Actes des Ateliers d'EGC 2012

CIDN : Classifications incrémentales et méthodes de détection de nouveauté

Pascal Cuxac ; Jean-Charles Lamirel ; Vincent Lemaire

Le développement de méthodes d'analyse dynamique de l'information, comme le clustering incrémental et les méthodes de détection de nouveauté, devient une préoccupation centrale dans un grand nombre d'applications dont le but principal est de traiter de larges volumes d'information variant au cours du temps. Ces applications se rapportent à des domaines très variés et hautement stratégiques, tels que l'exploration du Web et la recherche d'information, l'analyse du comportement des utilisateurs et les systèmes de recommandation, la veille technologique et scientifique, ou encore, l'analyse de l'information génomique en bioinformatique...
L'objectif de cet atelier est de réunir des chercheurs confirmés, ainsi que des jeunes chercheurs, autour des problématiques et des applications de la "classification incrémentale", et de la "détection de nouveauté" sur des types de données variées, afin d'échanger nos réflexions sur les travaux en cours ainsi que sur les points bloquants.
Initialement axé sur le clustering, cet atelier s'ouvre sur toute méthode de classification de données évolutives.

Vers le site de l'atelier --> CIDN

Introduction à la détection des évolutions dans les corpus textuels

Pascal Cuxac
INIST-CNRS, Vandoeuvre les Nancy, France
Jean-Charles Lamirel
TALARIS-LORIA, Vandoeuvre les Nancy, France

Avec l'augmentation croissante de documents nécessaires aux entreprises ou aux administrations, ainsi que la profusion de données disponibles via Internet, les méthodes automatiques de fouilles de données (text mining, data mining) sont devenues incontournables. Nous nous intéressons ici aux méthodes de classification non supervisée appliquées aux textes et essentiellement à des textes scientifiques.

L'un des objectifs principaux de l'analyse de l'information scientifique et technique est d'identifier les changements majeurs liés aux évolutions de la science. Les technologies émergentes jouent en effet un rôle essentiel dans les avancées scientifiques ou, industrielles.

La détection de technologies émergentes demeure néanmoins un processus complexe, et fait donc l'objet d'études dans un large spectre de domaines.

Afin de repérer et analyser les émergences à large échelle, il est notamment important de développer les méthodes d'analyse diachronique automatisées permettant de suivre les évolutions temporelles des données sur des corpus de texte de taille importante. Dans le domaine des méthodes de classification non supervisées, deux approches sont possibles (1) réaliser des classifications à différentes périodes de temps et analyser les évolutions entre les différentes phases (approche par pas de temps), (2) développer des méthodes de classification qui permettent de suivre directement les évolutions (les méthodes de classifications dynamiques ou incrémentales).

Après un bref tour d'horizon des méthodes de classification non supervisées nous illustrerons les problèmes posés et les résultats obtenus dans le cadre de l'analyse statique et dynamique des textes scientifiques. Nous présenterons finalement les perspectives de recherche dans ce domaine et ferons quelques propositions concernant l'évolution des méthodes dynamiques.

Diagnostic adaptatif en univers évolutif

René Quiniou*

avec la collaboration de

Thomas Guyet**

*INRIA/IRISA Rennes
**Agrocampus Ouest/IRISA Rennes

Les systèmes de surveillance ou de diagnostic à base de modèles observent à intervalles plus ou moins réguliers un système pour déterminer son état courant, diagnostiquer les éventuels dysfonctionnements et prédire son état futur. Pour ce faire, les systèmes de surveillance utilisent une représentation du fonctionnement du système observé sous forme d'un modèle de bon fonctionnement ou d'un modèle de panne.

Dans un univers évolutif ou lorsque le système observé peut s'altérer, le modèle doit être modifié en cours de fonctionnement. Lorsque le modèle est construit par apprentissage automatique cela signifie qu'il faut réapprendre ou modifier incrémentalement le modèle au cours de la surveillance du système. Certaines approches proposent une adaptation continue des modèles.
Nous nous intéressons plutôt aux approches révisant le modèle uniquement lorsque nécessaire.
Le système de surveillance doit alors effectuer deux tâches en ligne :

1) déterminer si l'adéquation du modèle au concept cible, i.e. la configuration courante du système observé, n'est plus suffisante,
2) puis adapter le modèle à la situation courante.

Une difficulté spécifique au diagnostic en univers évolutif est la capacité du système de surveillance à distinguer entre occurrence d'une panne et apparition d'un changement marquant une dérive du concept cible. Ce diagnostic est d'autant plus sensible qu'il est lié à l'adaptation du modèle.

Dans cet exposé, nous présenterons les problématiques liées à la détection de changement et à l'adaptation de modèles. Nous présenterons différentes formulations et solutions proposées pour résoudre ces problèmes, en particulier, par des méthodes d'apprentissage incrémental.
Nous illustrerons ces propositions par des applications traitées dans l'équipe DREAM de l'INRIA/IRISA dans des domaines tels que le monitoring cardiaque, la détection d'intrusion sur serveur HTTP ou la surveillance en environnement.

Arbres en ligne basés sur des statistiques d'ordre

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

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

Résumé. De nouveaux domaines d'application émergent où les données ne sont plus persistantes mais "passagères" : gestion de réseaux, modélisation de profils web... Ces données arrivent rapidement, massivement, et ne sont visibles qu'une fois. Il est donc nécessaire de les apprendre au fur et à mesure de leurs arrivées. Pour les problèmes de classification, les modèles à base d'arbres de décision en ligne sont performants et très largement utilisés sur ces problématiques. Dans cet article, nous proposons une nouvelle approche de construction d'un arbre uniquement basée sur des statistiques d'ordre. Le résumé habituellement utilisé dans la construction d'un arbre en ligne est basé sur une estimation de quantiles. Une méthode performante et robuste de discrétisation supervisée utilise ce résumé pour fournir à la fois un critère de coupure et un estimateur de densité par intervalle. Cette estimation permet la construction d'un classifieur Bayésien naïf dans les feuilles qui améliore la prédiction en début d'apprentissage.

1 Introduction

Les techniques d'apprentissage 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 gros effort a été réalisé pour des analyses de données homogènes et stationnaires. La plupart des approches d'apprentissage statistique (machine learning) utilisent des bases d'apprentissage de taille finie et produisent des modèles statiques.

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", sous la forme de flux. Parmi ces domaines on citera : la gestion de réseaux de télécommunications, la modélisation des utilisateurs au sein d'un réseau social, le web mining... L'un des défis techniques est de concevoir des algorithmes permettant de traiter ces nouvelles contraintes applicatives. Les données arrivant rapidement et n'étant visibles qu'une fois, il est nécessaire de les apprendre au fur et à mesure de leur arrivée. L'apprentissage incrémental apparaît dès lors comme une solution naturelle à ces problèmes de flux.

Parmi les méthodes d'apprentissage incrémental, les modèles à base d'arbres de décision inspirés de l'algorithme « Very Fast Decision Tree » (VFDT) (Domingos et Hulten, 2000)

sont très largement utilisés. La construction de l'arbre est incrémentale et les feuilles se transforment en nœud au fur et à mesure de l'arrivée des exemples. Les nouveaux exemples descendent l'arbre et sont agrégés par variable dans un résumé. Un critère (indice de Gini ou entropie) est ensuite appliqué sur ce résumé afin de trouver les points de coupure pour transformer une feuille en nœud. La qualité de prédiction de l'arbre peut encore être améliorée par l'ajout d'un modèle local dans chacune des feuilles comme dans VFDTc (Gama et al., 2003).

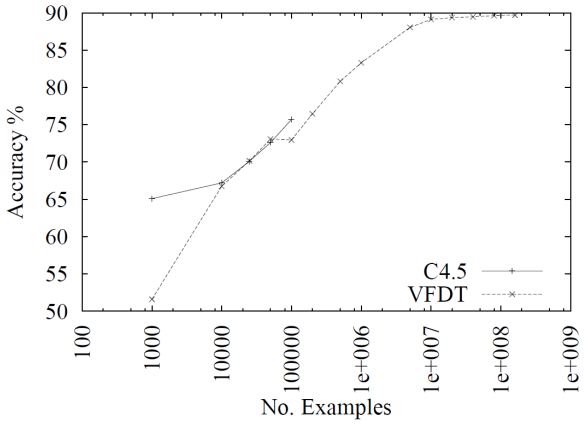

FIG. 1 – *Comparaison de C4.5 avec VFDT (extraite de (Domingos et Hulten, 2000))*

Le taux d'erreurs de l'algorithme est plus important en début d'apprentissage qu'un algorithme « batch » comme C4.5. Mais 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 illustre ce comportement et l'intérêt de VFDT par rapport à C4.5.

Dans cet article nous nous intéressons à la construction d'arbres de décisions en ligne. La section 2 présente notre positionnement par rapport aux approches existantes en termes de critère de coupure, résumé dans les feuilles, et de modèles locaux. En section 3 notre approche exclusivement basée sur des statistiques d'ordre est détaillée. La partie expérimentale comparant notre approche à celles existantes se trouve en section 4. La section 5 discute des travaux futurs et des possibilités de gestion de la dérive de concept issue de notre approche. La dernière partie conclut cet article.

2 Travaux antérieurs

La construction d'arbres de décision en ligne repose sur trois choix principaux. Premièrement, il est impossible dans le contexte des flux de données potentiellement de taille infinie de conserver tous les exemples. L'utilisation de résumé de données de taille finie est nécessaire afin de pouvoir contrôler la consommation mémoire de l'arbre. Le fait que les décisions soient locales à chaque feuille justifie le stockage des résumés dans chacune des feuilles. Deuxièmement le choix du point de coupure se fait par l'évaluation dans chaque feuille d'un critère

(généralement l'indice de Gini ou l'entropie). Ce choix étant une action définitive il faut s'assurer de sa robustesse et qu'il soit fait avec une certaine confiance. Finalement avant qu'une coupure ne se produise, l'information disponible dans les feuilles doit pouvoir être exploitée. L'utilisation d'un modèle local dans chacune des feuilles de l'arbre permet d'exploiter cette information afin d'améliorer la prédiction globale de l'arbre. La figure 2 illustre par un exemple simple l'approche des arbres en ligne et leurs différents composants.

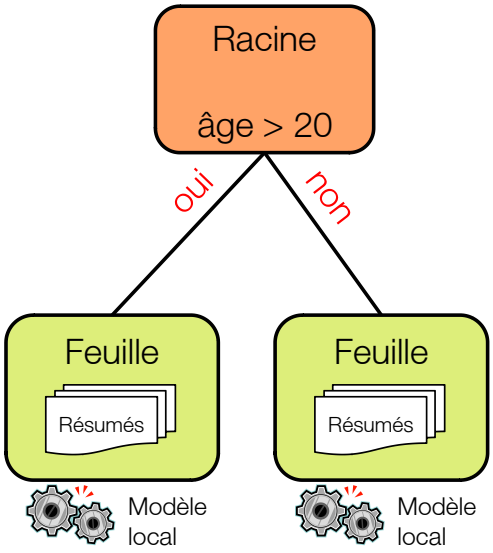

FIG. 2 – *Différents composants d'un arbre en ligne*

Cette section présente les différentes approches utilisées dans la littérature pour répondre aux trois points clés énoncés ci-dessus. La qualité d'un arbre en ligne dépend (i) des résumés dans les feuilles, (ii) du critère de coupure utilisé, (iii) du modèle local choisi.

2.1 Résumés dans les feuilles

Le but des résumés de données dans les feuilles d'un arbre de décision en ligne est de mémoriser les caractéristiques du flux et ainsi de pouvoir par la suite trouver le meilleur point de coupure pour transformer la feuille en nœud. Ce résumé peut aussi être utilisé pour construire un modèle local dans la feuille. Dans certains domaines applicatifs, comme par exemple la gestion d'un réseau de télécommunication ou d'énergie, le flux est potentiellement de taille infinie. L'arbre généré possèdera alors de fait un grand nombre de nœuds de décision. Cependant la mémoire disponible est elle souvent de taille finie. Il en découle que ces résumés doivent être de faibles tailles et gérer le compromis précision/erreur.

Les paragraphes ci-dessous illustrent certaines de ces techniques de résumés, où les attributs numériques et catégoriels sont en général traités à l'aide de techniques de résumés différentes.

2.1.1 Attributs numériques

Gama et Pinto proposent une méthode de résumé des attributs numériques appelée Partition Incremental Discretization - PiD (Gama et Pinto, 2006) qui est partiellement incrémentale. Cette solution est basée sur deux niveaux. Le niveau 1 réalise une première discrétisation où les comptes sont stockés par intervalle. Ce premier niveau implémente un algorithme incrémental qui combine les méthodes « Equal Frequency » et « Equal Width » et il doit contenir plus d'intervalles que le niveau 2. Le deuxième niveau utilise la discrétisation réalisée au niveau 1 pour effectuer la discrétisation finale qui peut être de plusieurs types : « Equal Frequency », « Equal Width », K-means, Recursive entropy discretization, Proportional discretization. La consommation mémoire de cette méthode dépend principalement du nombre d'intervalles du premier niveau. Le deuxième niveau n'est lui pas incrémental.

Dans le cadre des arbres en ligne (Pfahringer et al., 2008) ont réalisé une étude sur les résumés pour les attributs numériques ; étude dédiée aux arbres utilisant la borne d'Hoeffding. Ils ont testé différentes méthodes pour la réalisation de ces résumés :

- VFML (Very Fast Machine Learning) : cette méthode, très simple, prend les k premières valeurs du flux et s'en sert pour construire $k + 1$ intervalles. Les valeurs suivantes sont agrégées dans l'intervalle le plus proche défini par les premières valeurs. La consommation mémoire dépend du k choisi. Cette méthode provient du code source (Hulten et Domingos, 2003) fourni par Domingos et Hulten créateur de VFDT.
- Approximation par une Gaussienne : la distribution des données est supposée être une loi normale que l'on va chercher à approximer. Pour cela il suffit de ne conserver que les trois éléments qui définissent cette Gaussienne : la moyenne, l'écart type (ou la variance) et le nombre d'éléments. Le maintien de ces trois valeurs pouvant se faire de manière incrémentale cette méthode est parfaitement adaptée à une utilisation en ligne. Elle est choisie par Kirkby (Kirkby, 2008) dans toutes ses expérimentations. La consommation mémoire est constante quel que soit la nature du flux observé.
- Arbres binaires exhaustifs : Gama et al. utilisent cette méthode pour leur arbre VFDTc (Gama et al., 2003). Un arbre binaire de recherche, par variable numérique, est construit de manière incrémentale dans lequel toutes les valeurs observées sont stockées. La structure de l'arbre et la mémorisation des comptes des valeurs observées permet un accès immédiat aux comptes de part et d'autre d'un point de coupure. La consommation mémoire de l'arbre dépend du nombre de valeurs différentes arrivant dans la feuille.
- GK : cette méthode est un résumé basé sur des quantiles. Elle maintient un tableau trié d'intervalles et contrôle l'erreur sur la position du quantile. Elle sera détaillée dans la section 3.1.1. La consommation mémoire dépend soit de l'erreur maximale tolérée sur les quantiles soit du nombre fixé de quantiles à conserver.

