Classification incrémentale supervisée : un panel introductif

Christophe Salperwyck[*,**], Vincent Lemaire[*]

[*]Orange Labs
2, Avenue Pierre Marzin 22300 Lannion
prenom.nom@orange.ftgroup.com
[**] LIFL (UMR CNRS 8022) - Université de Lille 3
Domaine Universitaire du Pont de Bois
59653 Villeneuve d'Ascq Cedex

Résumé. Les dix dernières années ont été témoin du grand progrès réalisé dans le domaine de l'apprentissage statistique et de la fouille de données. Il est possible à présent de trouver des algorithmes d'apprentissage efficaces et automatiques. Historiquement les méthodes d'apprentissage faisaient l'hypothèse que toutes les données étaient disponibles et pouvaient être chargées en mémoire pour réaliser l'apprentissage. Mais de nouveaux domaines d'application de la fouille de données émergent telles que : la gestion de réseaux de télécommunications, la modélisation des utilisateurs au sein d'un réseau social, le web mining... La volumétrie des données explose et il est nécessaire d'utiliser des algorithmes d'apprentissage incrémentaux. Cet article a pour but de présenter les principales approches de classification supervisée incrémentale recensées dans la littérature. Il a pour vocation de donner à un lecteur débutant des indications de lecture sur ce sujet; sujet qui connaît déjà des applications industrielles.

1 Introduction

Les dix dernières années ont été témoin des grands progrès réalisés dans le domaine de l'apprentissage automatique et de la fouille de données. Ces techniques ont montré leurs capacités à traiter des volumétries importantes de données et ce sur des problèmes réels (Guyon et al., 2009; Féraud et al., 2010). Néanmoins le plus important effort a été réalisé pour des analyses de données homogènes et stationnaires et à l'aide d'algorithmes centralisés. La plupart des approches d'apprentissage automatique supposent que les ressources sont illimitées, par exemple que les données tiennent en mémoire vive. Dans ce contexte les algorithmes classiques d'apprentissage utilisent des bases d'apprentissage de taille finie et produisent des modèles statiques. Cependant la volumétrie des données continue de croître et ce plus vite que les capacités de traitement.

De nouveaux domaines d'application de la fouille de données émergent où les données ne sont plus sous la forme de tableaux de données persistants mais plutôt sous la forme de données "passagères". Parmi ces domaines on citera : la gestion de réseaux de télécommunications, la modélisation des utilisateurs au sein d'un réseau social, le web mining... Le défi scientifique principal est alors d'automatiser l'ensemble des étapes du processus d'apprentissage mais aussi

les étapes nécessaires au déploiement du modèle et ce sur des volumétries très larges. L'un des défis techniques est de concevoir des algorithmes permettant ce passage à l'échelle. Parmi l'ensemble des axes d'études, qui permettent ce passage à l'échelle, l'apprentissage incrémental semble répondre naturellement à cette problématique.

Lorsque le terme d'apprentissage incrémental est utilisé en informatique il fait référence aux algorithmes permettant d'entraîner un modèle de manière incrémentale. Au fur et à mesure que de nouvelles informations (données) sont présentées à l'algorithme, celui-ci apprend et ce sans avoir besoin de réapprendre le modèle à partir de zéro ni même à avoir à stocker l'ensemble des données.

Cet article a pour but de présenter pour différents types de classifieurs supervisés une liste de lectures introductives qui présente les principales approches de classification supervisée incrémentale recensées dans la littérature. Il a pour vocation à donner à un lecteur débutant des bases sur ce sujet ; sujet qui connaît déjà des applications industrielles (Almaksour et al., 2009; Saunier et al., 2004).

La section (Section 2) qui suit cette introduction a pour but de situer l'apprentissage incrémental parmi les différentes familles d'algorithmes d'apprentissage existantes mais aussi de poser un certain nombre de définitions, et notions utiles pour la compréhension de cet article. Les notations utilisées dans la suite du document sont également détaillées dans cette section.

La troisième section décrit les principaux algorithmes d'apprentissage incrémentaux classés par typologie de classifieurs (arbre de décision, SVM...) ; ces derniers n'étant pas tous naturellement incrémentaux. On se place ici principalement dans le cadre où les données sont stockées seulement sur disque mais ne peuvent être toutes placées en mémoire de manière à utiliser un algorithme hors-ligne (voir section 2.1.1).

Lorsque les données ne sont plus stockables sur disque il est alors impératif de travailler directement sur le flux entrant des données. Dans ce cas une adaptation des algorithmes d'apprentissage incrémentaux est requise. Ces adaptations sont décrites au cours de la section 4 pour les principales techniques de classification.

L'intérêt croissant de la communauté pour l'apprentissage incrémental ces dernières années (on peut en juger par le nombre croissant de workshop ou de conférences organisés sur ce sujet) a produit de nombreux algorithmes mais aussi différents indicateurs de comparaison entre algorithmes. Le but de la section 5 est de présenter les métriques et/ou indicateurs de comparaison les plus communément utilisés. Enfin avant de conclure il n'est pas possible de parler d'apprentissage incrémental sans consacrer une section à la notion de contexte. C'est le but de la section 6.

2 Préambule

2.1 Hypothèses et contraintes

En matière d'apprentissage automatique, il existe différentes hypothèses ou contraintes qui portent à la fois sur les données et sur le type de concept que l'on cherche à modéliser. Cette partie en décrit quelques unes dans le contexte de l'apprentissage incrémental de manière à comprendre les approches présentées dans les sections 3 et 4 de cet article.

Dans la suite de cet article on s'intéresse aux problèmes de classification binaire. On appelle exemple une observation x. L'espace de tous les exemples possibles est noté $\mathcal{X}$. Les exemples sont supposés être indépendants et tirés aléatoirement au sein d'une distribution de

probabilité notée $\mathcal{D}_{\mathcal{X}}$, et être munis de leur étiquette ($y \in \{-1, +1\}$). On appelle modèle un classifieur (f) qui détermine la classe qui doit être assignée à un exemple x ($f : \mathcal{X} \rightarrow \{-1, +1\}$. Chaque exemple est décrit par un ensemble d'attributs.

2.1.1 Sur les exemples d'apprentissage

Les modèles que l'on cherche à apprendre sont basés sur des exemples représentatifs d'un problème de classification (dans le cadre de cet article). L'algorithme d'apprentissage utilise ces exemples de manière à entraîner le modèle. Mais ces exemples sont plus ou moins disponibles : tous dans une base de données, tous en mémoire, partiellement en mémoire, un par un dans un flux... On trouve dans la littérature différents types d'algorithmes selon les types de disponibilité d'exemples différents (peu d'exemples, beaucoup, énormément et en continu).

Le cas le plus simple correspond à avoir un modèle basé sur une quantité d'exemples représentatifs qui peuvent être chargés en mémoire et exploités directement.

Dans d'autres cas, la quantité d'exemples est très importante et il est impossible de tous les charger en mémoire. Il faut donc concevoir un algorithme qui puisse générer un modèle sans avoir besoin que tous les exemples soient en mémoire. On peut alors chercher à découper les données en plusieurs petits ensembles (chunks) de manière à pouvoir les traiter les uns après les autres et ne pas avoir à tout charger en mémoire et / ou à utiliser des techniques de parallélisation de l'algorithme d'apprentissage. Le lecteur trouvera dans (Provost et Kolluri, 1999) une étude sur les méthodes utilisées pour traiter cette problématique de volumétrie : parallélisation, partitionnement de l'espace...

Dans le pire des cas, les données sont très volumineuses et arrivent de manière continue, on parle alors de flux de données. Les exemples ne peuvent être vus qu'une seule fois et dans l'ordre dans lequel ils arrivent. L'algorithme doit réaliser l'apprentissage très rapidement afin de ne pas ralentir le flux de données.

2.1.2 Sur la disponibilité du modèle

La mise en place d'un modèle de prédiction se réalise en deux étapes : 1) une étape d'apprentissage du modèle, 2) une étape d'exploitation du modèle. Dans le cas d'un apprentissage non incrémental ces étapes sont réalisées l'une après l'autre mais dans le cas de l'apprentissage incrémental, on repasse en phase d'apprentissage dès qu'un nouvel exemple arrive. Cette phase peut être plus ou moins longue et il est parfois nécessaire de pouvoir en maîtriser la complexité calculatoire. (Dean et Boddy, 1988) s'intéresse à cette problématique et définit le concept d'algorithme de prédiction anytime. C'est un algorithme (i) qui est capable d'être arrêté à tout moment et de fournir une prédiction ; et (ii) dont la qualité de la prédiction est proportionnelle au temps consommé.

2.1.3 Sur le concept

Supposons un problème de classification supervisée où l'algorithme d'apprentissage observe une séquence d'exemples munis de leurs étiquettes. La classe d'appartenance des exemples suit une loi de probabilité notée $P_{\mathcal{Y}}$. On appelle concept cible pour un exemple x_i la probabilité jointe $P(x_i, y_i) = P(x_i)P(y_i|x_i)$.

Le caractère non stationnaire du problème à résoudre peut être présent (principalement) sous deux principales formes de dérives.

Le concept cible n'est pas toujours constant dans le temps, parfois il se modifie ou dérive, on parle alors de dérive de concept (Michalski et al., 1986). Gama dans (Gama, 2010) divise cette dérive en deux sous catégories : soit elle est douce on parle alors de dérive de concept ("concept drift"), soit elle est abrupte et on parle alors de saut de concept ("concept shift"). Ces deux types de dérives correspondent à un changement de la probabilité $P(Y|X)$ au cours du temps. Les algorithmes de la littérature peuvent être classés selon qu'ils supportent ou non le changement de concept.

La distribution des données peut varier au cours du temps ($P(X)$) sans modification des $P(Y|X)$, on parle alors de covariate shift (Joaquin Quinonero-Candela et al., 2009). Le "covariate shift" apparaît aussi dans le cas où l'on effectue une sélection d'exemples non i.i.d. comme par exemple une sélection de données d'apprentissage artificiellement équilibrée (mais non équilibré dans le jeu de test) ou encore dans le cadre de l'apprentissage actif (Bondu et Lemaire, 2008). Il existe un débat sur cette notion de covariate shift, on peut en effet supposer que la distribution sous-jacente des exemples (D_X) ne change pas mais que ce sont que les exemples que l'on observe effectivement qui changent.

Dans la suite de cet article et en particulier dans la section 6 on s'intéressera principalement à la dérive de $P(Y|X)$. On supposera, étant donné un contexte, qu'il n'y a pas de covariate shift. Le lecteur intéressé trouvera dans (Joaquin Quinonero-Candela et al., 2009) et (Cornuéjols, 2009) des éléments sur le sujet du covariate shift.

