nouvelles notions en intelligence artificielle. Réfléchir à l'efficacité de différents algorithmes est un travail très intéressant et la poursuite d'un résultat marquant est très motivante.

Travailler sur la base d'un programme déjà compliqué n'est pas facile mais m'a permis d'aboutir rapidement à de bons résultats. De plus, le processus d'analyse et de réutilisation d'un code existant m'a permis de vite progresser en algorithmique et en C++, ce qui constitue un complément de formation important. J'ai également découvert le logiciel R largement répandu dans les projets incluant des statistiques.

Le travail en équipe m'a donné régulièrement l'occasion de présenter mon travail et mes résultats pour en discuter et mieux les interpréter. Lors de mes différentes études, j'ai appris à lire et à interpréter des résultats en fonction de tout leur contexte (par exemple la superposition de bornes ou encore le phénomène d'« overfitting »). Le travail en équipe nous a également permis, tout au long de mon stage, de partager les idées de tous pour se diriger rapidement vers les meilleures approches.

Pour conclure, ce stage m'a procuré une expérience très intéressante qui m'a notamment permis de découvrir l'environnement de la recherche. J'ai eu la chance de découvrir de nouvelles notions en intelligence artificielle dans un cadre idéal. De plus, sur le plan personnel, ce stage s'est accompagné d'un très beau voyage qui m'a permis de passer cinq mois au Québec.

8. REMERCIEMENTS

Je remercie mon maître de stage, François Laviolette, pour m'avoir donné l'occasion de réaliser un stage sur un sujet très intéressant et pour m'avoir si bien accueilli au Québec. Je

tiens également à le remercier pour toute son aide et son attention pendant mon stage. J'ai beaucoup apprécié ces mois de stage avec ce groupe soudé.

Mes remerciements s'adressent aussi à Mario Marchand qui s'est toujours porté disponible pour m'aider.

Je remercie Stéphane Simard pour toutes ses explications sur le projet et ses travaux. Son aide pour l'introduction au R et pour la reprise du programme a été capitale.

Je remercie Philippe Choquette pour son accueil au sein du groupe et je l'encourage dans la poursuite de son étude sur les Set Covering Machines.

Je remercie ma tuteur de stage, Christine Régis, pour sa disponibilité et ses conseils au cours de mon stage.

9. REFERENCES

Mario Marchand, cours IFT-65764 – Apprentissage automatique, automne 2004,
URL : http://www.ift.ulaval.ca/~mmarchand/ift65764/ift65764-nc.htm

Mario Marchand et Marina Sokolova, Learning with Decision Lists of Data-Dependent Features, Journal of Machine Learning Research, vol. 6, pp. 427-451 (2005),
URL : http://jmlr.csail.mit.edu/papers/volume6/marchand05a/marchand05a.pdf

François Laviolette, Mario Marchand et Mohak Shah, A PAC-Bayes approach to the Set Covering Machine.

Jigang Wang, Predrag Neskovic et Leon N Cooper, Learning class regions by sphere covering.

Site officiel du projet R : R project,
URL : http://www.r-project.org/

Manuel d'introduction à R : An Introduction to R,
URL : http://cran.r-project.org/doc/manuals/R-intro.html

Manuel sur le langage de script R : R Language Definition,
URL : http://cran.r-project.org/doc/manuals/R-lang.html

Guide sur le logiciel R (département de mathématiques de l'université Laval) : Le logiciel R,
URL : http://www.mat.ulaval.ca/informatique/R/R.html

10. ANNEXES

A. ANNEXE A : REDUCED COULOMB ENERGY, EXEMPLE SIMPLE

```
0) Méthode de validation : Learn and Test (!=CV).
1) Méthode de construction des boules: bordures de classe opposée.
2) Seuil utilisé (on ignore les boules qui couvrent moins d'exemples): 1
2prime) Coefficient rayon utilise : 1
3) Nombre de fois ou on enlève les exemples non couvert avant
réaprentissage: 0
4) Pas de réduction des boules.
5) Méthode RCE : on garde toutes les boules.
6) Critère Intersection : rce. C'est a dire : min(d(ex,c)/r).
7) Critère Vide : rce. C'est a dire : min(d(ex,c)/r).

Number of training examples: 105
 - RCE learn - done.

Number of testing examples = 103 (55 + and 48 -).
 - RCE test - done.

The RCE contains 49 -nodes and 56 +nodes.
                sécurité vide    inter   total
discussions     62       10      31      103
err + et -      9        2       5       16
err   +         2        1       1       4
err   -         7        1       4       12
Risque=15.534%
```