2.1.2 Attributs catégoriels

A notre connaissance toutes les publications précédemment citées dans le cadre des arbres en ligne utilisent une seule et unique méthode de résumé pour les attributs catégoriels qui est basée sur un comptage exhaustif. Cette méthode consiste à compter le nombre d'occurrences d'une valeur par variable. Elle peut être suffisante si le nombre de valeurs est limité et donc que ce comptage exhaustif peut tenir dans l'espace mémoire accordé pour le comptage. Sa consommation mémoire dépend donc du nombre de valeurs différentes du flux de données.

2.2 Critère de coupure

Lors de la construction de l'arbre de décision un critère de pureté 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 transformer une feuille en un nœud il faut déterminer à la fois l'attribut sur lequel couper ainsi qu'un point de coupure.

La littérature des arbres de décision hors ligne et en ligne utilise principalement deux critères de coupure : l'indice de Gini que l'on retrouve dans l'algorithme CART et le « gain ratio » basé sur l'entropie utilisé dans C4.5. Ces deux critères permettent de trouver le meilleur point de coupure pour un attribut et fournissent un score. Il suffit ensuite de réaliser la coupure sur l'attribut ayant le meilleur score. Ce processus de transformation d'une feuille en nœud se répète afin d'induire l'arbre de décision final.

La différence entre un arbre en ligne et hors-ligne réside dans le fait que l'arrivée des données se fait en continu. Le choix de l'attribut de coupure se fait d'après le résumé construit et non à l'aide de toute la distribution des exemples. Le choix permettant la transformation d'une feuille en nœud est une action définitive. Afin de s'assurer que ce choix soit réalisé avec une certaine confiance, Domingos et Hulten propose dans VFDT d'utiliser la borne d'Hoeffding (Hoeffding, 1963). Cette borne apporte une garantie sur le choix du bon attribut. La borne d'Hoeffding a par la suite été très souvent utilisée pour la réalisation d'arbre de décision en ligne : VFDTc (Gama et al., 2003), CVFDT (Hulten et al., 2001), IADEM (Ramos-Jimenez et al., 2006), « ensemble d'arbres d'Hoeffding » (Kirkby, 2008)... Nous utiliserons aussi dans la suite de cet article cette borne dans le cadre de la construction de l'arbre en ligne proposé. Nous la détaillons ci-dessous.

Borne d'Hoeffding 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 le tirage de 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 que : (i) de la plage de valeurs R, (ii) du nombre d'observations n, (iii) de la confiance désirée δ. Cette borne est plus conservatrice qu'une borne faisant une hypothèse sur la distribution des valeurs.

Cette borne permet de s'assurer (avec une probabilité de $1-\delta$) que le choix de l'attribut de coupure dans un nœud est le bon. Pour cela la borne est utilisée sur la moyenne d'un critère G d'évaluation de la qualité d'une coupure. Le meilleur attribut a est considéré définitivement meilleur que le deuxième meilleur attribut b si $\bar{G}_a - \bar{G}_b > \epsilon$.

2.3 Modèles locaux

Gama et al. (Gama et al., 2003) observent empiriquement 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 global tant que la feuille n'est pas transformée en nœud. Ils proposent alors d'utiliser ces exemples en ajoutant dans chaque feuille un modèle local. Cet algorithme, dénommé VFDTc, supporte par ailleurs les attributs numériques. On peut noter que cette technique d'utilisation de modèles locaux positionnés dans les feuilles d'un arbre existe depuis longtemps. Par exemple l'algorithme NBTree (Kohavi, 1996) utilise un arbre de décision avec un classifieur Bayésien naïf dans les feuilles.

Un bon modèle local pour les arbres en ligne doit consommer peu d'espace mémoire, être rapide à construire et à retourner une prédiction. Il doit de plus être en adéquation avec les résumés construits dans les feuilles. Une bonne capacité de prédiction du modèle local avec un faible nombre d'exemples est nécessaire car les résumés dans les feuilles peuvent être basés sur peu d'exemples. Une étude (Salperwyck et Lemaire, 2010) menée sur la vitesse (en nombre d'exemples) d'apprentissage de différents classifieurs montre que les forêts d'arbre et le classifieur Bayésien naïf sont des classifieurs qui requièrent peu d'exemples pour bien apprendre. Parmi ces deux classifieurs seul le classifieur Bayésien naïf respecte aux mieux les conditions imposées par la construction en ligne. En effet, il ne nécessite aucun espace mémoire supplémentaire si le résumé permet de retourner une estimation de densité par intervalle (ce qui est le cas de tous les résumés présentés précédemment). De plus il est rapide à construire et possède une faible complexité algorithmique en prédiction. Ce classifieur est d'ailleurs celui choisi dans VFDTc et il améliore la prédiction de l'arbre sur les jeux de données WaveForm et LED de l'UCI (Gama et al., 2003).

3 Approche proposée

Cet article introduit des travaux en cours sur la construction d'arbres de décisions en ligne basés sur les statistiques d'ordre. Pour les résumés nous avons choisi des méthodes consommant peu de mémoire et contrôlant leurs erreurs en plus d'être basées sur les statistiques d'ordre. Pour le critère et les modèles locaux, le choix s'est porté sur la méthode MODL qui permet à la fois d'évaluer un point de coupure et de fournir une discrétisation ensuite utilisable par un classifieur Bayésien naïf.

3.1 Résumés dans les feuilles

Les résumés utilisés par notre approche ont à la fois une consommation mémoire fixe et des garanties fortes en termes de contrôle de l'erreur sur les comptes. Ces résumés sont présentés ci-dessous.

3.1.1 Attributs numériques : résumé de quantiles (GK)

Les quantiles fournissent une statistique d'ordre sur les données. Le ϕ-quantile, avec $\phi \in [0, 1]$ est défini comme étant l'élément à la position $\lceil \phi N \rceil$ dans la suite triée des N valeurs vues jusqu'à présent. Soit ϵ l'erreur que l'on s'autorise sur la position de l'élément. Un élément est une ϵ approximation d'un ϕ-quantile si son rang est entre $\lceil (\phi - \epsilon)\, N \rceil$ et $\lceil (\phi + \epsilon)\, N \rceil$. Ceci revient au même que de réaliser une discrétisation en « Equal Frequency » ; le nombre de quantiles étant le nombre d'intervalles dans ce cas là.

Le résumé de quantiles GK (Greenwald et Khanna, 2001) est un algorithme de calcul de quantile dont la consommation de mémoire est en $O(\frac{1}{\epsilon} log(\epsilon N))$ dans le pire des cas. Cette méthode ne nécessite pas de connaître la taille des données à l'avance et est insensible à l'ordre d'arrivée des données. Selon le besoin, et pour un nombre de quantiles fixé, on peut soit définir une borne d'erreur maximale, soit définir une borne de mémoire maximale à utiliser. L'algorithme est basé sur un tableau de tuples $< v_i, g_i, \Delta_i >$ où :

– v_i est une valeur du flux de données

– g_i correspond au nombre de valeurs vues entre v_{i-1} et v_i
– Δ_i est l'erreur maximale possible sur g_i

Cette méthode est évaluée dans (Pfahringer et al., 2008) en conservant un résumé par classe et par attribut. Elle est évaluée comme étant moins performante que la méthode par Gaussienne ou la méthode VFML. Cependant nous avons adapté le résumé GK afin de n'avoir qu'un résumé par attribut mais dans lequel chaque tuple contient le nombre d'individus par classe. Expérimentalement cette modification améliore la qualité de prédiction (pour des raisons de place les expérimentations sur ce point ne sont pas détaillées dans cet article).

3.1.2 Attributs catégoriels : Count-min Sketch (CMS)

Le but du Count-min Sketch (Cormode et Muthukrishnan, 2005) est de trouver les top-k plus importantes occurrences d'une valeur avec une erreur maximale ϵ dans un flux de données. Une matrice de comptage de taille $t \times b$ est utilisée pour son stockage. Il utilise t fonctions de hachage h_i dans $\{1, \ldots, b\}$ qui sélectionnent la cellule de la matrice à incrémenter :

$$\forall i = 1, \ldots, t \quad h_i(e) = h_i(e) + 1$$

Le choix de t et b se fait à l'aide de deux paramètres δ et ϵ. Si l'on veut que l'estimation de la fréquence d'apparition $\hat{f}$ d'un item n'ait pas une erreur supérieure à ϵn avec une probabilité d'au moins $1 - \delta$ alors il faut prendre $t = log\frac{1}{\delta}$ et $b = O(\frac{1}{\epsilon})$. La fréquence d'un élément e est estimée par le minimum de $h_i(e)$.

$$\hat{f} = \underset{i}{argmin}(h_i(e))$$

L'utilisation du Count-min Sketch est pertinente quand le nombre de valeurs différentes est important. Dans le cas contraire l'utilisation d'un simple comptage est un meilleur choix qui n'introduit pas d'erreurs et consomme peu de mémoire.

3.2 Critère

Afin d'être cohérent avec nos résumés (GK et CMS), le critère MODL qui est basé sur les statistiques d'ordre et les comptes a été choisi pour réaliser les coupures et les groupements. L'approche MODL, initialement appliquée à la discrétisation et au groupage de valeurs, permet d'évaluer la qualité d'une coupure ou d'un groupement de valeurs respectivement pour une variable numérique et catégorielle. Son indice d'évaluation $l \in [0..1]$ correspond à un taux de compression qui indique l'informativité d'une variable numérique ou catégorielle.

Les méthodes MODL de discrétisation (Boullé, 2006) et de groupage de valeurs (Boullé, 2005) sont supervisées et sans paramètre. Elles se basent sur les comptes des classes par rapport à tous les découpages possibles en intervalles pour les variables numériques et en groupes pour les variables catégorielles. L'évaluation de la qualité du modèle est basée sur une approche Bayesienne. Son but est de trouver les meilleurs paramètres de la discrétisation/groupage : nombre d'intervalles, bornes des intervalles et répartition des classes dans les intervalles au sens Bayésien.

L'arbre, proposé dans cet article, est construit en ligne à la manière de VFDT en utilisant la borne d'Hoeffding mais le critère MODL remplace le critère du gain en Entropie. L'utilisation

du critère MODL a un intérêt supplémentaire car il retourne une valeur de critère $l > 0$ si et seulement si le modèle de discrétisation/groupage est meilleur que le modèle qui retourne la classe majoritaire. Cette propriété permet un pré élagage automatique de l'arbre alors que dans VFDT ce pré élagage a dû être implémenté séparément. De plus ce critère n'évalue pas seulement une coupure binaire mais peut aussi évaluer les coupures n-aires, ce qui permet de construire des arbres ayant des nœuds avec plus de deux fils.

3.3 Modèles locaux : approche à deux niveaux basée sur les statistiques d'ordre

Nous avons choisi le classifieur Bayésien naïf comme modèle local dans les feuilles pour ses performances lorsqu'il est construit avec peu de données ainsi que pour sa faible complexité algorithmique (voir section 2.3). Ce classifieur nécessite une estimation de la densité conditionnelle aux classes. Cette densité est estimée au sein de chaque intervalle ou groupe de valeurs calculés par la méthode MODL appliquée respectivement aux résumés GK ou CMS contenus dans les feuilles. Cette discrétisation sur les résumées correspond à la méthode de discrétisation à deux niveaux proposés par Gama avec sa méthode PiD (voir 2.1.1). Toutefois dans notre cas, notre approche à deux niveaux est basée entièrement sur des statistiques d'ordre et peut traiter indifféremment des attributs numériques ou catégoriels. Pour les variables numériques, le premier niveau correspond au résumé de quantiles GK et le second à la méthode de discrétisation MODL. Pour les variables catégorielles, le premier niveau correspond au Count-min Sketch (ou a un comptage) suivi du groupage MODL.

L'un des intérêts de la méthode MODL réside dans sa robustesse et son absence de paramètres. L'approche MODL discrétise/groupe une variable numérique/catégorielle en plusieurs intervalles/groupes si et seulement si cette discrétisation/groupage est meilleure (au sens Bayésien) que le modèle à un seul intervalle/groupe. Dans le cas ou le modèle le plus probable est le modèle à un intervalle (ou à un seul groupe de valeurs) cela signifie que l'attribut n'est pas informatif étant donné les données observées. Dans ce cas c'est la classe majoritaire qui est prédite. Kirkby dans (Kirkby, 2008) étudie justement ce comportement en comparant des arbres d'Hoeffding prédisant la classe majoritaire et ceux ayant un classifieur Bayésien naïf dans les feuilles. Il observe que parfois l'un est meilleur parfois l'autre. De ce fait il propose une méthode choisissant automatiquement soit l'un soit l'autre en évaluant leur taux d'erreur sur les exemples qui passent dans les feuilles. La méthode MODL devrait permettre de retrouver automatiquement ce comportement.

4 Expérimentations

Cette section constitue une évaluation préliminaire de la pertinence de notre approche par rapport aux méthodes décrites en section 2. Nous présentons tout d'abord les flux utilisés, puis les différentes méthodes évaluées et enfin les résultats obtenus.

4.1 Flux utilisés

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

l'art de ces générateurs est présenté dans (Kirkby, 2008) ou (Gama, 2010). Pour nos expérimentations nous allons utiliser les générateurs suivants :

- Random RBF (Radial Basis Function) : plusieurs centroïdes définies par leur centre et leur diamètre sont tout d'abord générées à différentes positions dans l'espace. Puis des exemples appartenant à ces bulles sont générés en s'éloignant du centre d'une distance générée aléatoirement à partir d'une loi gaussienne.
- Random Tree : génération d'un arbre avec certains paramètres (profondeurs minimale et maximale, nombres de variables numériques et catégorielles) puis génération de données à partir de cet arbre. Ce générateur favorise les algorithmes d'apprentissage basés sur les arbres.
- Waveform : problème à trois classes généré à partir de fonctions en forme d'ondes différentes et combinées. Ce générateur est basé sur une loi Gaussienne et favorise donc les classifieurs faisant une hypothèse Gaussienne.
- Function (F6) : génération d'exemples à partir d'une fonction basée sur 6 attributs numériques et 3 attributs catégoriels. Dans nos expérimentations nous avons choisi d'utiliser la fonction N°6 proposé par Agrawal avec 5% de bruit.

4.2 Algorithmes testés

Pour nos expérimentations nous avons utilisé la boîte à outils MOA : Massive Online Analysis (Bifet et al., 2009) développée par l'université de Waikato qui reprend la librairie VFML fourni par Hulten et Domingos (Hulten et Domingos, 2003). Cette boîte à outils permet de générer les flux et fournit les algorithmes de l'état de l'art auxquels nous souhaitons nous comparer. Nous y avons ajouté nos arbres basés sur les statistiques d'ordre.

Tous les arbres évalués partent du même algorithme basé sur les arbres d'Hoeffding. Cet algorithme contient trois paramètres : (i) le nombre d'exemples à considérer avant d'essayer de trouver un point de coupure ; (ii) la précision que l'on souhaite avoir sur la moyenne dans la borne d'Hoeffding et (iii) le paramètre τ qui est utilisé pour imposer le choix entre deux attributs dont la différence de critère est trop proche pour que la borne d'Hoeffding puisse les départager. Le paramétrage de toutes les versions des arbres testés ici est le même à savoir : $n = 200$, $\delta = 10^{-6}$, $\tau = 0.05$.