Gama dans (Gama, 2010) traite par ailleurs de la notion de contexte. Un contexte est défini par un ensemble d'exemples pour lesquels il n'y a pas de dérive de concept. Un flux de données peut donc être vu comme une séquence de contexte. Etre capable de traiter ce flux consiste alors à être capable de détecter les dérives de concepts et/ou d'être capable de travailler avec plusieurs contextes en simultané (voir Section 6).

2.1.4 Questions à se poser

Les différents paragraphes ci-dessus montrent qu'il est nécessaire, lors de la mise en place d'un système basé sur un classifieur supervisé, de se poser certaines questions :

- Les exemples peuvent-ils être stockés en mémoire ?
- Quelle est la disponibilité des exemples : tous présents ? en flux ? visibles une seule fois ?
- Le concept est-il stationnaire ?
- L'algorithme doit-il être anytime ?
- Quel est le temps disponible pour mettre à jour le modèle ?

Les réponses à ces questions doivent permettre de sélectionner les algorithmes adaptés à la situation et de savoir si on a besoin d'un algorithme incrémental, voire d'un algorithme spécifique aux flux.

2.2 Les familles d'algorithmes d'apprentissage

La partie précédente a présenté les différentes contraintes par rapport aux exemples qui peuvent parfois arriver en continu, en quantité et vitesse importantes. Selon ces différents cas de figures il existe différents types d'algorithmes d'apprentissage qui peuvent être utilisés.

2.2.1 Apprentissage hors ligne

L'apprentissage hors ligne correspond à l'apprentissage d'un modèle sur un jeu de données représentatif du problème et disponible au moment de l'apprentissage. Ce type d'apprentis-

sage est réalisable sur des volumes de taille faible à moyenne (jusqu'à quelques Go). Au delà le temps d'accès et de lecture des données devient prohibitif et il devient difficile de réaliser un apprentissage rapide (qui ne prennent pas des heures ou des jours). Ce type d'algorithme montre ses limites dans le cas où (i) les données ne sont pas entièrement chargeables en mémoire ou arrive de manière continue ; (ii) la complexité calculatoire de l'algorithme d'apprentissage est supérieure à une complexité dite quasi-linéaire. L'apprentissage incrémental est bien souvent une alternative intéressante face à ce genre de problème.

2.2.2 Apprentissage incrémental

L'apprentissage incrémental correspond à un système capable de recevoir et d'intégrer de nouveaux exemples sans devoir réaliser un apprentissage complet. Un algorithme d'apprentissage est incrémental si, pour n'importe quels exemples $x_1, ..., x_n$ il est capable de produire des hypothèses $f_1, ..., f_n$ tel que f_{i+1} ne dépend que de f_i et de l'exemple courant x_i. Par extension de la définition la notion "d'exemple courant" peut être étendu à un résumé des derniers exemples vus, résumé utile à l'algorithme d'apprentissage utilisé. Les propriétés désirées d'un algorithme incrémental sont un temps d'apprentissage beaucoup plus rapide par comparaison à l'apprentissage hors ligne. Pour atteindre cet objectif les algorithmes ne lisent souvent qu'une seule fois les exemples ce qui permet en général de traiter de plus grandes volumétries.

2.2.3 Apprentissage en ligne

Le qualificatif "en ligne" est ajouté lorsque que l'arrivée des exemples se fait de manière continue pour réaliser cet apprentissage et que l'algorithme est capable de fournir un modèle intégrant ce nouvel exemple. Les exigences en termes de complexité calculatoire sont plus fortes que pour l'apprentissage incrémental. Par exemple on cherchera à obtenir une complexité calculatoire constante ($O(1)$) si l'on désire réaliser un algorithme d'apprentissage à partir de flux. Il s'agit d'apprendre et de prédire à la vitesse du flux. Bien souvent s'ajoutent à cette différence essentielle des contraintes de mémoire et des problèmes de dérive de concept.

2.2.4 Apprentissage anytime

La définition de l'apprentissage anytime est ici restreinte au fait d'apprendre le meilleur modèle possible (étant donné un critère d'évaluation) jusqu'à une interruption (qui peut être l'arrivée d'un nouvel exemple). La famille des algorithmes par contrat est assez proche de celle des algorithmes anytime. (Zilberstein et Russell, 1996) proposent un algorithme s'adaptant aux ressources (temps / processeur / mémoire) qui lui sont passées en paramètres.

3 Apprentissage incrémental

3.1 Introduction

De nombreux algorithmes d'apprentissage sont adaptés au problème de la classification supervisée. Sans tous les citer il existe : les Séparateurs à Vastes Marges (SVM), les réseaux de neurones, les méthodes des k plus proches voisins, les arbres de décision, la régression logistique, l'analyse discriminante linéaire, etc. L'utilisation de tel ou tel algorithme dépend fortement de la tâche à résoudre et du désir d'interprétabilité du modèle. Dans cet article il

est impossible de tous les passer en revue. Les sections ci-dessous se concentrent sur les algorithmes les plus utilisés dans le cadre de l'apprentissage incrémental.

3.2 Arbre de décision

Un arbre de décision (Quinlan, 1986; Breiman et al., 1984) est un modèle de classification (pour cet article) présenté sous la forme graphique d'un arbre. L'extrémité de chaque branche est une feuille qui présente le résultat obtenu en fonction des décisions prises à partir de la racine de l'arbre jusqu'à cette feuille. Les feuilles intermédiaires sont appelées des nœuds. Chaque nœud de l'arbre contient un test sur un attribut qui permet de distribuer les données dans les différents sous-arbres. Lors de la construction de l'arbre un critère de pureté comme l'entropie (utilisé dans C4.5) ou Gini (utilisé dans CART) est utilisé pour transformer une feuille en nœud. L'objectif est de produire des groupes d'individus les plus homogènes possibles du point de vue de la variable à prédire (pour plus de détails voir par exemple (Cornuéjols et Miclet, 2010) chapitre 13). En prédiction, un exemple à classer "descend" l'arbre depuis la racine jusqu'à une unique feuille. Son trajet dans l'arbre est entièrement déterminé par les valeurs de ses attributs. Il est alors affecté à la classe dominante de la feuille avec pour score la proportion d'individus dans la feuille qui appartiennent à cette classe.

Les arbres de décision possèdent les avantages suivant : (i) la lisibilité du modèle, (ii) la capacité à trouver les variables discriminantes dans un important volume de données. Les algorithmes de références de la littérature sont ID3, C4.5, CART mais ils ne sont pas incrémentaux.

Des versions incrémentales des arbres de décision sont assez rapidement apparues. (Schlimmer et Fisher, 1986) propose ID4 et (Utgoff, 1989) propose ID5R qui sont basés sur ID3 mais dont la construction est incrémentale. ID5R garantit la construction d'un arbre similaire à ID3 alors qu'ID4 peut dans certains cas ne pas converger et dans d'autre cas avoir une prédiction médiocre. Plus récemment (Utgoff et al., 1997) propose ITI dont le fonctionnement est basé sur le maintien de statistiques dans les feuilles permettant une restructuration de l'arbre lors de l'arrivée des nouveaux exemples.

La lisibilité des arbres ainsi que leur rapidité de classement en font un choix très pertinent pour une utilisation sur d'importantes quantités de données. Cependant les arbres ne sont pas très adaptés aux changements de concept car dans ce cas d'importantes parties de l'arbre doivent être élaguées et réapprises.

3.3 Séparateurs à Vaste Marge

Les bases des Séparateurs à Vaste Marge (SVM) datent de 1963 et ont été proposées par Vapnik, père de cette théorie. Cependant les premières vraies publications basées sur ce procédé de classification sont apparues dans (Boser et al., 1992; Cortes et Vapnik, 1995). L'idée de base est de trouver l'hyperplan qui maximise la distance (la marge) entre les exemples de classes différentes.

Des versions incrémentales des SVM ont été proposées, parmi celles-ci (Domeniconi et Gunopulos, 2001) propose un découpage des données en partitions et quatre techniques différentes pour réaliser l'apprentissage incrémental :

– ED - Error Driven technique : lors de l'arrivée de nouveaux exemples, ceux qui sont mal classifiés sont conservés pour modifier le SVM.

– FP - Fixed Partition technique : un apprentissage sur chacune des partitions est réalisé et les vecteurs supports résultants sont agrégés ensemble (Syed et al., 1999).
– EM - Exceeding-Margin technique : lors de l'arrivée de nouveaux exemples, ceux qui se situent dans la zone de marge de l'hyperplan sont conservés. Lorsqu'un nombre assez important de ces exemples est collecté le SVM est mis à jour.
– EM+E - Exceeding-margin+errors technique : utilisation de "ED" et "EM", les exemples qui se situent dans la zone de marge et ceux qui sont mal classifiés sont conservés.

(Fung et Mangasarian, 2002) propose un PSVM - Proximal SVM, qui au lieu de voir une frontière comme étant un plan, la voit comme plusieurs plans (un espace) à proximité du plan de frontière. Les exemples supportant les vecteurs supports ainsi qu'un ensemble de points situés dans l'espace proche autour de l'hyperplan frontière sont conservés. En faisant évoluer cet ensemble, on enlève certains exemples trop anciens et on rajoute les nouveaux exemples ce qui permet de rendre le SVM incrémental.

Plus récemment l'algorithme LASVM (Bordes et Bottou, 2005; Bordes et al., 2005; Loosli et al., 2006) a été proposé. Il s'appuie sur une sélection active des points à intégrer dans la solution et donne des résultats très satisfaisants pour une utilisation en ligne. L'apprentissage peut être interrompu à tout moment, avec une étape éventuelle de finalisation (qui correspond à éliminer les vecteurs supports devenus obsolètes). Cette étape de finalisation faite régulièrement au cours de l'apprentissage en ligne permet de rester proche des solutions optimales. Du côté de la complexité calculatoire il n'y a qu'une résolution analytique à chaque étape. Du côté mémoire, l'algorithme peut être paramétré et c'est alors une question de compromis entre temps de calcul et espace mémoire.

3.4 Système à base de règles

En informatique, les systèmes à base de règles (Buchanan, 1984) sont utilisés comme un moyen de stocker et de manipuler des connaissances pour interpréter l'information de manière utile. Un exemple classique d'un système fondé sur des règles est le système expert d'un domaine spécifique qui utilise des règles pour faire des déductions ou des choix. Par exemple, un système expert peut aider un médecin à choisir le bon diagnostic qui repose sur un ensemble de symptômes. Un système à base de règles possède une liste de règles ou base de règles, qui constitue la base de connaissances. Cette base connaissance est soit donnée par un expert du domaine soit élaborée (extraite) à partir d'un ensemble d'apprentissage. Si elle est extraite de manière automatique il existe un certain nombre de critères qui permettent de juger de la qualité des règles. Nous invitons le lecteur à se reporter à (Lallich et al., 2007) pour une revue et une analyse d'un grand nombre de mesures existantes.

