Université du Québec à Montréal
Département d'informatique

Les algorithmes d'apprentissage automatique offerts par l'environnement Weka

Fait par : H. Yazid

Dirigé par : Prof. Hakim Lounis

Année 2006

Table des matières

1 Introduction

En plus des outils offerts (Explorer, Expérimenter, KnowledgeFlow), Weka (Waikato Environment for knowledge Analysis) met à la disposition d'un utilisateur une collection d'algorithmes d'apprentissage automatique. Parmi ces algorithmes, nous allons présenter dans ce document les plus communément utilisés (voir le tableau suivant). Ces algorithmes peuvent être utilisés soit pour une régression ou une classification. La régression s'intéresse à trouver une relation de corrélation entre les entrées (les différents variables indépendantes) et les sorties (la ou les variables dépendantes). La classification a pour but de prédire la classe d'une instance de test. Ces algorithmes sont appliqués en mode supervisé, guidé par la sortie (ou la classe ciblée).

Ces algorithmes sont répartis sur sept ou six familles dépendant de la version utilisée de Weka. Les sept familles trouvée dans la version 3.4 sont les réseaux Bayésiens, les classificateurs 'function', les classificateurs paresseux, les méta classificateurs, les classificateurs divers, les classificateurs basés sur les arbres de décision et les classificateurs basés sur les règles. Dans la version 3.4.6, toutes les familles sont présentes sauf la famille des classificateurs divers.

Dans la catégorie réseaux Bayésiens, on présentera les algorithmes *BayesNet* et *NaifBayes*. Ces deux algorithmes se basent sur la formule de Bayes avec la différence que dans le premier cas, l'hypothèse d'indépendance des attributs n'est pas prise en compte. Par contre, *NaifBayes* considère que les attributs (entrées) sont indépendants. Dans la seconde catégorie (classificateurs 'function'), on s'intéressera à six algorithmes qui sont *LeastMedSq*, *LinearRegression*, *Logistic* (régression logistique), *MultilayerPerceptron* (les réseaux de neurones multicouches), *RBFNetwork* (les réseaux à fonction radiale) et *SMO* (Sequential Minimal Optimization). Cette catégorie se caractérise par le fait que tous ses classificateurs procèdent au calcul d'une fonction donnée lors de la construction du modèle d'où le nom attribué à cette famille.

Dans la famille classificateurs paresseux, on considèrera trois algorithmes : *IB1*, *IBk* (K=4) et *KStar*. Ces derniers sont basés sur le même concept qui est la classification d'une nouvelle instance selon celles déjà classifiées en utilisant soit une fonction distance (cas de *IBk*) ou une fonction de similitude (cas de *KStar*).

La famille des méta-classificateurs regroupe les algorithmes qui permettent de combiner plusieurs classificateurs dans le but d'obtenir de meilleurs performances. Quatre algorithmes *AdaBoostM1*, *Bagging*, *Stacking* et *Vote* sont présentés dans ce document. Les classificateurs divers considérés sont *FLR* (Fuzzy Lattice Reasoning), *HyperPipes* et *VFI*. Ces trois algorithmes sont assez similaire et se basent sur la notion des limites d'un intervalle associé à chaque attribut.

Les classificateurs basés sur les arbres de décision génèrent des modèles sous forme d'arbre dont les nœuds sont des tests sur des attributs et les feuilles sont les classes. Parmi ces algorithmes, on décrira les algorithmes *ADTree*, *NBTree* et *J48* qui sont des variantes des arbres de décision. La dernière famille est constituée par les classificateurs basés sur les règles qui génèrent des modèles en forme de règles dont la partie condition est une combinaison de tests sur des attributs et la conclusion est la classe prédite. Un aperçu sur ces algorithmes est donné par les trois algorithmes *Conjunctive*, *PART* et *DecisionTable*.

Tableau I

Répartition des algorithmes choisis et leurs domaines

Code	**Groupe**	**Algorithmes**	**Domaines**
1	Réseaux bayésiens	BayesNet	Classification (classe nominale)
2		NaiveBayes	Classification
3	Classificateurs 'function'	LeastMedSq	Régression
4		LinearRegression	Régression
5		Logistic	Classification
6		MultilayerPerceptron	Classification, régression
7		RBFNetwork	Classification, régression
8		SMO	Classification
9	Classificateurs paresseux	IB1	Classification, régression
10		IBK (k=4)	Classification, régression
11		KStar	Classification, régression
12	Méta-classificateurs	AdaBoostM1	Classification
13		Bagging	Classification, régression
14		Stacking	Classification, régression
15		Vote	Classification
16	Classificateurs divers	FLR	Classification
17		HyperPipes	Classification
18		VFI	Classification
19	Classificateurs basés sur les arbres de décisions	ADTree	Classification
20		J48	Classification
21		NBTree	Classification
22	Classificateurs à base de règles	ConjunctiveRule	Classification, régression
23		DecisionTable	Classification, régression
24		PART	Classification

2 Réseaux Bayésiens

L'ensemble des algorithmes de ce groupe est basé sur la loi de Bayes. Bayes (Cornuéjols, 2002) propose, d'une part, que la connaissance sur le monde soit traduite par un ensemble d'hypothèses (fini ou non), chacune d'entre-elles étant affectée d'une probabilité reflétant le degré de croyance de l'apprenant dans l'hypothèse en question. La connaissance sur le monde est ainsi exprimée sous la forme d'une distribution de probabilités sur un espace d'hypothèses.

L'apprentissage consiste alors à trouver une structure de dépendances entre variables ou à estimer les probabilités conditionnelles définissant ces dépendances. Ainsi, les modèles basés sur cette formule, permettent d'exprimer des relations probabilistes entre les ensembles de faits. Le raisonnement adopté dans ce cas est conditionnel; deux faits peuvent être en relation causale.

Plusieurs variantes existent entre autre l'algorithme *BayesNet* et *NaiveBayes*.

2.1 BayesNet

Formellement la loi de Bayes (John, 1995) s'écrit de la manière suivante :

$$p_H(h \mid S) = \frac{p_H(h) P_X^m(S \mid h)}{P_X(S)}$$

Où :

p_H : densité de probabilité définie sur l'espace des hypothèses $h \in H$;

P_X : mesure de probabilité des événements sur X;

P_X^m : mesure de probabilité d'un ensemble d'apprentissage S= $\{(x_1, u_1), \ldots, (x_m, u_m)\}$.

De plus, $p_H(h)$ et $p_H(h \mid S)$ désignent respectivement la probabilité à priori de h et la probabilité à posteriori de h après considération des données de S. $P_X^m(S \mid h)$ est la probabilité conditionnelle de l'événement S si l'on suppose vrai l'état du monde correspondant à h. $P_X(S)$ est la probabilité

à priori de l'événement S.

L'importance en pratique de la règle de Bayes tient au fait qu'elle permet de ré-exprimer la probabilité à posteriori qui est difficile à calculer, en terme de probabilité à priori et conditionnelle, facile à obtenir.