Les résumés dans les feuilles pour les attributs catégoriels n'utilisent pas, dans ces expériences, le count-min sketch et le groupage MODL car les jeux de données utilisés comportent peu de valeurs différentes. Le résumé est donc basé sur un simple comptage. Les attributs numériques sont eux résumés soit par une estimation basée sur une Gaussienne soit par des quantiles GK.

L'attribut et le point de coupure sélectionnés pour transformer une feuille en nœud sont ceux possédant la valeur de critère maximale non nulle. Ce critère évalue, pour les variables catégorielles le modèle créant une feuille par valeur de la variable et pour les variables numériques uniquement les coupures binaires.

Les différentes approches ont été testées soit sans **modèles locaux** soit avec un classifieur Bayésien naïf dans chaque feuille. Le tableau 1 présente une vue synthétique des algorithmes présentés ci-dessous :

- **HT : Arbre d'Hoeffding.** Cet algorithme correspond à l'arbre d'Hoeffding de VFDT (Domingos et Hulten, 2000). Les résumés numériques sont basés sur l'estimation de densité par une Gaussienne par classe (évalué comme étant le plus performant dans

(Pfahringer et al., 2008)). Pour les attributs numériques 10 coupures binaires par classe, prises entre le minimum et le maximum observés, sont évaluées. Cet arbre ne possède pas de classifieur dans les feuilles.

- **HTNB : Arbre d'Hoeffding avec résumé Gaussien et classifieur Bayésien naïf.** Cette version est identique à la précédente mais possède un classifieur Bayésien naïf dans les feuilles. Les densités conditionnelles sont estimées pour (i) les variables numériques à l'aide de la loi Gaussienne paramétrée par les 3 éléments du résumé (μ, σ, n) ; et pour (ii) les variables catégorielles à l'aide des comptes par classe et par valeur.
- **HTmGKc : Arbre d'Hoeffding avec critère de coupure MODL.** Cette version correspond à la version que nous avons décrite dans la section 3. Le résumé pour les variables numériques est basé sur le résumé de quantiles GK avec des tuples contenant directement les comptes par classe. Chaque variable est résumée par 10 tuples. Le critère de coupure utilisé est basé sur le critère MODL décrit section 3.2. Les 10 coupures binaires évaluées proviennent des quantiles du résumé GK. Cette version ne possède pas de modèle local dans les feuilles.
- **HTmGKcNBm : Arbre d'Hoeffding avec critère de coupure MODL et classifieur Bayésien naïf.** Ce classifieur est identique à la version « HTmGKc » mais les feuilles contiennent un classifieur Bayésien naïf utilisant les probabilités conditionnelles aux classes. Celles-ci sont estimées pour les variables numériques par intervalles de valeurs issues du modèle de discrétisation MODL ((Boullé, 2006)) et pour les variables catégorielles à l'aide des comptes par classe et par valeur.

Méthode	Résumé numérique	Critère	Discrétisation numérique	Modèle local
HT	Gaussien	Gain en Entropie	Aucune	Aucun
HTNB	Gaussien	Gain en Entropie	Aucune	NB
HTmGKc	GK 10 tuples	Level MODL	Aucune	Aucun
HTmGKcNBm	GK 10 tuples	Level MODL	MODL sur GK	NB

TAB. 1 – *Spécifications des différents algorithmes testés (NB : Bayésien naïf)*

4.3 Résultats

Les résultats de nos expérimentations sont présentés dans la figure 3 sous la forme d'une courbe par jeu de données. Chaque générateur crée un flux de 11 millions d'exemples. Parmi les méthodes d'évaluation (Gama et al., 2009) nous avons choisi la méthode basée sur un ensemble de test. Le premier million d'exemples générés est utilisé comme ensemble de test et les 10 millions suivants comme ensemble d'apprentissage. Le critère d'évaluation utilisé est la précision et les classifieurs sont évalués tous les 300.000 exemples. De manière générale notre méthode se comporte bien sur les différents flux testés et est compétitive comparée aux autres méthodes. Le critère de coupure MODL basé sur les résumés numériques GK et les comptes par classe pour les variables catégorielles est globalement meilleur que celui basé sur le gain d'entropie sur un résumé numérique Gaussien et les comptes par classe pour les variables catégorielles

Modèle local L'apport du classifieur Bayésien naïf dans les feuilles est discutable dans le cas de l'utilisation du résumé avec une Gaussienne car parfois il peut très significativement soit améliorer (WaveForm) soit dégrader (F6) les performances du classifieur global. Dans le cas de l'utilisation de nos résumés à deux niveaux, le classifieur Bayésien naïf améliore toujours le résultat surtout en début d'apprentissage. Cette non dégradation est dûe à la robustesse de l'approche MODL. Celle-ci crée des intervalles seulement s'ils contiennent de l'information. Si la variable n'est pas informative aucun modèle de discrétisation n'est proposé. L'estimation basée sur ces intervalles est ensuite fournie au classifieur Bayésien naïf.

L'estimation de densité à base de Gaussienne donne de meilleurs résultats sur les flux de données « WaveForm » et « Random RBF ». Ce phénomène est particulièrement marqué en début d'apprentissage. Ce résultat n'est pas surprenant car il paraît normal que le classifieur Bayésien naïf ayant comme estimateur de densité une loi Gaussienne fonctionne bien sur des données issues des générateurs « WaveForm » et « Random RBF » qui utilisent des Gaussiennes pour générer les données.

5 Discussion

5.1 Mémoire limitée et critère global

Le comportement des algorithmes pour une quantité de mémoire limitée n'a pas été étudié dans cet article. Cependant il existe plusieurs techniques pour améliorer un classifieur en ligne utilisant des arbres qui seraient contraints de travailler avec une mémoire limitée.

L'une des plus simples consiste à supprimer le résumé dans les feuilles des variables les moins intéressantes. Le level MODL calculé pour chaque variable fournit une indication précise de l'information contenue dans une variable. Il serait donc intéressant de regarder si son utilisation donne de meilleurs résultats que celle du gain en entropie utilisée dans la littérature.

Une deuxième technique consiste à désactiver des feuilles et donc à limiter l'accroissement d'une partie de l'arbre. Ceci est réalisé en conservant les feuilles contenant le plus d'erreurs et dont on espère le plus d'amélioration. Une autre possibilité serait d'utiliser un critère global qui permettrait de savoir quelle partie de l'arbre peut potentiellement améliorer le plus sa qualité. Cette approche a déjà été utilisée pour une construction d'arbre « hors-ligne » (Voisine et al., 2009). Il s'agirait d'adapter le critère pour une utilisation « en-ligne ».

La troisième technique est de contraindre dynamiquement la quantité de mémoire utilisée par les résumés. Dans notre cas cette approche serait assez simple à réaliser en fusionnant des quantiles (GK) ou en diminuant la taille des nouveaux CMS.

5.2 Gestion de la dérive de concept

Cette section présente brièvement l'intérêt d'utiliser les statistiques d'ordre et le critère MODL dans les arbres en ligne pour détecter le changement de concept. En effet si le processus sous-jacent qui « génère » les données n'est pas stationnaire, le concept cible à apprendre et/ou la distribution des exemples peuvent varier au cours du temps. L'algorithme d'apprentissage doit donc être capable de détecter ces changements et d'adapter le modèle en conséquence.

On considère 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

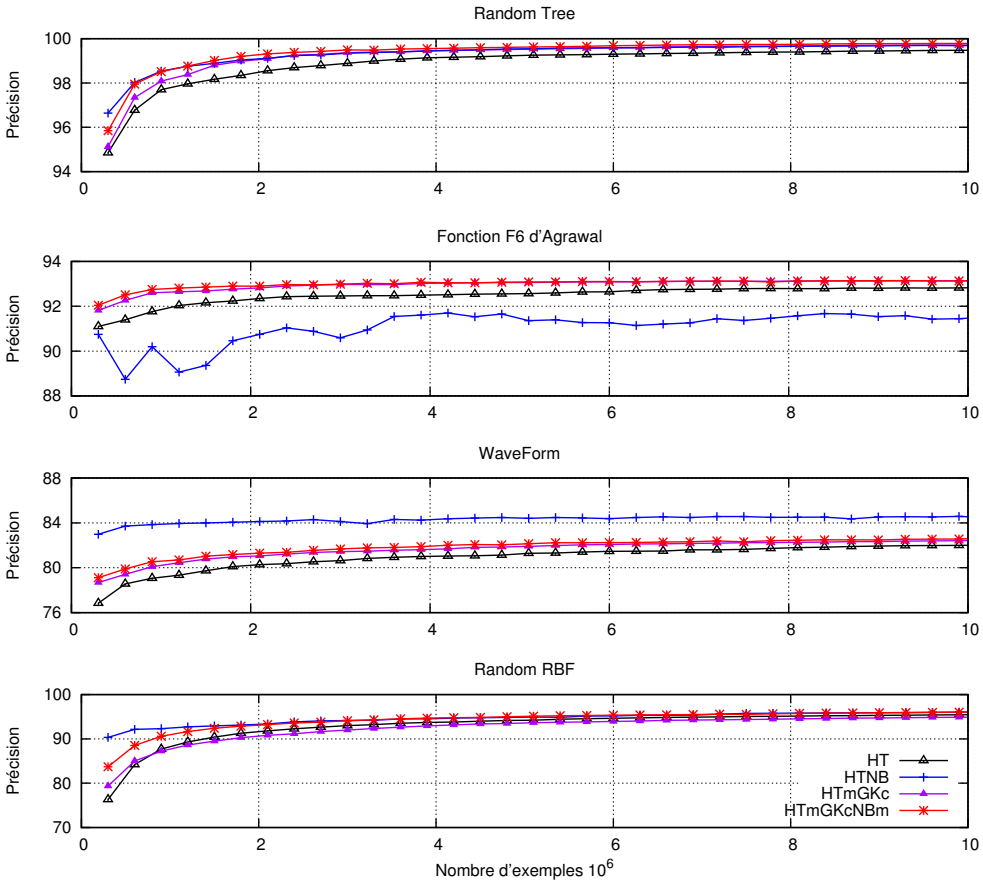

FIG. 3 – *Précision en fonction du nombre de données apprises*

entre deux changements de concept est appelée contexte (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 il existe un concept 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.

La détection du changement est étudiée dans l'état de l'art à l'aide de deux voies prédominantes : (i) la surveillance des performances du modèle courant de classification et (ii) la surveillance de la distribution des exemples. Le lecteur intéressé trouvera dans le tutorial présenté à PAKDD 2011 une excellente introduction sur ce sujet (voir `http://www.cs.waikato.ac.nz/~abifet/PAKDD2011/`).

Dans les sections précédentes de cet article on a présenté les résumés utilisés, les critères

de coupures (ou de groupage pour les attributs catégoriels) ainsi que les modèles locaux utilisés dans les feuilles de l'arbre. Ces trois éléments clefs peuvent aider à gérer le drift. Nous présentons ci-dessous des éléments de discussion de futurs travaux de recherche qui pourront être réalisés.

- **Utilisation des résumés dans les feuilles** : Lors de la décision de couper une feuille pour en faire un nœud de décision il est possible de conserver le résumé utilisé à ce moment précis. Un nouveau résumé est alors construit (mise à jour) en partant éventuellement du clone du résumé initial. Le résumé initial n'étant plus lui mis à jour. Muni de ces deux résumés on peut alors utiliser une méthode similaire à celle proposée dans (Bondu et Boullé, 2011) et qui permet de détecter des changements dans la distribution des exemples.
- **Utilisation du critère de coupure / groupage** : Le critère de coupure utilisé pour les variables numériques ainsi que le critère de groupage pour les variables catégorielles est un critère Bayésien. Ce critère évalue la probabilité d'un modèle, M_d de discrétisation (ou de groupage), $P(M_d|D)$, étant données les données observées par la feuille. Ce critère Bayésien pourrait être utilisé pour détecter le concept drift.
  A l'aide de la formule de Bayes il serait possible de calculer la probabilité que chaque exemple (d) ait été généré par le modèle de discrétisation (ou de groupage) : $P(d_j|M_{d_j})$; j étant l'indice de la variable explicative ayant été utilisée pour réaliser la coupure et M_{d_j} le modèle de discrétisation ou de groupage de la variable j. La comparaison, par nœud de l'arbre, de la distribution des $P(d_j|M_{d_j})$ au moment de la coupure et de la distribution des $P(d_j|M_{d_j})$ sur une fenêtre glissante (les z dernières valeurs de $P(d_j|M_{d_j})$ peuvent alors être stockées dans les feuilles) donnera une indication de la présence d'un concept drift sur j.
- **Utilisation des modèles locaux** : Le classifieur naïf de Bayes présent dans les feuilles utilise comme prétraitement la discrétisation à deux niveaux présentés lors de la section 3.3. Les valeurs des variables numériques et catégorielles sont respectivement discrétisées ou groupées pour produire des probabilités conditionnelles aux classes ($P(d_j|C)$). Ces probabilités conditionnelles sont utilisés pour élaborer le modèle de classification, M_c, le classifieur naïf de Bayes. Sous la condition d'indépendance des variables on pourrait calculer la probabilité que chaque exemple (d) ait été généré par le classifieur naïf de Bayes : $P(d|M_c) \approx \prod_{j=1}^{J} p(d_j|M_c) \approx \prod_{j=1}^{J} p(d_j|M_{d_j})$ (J étant le nombre de variables explicatives utilisées par le classifieur). Là encore la comparaison, par feuille, de la distribution des $P(d|M_c)$ sur une fenêtre de référence et sur une fenêtre glissante donnera une indication de la présence d'un concept drift sur M_c.

Il est ici intéressant de noter que les détections de drift présentées ci-dessus sont réalisées à l'aide des constituants de l'arbre et non à l'aide des performances de l'arbre. Les critères de construction de l'arbre peuvent être utilisés pour replier tout ou partie de l'arbre. Cette approche paraît plus consistante que celles basées sur les techniques de détection basées sur la performance du classifieur ; performance du classifieur rarement utilisée comme critère de construction de ce dernier.

6 Conclusion

Cet article constitue un travail en cours sur une méthode d'arbre de décision incrémentale construit en ligne sur des statistiques d'ordre. Notre approche a été présentée et située par rapport à l'état de l'art sur les différents points le constituant. Les résumés de quantiles GK et le résumé CMS sont utilisés dans les feuilles. Ces résumés sont par la suite utilisés par la méthode MODL pour fournir à la fois un critère de coupure (ou de groupage) et un estimateur de densité par intervalle (ou groupe). Cette estimation permet par la suite de construire un classifieur Bayésien naïf dans les feuilles. Les différentes expérimentations ont montré que notre arbre était performant comparé à quelques méthodes de l'état de l'art. Tout particulièrement le classifieur local améliore toujours la prédiction en début d'apprentissage.

Des expérimentations plus approfondies sont nécessaires afin de pouvoir tester notre approche sur plus de jeux de données et dans un contexte de mémoire limitée. Par ailleurs il serait intéressant de pouvoir comparer notre méthode sur des flux de données réels plutôt que seulement sur des jeux de données générés artificiellement.