Plusieurs algorithmes à base de règles ont été développés pour être mis à jour de manière incrémentale, parmi ceux-ci on peut citer :

– STAGGER (Schlimmer et Granger, 1986) est le premier système à avoir été conçu pour traiter le concept drift. Il se base sur deux processus : le premier modifie le poids des attributs des règles et le deuxième ajoute de nouveaux attributs dans les règles.
– FLORA, FLORA3 (Widmer et Kubat, 1996) dont le principe de fonctionnement repose sur des fenêtres et un système de gestion des contextes pour les enregistrer et les réactiver si nécessaire.

– AQ-PM (Maloof et Michalski, 2000) est basé sur un système de conservation des exemples aux frontières des règles et d'un mécanisme d'oubli lui permettant de faire face aux changements de concepts.

3.5 Approche Bayesienne naïve

Dans cette section on s'intéresse au prédicteur classifieur (ici restreint au classifieur) naïf de Bayes. Le classifieur Bayesien naïf (Langley et al., 1992) suppose que les variables explicatives sont indépendantes sachant la classe cible. Cette hypothèse réduit drastiquement les calculs nécessaires. Ce prédicteur s'est avéré très compétitif sur de nombreux jeux de données réels. Ses performances dépendent généralement d'une estimation précise des probabilités conditionnelles univariées et d'une sélection de variables efficace.

Le principal avantage de cette approche est sa vitesse d'apprentissage et sa faible variance. Avec très peu de données la précision est bien souvent meilleure qu'avec d'autres algorithmes comme l'expérimente Domingos dans (Domingos et Pazzani, 1997). Cette qualité fait que le Bayesien naïf est assez souvent utilisé en combinaison avec d'autres algorithmes, comme par exemple les arbres de décision dans (Kohavi, 1996) : NBTree. De plus par nature l'approche Bayesienne naïve est naturellement incrémentale et peut être mis à jour sans qu'il soit nécessaire de tout recalculer. En effet, il suffit de mettre à jour les comptes permettant de calculer les probabilités conditionnelles univariées. Ces probabilités étant basées sur une estimation des densités, le problème réside dans l'estimation incrémentale des densités conditionnelles. (John et Langley, 1995) expérimentent deux méthodes d'estimation de densité : une gaussienne unique et des noyaux multiples. Il présente leurs complexités spatiales et temporelles ainsi que l'incrémentalité naturelle de l'estimation par une gaussienne en mettant à jour la moyenne et la variance. (Lu et al., 2006) propose IFFD, une méthode de discrétisation incrémentale permettant de ne pas réaliser une discrétisation complète à l'arrivée de chaque exemple et donc de ne pas recalculer toutes les probabilités conditionnelles. IFFD ne réalise que deux opérations : ajout d'une valeur à un intervalle ou éclatement d'un intervalle en deux.

Les méthodes Bayesiennes non naïves réalisent des agrégations de variables et/ou valeurs mais ne sont pas aisément incrémentales.

3.6 Plus proches voisins - approche passive

Les méthodes passives (lazy learning (Aha, 1997)), comme par exemple les k plus proches voisins (k-ppv), sont appelées passives car il n'y a pas réellement de phase d'apprentissage mais simplement une conservation de certains exemples. A proprement parler, on ne construit jamais de modèle « global » des données, on se contente d'un modèle local construit à la demande au moment de l'apparition d'un nouveau motif. Rendre ces méthodes incrémentales est donc assez simple car il suffit de mettre à jour la base d'exemples conservée voire de n'en garder qu'une partie comme le propose (Brighton et Mellish, 2002). Tout le traitement a lieu pendant la phase de classification lors de la recherche du plus proche voisin. L'approche typique de ce type de méthodes est de trouver au sein de l'ensemble des exemples d'apprentissage conservés ceux qui sont les plus proches de la donnée que l'on souhaite étiqueter. La(es) classe(s) des exemples proches trouvée(s) donne une bonne indication de la classe à prédire pour la donnée présentée.

La littérature s'est consacrée en grande partie d'une part (i) à accélérer la recherche des k-ppv (V. Hooman et al., 2000; Moreno-Seco et al., 2002) et d'autre part (ii) à l'apprentissage de métrique (Kononenko et Robnik, 2003; Globersonn et Roweis, 2005; Weinberger et Saul, 2009). On trouvera aussi dans (Sankaranarayanan et al., 2007) un travail très intéressant dont une comparaison "incrémental versus non incrémental" d'une méthode à k-ppv.

4 Apprentissage incrémental sur flux

4.1 Introduction

Depuis les années 2000, le volume de données à traiter a fortement augmenté avec l'essor d'internet et plus récemment des réseaux sociaux. Ces données arrivent séquentiellement et en continu et ne sont accessibles qu'au moment de leur passage : on parle alors de flux de données. Des algorithmes spécifiques pour traiter l'apprentissage supervisé de manière optimale dans ces conditions ont été proposés. Cette partie est une présentation des principales méthodes de la littérature. Dans cette section la partie sur les arbres de décision est plus détaillée, ceci étant dû au nombre d'articles dédiés à ces techniques.

On peut se demander en premier lieu quelles sont les propriétés désirées d'un algorithme de classification incrémentale sur les flux. Domingos dans un article de réflexion sur l'apprentissage sur les flux (Domingos et Hulten, 2001) propose les critères suivants :

- durée faible et constante pour apprendre les exemples ;
- lecture d'une seule fois des exemples et dans leur ordre d'arrivée ;
- utilisation d'une quantité de mémoire fixée à priori ;
- production d'un modèle proche de celui qui aurait été généré s'il n'y avait pas eu la contrainte de flux ;
- possibilité d'interroger le modèle à n'importe quel moment (anytime) ;
- possibilité de suivre les changements de concept.

Des critères assez similaires avaient déjà été abordés par (Fayyad et al., 1996) dans le cadre des bases de données de taille importante. Plus récemment, (Stonebraker et al., 2005) propose huit propriétés d'un bon algorithme de traitement temps réel des flux. Cependant il se situe plutôt dans le cadre de la base de données que dans celui de l'apprentissage. Le tableau 1 présente une comparaison des différentes propriétés désirées selon les auteurs.

	(1)	(2)	(3)
itératif/incrémental	x	x	
lecture une seule fois des données	x	x	x
gestion de la mémoire/ressources	x	x	x
anytime	x	x	x
gestion de la dérive de concept		x	

TAB. 1 – *Propriétés d'un bon algorithme d'apprentissage sur flux de données : (1)= (Fayyad et al., 1996) ; (2)= (Hulten et al., 2001) ; (3)= (Stonebraker et al., 2005) .*

Dans un deuxième temps on s'intéresse aux éléments de comparaison des différents algorithmes. Le tableau 2 compare les différents algorithmes par rapport aux critères généralement utilisés (Zighed et Rakotomalala, 2000) pour évaluer un algorithme d'apprentissage.

La complexité en déploiement est donnée dans ce tableau pour un problème à deux classes et représente la complexité au pire cas pour obtenir l'une des probabilités conditionnelles pour un exemple en test ($P(C_k|X)$). On notera que cette complexité maximale est rarement atteinte pour certain classifieur : à titre d'exemple pour un arbre elle est de fait (si l'arbre n'est pas complètement déséquilibré) en $O(h)$ où h désigne la hauteur de l'arbre.

Critère	Arbre de décision	SVM	Plus proche voisin	Système à base de règles	Bayesien naïf
Qualité de l'algorithme d'apprentissage					
Rapidité d'apprentissage	+	- -	+ +	-	+
Complexité en déploiement	$O(a)$	$O(sj)$	$O(nj)$	$O(ab)$	$O(j)$
Rapidité et facilité de mise à jour	- -	-	+ +	+	+ +
CPU - mémoire	+	+	- -	+	+
Pertinence du classifieur obtenu					
Précision	+ +	+ +	+	+ +	+
Simplicité (nombre de paramètres)	-	-	+ +	+	+ +
Rapidité de classement	+ +	-	-	+	+ +
Compréhensibilité	+ +	-	+ +	+ +	+ +
Généralisation - Sensibilité au bruit	- -	+ +	-	-	+ +

TAB. 2 – *Comparatif des principaux algorithmes. On note dans ce tableau : n le nombre d'exemples ; j le nombre d'attributs ; a le nombre de règles ; b le nombre moyen de prémices par règle et s le nombre de vecteurs supports.*

Finalement les différences entre apprentissage incrémental et apprentissage incrémental pour flux de données sont présentées dans le tableau 3. On appelle phase de "Post-optimisation" une phase qui aurait pour but d'améliorer la solution trouvée après la première passe sur les exemples. Cette passe de post-optimisation peut chercher à améliorer le modèle obtenu sans nécessairement avoir besoin de relire les exemples.

	Incrémental	Incrémental sur flux
Réglage des paramètres du classifieur via une validation croisée	Non	Non
Lecture d'une seule fois des données	Oui	Oui
Post-optimisation en fin d'apprentissage	Oui	Non
Complexité calculatoire en apprentissage et en prédiction	Faible	Très faible
Gestion de la mémoire/ressources	Oui	Oui
Gestion (ou estimation) de la précision vs algorithme offline	Oui	Oui
Gestion précision versus temps pour apprendre	Non	Oui
Gestion de la dérive de concept	Non	Oui
Anytime (interruptible)	Non	Recommandé

TAB. 3 – *Propriétés de la classification incrémentale vis-à-vis de la classification incrémentale sur flux (oui=requis, non=non requis).*

La suite de cette section propose des algorithmes qui non seulement sont incrémentaux mais semblent aussi adaptés aux flux de données : capable d'apprendre sans ralentir le flux

(complexité plus basse que ceux de la section précédente), capable d'être réglés pour aller vers un compromis temps/mémoire/précision ; et s'attaquant au problème de la dérive de concept. Ils remplissent tout ou partie des éléments du tableau 3.

4.2 Arbre de décision

4.2.1 Préambule

Limitation des anciens algorithmes : SLIQ (Mehta et al., 1996), SPRINT (Shafer et al., 1996), RAINFOREST (Gehrke et al., 2000) sont des algorithmes spécialement conçus pour fonctionner sur des bases de données de taille importante. Cependant il leurs est nécessaire de voir plusieurs fois les données et ils ne satisfont donc pas les contraintes des flux. Certains algorithmes n'ont besoin de voir les données qu'une seule fois comme ID5R (Utgoff, 1989) ou son successeur ITI (Utgoff et al., 1997). Cependant l'effort pour mettre à jour le modèle est parfois plus conséquent que celui pour reconstruire un modèle à partir de rien. Ces algorithmes ne peuvent alors plus être considérés comme "en ligne" ou "anytime".