L'algorithme *BayesNet* implémenté au niveau de Weka permet de réaliser l'apprentissage avec les réseaux Bayésiens en utilisant de nombreux algorithmes de recherche, e.g., *GeneticSearch*. Il a de bonnes performances, particulièrement lorsque les attributs corrélés sont définis par l'utilisateur; mais il peut aussi apprendre avec succès les corrélations. Cependant, un fait important est que les réseaux Bayésiens ne tiennent pas compte de l'hypothèse d'indépendance des attributs.

2.2 NaiveBayes

Un réseau Bayésien naïf est basé sur deux hypothèses. La première suppose que les attributs sont indépendants et la deuxième énonce qu'aucun attribut caché ou latent ne peut influencer le processus de la prédiction (John, 1995). La probabilité combinée des variables revient donc à la multiplication des probabilités des différentes variables. Dans certains travaux, la probabilité est définie par la fréquence d'apparition d'une variable (basé sur l'estimateur de Laplace).

Le classificateur Bayésien naïf fournit une approche simple avec une sémantique claire pour représenter, utiliser et apprendre des connaissances probabilistiques. Cette méthode est utilisée dans le cadre de l'apprentissage supervisé. La performance est de prédire avec exactitude la classe des instances de test.

Les réseaux Bayésiens naïfs donnent de bons résultats, spécialement après l'exécution de la procédure de la sélection des attributs afin de supprimer les variables redondantes et dépendantes. Il est aussi compétitif que les meilleurs algorithmes d'apprentissage. L'inconvénient majeur de cette méthode est l'hypothèse d'indépendance des variables qui ne représente pas la réalité du monde réel.

3 Classificateurs "function"

Dans cette catégorie de classificateurs, nous présentons les algorithmes basés sur la régression linéaire, la régression logistique, les réseaux de neurones multicouches, les *SVM* (Sequential Minimal Optimization) et *RBFN* (Radial Basis Function Network).

3.1 La régression linéaire

La régression linéaire (Curvefit,1999) analyse la relation entre deux variables X et Y, le but étant est de trouver une droite qui prédit de la meilleure manière Y à partir de X. Elle réalise ce but en trouvant la droite minimisant la somme des carrés des distances verticales des points représentant l'ensemble d'apprentissage de la droite de régression. Elle suppose que les données sont linéaires et cherche les coordonnés à l'origine et la pente de la droite qui correspond au mieux à l'ensemble des données.

3.1.1 LeastMedSq

Cet algorithme réalise une régression linéaire à moindre carré médiane (Witten, 2000). Les fonctions de régression de moindre carré sont générées à partir de sous échantillons de données aléatoirement tirés de l'ensemble de données d'apprentissage. De ces fonctions, la régression présentant la plus basse erreur médiane carrée est choisie comme modèle final. Plus précisément, cet algorithme consiste à minimiser la médiane des carrés des distances des points représentant les données de l'ensemble d'apprentissage à la droite de régression. Géométriquement, ceci correspond à trouver la bande couvrant la moitié des observations, où l'épaisseur de la bande est mesurée dans la direction verticale. Il est à noter que cette notion est souvent plus facile à expliquer et à visualiser que la régression des moindres carrés moyenne.

Cette technique s'avère très robuste et fait face aux points isolés. Ce qui n'est pas le cas de la méthode des moindre carré moyenne qui est influencée par les points atypiques.

Malheureusement, la méthode médiane à moindre carrés est très coûteuse en temps de calcul ce qui souvent la rend inaccessible pour des problèmes pratiques.

3.1.2 LinearRegression

La régression linéaire (Curvefit,1999) est une méthode simple et recommandée pour la prédiction numérique (domaine de la régression) qui a été largement utilisée dans les applications statistiques pendant des décennies. Cependant, le modèle linéaire souffre de cet inconvénient qui est la linéarité. Si les données montrent une dépendance non linéaire, la ligne trouvée comme ligne de régression linéaire ne correspond pas alors à la meilleure approche. L'algorithme *LinearRegression*, implantée dans l'outil Weka, permet d'accomplir la régression linéaire sur un ensemble de données en utilisant le critère de Akaike (StatSoft, 2004) pour la sélection du modèle. Elle est capable de traiter des instances pondérées.

3.1.3 Régression logistique

La régression logistique est définie comme la généralisation d'une régression linéaire simple. L'objectif de la régression linéaire simple est de modéliser la relation entre une variable dépendante *quantitative* et une variable explicative (ou indépendante) quantitative. La régression logistique correspond à une régression linéaire où la variable dépendante (ou à expliquer) est *binaire* (c'est à dire qui ne peut prendre que deux valeurs 0/1 ou Oui/Non) (Alpaydin, 2004). Elle est très utile pour comprendre ou prédire l'effet d'une ou de plusieurs variables sur une variable à réponse binaire, e.g., modéliser l'effet du dosage d'un médicament.

Le classificateur "*Logistic*", fournit par l'outil Weka, permet la construction et l'utilisation du modèle de régression logistique multi-nominale avec l'estimateur de Ridge (Breiman, 1984). L'algorithme utilisé dans ce cas est une version modifiée de la régression logistique usuelle où la pondération des instances est prise en compte.

3.2 Réseaux de neurones multicouches

Les réseaux de neurones multicouches ont une architecture composée de plusieurs couches superposées; les nœuds d'une couche i sont reliés à ceux de la couche supérieure i-1. Nous trouvons dans un réseau multicouches :

- des neurones d'entrées : chargés de transmettre les données. La couche d'entrée permet la lecture des données qui sont des vecteurs de $\square^d$, notés $x^T=(x_1,\ldots,x_d)$. En génie logiciel, un x_j peut représenter une valeur donnée d'une métrique;
- des neurones cachés : ce sont des unités de traitements intermédiaires caractérisés par une fonction d'activation qui peut être une fonction seuil, un échelon ou une sigmoïde;
- des neurones de sorties : fournissent un résultat d'apprentissage traduisant une décision; généralement une décision de classification qui est interprétée par l'appartenance des données à une classe ou non;
- à chaque lien entre deux nœuds i et j appartenant à deux couches superposées, un poids est associé, noté par w (i,j) et qui est appelé poids synaptique (w pour 'weight').

Les réseaux de neurones utilisent le principe de la propagation d'informations entre neurones pour effectuer une tâche désirée. Plus précisément, une connaissance est décrite par des interconnexions de neurones et leurs poids synaptiques. Lors de l'apprentissage, ces poids peuvent être ajustés sur un ensemble de règles : le réseau ainsi entraîné pourra réaliser des tâches de classification (données nominales) ou de régression (données numériques) (Cornuéjols, 2002).

La figure 1 illustre un réseau de neurones généré à partir de données sur un produit logiciel.