Références

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.

Bondu, A. et M. Boullé (2011). Détection de changements de distribution dans un flux de données : une approche supervisée. In *Extraction et gestion des connaissances (EGC'2011)*, pp. 191–196.

Boullé, M. (2005). A grouping method for categorical attributes having very large number of values. *Machine Learning and Data Mining in Pattern Recognition*, 228–242.

Boullé, M. (2006). MODL : A Bayes optimal discretization method for continuous attributes. *Machine Learning 65*(1), 131–165.

Cormode, G. et S. Muthukrishnan (2005). An improved data stream summary : the count-min sketch and its applications. *Journal of Algorithms 55*(1), 58–75.

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.

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.

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

Gama, J. et C. Pinto (2006). Discretization from data streams : applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 662–667.

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. Rodrigues, et R. Sebastião (2009). Evaluating algorithms that learn from data streams. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pp. 1496–1500. ACM New York, NY, USA.

Greenwald, M. et S. Khanna (2001). Space-efficient online computation of quantile summaries. *ACM SIGMOD Record 30*(2), 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. et P. Domingos (2003). VFML – a toolkit for mining high-speed time-changing data streams.

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.

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.

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

Pfahringer, B., G. Holmes, et R. Kirkby (2008). Handling numeric attributes in hoeffding trees. *Advances in Knowledge Discovery and Data Mining*, 296–307.

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.

Salperwyck, C. et V. Lemaire (2010). Impact de la taille de l'ensemble d'apprentissage : une étude empirique. In *Atelier CIDN de la conférence ECG 2011*, Brest.

Voisine, N., M. Boullé, et C. Hue (2009). Un critère d'évaluation Bayésienne pour la construction d'arbres de décision. *Extraction et gestion des connaissances (EGC'2009)*, 259–264.

Summary

New application domains generate data which are not any more persistent but volatile: network management, web profile modeling... These data arrive quickly, massively and are visible just once. It is thus necessary to learn them according to their arrivals. For classification problems on-line decision trees are known to perform well and are widely used on streaming data. In this paper, we propose a new decision tree method based on order statistics. The summary, usually used in the construction of an on-line tree, is here based on bounded error quantiles. A robust and performing discretization or grouping method uses this summary to provide at the same time a criterion to perform a split and interval density estimations. This estimation is then used to build a naive Bayes classifier in the leaves to improve the prediction in the early learning stage.

Un algorithme non supervisé pour la détection de changements dans les flux de données

Guénaël Cabanes*, Younès Bennani*

*LIPN-CNRS, UMR 7030
99 Avenue J-B. Clément, 93430 Villetaneuse, France
cabanes@lipn.univ-paris13.fr

Résumé. Dans de nombreux cas, les bases de données sont en perpétuelle évolution, de nouvelles données arrivant constamment. Les flux de données posent plusieurs problèmes uniques qui rendent caduques les applications de technique standard d'analyse de données. En effet, ces bases de données sont perpétuellement en ligne, grossissant au fur et à mesure de l'arrivée de nouvelles données. De plus, la distribution de probabilité associée aux données peut changer au cours du temps ("dérive de concept"). Nous proposons donc dans cet article une méthode de représentations synthétiques de la structure des données permettant un stockage efficace des informations du flux, ainsi qu'une mesure de dissimilarité entre ces représentations pour la détection de dérive de concept.

1 Introduction

Dans de nombreux cas, les bases de données sont en perpétuelle évolution, elles sont caractérisées par une structure variable dans le temps, de nouvelles données arrivant constamment. Parfois, l'évolution et la masse des données sont tellement importantes qu'il est impossible de la stocker dans une base et que seule une analyse "à la volée" est possible. Ces processus sont appelés "analyse des flux de données", ils ont fait l'objet de nombreux travaux ces dernières années du fait du nombre important d'applications possibles dans de nombreux domaines (Guha et Harb, 2008; Balzanella et al., 2009). Cependant, l'étude des flux de données est un problème difficile à cause des coûts de calculs et de stockages associés aux volumes mis en jeux. Dans le domaine de la fouille de données, les principaux enjeux pour l'étude des flux de données sont d'une part la description condensée des propriétés d'un flux (Gehrke et al., 2001; Manku et Motwani, 2002) mais aussi la détection de variation ou de changement dans la structure du flux (Cao et al., 2006; Aggarwal et Yu, 2007).

Les flux de données posent plusieurs problèmes uniques qui rendent caduques les applications de technique standard d'analyse de données. En effet, ces bases de données sont perpétuellement en ligne, grossissant au fur et à mesure de l'arrivée de nouvelles données. De ce fait, les algorithmes efficaces doivent être capable de travailler avec une occupation mémoire constante malgré l'évolution des données, la base entière ne pouvant être conservée en mémoire. Cela implique éventuellement l'oubli d'information au court du temps. Un autre problème est connu sous le nom de "dérive de concept" : la distribution de probabilité associée

aux données peut changer au cours du temps. Tout algorithme d'apprentissage à partir de flux devrait être capable de détecter et gérer ce type de situation. Dans le cadre de l'apprentissage supervisé (chaque donnée est associée à une classe, que l'algorithme doit apprendre à prédire), plusieurs solutions on été proposées pour la classification de flux de données en présence de dérive de concept. Ces solution sont en général basées sur une maintenance adaptative d'une structure discriminante. On peut mentionner des méthodes d'ensemble de règles binaires (Widmer et Kubat, 1996), des arbre de décision (Hulten et al., 2001) et des ensembles de classifieurs (Street et Kim, 2001; Kolter et Maloof, 2005).

Nous nous plaçons pour cet article dans un cadre non supervisé (les étiquettes de classes sont inconnues), qui nécessite aussi des adaptations à la présence de dérive de concept pour l'analyse de flux de données. Nous proposons donc une méthode de représentations synthétiques de la structure des données ainsi qu'une mesure heuristique de dissimilarité entre ces modèles permettant de détecter les variations temporelles de la structure du flux (les dérives de concepts). L'avantage de cette méthode est la comparaison des structures par l'intermédiaire des modèles qui les décrivent, ce qui permet des comparaisons à n'importe quelle échelle temporelle sans surcharge de la mémoire de stockage. Ainsi il est possible de comparer la structure du flux à deux périodes potentiellement très éloignées dans le temps, puisque les modèles décrivant ces périodes peuvent être stockés en mémoire à très faible coût.

La section 2 propose une description générale de la méthode non supervisée d'analyse du flux et de détection de dérive de concept. La section 3 présente l'algorithme de représentations synthétiques des informations structurelles des données. La construction d'un modèle de la distribution des données est proposée dans les section 4, puis une mesure de dissimilarité permettant la détection de dérive de concept est présentée dans la section 5. Finalement, la section 6 décrit une méthode adaptée de compression des informations.

2 Description générale de la méthode

La méthode d'analyse du flux et de détection de dérive de concept non supervisé que nous proposons procède en quatre étapes :

- Construction d'une représentation synthétique à intervalles réguliers sur un réservoir de données qui se vide au fur et à mesure que de nouvelles données sont enregistrées. Cette représentation synthétique est basé sur l'apprentissage d'une SOM et permet un clustering automatique des données du réservoir. Pendant l'apprentissage, chaque prototype de la SOM est associé à de nouvelles informations extraites des données. Ces informations structurelles seront utilisées durant la deuxième étape pour estimer la distribution des données. Plus particulièrement, les informations associées aux prototypes sont :
  - *Mode de densité*. C'est une mesure de la densité des données au voisinage du prototype (densité locale). La densité locale est une mesure de la quantité de données présente dans une région restreinte de l'espace de description. Nous utilisons un estimateur à Noyau Gaussien (Silverman, 1981) pour cette tâche.
  - *Variabilité locale*. Il s'agit d'une mesure de la variabilité des données représentées par le prototype. Cette variabilité peut être définie comme la distance moyenne entre le prototype et les données qu'il représente.

  - *Le voisinage*. Il s'agit d'une mesure du voisinage du prototype. La valeur de voisinage entre deux prototypes est le nombre de données ayant ces deux prototypes comme meilleurs représentants.
- Estimation de la distribution des donnés à partir de chaque représentation synthétique. Cette estimation se fait hors ligne et ne nécessite pas de stockage de données. Elle est modélisée sous la forme d'une fonction de densité à partir d'un modèle de mélange de noyaux Gaussiens sphériques.
- Comparaison des distributions pour la détection de dérive de concept. Nous proposons l'utilisation d'une mesure de dissimilarité capable de comparer les deux fonctions de densité estimées à l'étape précédente.
- Compression des informations enregistrées sur le flux pour un stockage à taille mémoire fixe tenant compte d'une arrivée permanente de nouvelles données.

3 Construction d'une représentation synthétique

Dans cette étape, certaines informations générales sont extraites à partir des données et stockées dans les prototypes lors de l'apprentissage de la SOM. Dans notre algorithme, les prototypes de la SOM vont être "enrichis" par l'addition de nouvelles valeurs numériques extraites de la structure des données.

L'algorithme d'enrichissement procède en trois étapes :

Entrée :
- Les données $X = \{x^{(k)}\}_{k=1}^{N}$.

Sortie :
- Une estimation de la densité D_j et de la variabilité locale s_j associées à chaque prototype w_j.
- Une estimation des valeurs de voisinage $v_{i,j}$ associées à chaque paire de prototype w_i et w_j.

1. **Initialisation** :
   - Initialisation des paramètres de la SOM
   - $\forall i, j$ les densités locales (D_j), les valeurs de voisinages ($v_{i,j}$), les variabilités locales (s_j) et le nombre de données représentées par w_j (N_j) sont initialisés à zero.

2. **Tirage aléatoire d'une donnée** $x^{(k)} \in X$:
   - Calcul de $d(w_j, x^{(k)})$, la distance euclidienne entre la donnée $x^{(k)}$ et chaque prototype w_j.
   - Recherche des deux meilleurs prototypes (BMUs : Best Match Units) w_{u^*} et $w_{u^{**}}$:

$$u^* = \arg\min_i(d(w_i, x^{(k)})) \text{ et } u^{**} = \arg\min_{i \neq u^*}(d(w_i, x^{(k)}))$$

   .

3. **Mise à jour des informations structurelles** :
   - Variabilité :

$$s_{u^*}(t) = s_{u^*}(t-1) - \varepsilon(t) r(t) \left(s_{u^*}(t-1) - d(w_{u^*}, x_k)\right)$$

– Densité :

$$\forall j, D_j(t) = D_j(t-1) - \varepsilon(t) r(t) \left(D_j(t-1) - e^{-\frac{\|x^{(k)} - w_j(t)\|^2}{2\sigma^2}} \right)$$

– Voisinage :

$$\begin{aligned} \nu_{u^* u^{**}}(t) &= \nu_{u^* u^{**}}(t-1) - \varepsilon(t) r(t) \left(\nu_{u^* u^{**}}(t-1) - 1 \right) \\ \nu_{u^* i}(t) &= \nu_{u^* i}(t-1) - \varepsilon(t) r(t) \left(\nu_{u^* i}(t-1) \right) \quad \forall i \text{ voisin de } u^* \end{aligned}$$

Avec $\varepsilon(t)$ le taux d'apprentissage et $r(t) = \frac{1}{1+e^{\left(-\frac{t}{t_{max}}\right)}}$.

4. **Mise à jour des prototypes de la SOM** w_i comme défini dans (Kohonen, 2001).

5. **Répéter** T **fois les étapes** 2 **à** 4**, jusqu'à ce que** $t = t_{max}$.

À la fin de cette étape, à chaque prototype est associée une valeur de densité et de variabilité, et à chaque paire de prototypes est associée une valeur de voisinage. De ce fait, une grande partie de l'information sur la structure des données est stockée dans ces valeurs. Il n'est plus nécessaire de garder les données en mémoire. Ces informations peuvent être utilisées directement pour effectuer un clustering de la SOM mettant à jour la structure du flux pour la période enregistrée dans le réservoir (voir Cabanes et Bennani (2008, 2010)). La complexité de l'ensemble du processus est linéaire selon le nombre de données, ce qui permet une analyse rapide du flux au cours de son évolution.

4 Estimation de la distribution des données

Cette étape porte sur l'estimation de la distribution sous-jacente des données par une méthode à deux niveaux. L'idée ici est de pouvoir estimer la distribution des données à partir d'un modèle topologique de ces données. Pour cela nous proposons d'estimer une fonction de densité qui associe une valeur de densité à chaque point de l'espace de représentation des données. Nous connaissons la valeur de cette fonction au niveau des prototypes (D_i). Il faut en déduire une approximation de la fonction.

Nous faisons ici l'hypothèse que cette fonction peut être correctement approximée sous la forme d'un mélange de noyaux Gaussiens sphériques ($\{K_i\}_{i=1}^{M}$), où K_i est une fonction Gaussienne centrée sur un prototype w_i et M est le nombre de prototype. La fonction de densité peut alors s'écrire :

$$f(x) = \sum_{i=1}^{M} \alpha_i K_i(x)$$

avec

$$K_i(x) = \frac{1}{\sqrt{2\pi}.h_i} e^{-\frac{|w_i - x|^2}{2{h_i}^2}} \text{ et } \sum \alpha_i = 1$$

La méthode la plus populaire pour estimer un modèle de mélange (C'est à dire trouver h_i et α_i) est l'algorithme EM (Expectation-Maximization, (Dempster et al., 1977)). Cependant, cet algorithme travaille dans l'espace des données. Ici nous avons seulement à disposition la

SOM enrichie au lieu de l'ensemble des données et nous ne pouvons pas utiliser l'algorithme EM (voir Cabanes et Bennani (2010)).

Nous proposons donc une heuristique pour choisir h_i :

$$h_i = \frac{\sum_j \frac{v_{i,j}}{N_i+N_j}(s_i N_i + d_{i,j} N_j)}{\sum_j v_{i,j}}$$

où N_i est le nombre de données représentées par le prototype w_i, s_i est la variabilité de w_i et $d_{i,j}$ est la distance euclidienne entre w_i et w_j. L'idée est que h_i est l'écart type des données représentées par K_i. Ces données sont aussi représentées par w_i et ses voisins. Ainsi h_i dépend de la variabilité s_i calculée pour w_i et les distances $d_{i,j}$ entre w_i et ses voisins, pondérées par le nombre de données représentées par chaque prototypes et par la connectivité entre w_i et ses voisins.