Taille des arbres : De par sa méthode de construction la taille d'un arbre de décision croît avec l'arrivée de nouvelles données (Oates et Jensen, 1997) sans pour autant toujours améliorer son taux de bonne prédiction ((Zighed et Rakotomalala, 2000) - p203). Dans le cas des flux, l'arbre n'arrêtera donc pas de grandir si aucun traitement n'est prévu pour limiter sa croissance. De nouveaux algorithmes ont été développés afin de traiter le cas de l'apprentissage sur les flux.

Dérive de concept : Dans le cas où une dérive de concept apparaît dans le flux de données l'arbre doit être élagé ou si possible restructuré (Utgoff, 1989; Utgoff et al., 1997).

Borne d'Hoeffding : De très nombreux algorithmes d'apprentissage incrémentaux sur les flux utilisent la borne d'Hoeffding (Hoeffding, 1963) pour déterminer le nombre minimal de données nécessaire à la transformation d'une feuille en nœud. La borne d'Hoeffding permet de s'assurer que la vraie moyenne d'une variable aléatoire comprise dans un intervalle R ne sera pas différente, à ϵ près, de sa moyenne estimée après n observations indépendantes, tout cela avec une probabilité de $1-\delta$: $\epsilon = \sqrt{\frac{R^2}{2n} ln(\frac{1}{\delta})}$. L'intérêt de cette borne est qu'elle ne dépend pas de la distribution des valeurs mais seulement de : (i) de la plage de valeurs R, (ii) du nombre d'observations n, (iii) de la confiance désirée δ. Par contre cette borne est plus conservatrice que des bornes prenant en compte la distribution des valeurs.

4.2.2 Les principaux algorithmes rencontrés

On cite ci-dessous les principaux et récents algorithmes d'arbres incrémentaux pour flux de données trouvés dans la littérature :

• **VFDT** (Domingos et Hulten, 2000) est considéré comme un article de référence de l'apprentissage sur flux de données sachant gérer plusieurs millions d'exemples. Il est très largement cité et comparé aux nouvelles approches proposées depuis l'année 2000 sur la même problématique. Dans VFDT, la création de l'arbre est incrémentale et aucun exemple n'est conservé. Le taux d'erreurs de l'algorithme est plus important en début d'apprentissage qu'un algorithme comme C4.5. Cependant après avoir appris plusieurs centaines de milliers d'exemples, ce taux d'erreur devient plus faible car C4.5 n'est pas capable de travailler avec

des millions d'exemples et doit donc n'en utiliser qu'une partie. La figure 1 extraite de l'article de VFDT montre ce comportement et l'intérêt de VFDT par rapport à C4.5. De plus, Domingos et Hulten ont prouvé que les "Hoeffding Trees" sont proches de l'arbre appris qui aurait été généré par un algorithme hors ligne. Afin de pouvoir mieux répondre à la problématique des flux VFDT peut être paramétré. Les deux principaux paramètres sont : (i) la quantité maximale de mémoire à utiliser, (ii) le nombre minimal d'exemples à voir avant de réaliser le calcul du critère.

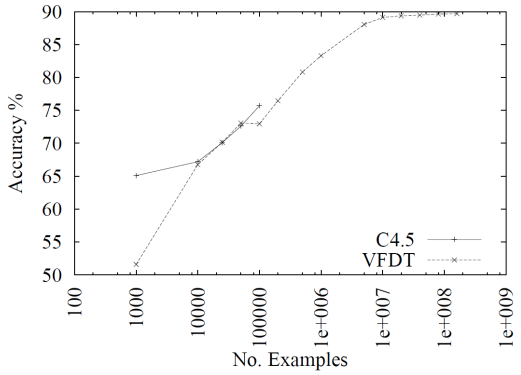

FIG. 1 – *Comparaison de C4.5 avec VFDT (extraite de (Domingos et Hulten, 2000))*

• **CVFDT** (Hulten et al., 2001) est une extension de VFDT pour gérer les changements de concept. Des sous arbres alternatifs sont construits si un changement de concept est détecté. Ces sous-arbres alternatifs remplacent le sous arbre original quand leurs taux d'erreurs deviennent plus faibles. Pour limiter l'utilisation de la mémoire seuls les sous arbres les plus prometteurs sont conservés. Malgré l'utilisation d'une fenêtre temporelle sur les exemples, la complexité reste en O(1) car on ne parcours pas à nouveau toutes les données de la fenêtre mais on utilise seulement la nouvelle donnée arrivée. D'après les expérimentations numériques de (Hulten et al., 2001) sur le jeu de données des "hyperplans en mouvement" (voir Tableau 4). On constate que le modèle contient quatre fois moins de nœuds mais consomme cinq fois plus de temps comparativement à VFDT.

• **VFDTc** (Gama et al., 2003) est une extension de VFDT qui gère les attributs numériques continus et non plus seulement catégoriels. Dans chaque feuille on conserve par attribut les comptes des valeurs numériques qui ont atteint cette feuille. Cela permet par la suite de trouver la meilleure valeur de coupure pour cet attribut pour transformer un nœud en feuille. De plus un classifieur Bayesien naïf est ajouté dans les feuilles afin d'améliorer la prédiction. (Gama et al., 2003) observe qu'il faut de 100 à 1000 exemples avant de transformer une feuille en nœud. Ces exemples dans les feuilles ne sont pas utilisés pour améliorer le modèle tant que la feuille n'est pas transformée en nœud. VFDTc propose d'utiliser ces exemples en ajoutant dans chaque feuille un modèle local. Le classifieur Bayesien naïf est connu pour apprendre bien et vite sur peu de données (cf. 3.5). C'est donc le classifieur choisi pour être utilisé dans les feuilles. Cette modification améliore la prédiction de l'arbre sur les jeux de tests WaveForm et LED de l'UCI.

• Dans **IADEM** (Ramos-Jimenez et al., 2006) et **IADEMc** (del Campo-Avila et al., 2006) la construction de l'arbre est aussi basée sur la borne d'Hoeffding. La croissance de l'arbre est gérée à l'aide du taux d'erreurs de celui-ci. L'arbre se développe jusqu'à arriver au taux d'erreurs maximum passé en paramètre de l'algorithme. IADEMc est une extension qui permet de gérer les attributs continus et qui possède un classifieur Bayesien naïf dans les feuilles (comme pour VFDTc). Les expérimentations réalisées sur les jeux de tests WaveForm et LED montrent les différences suivantes par rapport à VFDTc : une prédiction légèrement moins bonne mais des arbres dont la taille évolue beaucoup moins.

• Enfin Kirkby dans (Kirkby, 2008) réalise une étude sur les "Hoeffding Tree" et propose des améliorations à l'aide **d'ensemble d'"Hoeffding Tree"**. Sa première modification consiste à avoir dans les feuilles soit un modèle basé sur la classe majoritaire soit un modèle Bayesien naïf. Ses expériences ont montré que le Bayesien naïf n'était pas toujours le meilleur choix. Le classifieur qui a la plus faible erreur est conservé comme classifieur dans les feuilles. Sa deuxième modification aboutit à la proposition des "Hoeffding Option Trees" (inspiré de (Kohavi et Kunz, 1997)) qui sont une variante des "Hoeffding trees" possédant plusieurs sous-arbres dans chaque nœud. Les mêmes exemples d'apprentissage peuvent mettre à jour plusieurs arbres en même temps. La prédiction se fait par un vote à la majorité qui détermine la classe prédite. Les techniques de bagging et de boosting sont aussi expérimentées : le bagging permet une amélioration de la prédiction mais pour le boosting le résultat est plus mitigé.

4.2.3 Discussion

Limite des arbres : Du fait de la structure en arbre, le premier nœud a une très grande importance et si un changement de concept se produit sur ce nœud ou sur des nœuds assez haut, l'évolution de l'arbre est plus difficile, voire une reconstruction complète peut être nécessaire comme l'indique (Wang et al., 2003). Dans ce cas particulier, l'utilisation des arbres pour la classification sur un flux non stationnaire peut s'avérer coûteuse. La capacité à se restructurer d'un arbre construit de manière incrémentale est donc certainement un élément clef de réussite.

Du coté "anytime", (Seidl et al., 2009) propose l'utilisation d'un arbre Bayesien contenant dans ses nœuds des informations sur la meilleure prédiction. Ainsi s'il n'est pas possible d'arriver à une feuille de l'arbre on utilise la prédiction disponible dans les nœuds.

4.3 Séparateurs à Vaste Marge

Les SVMs sont assez peu abordés dans la littérature comme classifieur pour les flux de données. Dans (Tsang et al., 2006), une version des SVMs pour apprendre sur d'importantes quantités d'exemples est présentée. Elle est basée sur une approximation de la solution optimale à l'aide de la résolution de problèmes de type "MEB - Minimum Enclosing Balls". Sa complexité spatiale est indépendante de la taille de l'ensemble d'apprentissage. Un paramètre permet de régler la préférence pour la rapidité ou la précision du classifieur.

(Dong et al., 2005) présentent une optimisation des méthodes existantes afin d'augmenter la capacité des SVMs. Cette optimisation utilise la technique "diviser pour régner" en séparant le problème en sous problème et en trouvant et en éliminant rapidement les exemples n'étant pas des vecteurs supports.

On pourrait aussi utiliser LASVM (Bordes et Bottou, 2005; Bordes et al., 2005; Loosli et al., 2006) sur chaque point du flux présenté : soit il est éligible pour devenir "vecteur sup-

port" (VS) et alors on met à jour la solution courante ; soit il n'est pas éligible (loin des frontières courantes indépendamment de son étiquette), on ne fait rien. Au pire des cas lorsqu'un exemple est éligible, la complexité est au mieux en $O(s^2)$ avec s le nombre courant de vecteur supports. Dans les deux cas (éligible ou non), le calcul des éléments du noyau pour ce point et le calcul des vecteur supports doivent être réalisés. On s'aperçoit donc que la complexité n'est pas négligeable ce qui explique peut être que d'un point de vue applicatif on trouve peu de "LASVM" appliqués sur des flux de données (comparé à la complexité des arbres incrémentaux par exemple). Un article récent présente les garanties des algorithmes en ligne du type LASVM (Usunier et al., 2010).

L'une des voies d'amélioration est l'utilisation de SVM linéaire et la parallélisation des calculs comme proposé par Poulet et al. (Do et al., 2009). L'utilisation de ces SVMs sur GPUs permet d'apprendre sur des flux rapides et d'obtenir des améliorations, en termes de rapidité, supérieures à un facteur 100.

4.4 Système à base de règles

On trouve peu d'articles concernant l'approche à base de règles appliquée aux flux. Des algorithmes incrémentaux basés sur les systèmes à bases de règles existent mais ils ne sont pas prévus pour des flux de données. On trouve néanmoins une série de trois articles de Ferrer et al. sur ce sujet (Ferrer-Troyano et al., 2005, 2006). Ces derniers proposent l'algorithme FACIL qui est la première approche de base de règles incrémentale sur flux de données et est basé sur AQ-PM (Maloof et Michalski, 2000).