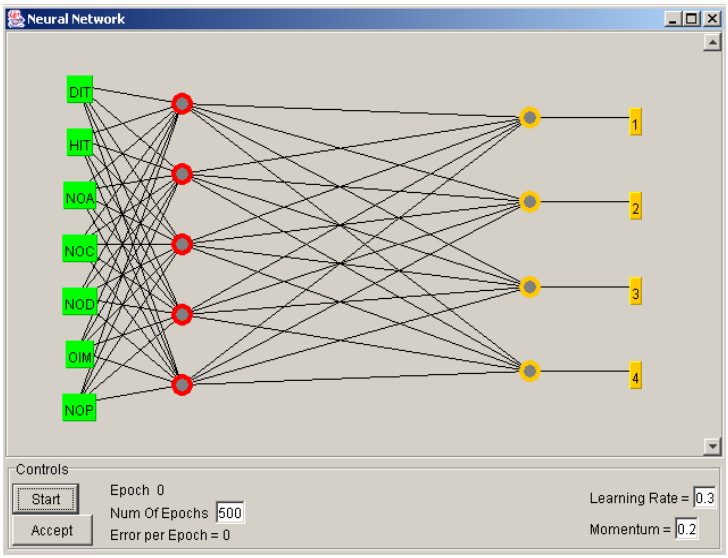

Figure 1 Réseau de neurones

Les nœuds de sorties (à droite) sont liés aux *classe1*, *classe2*, *classe3* et *classe4* et les nœuds ronds (à gauche) sont des nœuds cachés reliés aux nœuds d'entrées représentant des métriques d'héritage.

Le classificateur "réseau de neurones multicouches", implanté au niveau de l'outil Weka, utilise l'algorithme rétro-propagation pour réduire l'erreur d'apprentissage. Cet algorithme, facile à comprendre, consiste à remonter couche par couche en partant de la couche de sortie vers la couche d'entrée, en modifiant les poids synaptiques en amont de chaque couche afin de réduire l'erreur en sortie du réseau.

L'outil Weka permet aussi à un utilisateur de construire un réseau de neurones manuellement ou de le créer par algorithme. De plus, le réseau peut être contrôlé ou modifié au cours de l'apprentissage (exemple ajout de nœuds et de liens). Les nœuds cachés du réseau ont tous une fonction d'activation sigmoïde.

3.3 Le classificateur SVM (Sequential Minimal Optimization)

Afin de décrire l'algorithme *SMO* (Sequential Minimal Optimization), nous présentons tout d'abord les algorithmes d'apprentissage *SVM* (Support Vector Machines). L'origine des algorithmes *SVM* se trouve dans les méthodes développées dans les années 60. Le principe est la séparation de l'espace d'apprentissage par un hyperplan (nommé aussi surface linéaire) en se basant sur l'hypothèse que l'ensemble d'apprentissage est constitué d'exemples et de contre exemples. La classification d'un exemple test lors de l'apprentissage revient à faire un calcul linéaire et le signe de ce calcul permet de déterminer l'appartenance de l'exemple test (l'espace des exemples ou l'espace des contre exemples). Une généralisation de ces méthodes a conduit aux réseaux connexionnistes vers les années 1980. Les méthodes *SVM*, apparues dans les années 1990, sont aussi une généralisation de ces méthodes hyperplans. Elles procèdent tout d'abord par une transformation non linéaire de l'espace d'apprentissage et puis elles cherchent un hyperplan. Les figures 2, 3 et 4 permettent d'illustrer ces concepts (StatSoft, 2004). Nous avons, dans la figure 2, les instances appartenant soit à la classe claire soit à la classe foncée séparée par un hyperplan (classificateur linéaire) qui permet de classifier une nouvelle instance.

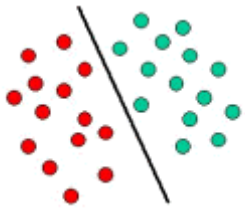

Figure 2 Séparation par un hyperplan linéaire

Seulement, cette représentation est trop simpliste puisque dans la majorité des cas de classification, il est nécessaire de trouver une séparation optimale pouvant classifier correctement les nouvelles instances. Ainsi, la séparation entre les deux classes claire et foncée de la figure 3 requiert une courbe.

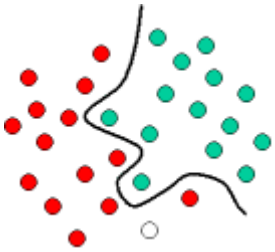

Figure 3 Séparation par une courbe

Les méthodes *SVM* permettent de prendre en compte ce problème de linéarité. Elles appliquent en premier une transformation mathématique sur l'espace d'apprentissage en utilisant des fonctions noyaux (ces fonctions sont non-linéaires). Une fois la transformation faite, les instances peuvent être séparées linéairement; il reste à trouver l'hyperplan optimal. Ce procédé est schématisé par la figure 4.

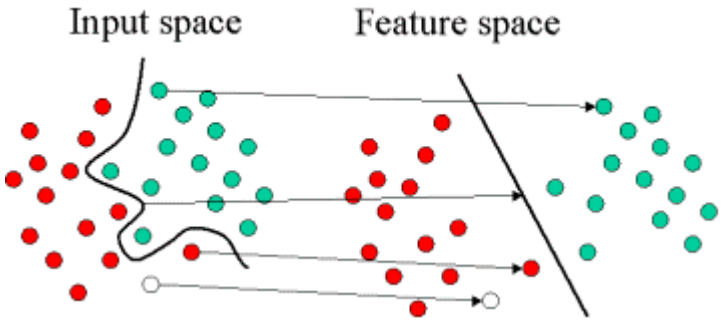

Figure 4 Transformation de l'espace d'apprentissage en SVM

Ce type de méthodes peut être utilisé pour construire plusieurs hyperplans dans un espace multidimensionnel permettant ainsi de classifier une nouvelle instance entre plusieurs classes. Différents types de méthodes *SVM* existent dépendant de la fonction noyau utilisée : linéaire, polynomiale, radiale (*RBF* : radial basis function) et sigmoïde. Les fonctions noyaux *RBF* sont le choix le plus populaire dans les méthodes *SVM*. Ces dernières peuvent être utilisées que ce soit dans la régression (variables numériques) ou la classification (variables nominales). Malgré la popularité de ces méthodes dans les domaines de l'apprentissage automatique et de la vision artificielle, il reste que les méthodes *SVM* s'avèrent lentes, en particulier pour les grands ensembles d'apprentissage. De plus, ces méthodes sont complexes et difficiles à implanter. L'utilisation des fonctions noyaux dans les méthodes *SVM* nécessite la résolution d'une

expression matricielle quadratique. Cette résolution est coûteuse en espace mémoire et en temps. La méthode *SMO* (Keerthi, 2001 ; Platt, 1998) est une nouvelle méthode d'apprentissage de type *SVM* qui est simple, facile à implanter et rapide. Elle propose une manière de résoudre cette expression quadratique en la subdivisant en plusieurs sous expressions faciles à résoudre; c'est une méthode d'optimisation. Elle adopte pour cela l'analyse quadratique lui permettant ainsi de passer plus de temps à évaluer la fonction de décision et exploiter au mieux les données d'apprentissage contenant un grand nombre de zéros (ces données d'apprentissages sont appelées creuses) et des valeurs binaires. La méthode *SMO* peut être un élément efficace dans l'utilisation de plus en plus importante des techniques d'apprentissage *SVM*.

L'algorithme implanté au niveau de l'outil Weka est la version *SMO* développée par John Platt (Platt, 1998). Il remplace toutes les valeurs manquantes et transforme les attributs nominaux en binaires. Par défaut, il normalise tous les attributs ainsi la sortie est basée sur ces attributs normalisés pas sur les données originales. Il est utilisé seulement pour des problèmes de classification.