Puisque la densité D au niveau des prototypes w est connue ($f(w_i) = D_i$), on peut déterminer les pondérations α_i. Ces pondérations sont solution du système d'équations linéaires suivant :

$$D = \sum_{i=1}^{M} \alpha_i K_i(w)$$

avec

$$D = [D_j]_{j=1}^{M} \text{ et } w = [w_j]_{j=1}^{M}$$

Cependant, il existe une infinité de solutions à cette équation, ce qui rend impossible toute résolution matricielle basée sur une inversion de matrice. De plus, la solution obtenue par calcul de la pseudo-inverse (Ben-Israel et Greville, 2003) n'est souvent pas satisfaisante, en particulier parce qu'elle peut contenir des valeurs de α négatives qui ne garantissent plus la contrainte : $\forall x, f(x) > 0$. Nous utilisons donc pour résoudre ce système une méthode très simple de descente de gradient. Les α_i sont initialisés par les valeurs de D_i, puis voient leur valeur réduite progressivement (avec une valeur minimum de 0) jusqu'à satisfaire au mieux $D = \sum_{i=1}^{M} \alpha_i K_i(w)$. Ainsi les valeurs de α restent en moyenne proportionnelles aux valeurs de D_i, ce qui satisfait l'hypothèse que chaque densité D_i est générée principalement par le prototype w_i. Pour cela, nous optimisons le critère suivant :

$$\alpha = \arg\min_{\alpha} \frac{1}{M} \sum_{i=1}^{M} \left[\sum_{j=1}^{M} (\alpha_j K_j(w_i)) - D_i \right]^2$$

Ainsi, nous obtenons une fonction de densité qui est un modèle des données représentées par la SOM. Ce type de méthode d'estimation de la distribution des données a été utilisé avec succès dans (Cabanes et Bennani, 2010) dans un cadre différent.

5 Comparaison de distributions pour la détection de dérive de concept

Nous proposons dans cette étape une mesure heuristique de dissimilarité entre deux distributions représentés par des fonctions de densité calculées à l'étape précédente. L'objectif de cette mesure est de pouvoir comparer les distributions de deux ensembles de données décrites dans les mêmes espaces, de façon à détecter si leurs distributions sont identiques, similaires ou nettement différentes (cf. Figure 1). Ce type de comparaisons permet la détection de changement dans la structure d'un flux de données (dérive de concept).

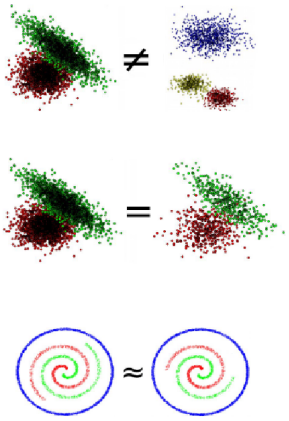

FIG. 1 – *Exemple de comparaisons de la distribution des données.*

5.1 Mesure de dissimilarité

Il est possible de définir une mesure de dissimilarité entre deux ensembles de données A et B, représenté chacun par une SOM et une fonction de densité :

$$SOM_A = \left[\{w_i^A\}_{i=1}^{M^A}, f^A\right]$$

et

$$SOM_B = \left[\{w_i^B\}_{i=1}^{M^B}, f^B\right]$$

Avec M^A et M^B le nombre de prototypes des modèles SOM_A et SOM_B, et f^A et f^B les fonctions de densité de A et B calculées à l'étape précédente.

La dissimilarité entre A et B est :

$$\begin{aligned} CBd(A,B) &= \frac{\sum_{i=1}^{M^A} f^A(w_i^A) log\left(\frac{f^A(w_i^A)}{f^B(w_i^A)}\right)}{M^A} + \frac{\sum_{j=1}^{M^B} f^B(w_j^B) log\left(\frac{f^B(w_j^B)}{f^A(w_j^B)}\right)}{M^B} \\ &= CBd_A + CBd_B \end{aligned}$$

L'idée est de comparer les fonctions de densité f^A et f^B pour chaque prototype w de A et B. Si les distributions sont identiques, ces deux valeurs doivent être très proches.

Cette mesure est une adaptation de l'approximation pondérée de Monté Carlo de la mesure symétrique de Kullback-Leibler (voir (Hershey et Olsen, 2007)), en utilisant les prototypes de la SOM comme un échantillons de l'ensemble des données. L'idée est de comparer pour chaque prototype i d'un modèle la densité D_i estimée à partir des données et la densité théorique au niveau de ce prototype, $Fd(w_i)$, estimée par la fonction de densité de l'autre modèle. Si les modèles sont identiques, ces deux mesures de densité doivent être très proches.

En outre, cet indice vérifie bien les propriétés d'une mesure de dissimilarité :
- Positivité : $CBd(A, B) > 0$
- Symétrie : $CBd(A, B) = CBd(B, A)$
- Séparation : $CBd(A, A) = 0$ par construction.

De façon à démontrer les performances de la mesure de dissimilarité proposée, nous avons utilisé cinq générateurs de données artificielles.

Les générateurs "Ring 1", "Ring 2", "Ring 3", "Spiral 1" et "Spiral 2" génèrent cinq types de jeux de données non-convexes en deux dimensions, de densité et de variance différentes. "Ring 1" est un anneau de rayon 1 (forte densité), "Ring 2" un anneau de rayon 3 (faible densité) et "Ring 3" un anneau de rayon 5 (densité moyenne). "Spiral 1" et "Spiral 2" sont deux spirales parallèles. La densité des points dans les spirales diminue avec le rayon. Des données des différentes distributions peuvent être générées aléatoirement à volonté (Figure 5.1).

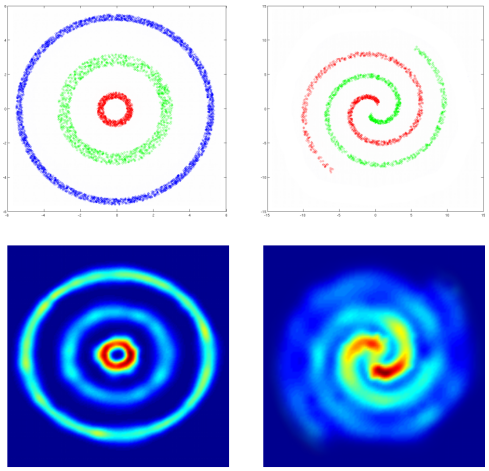

FIG. 2 – *Visualisations des données "Rings" 1 à 3 et "Spirals" 1 et 2 (en haut) et de leur fonction de densité estimée par notre algorithme (en bas).*

Lorsque le nombre de prototypes est suffisant, les différentes distributions sont parfaitement différenciées. Comme on peut le voir sur la figure 3, les distances entre modèles correspondant à une même distribution sont nettement plus faibles que les distances entre modèles

de distributions différentes. Pour visualiser les similarités entre modèles, nous avons utilisé une projection de Sammon (Sammon Jr., 1969)) en deux dimensions, qui respecte au mieux les similarités entre éléments dans l'espace de projection.

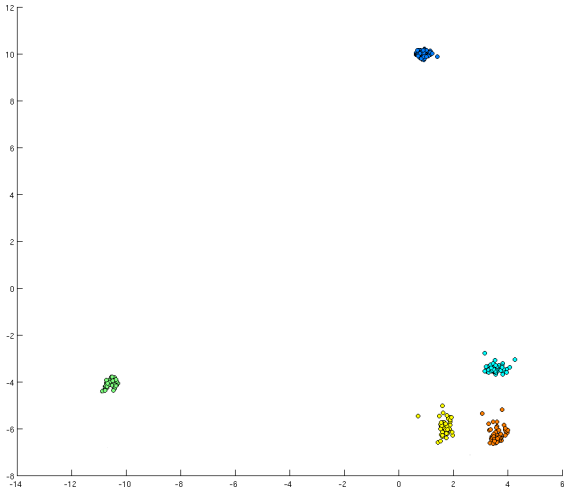

FIG. 3 – *Visualisations des similarités entre modèles (un point = un modèle). Bleu : modèles de l'Anneau 5, Vert : Anneau 1, Turquoise : Anneau 3, Jaune et Orange : Spirales 1 et 2.*

Pour tester la capacité de la méthode à détecter une dérive de concept, nous avons présenté au système des données aléatoires de type "Spirale 1" jusqu'au temps 5 (chaque pas de temps représente un millier de données), puis nous avons présenté des données de type "Anneau 5" jusqu'au temps 20, puis de type "Spirale 2" jusqu'au temps 25 et pour finir de type "Spirale 1" à nouveau jusqu'au temps 30. Le système apprend pour chaque période de temps une SOM enrichie et la compare à celle de la période précédente. Toutes nos expérimentations sont basées sur l'utilisation de la SOMToolbox (Vesanto, 2000) en prenant les paramètres par défaut pour l'apprentissage des SOM.

La figure 4 représente la différence entre les deux modèles de deux jeux consécutifs au cours du temps. Les variations dans la structure du flux sont parfaitement détectées par le système. En effet, lorsque le flux ne varie pas, les modèles correspondants sont très proches et la mesure de dissimilarité donne une valeur très faible. Au contraire, si la structure du flux varie, les modèles correspondants sont nettement moins similaires, et la mesure de dissimilarité est nettement plus élevée.

5.2 Extention aux comparaisons de clusters

L'extension de la méthode à la comparaison de segmentation est naturelle, puisque les modèles à base de SOM sont bien adapté au clustering des données. L'idée est donc de détecter, après clustering de deux ensembles de données, si certains clusters sont communs, si certains sont similaires avec quelques variations (dérive de concept) et si certains clusters sont propres à chaque base :

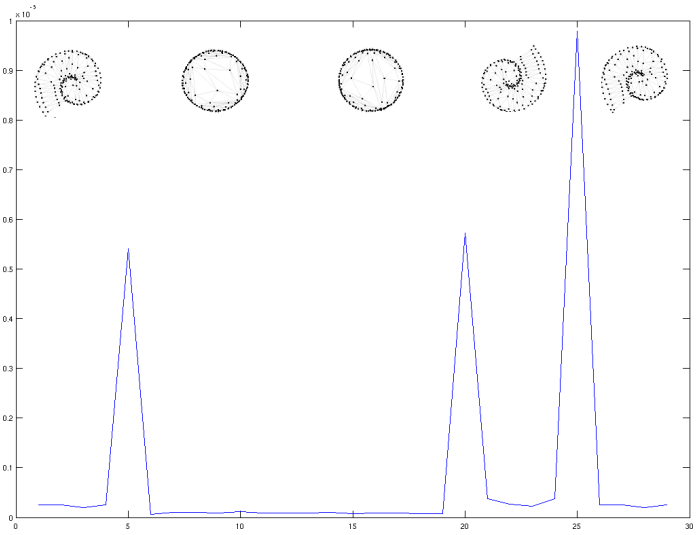

FIG. 4 – *Dissimilarité entre les deux modèles de deux jeux consécutifs au cours du temps. Certains modèles ont été représentés pour illustrer les variations temporelles. Ces variations de la structure du flux (temps 5, 20 et 25) sont parfaitement détectées.*

1. Définir la fonction de densité de chaque cluster C_k des bases A et B :
$$f_k(x) = \sum_{i \in C_k} \alpha_i K_i(x))$$
2. Pour chaque cluster k appartenant à une des bases, trouver le cluster k' le plus similaire dans l'autre base :
$$k' = \arg\min_i CBd(k, i)$$
3. Pondérer la distance entre k et k' par la distance moyenne entre k et les autres clusters de la même base de données que k.
$$CBd_{pond}(k, k') = \frac{CBd(k, k')}{\frac{1}{|C_k|} \sum_{i \in C_k, i \neq k} CBd(k, i)}$$
Remarque : CBd_{pond} n'est pas symétrique.
4. Il est alors possible de comparer les bases de données :
   - Si $CBd_{pond}(k, k') < 1$, il s'agit d'une dérive de concept entre k et k'.
   - Si $CBd_{pond}(k, k') \approx 0$, k et k' sont les mêmes clusters.
   - Si $CBd_{pond}(k, k') \geq 1$, k et k' sont des clusters différents.

De cette façon il est facile de détecter des clusters communs aux deux bases et des clusters propres à chacune.

Il est aussi possible de détecter des fusions ou des fissions de clusters d'une base à l'autre :

1. Décomposer l'indice : $CBd(k, k') = CBd_k + CBd_{k'}$ comme en 5.1.
2. Définir un indice de fusion-fission entre k et k' :

$$F(k, k') = \frac{CBd_k}{CBd_{k'}}$$

3. Analyser F :
   - Si $F(k, k') \approx 1$, les deux clusters sont bien séparés ou très similaires.
   - Si $F(k, k') \ll 1$, k est une partie du cluster k'.
   - Si $F(k, k') \gg 1$, k' est une partie du cluster k.

Ainsi si deux clusters k_1 et k_2 d'une base A ont tous deux pour plus proche correspond un cluster k' d'une base B et que $F(k_1, k') \geq 1$ et $F(k_2, k') \geq 1$, alors k est la fusion de k_1 et k_2, et k_1 et k_2 sont la fission de k.

Cette méthode permet donc une analyse très fine des variations de structures entre deux ensembles de données. En particulier, ce type d'analyse est très utile pour comprendre les variations de la structure d'un flux de données, telles que les apparitions et disparitions de clusters, ainsi que les phénomènes de fusions, fissions ou dérives de concepts.

6 Compression et stockage de la structure du flux

Une difficulté de l'analyse d'un flux de données est que ce flux est potentiellement infini. On ne peut donc pas se permettre de stoker indéfiniment des cartes SOM représentant différents instants, puisque les capacités de stockage sont limitées. Nous souhaitons donc proposer une méthode de fusion de SOM enrichies, qui permettra de compresser l'information stockée.

L'idée est de fusionner deux SOM ou plus représentant des instants successifs si la structure des données représentées par ces deux cartes est suffisamment similaire. Nous proposons, si la puissance de calcul disponible le permet, de construire et de mettre à jours une matrice de similarité entre les fonctions de densité représentant la structure du flux pour des périodes consécutives. Pour cela il suffit de comparer chaque nouvelle SOM enrichie stockée avec la SOM la plus récente déjà stockée. Puis, si la place manque, il est possible de fusionner les deux SOM adjacentes les plus similaires. Ainsi, à chaque compression des informations stockées, on conserve le maximum d'information sur les variations du flux.

La fusion de SOM peut se faire en générant des données à partir des fonctions de densités et en lançant l'algorithme SOM enrichi sur ces données.

Soit N SOM enrichies et leur fonction de densité : $SOM^1 = \{N_i^1, w_i^1, \alpha_i^1, h_i^1\}_{i=1}^{M^1}$, ..., $SOM^N = \{N_i^N, w_i^N, \alpha_i^N, h_i^N\}_{i=1}^{M^N}$. L'algorithme de génération des données est le suivant :

1. Sélectionner aléatoirement une des SOM. Chaque SOM A, composée de M^A neurones, a une probabilité d'être choisie de :

$$P(A) = \frac{\sum_{i=1}^{M^A} N_i^A}{\sum_{K=1}^{N} \sum_{i=1}^{M^K} N_i^K}$$

   Autrement dit, plus une SOM représente une grande quantité de données, plus elle a de chance d'être sélectionnée.

2. Sélectionner aléatoirement un neurone i de la SOM choisie en fonction du paramètre α, qui représente la contribution du neurone vis-à-vis de la fonction de densité :

$$P(i) = \frac{\alpha_i}{\sum_{j=1}^{M} \alpha_j}$$

3. Générer une donnée aléatoirement selon une distribution gaussienne sphérique centrée sur w_i et d'écart-type h_i.

Il suffit alors d'appliquer un algorithme SOM enrichi sur les données générées et d'estimer une fonction de densité pour obtenir une représentation condensée de la structure des SOM d'origine.