L'approche de l'algorithme FACIL se définit de la manière suivante :

- Lors de l'arrivée d'un nouvel exemple, on recherche pour toutes les règles ayant son étiquette lesquelles le couvrent et on incrémente leur support positif.
- Si l'exemple n'est pas couvert alors on recherche la règle qui nécessite une "augmentation" minimum de son espace couvert. De plus cette augmentation doit être limitée à une certaine valeur fixée par un paramètre κ pour être acceptée.
- Si aucune règle ayant la même étiquette que l'exemple ne le couvre alors on recherche dans l'ensemble de règles ayant une étiquette différente. Si on trouve des règles le couvrant on ajoute l'exemple aux règles comme exemple négatif et on incrémente leur support négatif.
- Si aucune règle ne couvre cet exemple, une nouvelle règle peut être créée.
- Chaque règle possède sa propre fenêtre d'oubli de taille variable.

4.5 Approche Bayesienne naïve

L'apprentissage par une approche Bayesienne naïve consiste dans un premier temps à réaliser une estimation des probabilités conditionnelles aux classes (voir section 3.5). Cette estimation est très souvent réalisée après une étape de discrétisation des variables explicatives. Dans le cadre des flux de données seule cette première étape est modifiée pour satisfaire la contrainte du flux. Dans l'état de l'art on trouve deux types de publications concernant la discrétisation incrémentale. Celles, assez peu nombreuses, qui concernent l'apprentissage et celles beaucoup plus nombreuses relatives aux systèmes de gestion de bases de données qui conservent des statistiques sous forme d'histogramme.

Dans le cas de l'apprentissage, Gama et Pinto dans (Gama et Pinto, 2006), propose PiD : "Partition Incremental Discretization" qui réalise une discrétisation incrémentale pour ensuite réaliser un apprentissage avec une méthode Bayesienne naïve. Cette discrétisation est une solution basée sur deux niveaux :

- au niveau 1, une première discrétisation est réalisée où l'on stocke les comptes des classes pour chaque intervalle et où l'on fait évoluer ces intervalles si nécessaires de la manière suivante : si jamais des valeurs en dehors du domaine arrivent alors on rajoute des intervalles (∼EqualWidth) ; si un intervalle contient trop de valeurs alors on le découpe en 2 intervalles (∼EqualFreq). La mise à jour se fait de manière incrémentale. Ce premier niveau doit contenir plus d'intervalles que le niveau 2. Le nombre d'intervalles est un paramètre.
- au niveau 2, la discrétisation réalisée au niveau 1 est utilisée pour réaliser la discrétisation finale qui peut être (au choix) : EqualWidth, EqualFreq, K-means, Proportional discretization, Recursive entropy discretization, MDL.

Le problème de cette méthode est que tout dépend de la discrétisation de niveau 1. On peut perdre de l'information sur les bornes de coupure (car on ne garde pas toutes les valeurs) et sur les comptes (quand on divise à nouveau un intervalle). En cas de distribution Zipfienne ou de présence d'observations aberrantes, on peut se retrouver avec un nombre d'intervalles beaucoup plus important que prévu sur le niveau 1.

Dans le cadre des systèmes de gestion des bases de données (SGBD) on retrouve les estimations par histogramme. Elles sont utilisées pour avoir des statistiques sur les données afin de créer les meilleurs plans d'exécution et savoir quelles optimisations utiliser. Ces techniques supportent la plupart du temps l'insertion et la suppression de données dans les bases, or dans le cas de l'apprentissage il n'est pas nécessaire de les supporter.

Gibbons et al (Gibbons et al., 2002) proposent des techniques incrémentales de mise à jour des histogrammes pour les systèmes de bases de données. Leur approche est basée sur deux structures :

- un réservoir de données contenant des données représentatives des données réelles (basé sur la technique du "Reservoir sampling" (Vitter, 1985)) ;
- des histogrammes de type "Equal Freq" résumant les données.

4.6 Plus proches voisins - approche passive

Les algorithmes des plus proches voisins sont facilement incrémentaux (cf. 3.6). Dans le cadre de la problématique d'apprentissage en flux, les solutions consistent à trouver la meilleure base d'exemples à conserver pour une faible mémoire en (i) oubliant des exemples les plus anciens, (ii) agrégeant les exemples proches. Dans la littérature on trouve différentes méthodes basées sur les plus proches voisins pour les flux :

- (Law et Zaniolo, 2005) utilisent une technique de discrétisation de l'espace d'entrée afin de limiter le stockage des exemples.
- (Beringer et Hüllermeier, 2007) conservent les exemples les plus récents et gèrent une fenêtre d'oubli.

5 Évaluation des algorithmes

Récemment de nombreux algorithmes ont été publiés pour l'apprentissage sur les flux de données. Cependant il est difficile de réaliser une comparaison de ces algorithmes entre eux. En effet les différents auteurs n'utilisent pas toujours les mêmes méthodes ni les mêmes jeux de tests. Gama dans (Gama et al., 2009) puis dans (Gama, 2010) (chapitre 3) propose une méthode d'expérimentation pour l'évaluation et la comparaison des algorithmes d'apprentissage sur les flux. Nous allons dans une première partie nous intéresser à la mesure du taux de prédiction utilisée et dans une seconde partie aux jeux de données utilisés. Ensuite nous discuterons d'autres points de comparaison possibles.

5.1 Mesures de précision et ensemble de test

Pour l'apprentissage "hors-ligne", la méthode la plus utilisée est la validation croisée en "k folds" (avec k=10). On découpe les données en 10 ensembles de taille identique. On se sert du premier ensemble comme jeu de test et des autres comme exemples d'apprentissage, puis on prend le deuxième comme jeu de test et les autres pour l'apprentissage, et ainsi de suite dix fois pour utiliser tous les jeux de tests différents. Les données d'un flux de par leur quantité et leur disponibilité ne s'accommode pas facilement de ce genre de méthode. Selon que le flux soit stationnaire ou non-stationnaire (présence de changement de concept), plusieurs méthodes d'évaluation sont envisageables. Elles sont basées sur des jeux de données construits différemment :

- un jeu de données indépendant pour réaliser le test et que l'on garde tout au long de l'expérimentation
- un jeu de données remis à jour régulièrement en tirant au hasard des exemples du flux qui n'ont pas encore été utilisés pour l'apprentissage
- un jeu de données pris dans les exemples d'apprentissage avant qu'ils ne soient appris (aussi appelé "tester puis apprendre" (Kirkby, 2008) ou "préquentiel" (Gama et al., 2009)). L'indicateur de précision est alors calculé de manière incrémentale. Cette méthode est plus pessimiste que celle du jeu de données indépendant. La figure 2 compare les erreurs de prédiction pour l'algorithme VFDT sur le jeu de test WaveForm entre l'approche "jeu de données indépendant" et "préquentiel" (Gama et al., 2009). L'utilisation de fenêtre ou de facteur d'oubli permet de rendre cet évaluateur moins pessimiste comme le propose (Gama et al., 2009) et comme illustré dans la figure 3. De cette manière les mauvaises prédictions du passé n'influencent plus (ou moins) les nouvelles mesures de la prédiction de l'algorithme.

Flux stationnaire : Dans le cas où le flux est stationnaire, les trois méthodes précédemment présentées peuvent être utilisées, néanmoins la méthode basée sur le jeu de test indépendant est celle que l'on retrouve majoritairement.

Flux non stationnaire : Dans le cas où le flux n'est pas stationnaire (avec changement de concept), on ne peut pas utiliser toutes les méthodes précédentes directement. La méthode basée sur le jeu de test indépendant "holdout" ne convient pas car elle ne sera pas capable de bien évaluer les différents concepts. Au contraire la méthode "préquentiel" est particulièrement bien adaptée car son jeu de test évolue avec le flux.

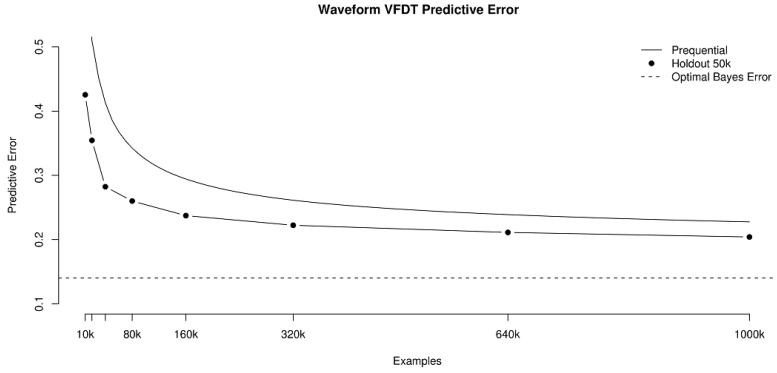

FIG. 2 – *Comparaison entre l'erreur avec le jeu de test indépendant et préquentiel*

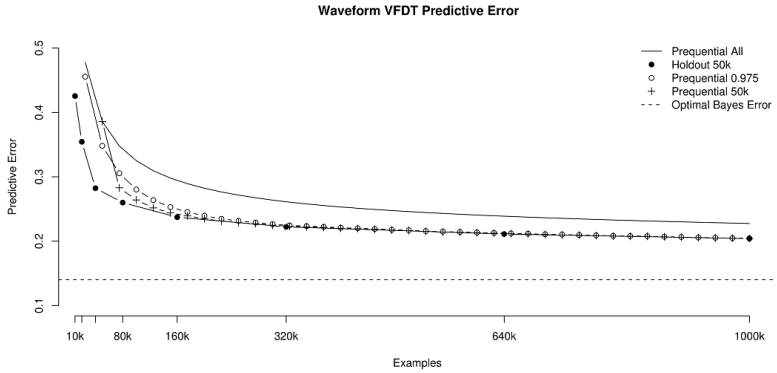

FIG. 3 – *Comparaison entre l'erreur avec jeu de test indépendant et préquentiel avec des facteurs d'oubli*

5.2 Jeux de données

Bien souvent dans la littérature, une première évaluation des algorithmes est réalisée sur les bases UCI classiques. Cela permet d'avoir une première idée des performances de l'algorithme bien qu'un algorithme sur les flux ne soit pas forcément prévu pour bien fonctionner sur des petits jeux de données. Ensuite pour simuler les flux, plusieurs approches sont possibles selon que l'on souhaite avoir un flux avec ou sans gestion de la dérive de concept. Le tableau 4 présente une synthèse des jeux utilisés dans les différentes publications. Le lecteur trouvera d'autres jeux de données dans (Gama, 2010) page 209.