3.4 Classificateur RBFN (Radial Basis Function Network)

Les réseaux à fonction radiale (*RBF*) sont un cas particulier des réseaux multi-couches et des réseaux SVM (Haykin, 1994). Ils sont composés de trois couches (Figure 5) où chaque nœud de la couche cachée utilise comme fonction d'activation une fonction noyau telle que "la Gaussienne". Cette fonction est centrée au point spécifié par le vecteur de poids associé au nœud. La position et la largeur des fonctions d'activation sont apprises à partir des données d'apprentissage. Plus spécifiquement, l'entrée de chaque neurone caché (unité radiale) correspond à la distance entre le centre de l'unité (déterminé par l'algorithme "clustering K-Means") et le vecteur d'entrée. La sortie du neurone caché est alors l'application de la fonction Gaussienne à cette distance. L'écart type de la fonction Gaussienne peut être calculé par certaines méthodes, e.g., la méthode "K-nearest neighbor" (voir plus loin les classificateurs paresseux *IBk*). Chaque nœud de sortie implante une combinaison linéaire de ces fonctions.

L'algorithme "clustering K-Means" (Weisstein, 2005) tente de sélectionner un ensemble de points optimaux qui sont considérés comme les centroïdes des regroupements de données d'apprentissage. Cet algorithme consiste, dans une première étape, à assigner des données d'apprentissage aléatoirement à k ensembles et dans une deuxième étape, à calculer le centroïde de chacun des k ensembles. Ces deux étapes sont répétées jusqu'à ce que le critère d'arrêt soit rencontré c'est-à-dire quand aucun changement ne se fait sur l'assignation des données d'apprentissage.

Les réseaux *RBF* ont plusieurs avantages dont, la capacité de modéliser n'importe quelle fonction non linéaire en utilisant une seule couche cachée. La transformation linéaire dans la couche de sortie peut être optimisée en utilisant les techniques rapides de modélisation linéaire traditionnelles. Ces réseaux sont aussi plus rapides que les réseaux multicouches dans l'apprentissage. Néanmoins, ils sont plus lents dans l'exécution et consomme plus d'espace que les réseaux traditionnels multicouches.

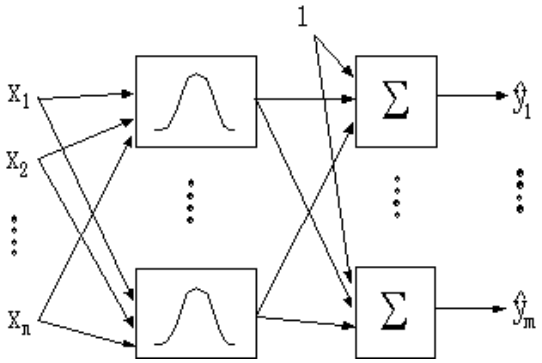

Figure 5 Réseau *RBFN* à plusieurs sorties (Orr, 1996)

Les problèmes pouvant être résolus par l'application de ces réseaux sont l'approximation de fonction, la classification, la modélisation des systèmes dynamiques, etc.

4 Classificateurs paresseux

Les méthodes basées instances débutent avec une instance particulière et déterminent comment elles peuvent être généralisées pour couvrir d'autres instances avoisinantes dans la même classe.

4.1 IBk (K instance-based learner: K-nearest neighbours classifier)

Les méthodes du plus proche voisin sont des techniques statistiques standard qui ont été adaptées par les chercheurs en apprentissage automatique pour améliorer les performances de la classification et rendre la procédure plus efficace en temps de calcul.

Dans l'apprentissage basé sur les instances, chaque nouvelle instance est comparée avec celles déjà existantes en utilisant une métrique distance (Witten, 2000). L'instance la plus proche est utilisée pour assigner la classe de la nouvelle instance. Ainsi, cette technique est nommée la méthode de classification du plus proche voisin. Dans certains cas, plus d'un plus proche voisin sont utilisés et la classe majoritaire des voisins les plus proches (ou la moyenne des distances pondérées si la classe est numérique) est assignée à la nouvelle instance. Cette technique est appelée la méthode des k voisins les plus proches.

Le calcul de la distance est généralement déterminé par la technique de la distance Euclidienne standard mais les attributs sont supposés être normalisés et d'importances égales. Un des principaux problèmes en l'apprentissage automatique est de déterminer quels sont les attributs qui pourraient être plus importants que d'autres; ceux-ci sont habituellement reflétés dans la métrique distance par certains types de pondération d'attributs. La dérivation adéquate des poids d'attributs à partir de l'ensemble d'apprentissage est le problème clé dans l'apprentissage basé sur les instances. L'inconvénient apparent dans les représentations basées instances est qu'elles ne construisent pas explicitement des structures apprises; les instances ne décrivent pas réellement les modèles (patterns) dans les données. Cependant, ces instances combinées avec la métrique distance permettent de distinguer une classe d'une autre, afin de découper l'espace des instances.

IBk représente l'implantation du classificateur des k voisins les plus proches qui utilise la métrique distance. Par défaut, il utilise juste le voisin le plus proche (k=1); le nombre k peut être spécifié manuellement ou déterminé automatiquement en utilisant la validation croisée « leave-one-out ». Il normalise les attributs par défaut et peut aussi pondéré les distances.

4.2 KStar (K*)

KStar est un classificateur basé instance qui classifie une instance de test selon la classe des instances, qui lui sont semblables, et ce en utilisant une fonction de similitude (Cleary, 1995). Il diffère des autres algorithmes d'apprentissage basés instance parce qu'il emploie une fonction de distance basée sur l'entropie. L'utilisation de l'entropie (Witten, 2000) comme mesure de distance a plusieurs avantages. Elle fournit une approche cohérente à la manipulation des attributs symboliques, des attributs à valeurs réelles et des valeurs manquantes. En utilisant une telle mesure, K* fournit des résultats rivalisant favorablement avec plusieurs algorithmes d'apprentissage automatique.

5 Les méta-classificateurs

Les méta-classificateurs sont des combinaisons de plusieurs classificateurs dans le but d'avoir de meilleures performances. Intuitivement, la combinaison de plusieurs modèles aide seulement quand ces modèles sont différents l'un de l'autre de manière significative et lorsque chacun traite correctement un pourcentage raisonnable de données. Idéalement, les modèles se complémentent, chacun étant un spécialiste dans une partie du domaine dans laquelle les autres ne produisent pas de bons résultats.

Dans ce cadre, nous présentons quatre de ces algorithmes à savoir : *AdaBoost.M1*, *Bagging*, *Stacking* et *Vote*.

5.1 Bagging

Le terme *Bagging* ("boostrap aggregating") a été inventé par Breiman (1996) en définissant les propriétés de *Bagging* d'un point de vue théorique et d'un point de vue empirique pour la classification et la prédiction numérique. Ce classificateur permet de combiner les décisions des différents modèles regroupant les diverses sorties en une seule prédiction. La façon la plus facile de combiner des modèles, dans la classification, est de prendre un vote ou éventuellement un vote pondéré. Dans le cas de la prédiction numérique, c'est le calcul de la moyenne ou peut être une

moyenne pondérée. Dans ce dernier cas, l'algorithme *Bagging* assigne aux modèles des poids égaux. Un autre point important est que *Bagging* exploite l'instabilité inhérente dans les algorithmes d'apprentissage en la neutralisant par la simulation du processus de décomposition *biais-variance* en utilisant un ensemble d'apprentissage donné. Il est à noter que pour les inducteurs (classificateurs de base) stables, ceux dont la sortie est insensible aux petits changements dans l'entrée, *Bagging* ne fonctionne pas.