B. Annexe B : Reduced Coulomb Energy, algorithme final

---------- Récapitulatif des options :

-Learn And Test
-Validation croisée puis Learn And Test

Apprentissage :
-Seuil
-Construction des boules:Bordures de même classe, bordures de classe opposée
-Possibilité d appliquer un coefficient au rayon de toutes les boules
-Possibilité de retirer les exemples non couvert et de relancer l apprentissage
-Possibilité de réduire les boules pour éviter les intersections (2 méthodes possibles)
-Sélection des boules : CSC, GSC, garder toutes les boules

Test :
-Vide : RCE, PPV, PPC, gaussienne sur le centre (PPCG), gaussienne sur le rayon (PPVG)
-Intersection : RCE, nombre de boules
-Possibilité d'utiliser rayon Min et rayon Max pour un vote de majorité

---------- Rapide présentation de l'algorithme :

LEARN :

```
SCM1->learn_allBall()  // apprentissage sur les -
SCM2->learn_allBall()  // apprentissage sur les +

for (i=0 ; i<RemoveSeuil ; i++) {
  // on supprime les exemples non couvert
  removeSeuil();

  // On refait l'apprentissage
  SCM1->learn_allBall(); // apprentissage sur les -
  SCM2->learn_allBall(); // apprentissage sur les +
}

// on determine rayon min
if (CutInter1)
  cutInter1();
else if (CutInter2)
  cutInter2();

// on determine le rayon max
if (RayonMinMax)
  rayonMax();

// Méthodes possibles : CSC, GSC ou rien (=AllBall)
if (CSC) {
  SCM1->learn_CSC(...);
  SCM2->learn_CSC(...);
} else if (GSC") {
  SCM1->learn_GSC(...);
  SCM2->learn_GSC(...);
}
```

Test :

```
Pour chaque exemple:
  Liste des boules + qui contiennent l'exemple: nBoulePos
  Liste des boules - qui contiennent l'exemple: nBouleNeg

if (nBouleNeg > 0 et nBoulePos==0)
  exemple classé negatif

if (nBouleNeg==0 et nBoulePos >0)
  exemple classé positif

if (nBouleNeg > 0 et nBoulePos >0)
  classe = intersection(ex);

if (nBouleNeg==0 et nBoulePos==0)
  if (rayonMinMax)
    classe = rayonMinMax(ex);
  else
    classe = vide();
```

C. Annexe C : Reduced Coulomb Energy, exemple final

```
0) Méthode de validation : Learn and Test (!=CV).
1) Méthode de construction des boules: bordures de classe opposée.
2) Seuil utilisé (on ignore les boules qui couvrent moins d'exemples): 2
2prime) Coefficient rayon utilise : 1
3) Nombre de fois ou on enlève les exemples non couvert avant
réaprentissage: 1
4) Réduction (1) des boules pour ne pas avoir d'intersections entre les
classes.
  si réduction, coefficient utilise sur les boules réduites: 1
5) Méthode de sélection des boules : CSC.
6) Critère Intersection : rce. C'est a dire : min( d(ex,c)/r ).
7) Critère Vide : + proche centre.
  si gaussienne (et run_RCE), variance = 0.21
8) Utilisation de l'idée rayons min, max: b-alpha/(b-a).

Number of training examples: 105
 - RCE learn -
remove seuil 1
done.

Number of testing examples = 103 (55 + and 48 - ).
 - RCE test - done.


The RCE contains 15 -nodes and 24 +nodes.
                sécurité vide    inter   total
discussions     52       51      0       103
err + et -      5        17      0       22
err   +         0        5       0       5
err   -         5        12      0       17
Risque=21.3592%

 Rayon Min Max, les exemples dans le vide sont repartages:
 zone sur:      2 erreurs: 0
 intersection:     0 erreurs: 0
 vrai vide:     49 erreurs: 17
 nb err en InterBis +: 0
 nb err en InterBis -: 0

 nb de boules retirées par le seuil (-): 15
 nb de boules retirées par le seuil (+): 8

 nb d'exemples négatifs retires:11
 nb d'exemples positifs retires:3

 nb de boules écrasées par le rétrécissement (-) : 0
 nb de boules écrasées par le rétrécissement (+) : 0

 nb de boules écrasées, non agrandies (-) : 0
 nb de boules écrasées, non agrandies (+) : 0

 nb d'exemples non couverts par le GSC/CSC (-) : 15
 nb d'exemples non couverts par le GSC/CSC (+) : 7
```