5.2.1 Générateur de flux sans présence de dérive de concept

Afin d'avoir assez de données pour tester les algorithmes d'apprentissage sur les flux, des générateurs artificiels existent pour permettre de générer des milliards d'exemples. Un état de

l'art de ces générateurs est présenté dans (Kirkby, 2008). Les principaux générateurs sont :

- Random RBF Generator (Radial Basis Function) : basé sur des "bulles" ayant un centre. On a plusieurs bulles de points de différents paramètres à différentes positions dans l'espace. Ensuite on génère des exemples appartenant à ces bulles.
- Random Tree Generator : génération d'un arbre avec certains paramètres et ensuite on génère les données à partir de l'arbre. Ce générateur favorise les algorithmes d'apprentissage basés sur les arbres.
- LED Generator : prédiction pour un afficheur LED.
- Waveform Generator : problème à trois classes généré à partir de fonctions en forme d'onde différentes et combinées.
- Function Generator : les exemples sont générés à partir de règles sur les attributs qui déterminent la classe.

5.2.2 Générateur de flux avec présence de dérive de concept

Afin de pouvoir tester les performances de leurs algorithmes sur des flux de données avec des changements de concept, des auteurs (voir Tableau 4) ont proposé des générateurs de flux pour lesquels on peut paramétrer la position et la quantité du changement (voir aussi (Gama, 2010) page 209). Il faut aussi noter que les concepts présentés section précédente peuvent être facilement modifiés pour contenir des dérives de concept.

Nom	Proposé par	Utilisé dans	Type	Taille
STAGGER	(Schlimmer et Granger, 1986)	(Beringer et Hüllermeier, 2007; Bifet et Kirkby, 2009; Kolter et Maloof, 2003; Bifet et al., 2009)	Artificiel	infini
SEA Concept	(Street et Kim, 2001)	(Gama et al., 2005)	Artificiel	infini
Hyperplan en mouvement	(Hulten et al., 2001)	(Bifet et al., 2009; Beringer et Hüllermeier, 2007; Wang et al., 2003; Ferrer-Troyano et al., 2005; Narasimhamurthy et Kuncheva, 2007)	Artificiel	infini
Forest Covertype	UCI	(Gama et al., 2003; Law et Zaniolo, 2005; Bifet et al., 2009)	Réel	581K
Poker Hand	UCI	(Bifet et al., 2009)	Réel	1M

TAB. 4 – *Flux de données pour les algorithmes pour les flux*

5.2.3 Jeu de données réelles

Les jeux de données réels "Forest Covertype" et "Poker Hand" de l'UCI, contiennent plusieurs centaines de milliers d'exemples. Dans ces deux jeux l'existence et la position du changement de concept ne sont pas connues. De part leur taille assez importante ils sont utilisés pour simuler un flux de données réelles. Les auteurs ci-dessous proposent eux des données qui leurs sont propres :

- proxy web de l'Université de Washington pour Domingos dans (Domingos et Hulten, 2000; Hulten et al., 2001)
- consommation électrique en Australie pour Gama dans (Gama et al., 2005)

Le principal reproche que l'on peut faire à ces jeux de données est qu'ils ne sont pas accessibles et donc qu'ils ne pourront pas être réutilisés pour réaliser des comparaisons.

5.3 Evaluation entre algorithmes

On trouve très majoritairement la même méthodologie de comparaison dans la littérature. Dans un premier temps, les auteurs d'un nouvel algorithme effectuent une comparaison avec des algorithmes connus mais non prévus pour fonctionner sur des flux de données comme : C4.5, ID3, Bayesien naïf, forêt d'arbres... L'idée est de voir les performances sur de petites quantités de données contre des algorithmes connus. Puis dans un second temps ils se confrontent aux autres algorithmes déjà présents dans la littérature de l'apprentissage sur les flux. Le lecteur pourra trouver dans (Kirkby, 2008) ainsi que dans (Gama et al., 2009) un tableau comparatif des méthodes d'évaluation entre certains des algorithmes cités dans cet article.

Environnement d'évaluation : L'un des problèmes lorsque l'on veut réaliser une comparaison entre divers algorithmes de classification est de réaliser facilement des expérimentations. En effet, bien souvent il est difficile d'obtenir le code source de l'algorithme puis de réussir à construire un exécutable et parfois les formats d'entrée et de sortie sont différents entre les divers algorithmes. Pour l'apprentissage hors-ligne l'environnement WEKA (Witten et Frank, 2005) proposé par l'université de Waikato permet de réaliser rapidement des expérimentations. Cette même université a proposé en 2009 une boîte à outils d'expérimentation pour les flux : MOA (Bifet et Kirkby, 2009). On y retrouve la majorité des générateurs vus précédemment ainsi qu'un nombre important d'algorithmes d'apprentissage sur les flux basés sur les arbres d'Hoeffding.

Autres points de comparaison : La mesure de précision basée sur l'AUC (Fawcett, 2004) est la mesure de comparaison principale dans le cadre de la classification supervisée. Dans le cadre des flux de données d'autres mesures sont parfois utilisées en complément :

- taille du modèle (nombre de nœuds pour les arbres, nombre de règles, espace mémoire...)
- vitesse d'apprentissage (nombre d'exemples appris par seconde)

La comparaison de la vitesse d'apprentissage de deux algorithmes peut poser des problèmes car il suppose que le code/logiciel est disponible publiquement. De plus, on n'évalue pas seulement la vitesse d'apprentissage mais aussi la plate-forme sur laquelle tourne l'algorithme et la qualité de l'implémentation. Ces mesures sont utiles pour donner un ordre d'idée mais doivent être prises avec précautions et non pas comme des éléments absolus de comparaison.

Kirkby (Kirkby, 2008) réalise ses expérimentations sous différentes contraintes de mémoire simulant divers environnements : réseau de capteurs, PDA, serveur. On peut donc comparer les différents algorithmes par rapport à l'environnement dans lequel il serait exécuté.

On peut aussi mentionner le fait que la mesure de précision n'a pas de sens dans l'absolu en cas de dérive de concept. Le lecteur pourra alors se tourner vers des protocoles d'apprentissage, comme le mistake-bound (Littlestone et Warmuth, 1989). Dans ce cadre, l'apprentissage se déroule en cycles où les exemples sont vus un par un ; ce qui cadre bien avec l'apprentissage sur des flux de données. Au début l'algorithme d'apprentissage ($\mathcal{A}$) apprend une hypothèse (f_t) et la prédiction pour l'instance courante est $f_t(x_t)$. Puis l'étiquette réelle de l'exemple (y_t) est révélée à l'algorithme d'apprentissage qui peut avoir fait une erreur ($y_t \neq f_t(x_t)$). Puis le cycle reprend jusqu'à un horizon de temps donné (T). La borne d'erreur est reliée à la mesure du maximum d'erreur commise sur la période de temps T.

Limites des techniques d'évaluation : Des techniques d'évaluation ont été récemment proposées par la communauté et une recherche de consensus est en cours (Gama et al., 2009). En effet, des critères comme le nombre d'exemples appris par seconde ou une vitesse de prédiction dépend de la machine et de la mémoire utilisée ainsi que de la qualité de codage. Il faut donc trouver des critères et/ou plateformes communs afin de pouvoir réaliser une comparaison aussi impartiale que possible.

6 Gestion de la dérive de concept

Dans cette section le problème d'apprendre quand la distribution des exemples varie au cours du temps est abordé. Si le processus sous-jacent qui "génère" les données n'est pas stationnaire le concept cible à apprendre peut varier au cours du temps. L'algorithme d'apprentissage doit donc être capable de détecter ces changements et d'adapter le modèle de classification supervisée en conséquence.

On considère dans cette section que le concept à apprendre peut varier dans le temps mais qu'il est persistant et consistant (voir (Lazarescu et al., 2004)) entre deux changements. La période de temps existante entre deux changements de concept est appelée contexte (voir la Section 2.1.3 (Gama, 2010)). La dérive de concept apparaît à travers les exemples qui sont générés : les anciennes observations du processus deviennent caduques vis-à-vis des observations actuelles du processus. Les anciennes observations n'appartiennent pas au même contexte que les observations actuelles.

Si on suppose qu'entre deux changements de contexte ($Cont_i$ i représentant l'index du contexte) il existe un concept ($Conc_i$) suffisamment persistant et consistant pour lequel on peut collecter assez d'exemples d'apprentissage alors gérer la dérive de concept se ramène souvent à la détection de ce changement. On précise que l'on s'intéresse dans cette section uniquement aux variations de $P(Y|X)$. La question de la détection de nouveautés (Marsland, 2003) n'est pas abordée.

Si l'on sait détecter ces changements de concepts au cours du temps on pourra alors :
- soit réapprendre le modèle de classification à partir de zéro ;
- soit adapter le modèle courant ;
- soit adapter un résumé des données sur lequel se fonde le modèle courant ;
- soit travailler avec la séquence des modèles de classification appris au cours du temps.

La suite de cette section est organisée en deux parties : la première partie décrit des méthodes pour détecter la dérive de concept ; la seconde partie décrit des méthodes pour adapter le modèle de classification parmi celles décrites ci-dessus.

6.1 Détection de la dérive de concept

On considère ici uniquement la partie détection de la dérive sans s'occuper de la gestion qui en est faite après. La détection de la dérive de concept peut être réalisée principalement à l'aide de deux voies : (i) la surveillance des performances du modèle courant de classification et (ii) la surveillance de la distribution des exemples.

Pour la première voie :
- (Widmer et Kubat, 1996) proposent de détecter le changement de concept en analysant le taux d'erreurs de classification et les modifications qui se produisent dans les struc-

tures de l'algorithme d'apprentissage (ajout de nouvelles définitions dans les règles dans son cas). A partir d'une certaine variation de ces indicateurs, on diminue la taille de la fenêtre d'apprentissage sinon on la laisse croître afin d'avoir plus d'exemples pour réaliser un meilleur apprentissage. Ce fut l'un des premiers travaux sur ce sujet et sur lequel s'appuient beaucoup d'autres auteurs comme Gama dans le paragraphe ci-dessous.
- Gama dans (Gama et al., 2009) propose d'utiliser le test statistique de Page-Hinkley (Page, 1954). Gama le modifie en y ajoutant des facteurs d'oubli. Ce test est basé sur le calcul des écarts par rapport à la moyenne d'une variable. Son intérêt réside dans le fait qu'il est à la fois performant, simple à calculer et incrémental.