En effet, la combinaison des hypothèses multiples peut être vue à travers le processus de décomposition *biais-variance* (Witten, 2000). Les erreurs sont usuellement présentes puisque aucun classificateur n'est parfait. Le taux d'erreurs pour un algorithme d'apprentissage particulier constitue la première erreur et est appelé *biais*. Il mesure le taux d'appariement correspondant de la méthode d'apprentissage. La seconde erreur dans le modèle appris, dans la pratique, provient de l'ensemble d'apprentissage utilisé. Cette erreur, sur tous les ensembles d'apprentissage et les ensembles de test, est appelée *la variance* de la méthode d'apprentissage. L'erreur totale d'un classificateur est composée de la somme des biais et de la variance d'où le nom de la *décomposition biais-variance*.

Dans ce processus, un grand nombre d'ensembles d'apprentissage indépendants et de même taille, utilisés pour générer un certain nombre de classificateurs, est supposé être disponible et une seule réponse est déterminée par le vote majoritaire. À chaque itération du processus, la méthode *Bagging* utilise un ensemble de données échantillonné, de même taille que l'ensemble de données d'apprentissage d'origine avec la réplication de certaines instances et la suppression d'autres.

Bagging peut être aussi appliqué à des schémas d'apprentissage pour la prédiction numérique basés sur la moyenne des prédictions individuelles. Le *biais* est défini comme l'erreur des moindres carrées sur la moyenne de tous les modèles construits, tandis que la variance est la composante de l'erreur d'un seul modèle, qui est due à l'ensemble d'apprentissage particulier.

5.2 AdaBoost.M1

La méthode *Boosting* recherche explicitement des modèles qui se complémentent pour combiner de modèles multiples (Witten, 2000). Cette méthode présente des similarités avec les méthodes *Bagging*. Ces deux méthodes utilisent la technique *vote* pour la classification ou *le calcul de la moyenne* pour la prédiction, afin de combiner la sortie des modèles individuellement. De même que *Bagging*, *Boosting* combine les modèles de même type, e.g., les arbres de décision. Cependant, *Boosting* est itératif alors que *Bagging* construit des modèles séparément. Dans l'algorithme *Boosting*, chaque nouveau modèle est influencé par la performance de ceux précédemment construits. La dernière différence est que la technique *Boosting* pondère la contribution du modèle par sa performance, plutôt que de donner des poids égaux à tous les modèles.

Plusieurs variantes de la méthode *Boosting* existent, parmi elles la méthode *AdaBoost.M1* conçue spécifiquement pour la classification. Elle a été développée par Freund (1996). La version utilisée dans Weka est celle avec des attributs nominaux. Comme la méthode *Bagging*, la méthode *AdaBoost.M1* peut utiliser n'importe quel algorithme d'apprentissage de classification. Par mesure de simplification, l'algorithme d'apprentissage doit être capable de manipuler des instances pondérées (le poids doit être un nombre positif). La présence des poids d'instances change la manière de calculer l'erreur par le classificateur qui correspond à la somme des poids des instances mal classifiées divisée par le poids total de toutes les instances, au lieu de la fraction des instances mal classifiées.

5.3 Stacking

Stacking présente une autre manière de combiner de multiples modèles. Contrairement aux techniques *Bagging* et *Boosting*, *Stacking* n'est pas utilisé pour combiner les modèles de même type (Witten, 2000). Il s'applique aux modèles construits par différents algorithmes d'apprentissage, e.g., inducteur à arbre de décision, un réseau Bayésien naïf et un schéma d'apprentissage basé instance. Une façon de combiner les sorties est par le vote. Cependant, le vote non pondéré a une signification que si les schémas d'apprentissage ont des performances

équivalentes. Le problème soulevé avec *vote* est la détermination du classificateur jugé correct. Une autre façon de procéder est celle de *Stacking* qui introduit la notion de méta-apprenant (learner) à la place de la procédure de *vote*. *Stacking* essaie d'apprendre quels classificateurs sont fiables en utilisant le Meta-apprenant (méta-modèle) pour découvrir la meilleure façon de combiner la sortie des apprenants de base. Les entrées du méta-modèle, appelé modèle de niveau 1, sont les prédictions des modèles de base qui sont aussi nommés modèles de niveau 0. Une instance de niveau 1 possède plusieurs attributs qui sont au nombre des modèles de niveau 0.

Pour la classification, une instance est en premier fournie aux modèles du niveau 0, et chacun de ces derniers estime une valeur classe. Ces estimés sont fournis ensuite au modèle de niveau 1 qui combine ces valeurs en la prédiction finale. Pour la construction du modèle de niveau 1, *Stacking* utilise une technique Hold-one-out. Cette technique consiste à réserver quelques instances pour former les données d'apprentissage pour le classificateur de niveau 1 et construire les classificateurs de niveau 0 à partir des données restantes. Une fois les classificateurs de niveau 0 construits, ils sont alors utilisés pour classifier les instances dans l'ensemble "hold-one-out", formant ainsi les données d'apprentissage de niveau 1.

Le fait que les classificateurs de niveau 0 n'aient pas été entraînés sur l'ensemble "hold-one-out", leur prédiction est non biaisée et ainsi les données d'apprentissage du niveau 1 reflètent avec précision la vraie performance des algorithmes d'apprentissage de niveau 0. Une fois les données du niveau 1 générées par cette procédure "hold-one-out", les classificateurs de niveau 0 peuvent être ré-appliqués pour générer les classificateurs à partir de l'ensemble global des données d'apprentissage. Ainsi, une utilisation légèrement meilleure des données est faite et de meilleures prédictions sont obtenues. Pour utiliser pleinement les données d'apprentissage, la validation croisée au niveau des classificateurs de niveau 0 est introduite avec le Stacking.

Enfin, pour le classificateur de niveau 1, n'importe quel algorithme d'apprentissage peut être appliqué. Cependant, vu que la plupart du travail est effectué par les classificateurs de niveau 0, le classificateur 1 est fondamentalement arbitraire, et il est donc conseillé de choisir un algorithme simple pour cet objectif, e.g., modèles linéaires qui fonctionnent mieux que les autres dans la pratique.

5.4 Vote

Plusieurs techniques plus simples qui visent à combiner différentes hypothèses dans une seule prédiction sont basées sur le *Vote* (Chan, 1995; Parhami, 1994). Le premier schéma d'apprentissage est la technique « vote simple », basée sur les prédictions de différents classificateurs de base. La prédiction finale est choisie comme classification avec une pluralité de votes. Une variante est le vote pondéré.

Chaque classificateur est associé à un poids, qui détermine le degré de précision du classificateur lors de la validation d'un ensemble de données où l'ensemble de validation A définit un ensemble d'instances aléatoirement choisies parmi tous les sous-ensembles. Puisque chaque classificateur est formé sur seulement un sous-ensemble, les instances des autres sous-ensembles, qui contribuent à l'ensemble de validation, fournissent une mesure de degré de prédiction. Chaque prédiction est pondérée par le poids assigné au classificateur. Les poids de chaque classification sont additionnés et la prédiction finale est la classification ayant le poids le plus grand.