7 Conclusion

Dans cet article nous proposons une méthode non supervisée d'analyse des flux de données. Cette méthode permet l'analyse, la compression et le stockage des informations de la structure du flux et de ses variations au cours du temps. Il est suffisamment rapide pour être applicable aux grands flux de données : la complexité de représentation synthétique est linéaire selon la taille du réservoir et les étapes ultérieures n'ont pas besoin des données pour être effectuées, puisqu'elles se basent exclusivement sur cette représentation synthétique. Nous avons montré à travers quelques exemples que la méthode est capable de détecter de manière assez fine les variations de la structure du flux et les dérives de concept.

Ces résultats préliminaires doivent maintenant être confirmés par un passage à l'échelle, au travers d'applications réelles sur des grands flux de données avec un nombre de dimentions plus important. Il est important de vérifier que ce type de méthode est capable de détecter des dérives de concept plus progressives que celles testées. De plus, nous travaillons actuellement sur une version adaptative de l'algorithme SOM enrichie pour la représentation synthétique des données, qui sera capable de suivre le flux en temps réel et d'adapter sa représentation au cours du temps, avec une fusion incrémentale des SOM en cours d'analyse.

Remerciements

Ce travail a été financé par l'Agence Nationale de la Recherche dans le cadre du projet E-Fraud Box (ANR-09-SECU-03).

Références

Aggarwal, C. et P. Yu (2007). A Survey of Synopsis Construction Methods in Data Streams. In C. Aggarwal (Ed.), *Data Streams : Models and Algorithms*, pp. 169–207. Springer.

Balzanella, A., Y. Lechevallier, et R. Verde (2009). A new approach for clustering multiple streams of data. In S. Ingrassia et R. Rocci (Eds.), *Classification and Data Analysis*.

Ben-Israel, A. et T. N. E. Greville (2003). *Generalized Inverse : Theory and Applications.* New York : Springer Verlag.

Cabanes, G. et Y. Bennani (2008). A local density-based simultaneous two-level algorithm for topographic clustering. In *Proceedings of IJCNN'08*, pp. 1176–1182.

Cabanes, G. et Y. Bennani (2010). Unsupervised topographic learning for spatiotemporal data-mining. *Advances in Artificial Intelligence 2010, Article ID 832542, 12 pages*.

Cao, F., M. Ester, W. Qian, et A. Zhou (2006). Density-based clustering over an evolving data stream with noise. In *2006 SIAM Conference on Data Mining*, pp. 328–339.

Dempster, A. P., N. M. Laird, et D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B 39*, 1–38.

Gehrke, J., F. Korn, et D. Srivastava (2001). On computing correlated aggregates over continual data streams. In *Special Interest Group on Management of Data Conference*, pp. 13–24.

Guha, S. et B. Harb (2008). Approximation algorithms for wavelet transform coding of data streams. *IEEE Transactions on Information Theory 54*(2), 811–830.

Hershey, J. R. et P. A. Olsen (2007). Approximating the Kullback Leibler divergence between Gaussian mixture models. In *Proceedings of IEEE ICASSP'07*, Volume 4, pp. 317–320.

Hulten, G., L. Spencer, et P. Domingos (2001). Mining time-changing data streams. In *International Conference on Knowledge Discovery and Data Mining*, pp. 97–106.

Kohonen, T. (2001). *Self-Organizing Maps*. Berlin : Springer-Verlag.

Kolter, J. Z. et M. A. Maloof (2005). Using additive expert ensembles to cope with concept drift. In *ICML*, pp. 449–456.

Manku, G. S. et R. Motwani (2002). Approximate frequency counts over data streams. In *Very Large Data Base*, pp. 346–357.

Sammon Jr., J. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computer 18*(5), 401–409.

Silverman, B. (1981). Using kernel density estimates to investigate multi-modality. *Journal of the Royal Statistical Society, Series B 43*, 97–99.

Street, W. N. et Y. Kim (2001). A streaming ensemble algorithm (sea) for large-scale classification. pp. 377–382. ACM Press.

Vesanto, J. (2000). Neural network tool for data mining : SOM Toolbox.

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

Summary

In many cases, databases are in constant evolution, new data arriving continuously. Data streams pose several unique problems that make obsolete the applications of standard data analysis methods. Indeed, these databases are constantly on-line, growing with the arrival of new data. In addition, the probability distribution associated with the data may change over time ("concept drift"). We propose in this paper a method of synthetic representation of the data structure for efficient storage of information, and a measure of dissimilarity between these representations for the detection of concept drift.

Reacting to Concept Changes using a Committee of Experts

Ghazal Jaber[*,**], Antoine Cornuéjols[*]
Philippe Tarroux[**]

[*]AgroParisTech, département MMIP et INRA UMR-518
16 rue Claude Bernard
F-75231 Paris Cedex 5, France
ghazal.jaber, antoine.cornuejols@agroparistech.fr,
http://www.agroparistech.fr/mia/equipes:membres:page:ghazal
[**]Université de Paris-Sud, LIMSI, Bâtiment 508,
F-91405 Orsay Cedex, France
ghazal.jaber, philippe.tarroux@limsi.fr

Abstract. We present a general framework to deal with concept changes in online machine learning. Our approach relies on a committee of experts where each expert is trained on a different history size. The experts change constantly based on their performance which creates a dynamic committee that can adapt to a large variety of concept changes. The experiments, based on synthetic data, simulate abrupt and global concept changes. We test different methods to weight the experts and to combine their predictions. The experimental results show that our ensemble algorithm learns a concept change better than when a single expert learns each concept separately. Different types of experts as neural networks, SVMs and others may coexist in the committee in order to increase the diversity and improve the overall performance. We show how our algorithm is robust to the existence of potentially "bad" types of experts in the committee.

1 Introduction

Classical machine learning algorithms suppose that the training data are independent and identically distributed and that the concept they aim to learn is stationary i.e. does not change with time. These conditions are not met in general in online learning where the training data is received in a stream generated by a system with potentially evolving states. This system, sometimes referred to as a hidden context (Widmer and Kubat (1996)), may introduce changes in the training data, producing a "concept change".

Different types of concept changes exist in the litterature: A *concept drift* (Zliobaite (2009)) refers to the change of the statistical properties of the concept's target value (Cornuéjols (2010)). For example, the behavior of customers in shopping might evolve with time and thus the concept capturing this behavior evolves as well. The speed of the change can be gradual or sudden. A sudden drift is sometimes referred to as a *concept shift* (Widmer and Kubat (1996), Yang et al. (2006)). Another type of change, known as *virtual concept drift* (Syed et al.

(1999)), *pseudo-concept drift* (Ruping (2001)), *covariate shift* (Bickel et al. (2009)) or *sample selection bias* (Fan et al. (2005)) occurs when the distribution of the training data in the feature space changes with time. Even if the underlying concept remains the same, the model should be updated in order to cope with the current distribution. In Minku et al. (2010), any kind of concept change is a concept drift. Thus, the same denomination might not necessarly mean to the same thing in the community. However, according to Stanley (2003), it is not important from a practical point of view to categorize concept changes based on the source of change (change in the feature space distribution, the target values or both) since the model should be revisited anyway. Minku et al. (2010) divides changes into different types, according to different criteria (severity, speed, frequency, recurrence and predictability) creating mutually-exclusive and non-heterogeneous categories.

A main issue when dealing with concept drift is to find the optimal history size to learn the concept. When the concept is stable, a long history of training data may allow for more precise predictions. However, a long history can also be misleading if the concept is changing since it will contain old and expired data. In such case, a small history allows for a fast adaptation to the change. This is the basis of the well-known stability-plasticity dilemma.

Time sliding windows on the training data have been suggested to deal with concept drift. The window size can be fixed or adaptative. When the window size is fixed, selecting a "good" window size requires an apriori knowledge about the change in order to deal with the stability-plasticity dilemma. An algorithm that fixes the window size also assumes that the environment behaves the same way all the time, a condition that is not met in the real world. Adaptative window sizes were a solution to this problem. In Widmer and Kubat (1996), the window keeps growing when the concept is stable. When a change is detected, the window size is shrinked accordingly. Example weighting and selecting techniques have also been suggested. The training instances can be weighted depending on the instance age or performance regarding the current concept.

Several online ensembles methods have been proposed for tackling changing concepts. Ensemble methods have the advantage of holding diverse experts in the ensemble. According to Minku et al. (2010), diversity helps reduce the initial drop in accuracy that happens just after the change. When the concept is stable, however, low diversity in the ensemble gives more accurate results. In many ensemble approaches, experts are removed from the ensemble when their performance drops under a threshold and are then replaced by new experts with a small window size. Expulsing an expert from the ensemble can be the result of a concept change which makes its training window unadapted to the current situation. Thus, we don't need to explicitly detect a change and adapt the expert's window size: this can be done implicitly with a committee of experts.

In this paper, we suggest an online ensemble method that deals with concept changes. We solve the stability-plasticity dilemma by using a committee of experts where each expert is trained on a different history size. The committee members change constantly based on their prediction performance: the lowest performing members are removed from the committee and replaced with new members with a small history size. The history size of the remaining

members is incremented. Thus, we create a committee of experts whose history size evolves dynamically, which allows the adaptation to different types of changes.

The rest of this paper is organized as follows. In Section 2, we discuss related work on ensemble methods and introduce our online ensemble algorithm that deals with concept drift. The algorithm is explained in detail in Section 3. We conducted several experiments on artificial data sets to describe the behavior of the algorithm in practice. The results are reported in Section 4. Finally, in Section 5, we summarize and suggest future research directions.

2 State of the Art

Many ensemble algorithms have been proposed to deal with concept drift. The ensemble methods can be grouped as follows:

Data receipt: The data is either received and processed one instance at a time or in sequential batches. A main problem when learning on sequential batches is that a drift might occur inside a batch. An expert trained on this batch will learn misleading and sometimes contradicting information. Choosing the batch size might also be a problem when we have no apriori knowledge of the training data [1].

Data pre-processing: The data is either used as is or it can be pre-processed by changing the data distribution using sampling and/or weighting methods.

Experts' learning extent: In most cases, when the data is received in batches, an expert learns on a block of data and its learning stops at this point. In other scenarios, however, experts don't stop learning: they constantly update their knowledge with newly observed training data.

Experts deletion: In some approaches, an expert is deleted from the ensemble when its performance drops under a predefined threshold. In some other approaches, whenever the experts in the ensemble are evaluated, the worst expert is removed. In such case, and if the data is processed one instance at a time, good experts might be removed if the expert didn't get the chance to observe enough training data or if the data is noisy. To avoid random and unmeaningful deletion operations, approaches may set a maturity age that prohibits the deletion of an expert if the expert is not mature [2]. Other approaches set a frequency parameter for evaluation, creation & deletion of experts. This deletion problem is not encountered when the data is processed one block at a time since the block size is normally large enough to learn a stable expert.

Ensemble's final prediction: The ensemble classifies a test instance using the experts' predictions and weights. In case of unweighted ensembles, the weights are set to the same value for all the experts. In case of weighted ensembles, each expert is assigned a weight that reflects its predictive performance. Different methods have been proposed for weights computing: most

1. If, for instance, we know that what we learn depends on the season of the year then we may set the batch size such that it covers each season separately.

2. The maturity age corresponds to the minimum number of training examples that the expert should learn in order to be considered as a mature expert.

methods evaluate an expert based on its classification performance on recently observed data. Other methods use local accuracy to weight an expert: the expert's classification performance is evaluated on the neighborhood of the test instance and the weight is set accordingly. The experts predictions are then integrated to give the ensemble's final prediction. Classical integration methods include: majority voting, weighted majority voting, selecting the prediction of the expert with the highest weight, using a roulette-wheel selection on the experts weights etc...

Next, we present five online ensemble approaches that have been proposed to deal with concept drift:

The **KBS-stream algorithm** (Scholz and Klinkenberg (2005)) is a boosting-like method that trains a classifier ensemble from data streams. In Scholz and Klinkenberg (2005), the data are received in batches. With each received batch, either a new classifier is added or the latest added classifier is updated. Before learning, the data distribution in the recent batch is changed using sample weighting and sampling strategy: the data is passed through the existing classifiers and the distribution is modified such that the last classifier learns something different than the remaining ones in the ensemble. The data is assumed to be independent and identically distributed in each batch and a drift is allowed to occur between two consecutive batches but never inside a batch. These conditions are not met in many real-world online problems.

The **ASHT-bagging algorithm** (Bifet et al. (2009)) is a variant of bagging designed to tackle with non-stationary concepts. The approach builds an ensemble of Hoeffding Trees (Domingos and Hulten (2000)) with different tree sizes: small trees to adapt more quickly to changes and large trees which are better for stationary concepts. In this approach, the data is received one instance at a time. A tree in the ensemble is updated with each newly observed training data. When a tree reaches its maximum size, it is pruned, even for stationary concepts. The diversity of the ensemble improves bagging and allows the adaptation to concept drifts.

The **DWM (Dynamic Weighted Majority) algorithm** (Kolter and Maloof (2007)) receives and processes the data one instance at a time. Each classifier in the ensemble is initially assigned a weight of one. The weight is decreased if the classifier misclassifies an instance. This ensemble method copes with concept drift by dynamically adding and removing classifiers: a new classifier is added if the ensemble misclassifies a test sample and a classifier is removed if its weight is lower than a predefined threshold. This strategy makes the ensemble size variable. As for the ensemble final prediction, it corresponds to the class label with the highest accumulated weight.

In the **dynamic integration of classifiers**, Tsymbal et al. (2008) proposes an ensemble integration technique to handle concept drift. In this approach, the data is received as a sequence of fixed-size batches. The data is then divided into overlapping or non-overlapping blocks. With each data block, a new expert is trained and added to the ensemble. When the ensemble size reaches its maximum size, the *replace the loser* pruning strategy is used: the worst expert in the ensemble is removed and replaced by the new one, trained on the most recent data block. In dynamic integration of classifiers, each classifier is given a weight proportionnal to its local accuracy with regard to the instance tested and the best classifier is selected or the classifiers are integrated using weighted voting. They show that dynamic integration gives better accuracy than the most commonly used integration techniques, specially in the case of local drifts.

The **committee of decision trees** in Stanley (2003) generates a weighted committee of decision trees that votes on the current classification. Each expert votes for a newly received test instance then it modifies its knowledge when the instance's label becomes available. The voting performance of each expert is evaluated. If some experts see their voting performance drop under a predefined threshold, the worst classifier is removed and replaced by a new classifier trained on the last observed training instance [3]. This strategy allows the experts to affine their knowledge when the concept is stable. When the concept changes, experts that learnt on old and misleading data will encounter a drop in their performance and will be replaced.

Our algorithm is inspired by the work of Stanley (2003). The main difference is that instead of adding one expert at a time, we add a set of different experts. This allows the increase of the diversity in the committee by using different types of experts. Thus, we can add several models as decision trees, SVMs, neural networks and others and we can also use different structures of the same model as neural networks with different activation functions or layers, decision trees with different sizes, etc... In addition, in our approach, we don't set a predefined threshold for expert deletion. The worst and mature experts are deleted all the time. We show in the experiments that this deletion strategy doesn't affect the committee's predictive performance even when the concept is stable. We test different ways of evaluating and weighting the experts; we also explore different ways of integrating their predictions. When a training instance is to be processed, the experts are evaluated twice: first before learning, when asked to classify the instance, the experts are evaluated for the committee's final prediction, and secondly after the true label of the instance is revealed, the experts are re-evaluated and their weights are used this time to delete the worsts among them. The weighting strategy for both evaluation steps can be different.