Pour la seconde voie on pourra utiliser des méthodes paramétriques, ou test statistique qui présuppose souvent de ce que l'on veut détecter : changement dans la moyenne, dans la variance, dans les quantiles, etc. Par exemple lorsque Gama utilise le test de Page-Hinkley il cherche à détecter une rupture dans une moyenne. Une méthode paramétrique est a priori imbattable quand le type de rupture rentre exactement dans son biais mais son paramétrage peut devenir un problème dans des conditions limites (biais non présent dans les données). De façon générale, il s'agit d'un compromis entre le biais particulier et l'efficacité de la méthode. Si on cherche à détecter n'importe quel type de rupture il semble difficile de trouver une méthode paramétrique qui en soit capable.

6.2 Méthodes adaptatives

Après avoir détecté la dérive de concept il faut : (i) soit réapprendre le modèle de classification à partir de zéro ; (ii) soit adapter le modèle courant ; (iii) soit adapter un résumé des données sur lequel se fonde le modèle courant ; (iv) soit travailler avec la séquence des modèles de classification appris au cours du temps.

Si on décide de réapprendre le modèle à partir de zéro : l'algorithme d'apprentissage s'appuie alors :
- soit sur une mémoire partielle des exemples : (i) un nombre fini des derniers exemples, (ii) un nombre adaptatif des derniers exemples (iii) un résumé des derniers exemples. La taille de la fenêtre des derniers exemples utilisée ne peut en aucun cas excéder l'horizon de l'avant dernière dérive détectée (Widmer et Kubat, 1992).
- soit sur une mémoire "complète" des exemples comme les méthodes capables de stocker des statistiques sur les exemples sous la contrainte d'une capacité mémoire donnée et d'une précision donnée (Greenwald et Khanna, 2001)

Les méthodes qui décident d'adapter le modèle de classification peuvent s'appuyer :
- soit sur plusieurs modèles de classification utilisés en même temps comme par exemple les méthodes ensemblistes ("bagging" (Breiman, 1996)) qui consistent à utiliser plusieurs classifieurs en même temps afin d'améliorer les capacités de prédiction de ceux-ci :
  - (Street et Kim, 2001) proposent l'algorithme SEA : "Streaming Ensemble Algorithm" qui utilise un ensemble de classifieurs sur les flux. Le flux remplit un tampon de taille définie et dès que le tampon est plein l'algorithme C4.5 est lancé dessus. On se retrouve donc avec une suite de classifieurs générés que l'on met dans un pool. Une fois le pool plein, si le nouveau classifieur améliore la prédiction du pool alors il est conservé, sinon il est rejeté. Dans (Wang et al., 2003) la même technique est

utilisée mais différents types de classifieurs composent l'ensemble : bayésien naïf, C4.5, RIPPER...
  - Dans (Kolter et Maloof, 2003), un poids est affecté à chaque classifieur, les poids sont mis à jour uniquement lors de la mauvaise prédiction du nouvel exemple. Dans ce cas un nouveau classifieur est ajouté et les poids des autres classifieurs sont diminués. Les classifieurs ayant un poids trop faible sont retirés de l'ensemble.
  - (Bifet et Gavalda, 2007; Bifet et al., 2009) propose aussi une méthode pour mettre à jour l'ensemble : ADWIN (ADaptative WINdowing). Elle est basée sur la détection des changements de concept à l'aide d'un réservoir de taille variable W. Le réservoir s'agrandit quand il n'y a pas de changements de concept et diminue lorsqu'il en rencontre un. Cette approche ne consomme comme mémoire que $log(W)$ et ne nécessite pas de paramètre. Quand un changement est détecté le plus mauvais classifieur est retiré et un nouveau classifieur est ajouté à l'ensemble.
- soit sur plusieurs modèles de classification mais les classifieurs sont utilisés **un à un**, les autres modèles déjà construits dans le passé sont considérés comme un pool de candidats potentiels (Gama et P., 2009; Gomes et al., 2010)

Ce sont aussi les exemples (et donc les modèles) qui servent à apprendre le modèle de classification qui peuvent aussi être pondérés au cours du temps. Depuis les années 80 des solutions comme les fenêtres et les facteurs d'oubli ont été proposées : (Schlimmer et Granger, 1986; Widmer et Kubat, 1992, 1996; Salganicoff, 1997). Le sujet est toujours d'actualité et il constitue une nouvelle problématique pour les algorithmes d'apprentissage en flux. On citera (Salganicoff, 1997) :

- TWF (Time-Weighted Forgetting) : plus l'exemple est ancien plus son poids est faible ;
- LWF (Locally-Weighted Forgetting) : à l'arrivée d'un nouvel exemple on augmente le poids des exemples qui sont proches de lui et on baisse ceux des autres. Les régions ayant des exemples récents sont ainsi conservées (ou créées si elles n'existaient pas) et les régions n'en ayant peu ou pas sont supprimées ;
- PECS (Prediction Error Context Switching) : l'idée de LWF est reprise mais tous les exemples sont gardés en mémoire et une vérification des étiquettes des nouveaux exemples par rapport aux anciens est effectuée. Une probabilité des exemples est calculée grâce aux comptes des exemples ayant ou n'ayant pas la même étiquette. Seulement les meilleurs exemples sont utilisés pour réaliser l'apprentissage.

7 Conclusion

Cet article synthétique introductif a présenté les principales approches de classification incrémentale recensées dans la littérature. Dans un premier temps les différentes problématiques de l'apprentissage ont été présentées ainsi que les nouveaux problèmes imposés par la contrainte des flux : quantité et vitesse importante des données et la possibilité de ne les voir qu'une fois. Ces caractéristiques imposent de se tourner vers des algorithmes spécifiques : incrémentaux voire spécialisés pour les flux. Ces principaux algorithmes ont été décrits par rapport à la classe à laquelle ils appartiennent : arbre de décisions, SVM... On remarque par ailleurs que dans la littérature les arbres de décisions incrémentaux, qui satisfont plus facilement les contraintes de vitesse de prédiction et/ou d'apprentissage, sont prédominants.

Au niveau des verrous à lever, l'adaptation aux changements de concept est l'un des challenges les plus importants et difficiles à résoudre. En effet, il faut d'une part trouver quand le

changement a lieu puis modifier le modèle en conséquence et ceci avec le moins de fausses détections possibles. Le compromis précision / vitesse / mémoire semble une voie intéressante si ce compromis pouvait être directement écrit comme un critère à optimiser lors de l'apprentissage du classifieur. On peut aussi citer la propriété anytime qui n'est que très peu implémentée en respectant sa définition stricte : "algorithme qui peut être arrêté à tout moment et qui retourne une prédiction". On peut aussi noter qu'un flux de données peut être vu comme un environnement changeant sur lequel s'exerce un modèle de classification. Les erreurs de classification sont ensuite accessibles dans un horizon de temps limité. Par conséquent on peut penser que les algorithmes liés à la théorie des jeux ou à l'apprentissage par renforcement devraient aussi pouvoir être formalisés pour l'apprentissage incrémental sur flux.

La plupart des tests sont réalisés sur des générateurs de données ou des jeux de données de l'UCI d'importante volumétrie (mais de moins d'un million d'exemples) mais rarement sur des flux réels. Certains auteurs proposent des flux provenant d'industriels (consommation d'électricité par exemple) mais on ne connaît pas précisément leurs caractéristiques : stationnaire ou non, niveau de bruit, type de modèle sous-jacent... Dès lors, les résultats obtenus sont problèmes (ou flux) dépendants. Il serait intéressant à terme d'avoir, comme cela existe déjà avec le dépôt de l'UCI, une liste de flux caractérisés et librement accessibles. Il en va de même pour une plateforme d'expérimentation même si l'université de Wakaito nous propose déjà l'environnement MOA qui s'inspire de WEKA. Etant à un jeune stade de son développement MOA nécessite d'être complété avec les nouvelles méthodes comme le fut WEKA à ses débuts.

Références

Aha, D. W. (Ed.) (1997). *Lazy Learning*. Springer.

Almaksour, A., H. Mouchère, et E. Anquetil (2009). Apprentissage incrémental et synthèse de données pour la reconnaissance de caractères manuscrits en-ligne. In *Dixième Colloque International Francophone sur l'écrit et le Document*.

Beringer, J. et E. Hüllermeier (2007). Efficient instance-based learning on data streams. *Intelligent Data Analysis 11*(6), 627–650.

Bifet, A. et R. Gavalda (2007). Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, pp. 443–448.

Bifet, A., G. Holmes, B. Pfahringer, R. Kirkby, et R. Gavaldà (2009). New ensemble methods for evolving data streams. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, 139.

Bifet, A. et R. Kirkby (2009). DATA STREAM MINING A Practical Approach. *Journal of empirical finance 8*(3), 325–342.

Bondu, A. et V. Lemaire (2008). Etat de l'art sur les methodes statistiques d'apprentissage actif. *RNTI A2 Apprentissage artificiel et fouille de données*, 189.

Bordes, A. et L. Bottou (2005). The Huller : a simple and efficient online SVM. *Proceedings of the 16th European Conference on Machine Learning (ECML2005)*.

Bordes, A., S. Ertekin, J. Weston, et L. Bottou (2005). Fast kernel classiffiers with online and active learning. *Journal of Machine Learning Research 6*, 1579–1619.

Boser, B., I. Guyon, et V. Vapnik (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152.

ACM New York, NY, USA.

Breiman, L. (1996). Bagging predictors. *Machine learning 24*(2), 123–140.

Breiman, L., J. Friedman, R. Olshen, et C. Stone (1984). *Classification and regression trees*. Chapman and Hall/CRC.

Brighton, H. et C. Mellish (2002). Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery 6*(2), 153–172.

Buchanan, B. G. (1984). *Rule Based Expert Systems : The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Cornuéjols, A. (2009). *Blueprint on Ubiquitous Knowledge Discovery*, Chapter On-line learning : Where are we so far ? May M. and Saitta L. eds. - Springer.

Cornuéjols, A. et L. Miclet (2010). *Apprentissage artificiel - Concepts et algorithmes*. Eyrolles.

Cortes, C. et V. Vapnik (1995). Support-vector networks. *Machine Learning 20*(3), 273–297.

Dean, T. et M. Boddy (1988). An analysis of time-dependent planning. In *Proceedings of the seventh national conference on artificial intelligence*, pp. 49–54.

del Campo-Avila, J., G. Ramos-Jiménez, J. . Gama, et R. Morales-Bueno (2006). Improving Prediction Accuracy of an Incremental Algorithm Driven by Error Margins. *Knowledge Discovery from Data Streams*, 57.

Do, T., V. Nguyen, et F. Poulet (2009). GPU-based parallel SVM algorithm. *Jisuanji Kexue yu Tansuo 3*(4), 368–377.

Domeniconi, C. et D. Gunopulos (2001). Incremental support vector machine construction. In *ICDM*, pp. 589–592.

Domingos, P. et G. Hulten (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80. ACM New York, NY, USA.

Domingos, P. et G. Hulten (2001). Catching up with the data : Research issues in mining data streams. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*.