Littlestone (1989) propose plusieurs algorithmes pondérés de majorité pour combiner différents classificateurs. Ces algorithmes sont semblables à la méthode de vote pondéré décrite précédemment; la différence principale est comment les poids sont obtenus. L'algorithme de base associe à chaque classificateur appris un premier poids. Chaque instance dans l'ensemble d'apprentissage est alors traitée par les classificateurs. La prédiction finale pour chaque instance est produite comme dans le vote pondéré. Si la prédiction finale est erronée, les poids des classificateurs, dont les prédictions sont incorrectes, sont multipliés par un nombre compris entre 0 et 1 diminuant ainsi leur contribution aux prédictions finales.

6 Classificateurs divers (misc)

Cette catégorie d'algorithmes d'apprentissage ne peut être utilisée que dans l'ancienne version de Weka (version 3.4) - la nouvelle version 3.4.6 n'inclut pas cette catégorie d'algorithmes.

6.1 Fuzzy Lattice Reasoning Classifier (FLR)

Un treillis (Lattice) est défini comme un ensemble partiellement ordonné. N'importe quel couple d'éléments de cet ensemble a une plus grande limite inférieure dénotée "meet" et une plus petite limite supérieure dénotée "join". Un tel ordonnancement est appelé ordonnancement de treillis (Petridis, 1999). Un treillis est nommé complet lorsque chacun de ses sous ensembles a une limite inférieure plus grande et une limite supérieure plus petite.

Un treillis complet non vide contient un plus petit élément et un plus grand élément dénotés respectivement "*O*" et "*I*". Dans d'autres ouvrages, d'autres symboles sont adoptés pour représenter le plus petit élément et le plus grand élément.

Par conséquent, un treillis est noté par une paire ordonnée $(A, \leq_A)$ où A est l'ensemble considéré et $\leq_A$ une relation binaire d'ordonnancement de treillis sur l'ensemble A; $\leq_A$ est un sous ensemble de AxA selon les lois de transitivité, d'antisymétrie et de réflexivité. Deux treillis partageant le même ensemble, appelé A, sont distingués par l'indice de leur relation d'ordre, exemple $(A, \leq_{A1})$ et $(A, \leq_{A2})$. Un treillis $(A, \leq_A)$ est nommé alors treillis conventionnel. Les opérations 'meet' et 'join' dans un treillis conventionnel sont notées respectivement $\vee_A$ et $\wedge_A$.

La version, implantée au niveau de l'outil Weka, peut être utilisée pour la classification en utilisant des prédicateurs numériques.

6.2 HyperPipes

Dans Weka, le classificateur *HyperPipe* construit, pour chacune des classes d'instances, un *HyperPipe* contenant tous les points de cette classe. Il enregistre essentiellement les limites d'attribut observées pour chaque classe d'instances. Des instances de test sont classifiées selon la classe qui contient le plus d'instances (Kalapanidas, 2003). Ce classificateur ne permet pas la manipulation des classes numériques, ou les valeurs manquantes dans des instances de test. L'algorithme est extrêmement simple et à l'avantage d'être extrêmement rapide. Il fonctionne tout à fait bien en présence d'un grand nombre d'attributs.

6.3 VFI (Voting Feature Interval)

VFI (Dermiröz, 1997) est un classificateur basé sur le concept d'intervalle de vote d'attributs. Il est assez semblable à *HyperPipes*. À partir de l'ensemble d'apprentissage, l'algorithme *VFI* construit des intervalles pour chaque attribut. Un intervalle est soit une échelle soit un intervalle de point. Pour des intervalles de point, seule une valeur simple est employée pour définir cet intervalle. Pour des intervalles d'échelle, il suffit de maintenir seulement la limite inférieure pour l'échelle des valeurs, puisque tous les intervalles d'échelle sur une dimension d'attributs sont linéairement ordonnés. Pour chaque intervalle, une seule valeur et les votes de chaque classe dans cet intervalle sont retenus. Ainsi, un intervalle peut représenter plusieurs classes en stockant le vote pour chaque classe. La classification d'une nouvelle instance est basée sur un vote parmi les classifications faites par la valeur de chaque attribut séparément.

Dans la phase d'apprentissage, sont construits les intervalles d'attributs pour chaque attribut. Les votes de chaque classe pour un intervalle d'attributs sont déterminés dans la phase de classification.

Les algorithmes *VFI* considèrent aussi chaque attribut séparément comme dans le cas du classificateur Bayésien naïf. Le schéma vote est utilisé dans la classification pour combiner les classifications individuelles de chaque attribut. Le vote final est la somme de tous les votes individuels donnés par les attributs. Cette méthode a pour avantages la vitesse de classification et la manipulation des valeurs manquantes d'attributs.

7 Classificateurs basés sur les arbres de décision

Un arbre de décision est un arbre dans lequel chaque noeud représente un choix entre un certain nombre de solutions alternatives, et chaque noeud feuille représente une classification ou une décision.

Dans cette section, nous traitons les algorithmes suivants : *ADTree*, *J48* et *NBTree*.

7.1 ADTree (arbres alternatifs de décision)

Les résultats de l'application de la technique Boosting aux algorithmes d'arbre de décision ont montré que les classificateurs produits sont d'une grande précision. Ces classificateurs sont sous la forme de vote majoritaire sur un nombre d'arbres de décision. Malheureusement, ces classificateurs sont souvent de grande taille, complexes et difficiles à interpréter.

Une nouvelle combinaison des arbres de décision avec Boosting, *ADTree* (les arbres alternatifs de décision), génère des règles de classification qui sont souvent plus petites et plus faciles à interpréter que les règles produites en utilisant *C5.0* avec Boosting (Holmes, 2002). Les expérimentations ont montré qu'*ADTree* donnent des performances semblables à celles de *C5.0* avec Boosting.

Un arbre de décision simple est un graphe constitué de 1 ou plusieurs noeuds de décision et de 1 ou plusieurs feuilles de prédiction. Pour construire un arbre de décision alternatif, chaque nœud d'un arbre de décision simple est remplacé par un nœud de prédiction et un nœud séparateur. Un nœud séparateur est de même nature qu'un noeud de décision alors qu'un nœud de prédiction est un nombre réel.

Essentiellement, un arbre *ADTree* est un graphe AND/OR. La connaissance contenue dans l'arbre est distribuée comme de multiples chemins qui doivent être parcourus pour former des prédictions. Les instances, qui satisfont les multiples nœuds séparateurs, ont les valeurs de prédiction des nœuds atteints sommées pour former une valeur totale de prédiction. Une somme positive représente une classe et une somme négative représente l'autre classe dans le cas de deux classes.

La version actuelle de *ADTree*, implantée au niveau de Weka, supporte seulement des problèmes à deux classes. L'induction des arbres a été optimisée et des méthodes heuristiques de recherche ont été introduites pour rendre l'apprentissage plus rapide.