3 The Proposed Algorithm

In an online learning scenario, the training data is received in a stream and a training instance is processed once, on arrival. A training example is represented by a pair $(\mathbf{x}, y)$, where $\mathbf{x} \in R^p$ is a vector in a p-dimensional feature space and y is the desired output or target. In a classification problem, y is a discrete value whereas in a regression problem y is a real value. The desired output y can also be represented as a d-dimensional vector, in a more general formulation. A concept represents the distribution of the problem $p(\mathbf{x}, y)$ (Minku et al. (2010)). This distribution might evolve with time creating a concept change.

The concept can be a decision tree, a neural network, an SVM or any other model that can capture $p(\mathbf{x}, y)$. In this paper, the concept is an ensemble of classifiers that predict an instance's target or label. The ensemble is required to adapt rapidly to a concept change and to predict the target y of a test sample $\mathbf{x}$ at anytime. The main parts of the ensemble method are presented next. The pseudo-code is shown in Algorithm 1.

The pool of predictors. In our algorithm a *pool of predictors* is a set of experts with possibly different prediction models (ex: SVM, neural networks, decision trees), different model structures (ex: different activation functions or number of layers in a neural network) and/or

3. The approach uses incremental learning where the training data is learnt in a sequence i.e. one instance at a time.

different training algorithms. The pool of predictors has a size of n_{pool}. The predictors can be used for regression (Jaber et al. (2011)) or classification purposes. In this paper, a predictor predicts the label of an example in a classification task.

The committee of predictors. Our committee consists of pools of predictors. Instead of adding one predictor to the committee at a time, we add a pool of predictors, which allows the existence of various types of prediction models and structures in the committee. We define the maximum number of pools of predictors in the committee as max_{pool}.

The committee before reaching its maximum size. The committee is initially empty. At each time step, a new training data $e = (\mathbf{x}, y)$ is received where $\mathbf{x}$ is the example and y the corresponding label. A new pool of predictor trained on $e = (\mathbf{x}, y)$ is added to the committee. The predictors that are already in the committee are also trained incrementally on the new training data $e = (\mathbf{x}, y)$. For instance, at time step 1, when the first example e_1 is received, a pool b_1 trained only on $\{e_1\}$ is added to the committee. At time step 2, a new pool b_2 is added to the committee, trained on $\{e_2\}$ and the old pool b_1 is trained on $\{e_2\}$ also. Thus, b_1 is trained on $\{e_1, e_2\}$ and b_2 is trained on $\{e_2\}$ only. This operation is repeated until the committee reaches its maximum size. At time step t, the pool b_j is trained on $\{e_j, ..., e_t\}$, where $j \in [1, max_{pool}]$ and $j <= t$.

The committee after reaching its maximum size. The committee reaches its maximum size at time step $t = max_{pool}$. From this time on, the committee is updated by removing the worst predictors and adding new ones. The updating operation however cannot be executed unless all the predictors in the committee are mature i.e. they all have been trained on a minimum number of training data. When a deletion operation is to be executed, each predictor is evaluated and a weight is assigned accordindly. The larger the weight, the better the predictor. The predictors with the lowest weights are removed and replaced by a new pool trained on the last observed example. In order to keep the committee size fixed, the worst $pool_{size}$ predictors are removed from the committee. In this paper, we don't set a specific function to compute the weights; we leave it as a general function. We will explore several methods for weights computing in the experiments section. These weights, used to remove the worst predictors, will be refered to as "deletion weights" in the remainder of this paper.

The committee's final prediction. The committee predicts the label $\tilde{y}$ of an incoming example $\mathbf{x}$. The label of $\mathbf{x}$ is first unknown to the committee. It is just after the comittee predicts its label $\tilde{y}$ that the real label y is revealed. Since each predictor gives its own prediction of the label, how does the committee constructs a final prediction $\tilde{y}$? The "selection weights" are computed to evaluated each predictor in the committee. Then, according to the selection weights, the predictions are combined using an integration function (ex: voting, weighted voting, the prediction of the best predictor etc...). The result of the integration function gives the committee's final prediction. It is important to notice that an unmature predictor is not taken into account in the final prediction, unless all the predictors in the committee are still unmature. In such case, all the predictors share the same weight.

Algorithm 1 The Concept Change Adaptation Algorithm

Require: $\mathbf{x}$ is an incoming example with no corresponding label received yet; n_{pool} is the size of the pool of predictors; max_{pool} is the maximum number of pools in the committee; P is the committee and max_P is the committee's maximum size: $max_P = n_{pool} * max_{pool}$.
Ensure: $\tilde{y}$ is the committee's prediction of $\mathbf{x}$'s label

{PREDICTION}
$H \leftarrow \phi$ is the set of predictions
for $k = 1 \rightarrow$ size(P) **do**
  $p_k \in P$ is the k^{th} predictor
  $\tilde{y}_{p_k}$ is the prediction of p_k on $\mathbf{x}$'s label
  $H \leftarrow H \cup \tilde{y}_{p_k}$
end for

$WSEL \leftarrow \phi$ are the selection weights
for $k = 1 \rightarrow$ size(P) **do**
  $p_k \in P$ is the k^{th} predictor
  $wsel_{p_k} = evals\_fct(p_k)$ is p_k's selection weight
  $WSEL \leftarrow WSEL \cup wsel_{p_k}$
end for

$\tilde{y} = integ\_fct(WSEL, H)$ is the committee's final prediction
Read the real label y of $\mathbf{x}$

{LEARNING}
$WDEL \leftarrow \phi$ are the deletion weights
for $k = 1 \rightarrow$ size(P) **do**
  $p_k \in P$ is the k^{th} predictor
  $wdel_{p_k} = evald\_fct(p_k)$ is p_k's deletion weight
  $WDEL \leftarrow WDEL \cup wdel_{p_k}$
end for

if size(P) $>= max_P$ **and**
for all $p \in P$, p's age$>=$ maturity_age **then**
  for $k = 1 \rightarrow n\_pool$ **do**
    $p \in P$ is the predictor with the lowest deletion weight
    $P \leftarrow P \setminus p$
  end for
end if

if size(P) $< max_P$ **then**
  b is the pool of predictors; the age of each $p \in b$ is equal to 0
  $P \leftarrow P \cup b$
end if

for $k = 1 \rightarrow$ size(P) **do**
  $p_k \in P$ is the k^{th} predictor
  train p_k on $(\mathbf{x}, y)$
  increment p_k's age
end for

4 Experiments

We conducted several experiments to examine the behavior of the algorithm. In Section 4.1, we check if the algorithm gives more importance to plasticity over stability i.e. if it was designed to react well on changing concepts only or if it can perform well on stable concepts also. In Section 4.2, we observe the effect of "bad predictors" in the pool of committee on the predictive performance. Finally, in Section 4.3, we compare different ways of weighting the experts and integrating their predictions.

4.1 Experiment 1: *the stability-plasticity dilemma*

In this set of experiments, we examine the behavior of the committee when the concept is stable and when a sudden and severe drift occurs i.e. when the old concept is directly replaced by a completely new one. The deletion strategy should allow a fast adaptation to a concept change but we should ensure that deleting experts constantly doesn't affect the committee's predictive performance when the concept is stable. We also show that our committee learns a concept drift better than a single expert learns each concept individually.

Experimental Setup In the experiments, the target concept C is a hyperplane. A hyperplane in a d-dimensional space is represented by the equation:

$$\sum_{i=1}^{d-1} w_i x_i = w_0 \tag{1}$$

We simulate one drift event and thus we create two different concepts C_1 and C_2: The first concept C_1 is learnt from a data stream of 200 training examples $(\mathbf{x}, y_1)$ where $\mathbf{x} \in [0,1]^d$ is a randomly generated vector of dimension d and y is set as follows:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{d-1} w_i x_i - w_0 > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The value of w_0 is set to:

$$w_0 = 1/2 \sum_{i=1}^{d-1} w_i \tag{3}$$

so that nearly half of the y's are positive and the other half is negative and $d = 2$. At time step 201, C_1 is replaced by another concept C_2, represented by another sequence of 200 training examples $(\mathbf{x}, y_2)$ where $\mathbf{x}$ is the same as in C_1 and y_2 is the opposite class of y_1. Thus, we flip the classes from C_1 to C_2. Since all the input space is affected by the drift, this is called a *severe drift* (Minku et al. (2010)) or a *global drift* (Tsymbal et al. (2008)). The second concept replaces completely the first one at time step 201 causing what is called a *sudden drift* (Minku et al. (2010)).

In order to evaluate the predictive performance, we compute the *online error* which is calculated as follows: at time step t, the label of a training instance $\mathbf{x}_t$ is predicted before the instance is learnt. The prediction error is calculated and the average prediction error from time step 1 to t is updated. The average error is known as the online error (Minku et al. (2010)).

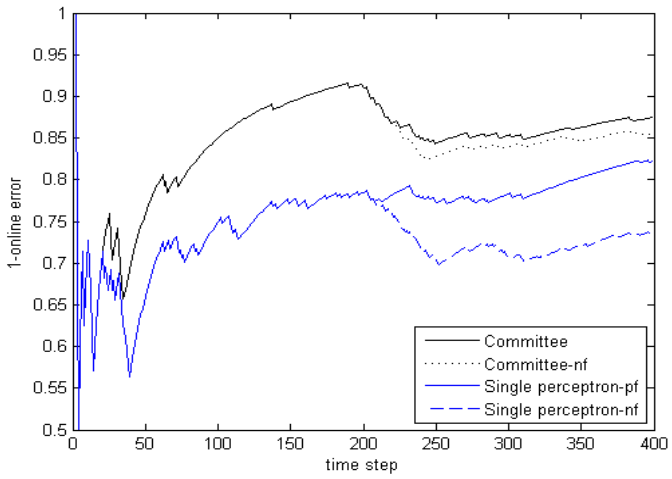

FIG. 1 – *The online predictive performance with the four experiments described in Section 4.1; nf and pf refer to "no forgetting" and "perfect forgetting", repectively*

This procedure is also called the progressive evaluation (Tsymbal et al. (2008)) or the interleaved test-then-train approach (Bifet et al. (2009)). In our experiments, the prediction error is the absolute value between the predicted value and the real value. The online predictive performance or accuracy (acc) and the online error (err) are related by the following equation: $acc = 1 - err$.

We compared the predictive performance using four approaches:

a **Committee**: The evolving committee described in Section 3. The pool of predictors contains a single perceptron and the maximum number of pools is set to 20 ($n_{pool} = 1, max_{pool} = 20$). Both deletion & selection weights are computed as the inverse of the online error on the recent data [4]. The expert with the largest selection weight is chosen for the final prediction and the expert with the lowest deletion weight is removed [5]. The maturity age is set to 20 [6].

b **Committee-nf**: The committee is the same as the previous one. However, we don't remove any classifier: all the committee members increase their history size as new training data is observed.

c **Single Perceptron-pf**: A single perceptron that learns the current concept. It erases its memory -by resetting its weights to 0- when the drift occurs [7]. This simulates a perfect forgetting of the old concept C_1 when it is replaced by the new concept C_2.

d **Single Perceptron-nf**: A single perceptron that learns the current concept without erasing its memory. It keeps learning as new training data becomes available.

4. We compute the error on the last 20 predictions.
5. Since we have one expert in the pool, only one expert is removed at deletion time.
6. The value 20 has been chosen randomly.
7. The perceptrons' initial weights are fixed to 0 in all the experiments for comparison purposes.

Results & Analysis The results are shown in figure 1. We notice that:

- When the concept is stable from time step 1 to 200, our committee is either comparable or better than the other approaches (b), (c) and (d). Thus, the deletion strategy doesn't affect the committee's predictive performance when the concept is stable. In general, when the concept is stable, the committee keeps the predictors with a large history size and expulses the predictor(s) with the smallest history size since they will perform poorly. The expulsed predictors will be replaced by new ones, trained on a small history size which will cause their expulse again from the committee and so on. Meanwhile, the remaing predictors increase their history size which allows learning the stable concept.
- In experiment (b), the online learning algorithm of the perceptron makes the perceptron able to forget C_1 when enough training data of C_2 has been observed. We see however, that the committee adapts faster to the change in experiment (a) than in (b). This suggests that the fast adaptation in experiment (a) is related to the approach forgetting strategy -realized by removing bad experts- and not to the perceptron's forgetting strategy embedded in its learning algorithm. The advantage of the deletion strategy is emphasized when using a lossless learning algorithm which keeps the memory of all previously learnt data. Lossless online algorithms are available for decision trees, Naive Bayes models and others (Oza and Russell (2001), Utgoff et al. (1997)).
- The committees of experts in experiments (a) & (b) outperform the single classifiers in experiments (c) & (d). By comparing the results of experiments (a) & (c), we also notice that our committee learns a concept drift better than the single perceptron learns each concept individually. Erasing the memory of the perceptron (experiment (c)), when the drift occurs, gives better predictive results than keeping the memory of the previously learnt concept (experiment (d)) since it takes time for the perceptron to forget its acquired knowledge.

4.2 Experiment 2: *the pool of predictors*

The pool of predictors allows the use of different training models, algorithms and structures in the committee. While this has the advantage of adding diversity, it might hurt the committee performance when badly chosen experts have been added to the pool. In this set of experiments, we observe the behavior of the committee when "bad" experts exist in the pool of predictors. We show how our approach increases the percentage of the "good" experts in the committee while decreasing the percentage of the "bad" experts. The predictive performance is not affected during this process.

Experimental Setup The same settings described in Section 4.1 were used (training data, maturity age, selection and deletion weights). However, we varied the pool of predictors in the following two experiments:

a The pool of predictors contains a single perceptron and the maximum number of pools is set to 20. Thus, the committee maximum size is equal to 20 ($n_{pool} = 1, max_{pool} = 20, max_P = 20$).

b The pool of predictors contains two perceptrons: The first one will be normally trained on its training data. We will refer to this perceptron as a "normal perceptron". The second

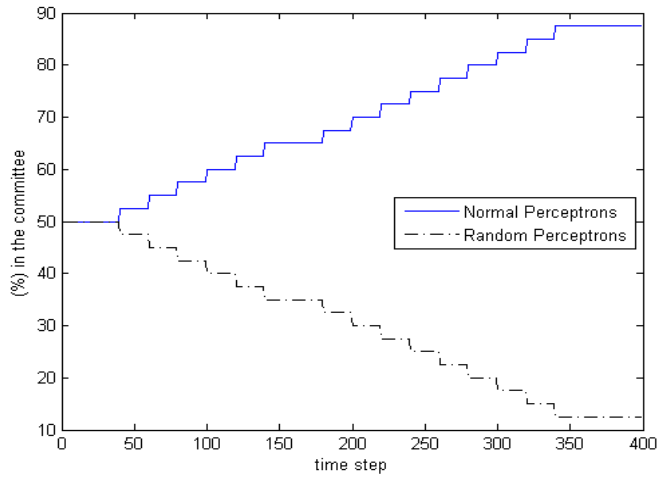

FIG. 2 – *The percentage of normal and random perceptrons in the committee as described in experiment (b) of Section 4.2.*

one represents a randomly generated hyperplane which classifies randomly half of the examples as positive and the other half as negative. This perceptron -referred to as a "random perceptron"- stays as is in the committee until it is removed.
The maximum number of pools is also set to 20. Thus, the committee maximum size is equal to 40 ($n_{pool} = 2, max_{pool} = 20, max_P = 40$).