Domingos, P. et M. Pazzani (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning 130*, 103–130.

Dong, J.-x., A. Krzyzak, et C. Y. Suen (2005). Fast SVM training algorithm with decomposition on very large data sets. *IEEE transactions on pattern analysis and machine intelligence 27*(4), 603–618.

Fawcett, T. (2004). ROC graphs : Notes and practical considerations for researchers. *Machine Learning 31*, 1–38.

Fayyad, U. M., G. Piatetsky-Shapiro, P. Smyth, et R. Uthurusamy (1996). *Advances in Knowledge Discovery and Data Mining*. Menlo Park, CA, USA : American Association for Artificial Intelligence.

Féraud, R., M. Boullé, F. Clérot, F. Fessant, et V. Lemaire (2010). The orange customer analysis platform. In *Industrial Conference on Data Mining (ICDM)*, pp. 584–594.

Ferrer-Troyano, F., J. Aguilar-Ruiz, et J. Riquelme (2006). Data streams classification by incremental rule learning with parameterized generalization. In *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 661. ACM.

Ferrer-Troyano, F., J. S. Aguilar-Ruiz, et J. C. Riquelme (2005). Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 ACM symposium on Applied computing*, pp. 572. ACM.

Fung, G. et O. Mangasarian (2002). Incremental support vector machine classification. In *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, Virginia*, pp. 247–260.

Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC Press.

Gama, J., P. Medas, et P. Rodrigues (2005). Learning decision trees from dynamic data streams. *Journal of Universal Computer Science 11*(8), 1353–1366.

Gama, J. et K. P. (2009). Tracking recurring concepts with metalearners. In *Progress in Artificial Intelligence : 14th Portuguese Conference on Artificial Intelligence*, pp. 423.

Gama, J. et C. Pinto (2006). Discretization from data streams : applications to histograms and data mining. *Proceedings of the 2006 ACM symposium on Applied*.

Gama, J., R. Rocha, et P. Medas (2003). Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 523–528. ACM New York, NY, USA.

Gama, J., P. P. Rodrigues, R. Sebastiao, et P. Rodrigues (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 329–338. ACM New York, NY, USA.

Gehrke, J., R. Ramakrishnan, et V. Ganti (2000). RainForest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery 4*(2), 127–162.

Gibbons, P., Y. Matias, et V. Poosala (2002). Fast incremental maintenance of approximate histograms. *ACM Transactions on Database 27*(3), 261–298.

Globersonn, A. et S. Roweis (2005). Metric learning by collapsing classes. In *Neural Information Processing Systems(NIPS)*.

Gomes, J. B., E. Menasalvas, et P. A. S. Sousa (2010). Tracking recurrent concepts using context. In *Proceedings of the 7th international conference on Rough sets and current trends in computing (RSCTC'10)*.

Greenwald, M. et S. Khanna (2001). Space-efficient online computation of quantile summaries. In *SIGMOD*, pp. 58–66.

Guyon, I., V. Lemaire, G. Dror, et D. Vogel (2009). Analysis of the kdd cup 2009 : Fast scoring on a large orange customer database. *JMLR : Workshop and Conference Proceedings 7*, 1–22.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*.

Hulten, G., L. Spencer, et P. Domingos (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 97–106. ACM New York, NY, USA.

Joaquin Quinonero-Candela, J., M. Sugiyama, A. Schwaighofer, et N. D. Lawrence (2009). *Dataset shift in Machine Learning*. MIT Press.

John, G. et P. Langley (1995). Estimating continuous distributions in bayesian classifiers.

In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345. Morgan Kaufmann.

Kirkby, R. (2008). *Improving Hoeffding Trees*. Ph. D. thesis, University of Waikato.

Kohavi, R. (1996). Scaling up the accuracy of naive-Bayes classifiers : A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Volume 7. Menlo Park, USA : AAAI Press.

Kohavi, R. et C. Kunz (1997). Option decision trees with majority votes. In *ICML '97 : Proceedings of the Fourteenth International Conference on Machine Learning*, San Francisco, CA, USA, pp. 161–169. Morgan Kaufmann Publishers Inc.

Kolter, J. et M. Maloof (2003). Dynamic weighted majority : A new ensemble method for tracking concept drift. In *Proceedings of the Third International IEEE Conference on Data Mining*, pp. 123–130.

Kononenko, I. et M. Robnik (2003). Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning Journal 53*, 23–69.

Lallich, S., O. Teytaud, et E. Prudhomme (2007). Association rule interestingness : Measure and statistical validation. In F. Guillet et H. Hamilton (Eds.), *Quality Measures in Data Mining*, Volume 43 of *Studies in Computational Intelligence*, pp. 251–275. Springer Berlin / Heidelberg.

Langley, P., W. Iba, et K. Thompson (1992). An analysis of bayesian classifiers. In *International conference on Artificial Intelligence, AAAI*, pp. 223–228.

Law, Y. et C. Zaniolo (2005). An adaptive nearest neighbor classification algorithm for data streams. *Lecture notes in computer science 3721*, 108.

Lazarescu, M. M., S. Venkatesh, et H. H. Bui (2004). Using multiple windows to track concept drift. *Intelligent Data Analysis 8*(1), 29–59.

Littlestone, N. et M. Warmuth (1989). The weighted majority algorithm. *30th Annual Symposium on Foundations of Computer Science*, 256–261.

Loosli, G., S. Canu, et L. Bottou (2006). SVM et apprentissage des très grandes bases de données. *CAp Conférence d'apprentissage*.

Lu, J., Y. Yang, et G. Webb (2006). Incremental discretization for naive-bayes classifier. *Advanced Data Mining and Applications*, 223 – 238.

Maloof, M. et R. Michalski (2000). Selecting examples for partial memory learning. *Machine Learning 41*(1), 27–52.

Marsland, S. (2003). Novelty detection in learning systems. *Neural Computing Surveys 3*, 157–195.

Mehta, M., R. Agrawal, et J. Rissanen (1996). SLIQ : A fast scalable classifier for data mining. *Lecture Notes in Computer Science 1057*, 18–34.

Michalski, R. S., I. Mozetic, J. Hong, et N. Lavrac (1986). The Multi-Purpose incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1041–1045.

Moreno-Seco, F., L. Mico, et J. Oncina (2002). Extending laesa fast nearest neighbour algorithm to find the k nearest neighbours. In *SSPR & SPR*, pp. 718–724.

Narasimhamurthy, A. et L. Kuncheva (2007). A framework for generating data to simulate changing environments. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference : artificial intelligence and applications*, Volume 549, pp. 389. ACTA Press.

Oates, T. et D. Jensen (1997). The Effects of Training Set Size on Decision Tree Complexity. In *ICML '97 : Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 254–262.

Page, E. (1954). Continuous inspection schemes. *Biometrika 41*(1-2), 100.

Provost, F. et V. Kolluri (1999). A Survey of Methods for Scaling Up Inductive Algorithms. *Data Min. Knowl. Discov. 3*(2), 131—-169.

Quinlan, J. R. (1986). Learning Efficient Classification Procedures and Their Application to Chess End Games. *Machine Learning - An Artificial Intelligence Approach*, 463–482.

Ramos-Jimenez, G., J. del Campo-Avila, et R. Morales-Bueno (2006). Incremental algorithm driven by error margins. *Lecture Notes in Computer Science 4265*, 358.

Salganicoff, M. (1997). Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review 11*(1), 133–155.

Sankaranarayanan, J., H. Samet, et A. Varshney (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*.

Saunier, N., S. Midenet, et A. Grumbach (2004). Apprentissage incrémental par sélection de données dans un flux pour une application de sécurité routière. In *Conférence d'Apprentissage (CAP)*, pp. 239–251.

Schlimmer, J. et D. Fisher (1986). A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 496–501.

Schlimmer, J. et R. Granger (1986). Incremental learning from noisy data. *Machine learning 1*(3), 317–354.

Seidl, T., I. Assent, P. Kranen, R. Krieger, et J. Herrmann (2009). Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*, pp. 311–322. ACM.

Shafer, J., R. Agrawal, et M. Mehta (1996). SPRINT : A scalable parallel classifier for data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 544–555.

Stonebraker, M., U. Çetintemel, et S. Zdonik (2005). The 8 requirements of real-time stream processing. *ACM SIGMOD Record 34*(4), 42–47.

Street, W. et Y. Kim (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 377–382. ACM New York, NY, USA.

Syed, N., H. Liu, et K. Sung (1999). Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 317–321. ACM New York, NY, USA.

Tsang, I., J. Kwok, et P. Cheung (2006). Core vector machines : Fast SVM training on very large data sets. *Journal of Machine Learning Research 6*(1), 363.

Usunier, N., A. Bordes, et L. Bottou (2010). Guarantees for approximate incremental SVMs. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Volume 9, pp. 884–891.

Utgoff, P. (1989). Incremental induction of decision trees. *Machine Learning 4*(2), 161–186.

Utgoff, P., N. Berkman, et J. Clouse (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning 29*(1), 5–44.

V. Hooman, V., C.-S. Li, et V. Castelli (2000). Fast search and learning for fast similarity search. In *Storage and Retrieval for Media Databases*, Volume 3972, pp. 32–42.

Vitter, J. (1985). Random sampling with a reservoir. *ACM Trans. Math. Software 11*(1), 37–57.

Wang, H., W. Fan, P. S. Yu, et J. Han (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, New York, New York, USA, pp. 226–235. ACM Press.

Weinberger, K. et L. Saul (2009). Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research (JMLR) 10*, 207–244.

Widmer, G. et M. Kubat (1992). Learning flexible concepts from streams of examples : FLORA2. In *Proceedings of the 10th European conference on Artificial intelligence*, Number section 5, pp. 463–467. John Wiley & Sons, Inc.

Widmer, G. et M. Kubat (1996). Learning in the presence of concept drift and hidden contexts. *Machine learning 23*(1), 69–101.

Witten, I. H. et E. Frank (2005). *Data mining : practical machine learning tools and techniques*. Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, second edition.

Zighed, D. et R. Rakotomalala (2000). *Graphes d'induction : apprentissage et data mining*. Hermes Science Publications.

Zilberstein, S. et S. Russell (1996). Optimal composition of real-time systems. *Artificial Intelligence 82*(1), 181–213.

Summary

The last ten years were proficient in the statistical learning and data mining field and it is now easy to find learning algorithms which are fast and automatic. Historically a strong hypothesis was that data were all available or could be loaded into memory so that learning algorithms can use them straight away. But recently new use cases generating lots of data come up as for example: monitoring of telecommunication network, user modeling in dynamic social network, web mining... The volume of data increases very rapidly and it is now necessary to use incremental learning algorithms. This article presents the main approaches of incremental supervised classification available in the literature. It aims to give basics knowledge to the reader who begins in this subject.