7.2 J48

L'algorithme *J48* implante la méthode *C4.5* conçue par Quinlan (1986) dans le cadre de l'apprentissage supervisé. Cet algorithme permet la génération d'un arbre de décision, élagué ou non. Cette méthode a pour but global de générer un arbre de décision simple (et petit) capable de classifier les nouvelles instances. À partir de l'ensemble d'apprentissage, *C4.5* extrait la régularité de règles à partir d'instances et construit un arbre de décision qui classifiera les instances avec un certain degré d'erreur tolérée.

Pour une instance donnée, les noeuds de l'arbre de décision sont utilisés pour tester les valeurs d'attribut. Suivre une branche ou une autre d'un nœud donné dépend des résultats du test effectué au niveau du dit nœud.

À la fin du parcours de l'arbre de décision de la racine à une feuille selon les valeurs d'un problème donné, la classification trouvée à la feuille est la classification prédite du problème. Pour la construction des sous-arbres, *C4.5* utilise le taux de gain d'information (IGR : information gain rate) pour chacun des attributs possibles qui pourraient potentiellement être utilisés pour diviser les données. L'IGR (Witten, 2000) est une heuristique qui évalue la capacité d'un attribut de réduire l'aspect aléatoire dans des instances non classifiées. L'attribut avec le plus grand IGR est choisi comme racine d'un sous-arbre. Bien qu'IGR soit une métrique qui prend une décision locale par opposition à une décision globalement optimale, l'attribut sélectionné est souvent le plus discriminatoire. Cette méthode de construction de sous-arbres est appliquée récursivement jusqu'à ce que l'arbre résultant classifie entièrement toutes les instances d'apprentissage.

Les arbres de décision ont été utilisés comme classificateurs pour de nombreux domaines, e.g., la médecine. Pour plusieurs de ces domaines, les arbres produits par *C4.5* sont petits et précis, ayant pour résultat des classificateurs rapides et fiables. Ces propriétés font des arbres de décision un outil valable et populaire pour la classification.

Pour avoir un classificateur fiable, simple, petit et rapide, l'arbre de décision doit être élagué. En effet, la plupart des algorithmes d'arbre de décision utilisent une méthode "élagage", qui signifie qu'ils génèrent un arbre de grande taille et ensuite suppriment une certaine partie de cet arbre. Une méthode alternative consiste à arrêter la construction de l'arbre une fois que l'ensemble d'apprentissage ait été suffisamment subdivisé en utilisant un critère.

La méthode d'élagage de *C4.5* est basé sur l'estimation du taux d'erreur de chaque sous-arbre, et remplace le sous-arbre avec un noeud feuille si l'erreur estimée de la feuille est très basse.

7.3 NBTree (Naïve-Bayes decision-Trees)

Cet algorithme permet de générer un arbre de décision avec des classificateurs Bayésiens naïfs au niveau des feuilles. Kohavi (1996) propose *NBTree* comme une approche hybride combinant le classificateur Bayésien naïf et le classificateur d'arbre de décision. Ce classificateur hybride *NBTree* obtient fréquemment une très haute précision par rapport au classificateur Bayésien naïf ou au classificateur d'arbre de décision. Il utilise une structure arborescente pour diviser l'espace d'instances en sous-espaces et générer un classificateur Bayésien naïf pour chaque sous-espace.

Les noeuds des arbres de décision contiennent les tests à une seule variable. Chaque feuille d'un arbre Bayésien contient un classificateur Bayésien naïf local qui ne considère pas les attributs impliqués dans les tests au niveau des nœuds du chemin conduisant à la feuille. Dans un arbre de décision conventionnel, chaque feuille est marqué avec une seule classe et prédit cette classe pour les instances qui atteignent la feuille alors qu'un arbre Bayésien naïf utilise un classificateur Bayésien naïf local pour prédire les classes de ces instances. Une sélection appropriée des tests peut permettre à l'arbre de factoriser les interdépendances préjudiciables d'attributs hors des classificateurs Bayésiens naïfs locaux aux feuilles.

8 Les classificateurs basés sur les règles

Nous traitons dans ce groupe les algorithmes *Conjunctive*, *PART* et *DecisionTable*.

8.1 ConjunctiveRule

Ce classificateur implante une méthode d'apprentissage (Witten, 2000) à base de règles conjonctives simples pour des classes numériques et nominales. Une règle se compose d'antécédents liés ensemble par une conjonction AND et d'une conséquence. Dans le cas de la classification, la conséquence est la distribution des classes disponibles. Elle corresponds à une moyenne pour la régression. Si une instance de test n'est pas couverte par cette règle alors la prédiction est la classe distributions/valeurs par défaut qui couvre les données non couvertes par aucune règle. Ce classificateur sélectionne un antécédent en calculant le gain de l'information de chaque antécédent et élague la règle générée en utilisant l'erreur réduite d'élagage (REP : Reduced Error Pruning) ou un pré-élagage simple basé sur le nombre d'antécédents.
Pour la classification, l'information d'un antécédent est la moyenne pondérée des entropies (Witten, 2000) des données couvertes et non couvertes par la règle. Pour la régression, l'information est la moyenne pondérée des erreurs de la moyenne carrée des données couvertes et non couvertes par la règle.

8.2 PART

PART (Frank, 1998) permet d'inférer des règles par la génération itérative d'arbres de décision partiels en combinant deux paradigmes majeurs : les arbres de décision et la technique d'apprentissage des règles "diviser pour régner". En effet, il existe deux approches pour générer des règles soit à partir d'arbre de décision ou en utilisant la technique 'diviser pour régner'.

Le processus classique suivi par l'algorithme *C4.5* pour générer des règles est complexe et consomme beaucoup de temps. Il est composé de cinq phases pour aboutir à la génération de l'ensemble de règles. Ces phases sont : la construction de l'arbre de décision, la transformation en un ensemble de règles, la simplification de chacune des règles séparément, les règles pour chaque classe sont ensuite considérées et le bon sous-ensemble est recherché selon un critère basé sur le principe de « longueur minimum de description ». L'étape suivante est d'ordonner les sous-ensembles pour les différentes classes tout en évitant d'éventuels conflits. La classe par défaut est déterminée. Finalement, les règles sont supprimées de l'ensemble des règles une par une, tant que cela réduit l'erreur sur l'ensemble des règles de l'ensemble d'apprentissage.

Les algorithmes "diviser pour régner" représentent une approche plus directe aux règles de décision. Ils génèrent une règle à la fois, suppriment les instances couvertes par cette règle, et induisent itérativement plus de règles pour les instances restantes. Bien qu'à l'origine formulée pour des problèmes à deux classes, cette méthode peut être appliquée directement aux problèmes multi-classes par la construction des règles, séparément pour chaque classe, puis leur ordonnancement de manière appropriée.

À la différence des deux approches, PART n'a pas besoin d'effectuer une optimisation globale pour produire des ensembles de règles précis, ce qui apporte plus de simplicité. Il adopte la stratégie "diviser pour régner" parce qu'il construit une règle, supprime les instances couvertes par cette règle, et continue de créer des règles récursivement pour les instances restantes jusqu'à ce qu'il n'en reste aucune.