Results & Analysis We show in figure 2 the percentage of normal and random perceptrons in the committee for experiment (b). We notice the following:

- During time steps 1 to 20, there are 50% of each type of perceptrons in the committee. Since the committee has not reached its maximum size yet, pools are still added during this period.
- At time step 20, the lastly added pool requires 20 time steps to be mature. Thus, from time step 21 to 40, no deletion is performed on the committee members which keeps the percentage of each type of perceptrons fixed.
- The percentage of normal perceptrons increases with time while the percentage of random perceptrons decreases. This is explained by the fact that the number of random perceptrons deleted from the committee is higher than the added ones. For instance, in each of 15 deletion operations, the worst two experts -deleted from the committee- were random perceptrons and were replaced by a pool which contains one normal perceptron and one random perceptron. Thus, after these 15 deletion operations, 30 random perceptrons were deleted and 15 random perceptrons were added.

After deleting members from the committee and adding a new pool, 20 steps are required for the lastly added pool to be mature. Since deleting an expert cannot be performed unless all the

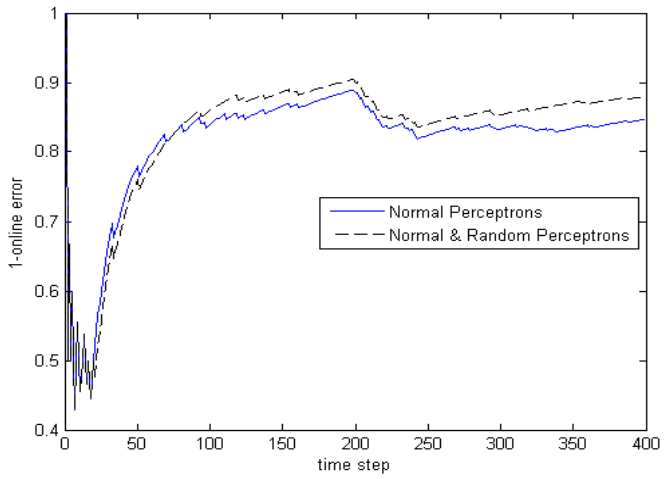

FIG. 3 – *The online predictive performance with the two experiments described in Section 4.2*

experts in the committee are mature, 20 steps are required between two consecutive deletion operations. This causes the step effect in figure 2.

In figure 3, we compare the predictive performance with experiments (a) and (b). We can see that adding random predictors didn't decrease the online error even when the committee contained a large number of random predictors (50%). In fact, the accuracy is improved. This is explained by the fact that the committee in experiment (b) is twice as big as the one in (a). The existence of clearly "bad predictors" in experiment (b) will cause their removal from the committee. However with only "good predictors" as in experiment (a), the relatively least performing predictors are removed even if they are globally good.

The results of the experiments suggest that if some predictors in the pool were badly selected (ex: unadapted model or structure to the current learning problem), there is not need to worry about a drop in the predictive performance [8]. The algorithm will manage to remove the worst predictors and keep the better ones. Even if bad experts remain in the committee, they won't be selected for the committee's final prediction. It is also important to note that if a "bad" model in the pool becomes "good" at a specific time, it will get a second chance to populate the committee since it is always included in each newly added pool.

4.3 Experiment 3: *the combination of experts' prediction*

In this set of experiments, we test two different ways of computing the *selection weights* and different ways of *combining* them for the final prediction.

8. Of course, if all the experts are badly selected in the pool, the approach won't solve the problem.

Selection Weights The selection weights are computed as follows:

- **Recent Window method (RW)**: As in the previous experiments, the weight of the expert is computed as the inverse of its online error on the recent training data. The weight value ranges from 0 to ∞.
- **Neighborhood Window method (NW)**: As suggested in Tsymbal et al. (2008), the weight of the expert is computed by evaluating its prediction performance on the neighborhood of the test data. The weight value ranges from -1 (when the expert misclassifies all the examples in the neighborhood) to +1 (when the expert correctly classifies all the examples in the neighborhood). The selection weight $wsel_{p_k}(\mathbf{x})$ of expert p_k given a test sample $\mathbf{x}$ is computed as follows:

$$wsel_{p_k}(\mathbf{x}) = \sum_{j=1}^{k} \big(\sigma(\mathbf{x}, \mathbf{x_j}).mr_{p_k}(\mathbf{x_j})\big) / \sum_{j=1}^{k} \sigma(\mathbf{x}, \mathbf{x_j}) \tag{4}$$

$$mr_{p_k} = \begin{cases} 1 & \text{if } \tilde{y}_{j,p_k} = y_j \\ -1 & \text{if } \tilde{y}_{j,p_k} \neq y_j \end{cases} \tag{5}$$

$$\sigma(\mathbf{x}, \mathbf{x_j}) = \frac{1}{||\mathbf{x} - \mathbf{x_j}||} \tag{6}$$

In the above equations, k is the size of the neighborhood, $\mathbf{x_j}$ is the j-th nearest neighbour from $\mathbf{x}$, y_j is $\mathbf{x_j}$'s real label and $\tilde{y}_{j,p_k}$ is $\mathbf{x_j}$'s label as predicted by the expert p_k.

It is important to note that we are testing the *selection weights* which are used to evaluate the experts for the final prediction. Not to be confused with the *deletion weights* which are used to remove the worst experts (see Algorithm 1). The deletion weights are computed the same way in all the experiments using the RW method [9].

Combination Rules After computing the weights, we tested different ways of combining the experts prediction:

- **V**: simple vote
- **WV**: weighted vote
- **WVD**: weighted vote after suppressing the predictions of the worst experts i.e. the experts with the performances that fall into the lower half of the performance interval.
- **S**: the prediction of the best expert is selected i.e. the expert with the largest weight.

Experimental Setup We conducted three experiments using the same training data as in Section 4.1 and 4.2. The experiments have the following settings:

a The selection weights are computed using the NW method: the size of the neighborhood k is set to 5 and the nearest neighbours are taken from the last 20 training samples. The pool of predictors contains one perceptron and the maximum number of pools is set to 20 i.e. $n_{pool} = 1, max_{pool} = 20, max_P = 20$. The perceptrons' weights are initialized to 0.

9. Using the same deletion weights allows one to evaluate the committee predictive performance based on the selection weights only.

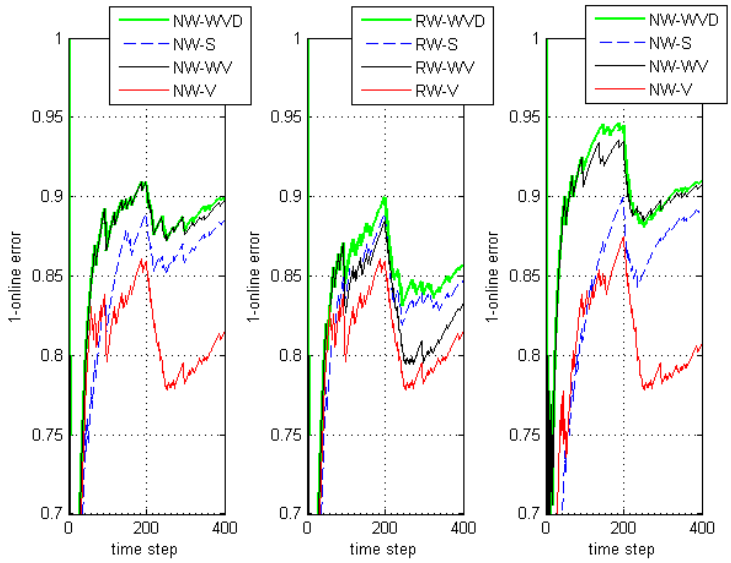

FIG. 4 – *From left to right: the online predictive error with the experiments* (a),(b) *and* (c) *described in Section 4.3, respectively. In each figure, we plot the online predictive performance with several selection & combination methods. The items in the legend are read as: weight computation method-combination method.*

b The selection weights are computed using the RW method: the online error is computed on the last 20 training data. The pool of predictors and the other settings are the same as in experiment (a).

c The selection weights are computed using the NW method with the same settings as in experiment (a). However, we use a different pool of predictors which contains two types of perceptrons: a normal perceptron and a random perceptron. The maximum number of pools is set to 10 i.e. $n_{pool} = 1, max_{pool} = 20, max_P = 20$. The perceptrons' weights are initialized randomly and their bias w_0 is computed as in equation 3.

Results & Analysis The results of the experiments are shown in figure 4. We notice the following:

– When comparing the combination methods in terms of prediction accuracy in the experiments (a) and (b), we notice that the simple voting (V) gives the highest online error. This is expected since it doesn't take into account the goodness/badness of the predictor. WVD always gives the best results while WV shows a different behavior when used with RW and NW weights. With the RW weights, we can see that shortly after the drift, the WV accuracy falls. This is explained by the fact that when a committee contains a relatively large number of bad experts, their weights will sum up and will be high enough to win the vote. This effect is not observed when WV is used with the NW weights because

bad experts have negative weights which will decrease the influence of their vote. Since WV dimishes the influence of the bad experts and WVD suppressed the votes of the bad experts, WVD and WV give very close results when used with the NW weights.
– When comparing NW and RW in experiment (a) and (b), we notice that NW gives a better prediction accuracy than RW when learning a stable concept. It also reacts and recovers faster from the concept drift. However, NW requires more computational time since, for each test instance $\mathbf{x}$, NW computes the predictive performance of each expert in the neighborhood of $\mathbf{x}$.
– In experiment (c), we added random predictors to emphasize the differences between the combination methods. We notice the unstable behavior of the S combination method: the gap increases between S and the best combination methods when random predictors exist in the committee. By comparing the results of experiment (c) with (a), we see an improved accuracy when learning the first concept C_1 in experiment (c). Initializing the perceptrons with different initial weights adds diversity to the committee which increases the overall predictive performance [10].

5 Conclusion

In this paper, we presented an approach that is able to deal with concept changes, using a dynamic and diverse committee of experts where each expert predicts the label of an incoming test sample. The diversity has an impact on the predictive performance of the committee: it improves the classification accuracy when the concept is stable and lowers the test error when the concept change happens (Minku et al. (2010)). In our approach, the committee is *diverse* since it can hold various types of experts in the committee. The experts can be different prediction models (neural networks, SVMs ...) with potentially different structures (number of layers in a neural network, the maximum size of a decision tree ...) and are also trained on different training data. Initializing the experts differently also increases the diversity in the committee. The committee is also *dynamic* since it constantly updates itself by removing the lowest performing experts and adding new experts with a small history size. As for the remaining experts which were not expelled from the committee, their history size is increased. This strategy avoids finding explicitly an appropriate window size in order to solve the stability-plasticity dilemma. Since the approach processes one instance at a time, there is no need for storage or reprocessing (unless NW method is used to weight the classifiers which requires to keep a small window of recent data for local accuracy evaluation).

The experimental results showed that our ensemble algorithm learns a concept change better than when a single expert learns each concept separately. Thus, our algorithm gives importance to plasticity by adapting rapidly to a concept change and it also gives importance to stability by learning a stable concept as a single online expert would do. In the experiments, we show how the algorithm is robust to the existence of potentially "bad" types of experts in the committee: the algorithm manages to remove the "bad" experts and increases the number of "good" experts in the committee without hurting the overall predictive performance during the process. Finally, we tested different methods to weight and combine experts. We noticed

10. In the experiments (a) and (b), the diversity comes from predictors trained on different training samples.

that even in a global drift scenario, evaluating an expert based on its local accuracy reflects better the expert's predictive performance on a test sample, than with its global accuracy.

For future work, we plan to test our algorithm on different types of changes and compare the results with already existing ensemble methods that deal with concept change. We also find it interesting to study the advantages of combining a pro-active approach which predicts how a concept will change in the near future, with a reactive approach which adapts passively to the current situation.

References

Bickel, S., M. Brückner, and T. Scheffer (2009). Discriminative learning under covariate shift. *The Journal of Machine Learning Research 10*, 2137–2155.

Bifet, A., G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pp. 139–148.

Cornuéjols, A. (2010). On-line learning: where are we so far? *Ubiquitous knowledge discovery*, 129–147.

Domingos, P. and 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.

Fan, W., I. Davidson, B. Zadrozny, and P. S. Yu (2005). An improved categorization of classifier's sensitivity on sample selection bias.

Jaber, G., A. Cornuéjols, and P. Tarroux (2011). Predicting concept changes using a committee of experts. In *ICONIP'11 International Conference on Neural Information Processing*, Shanghai, China, pp. 580–588.

Kolter, J. Z. and M. A. Maloof (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research 8*, 2755–2790.

Minku, L. L., A. P. White, and X. Yao (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering 22*(5), 730–742.

Oza, N. C. and S. Russell (2001). Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*.

Ruping, S. (2001). Incremental learning with support vector machines. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, San Jose, CA , USA, pp. 641–642.

Scholz, M. and R. Klinkenberg (2005). An ensemble classifier for drifting concepts. In *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*, pp. 53–64.

Stanley, K. O. (2003). Learning concept drift with a committee of decision trees. *UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA*.

Syed, N. A., H. Liu, and K. 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. 272–276.

Tsymbal, A., M. Pechenizkiy, P. Cunningham, and S. Puuronen (2008). Dynamic integration of classifiers for handling concept drift. *Information Fusion 9*(1), 56–68.

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

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

Yang, Y., X. Wu, and X. Zhu (2006). Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data mining and knowledge discovery 13*(3), 261–289.

Zliobaite, I. (2009). Learning under concept drift: an overview. Technical report, Vilnius University, Faculty of Mathematics and Informatics.

Résumé

Nous présentons une méthode d'ensemble capable de s'adapter aux changements d'un concept dans le cadre de l'apprentissage artificiel en ligne. Notre approche s'appuie sur un comité d'experts où chaque expert est formé sur des données différentes. Les experts évoluent constamment en fonction de leur performance ce qui crée un comité dynamique capable de s'adapter à une grande variété de changements de concept. Les expériences, basées sur des données artificielles, simulent un changement brusque de concept. Nous testons différentes méthodes pour évaluer les experts et combiner leurs prédictions. Les résultats montrent que notre méthode d'ensemble apprend un changement de concept mieux que quand un seul expert apprend chaque concept séparément. Différents types d'experts comme les réseaux de neurones, les arbres de décision et d'autres peuvent coexister au sein du comité afin d'augmenter la diversité et d'améliorer la performance globale. L'algorithme reste toutefois robuste à l'existence potentielle de "mauvais" types d'experts dans le comité.