Il diffère de l'approche standard dans la manière où chaque règle est créée. Essentiellement, pour construire une seule règle, un arbre de décision élagué est construit pour l'ensemble courant d'instances. La feuille avec la plus grande couverture est transformée en une règle et l'arbre est abandonné. Ceci évite la généralisation hâtive, en généralisant seulement une fois que les implications soient connues (c'est-à-dire tous les sous-arbres ont été développés).

En utilisant un arbre élagué pour obtenir une règle, au lieu de la construire de manière incrémentale en ajoutant des conjonctions une par une, le problème de sur-élagage de l'algorithme d'apprentissage basé sur le principe "diviser pour régner" est évité. La combinaison de la méthodologie "diviser pour régner" et les arbres de décision ajoute de la flexibilité et de la vitesse. Il est, en effet, inutile de construire un arbre de décision plein pour obtenir une règle simple. Le processus peut être accéléré sensiblement sans sacrifier les avantages ci-dessus. L'idée principale est de construire un arbre de décision *partiel* au lieu d'un arbre entièrement exploré. Un arbre de décision partiel est un arbre de décision ordinaire qui contient des branches aux sous-arbres non définis.

PART fournit des résultats aussi précis que ceux de l'algorithme *C4.5*. Il permet d'éviter l'optimisation globale de *C4.5* et fournit un ensemble de règles compactes et exactes.

8.3 DecisionTable

Cette approche a pour but de construire et d'utiliser un classificateur table de décision à majorité. Les tables de décision sont l'un des espaces d'hypothèses les plus simples possibles, et habituellement elles sont faciles à comprendre. Les résultats expérimentaux montrent que pour des problèmes réels et artificiels contenant seulement des attributs discrets, *IDTM* (inducing decision tables) peut parfois surpasser des algorithmes très connus tel que *C4.5*.

Une des représentations les plus simples possibles est une table de décision avec une règle par défaut appariant la classe majorité. Cette représentation, appelée *DTM* (Tableau de décision à majorité) a deux composants : un schéma qui est un ensemble d'attributs inclus dans la table, et un corps se composant d'instances étiquetées de l'espace défini par les attributs dans le schéma (Kohavi, 1995). Soit une instance non étiquetée, un classificateur de table de décision recherche les instances correspondantes exactes dans la table de décision en utilisant seulement les attributs dans le schéma. Si aucune instance n'est trouvée, la classe de majorité du *DTM* est retournée, sinon, la classe majorité de toutes les instances correspondantes est retournée. Pour construire un *DTM*, l'algorithme d'induction doit décider quels attributs sont à inclure dans le schéma et quelles instances sont à stocker dans le corps.

9 Conclusion

À travers ce document, nous avons essayé de présenter les différents algorithmes d'apprentissage qui sont offerts par l'environnement expérimental Weka. En effet, cet environnement renferme un vaste choix d'algorithmes d'apprentissage variant des algorithmes traditionnels (régression, réseau Bayésiens, etc.) aux plus récents les algorithmes *SVM*, par exemple. Ces algorithmes ont été répartis sur un certain nombre de catégories à savoir les réseaux Bayésiens, les classificateurs 'function', les classificateurs paresseux, les méta-classificateurs, les classificateurs basés sur les arbres de décision et enfin les classificateurs basés sur les règles. Ces algorithmes peuvent être utilisés pour résoudre des problèmes de classification ou de régression en mode supervisé.

10 Références

Alpaydin, E. (2004). *Introduction to Machine Learning*. MIT Press, 2004.

Breiman, L. (1996). *Bagging predictors*. Journal Machine Learning, 24(2):123-140.

Breiman, L., Friedman, J.H., Olshen, R.A. et Stone, C.J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.

Chan, P.K., Stolfo, S.J. (1995). *A Comparative Evaluation of Voting and Meta-learning on Partitioned Data*. (1995), International Conference on Machine Learning.

Cleary, J.G., Trigg, L.E. (1995). K* : An Instance-based Learner Using an Entropic Distance Measure. In proceeding of International Conference on Machine Learning, (1995) 108114.

Cornuéjols, A., Miclet, L. et Kodratoff Y. (2002). *Apprentissage Artificiel : Concepts et algorithmes*. Édition Eyrolles, 2002.

Curvefit, (1999). *Linear regression*. http://www.curvefit.com/linear_regression.htm.

Demiröz, G., Güvenir, H.A. (1997). *Classification by Voting Feature Intervals*. European Conference on Machine Learning.

Frank, E., Witten, I.H. (1998). *Generating Accurate Rule Sets Without Global* Optimization. In Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

Freund, Y., Schapire, R.E. (1996). *Experiments with a new boosting algorithm*. Proc International Conference on Machine Learning, pages 148-156, Morgan Kaufmann, San Francisco.

Haykin, S. (1994). *Neural Networks : A Comprehensive foundation*. IEEE Computer Society Press, 1994.

Holmes, G., Pfahringer, B., Kirkby, R., Frank, E., Hall, M. (2002). *Multiclass Alternating Decision Trees*. ECML, 2002.

John, G.H., Langley, P. (1995). *Estimating Continuous Distributions in Bayesian Classifiers*. In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Mateo, 1995.

Kalapanidas, E., Avouris, N., Craciun, M., Neagu, D. (2003). *Machine Learning algorithms: a study on noise sensitivity.* Proceedings of the 1st Balkan Conference on Informatics BCI 2003 (Y.

Manolopoulos, P. Spirakis eds.), 356-365, 21-23 November 2003, ISBN 960-287-045-1, Springer-Verlag., Thessaloniki, Greece.

Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K. (2001). *Improvements to Platt's SMO Algorithm for SVM Classifier Design*. Neural Computation, 13(3), pp 637-649, 2001.

Kohavi, R. (1996). *Scaling up the accuracy of naive-Bayes classifiers: a decision tree hybrid*. Procedings of the Second Internaltional Conference on Knoledge Discovery and Data Mining, (1996).

Littlestone, N. Warmuth, M.K., (1989). *The weighted majority algorithm*. In 30th Annual Symposium on Foundations of Computer Science, pages 256-261, October 1989.

Orr, M.J.L. (1996). *Introduction to Radial Basis Function Networks*. Technical Report. 1996. www.anc.ed.ac.uk/~mjo/intro.html.

Parhami, B. (1994). *Voting Algorithms*. IEEE Trans. On Reliability, Vol. 43, No. 4, 1994 December.

Petridis, V., Kaburlasos V.G. (1999). *Learning in the Framework of Fuzzy Lattices*. In the IEEE Transactions on Fuzzy Systems, vol. 7, no. 4, pp. 422-440, 1999.

Platt, J. (1998). *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. Advances in Kernel Methods - Support Vector Learning. Éditeurs Schölkopf, B., Burges, C., Smola, A. MIT Press.

Quinlan, R. (1986). *Induction of decision trees*. Machine Learning. Vol.1, No.1, pp. 81-106, 1986.

StatSoft, Inc. (2004). *Electronic Statistics Textbook*. Tulsa, OK: StatSoft. http://www.statsoft.com/textbook/stathome.html.

Weisstein, E.W. (2005). *K-Means Clustering Algorithm*. From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html.

Witten, I.H., Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation*. Morgan Kaufmann Publishers, San Francisco, California, 2000.