Lille 3 Université Sciences humaines et sociales Charles-de-Gaulle

Ecole doctorale régionale Sciences Pour l'Ingénieur
Université Lille Nord-de-France - 072

Apprentissage incrémental en-ligne sur flux de données

THÈSE

présentée et soutenue publiquement le 30 novembre 2012

pour l'obtention du grade de

Docteur, spécialité Informatique

par

Christophe Salperwyck

préparée à Orange Labs et dans l'équipe-projet SequeL dans le cadre d'une convention CIFRE

orange™ cnrs lifl Laboratoire d'Informatique Fondamentale de Lille Inria INVENTEURS DU MONDE NUMÉRIQUE

Composition du jury

Encadrant :	Vincent Lemaire - Orange Labs & HDR Paris Sud / Orsay
Directeur :	Philippe Preux - Professeur à l'Université de Lille
Rapporteurs :	Pascale Kuntz-Cosperec - Professeur à l'Université de Nantes Djamel Abdelkader Zighed - Professeur à l'Université de Lyon
Examinateurs :	René Quiniou - Chargé de Recherche à l'IRISA de Rennes Younès Bennani - Professeur à l'Université Paris 13

Table des matières

Table des figures

Liste des tableaux

Notations générales

X	le vecteur des variables explicatives
X_i	la $i^{ème}$ variable du vecteur X
d	la dimension du vecteur X
x_i	une instance de la variable X_i
C	la variable à prédire
K	le nombre de classes du problème
c_k	la classe d'étiquette k
$x = \{x_1, x_2, ..., x_d, c_k\}$	un exemple étiqueté
$f : X \rightarrow C$	un classifieur
$P(C\|X)$	un concept

Remerciements

Ce manuscrit conclut trois ans de travail, je tiens en ces quelques lignes à exprimer ma reconnaissance envers tous ceux qui de près ou de loin y ont contribué.

Je remercie tous les membres du jury pour l'intérêt qu'ils ont porté à mes travaux. Je remercie Pascale Kuntz et Djamel Zighed d'avoir accepté d'être rapporteurs de mes travaux et pour leurs nombreuses remarques constructives. Je remercie Younès Bennani et René Quiniou pour leur participation au jury.

Je souhaite remercier Vincent Lemaire pour m'avoir permis de réaliser cette thèse au sein d'Orange Labs. Je te remercie pour ton aide et tes précieux conseils au cours de ces trois années ainsi que tes nombreux encouragements. Je voudrais te dire toute ma gratitude pour ton enseignement et ton aide à la rédaction de mes articles.

Je remercie Philippe Preux d'avoir accepté d'être directeur de cette thèse. Je te remercie pour le temps que tu m'as consacré, ta réactivité et tes conseils, et cela malgré la distance importante entre Lannion et Lille.

Je tiens spécialement à remercier Marc Boullé pour sa disponibilité et les nombreux échanges qui m'ont grandement aidé pour la mise au point de nouvelles méthodes. Je te remercie également pour la rigueur scientifique que tu m'as permis d'acquérir, et pour tes nombreux conseils.

Je remercie Fabrice Clérot pour m'avoir guidé dans mes recherches bibliographiques et dans le positionnement de mes travaux. Nos nombreuses discussions et tes remarques toujours pertinentes m'ont aidé à bien situer les problèmes liés aux flux de données.

Je remercie Tanguy Urvoy pour m'avoir fait partager sa passion pour la recherche, tout particulièrement durant les trois mois du challenge ICML. Nos discussions m'ont permis d'avancer grandement dans mon sujet de thèse.

Je tiens tout particulièrement à remercier Carine Hue pour son soutien et sa patience à me réexpliquer certaines notions de statistique oubliées. Je la remercie également pour ses nombreuses relectures et conseils pour la rédaction de ce manuscrit.

Je remercie tous les membres de l'équipe pour leur soutien et nos nombreux échanges. Merci aussi à tous ceux (ils se reconnaîtront) qui m'ont permis de me détendre en tapant la balle ou en naviguant durant ces trois années dans le Trégor.

Un dernier remerciement se dirige tout naturellement vers Agnieszka et toute ma famille qui m'ont encouragé pendant toute la durée de cette thèse.


temps
numériques
naïf
bayésien
fenêtre
hors-ligne
détection
concept
discrétisation
variable
approche
apprentissage
résumé
taille
flux
classe
comptes
performances
critère
classifieur
en-ligne
changement
modèle
exemples
détecter
niveau
moyenne
MODL
mémoire
données
exemple
algorithme
classification
tuples
intervalle
arbre
distribution

Chapitre 1

Introduction

L'extraction de connaissances à partir de données (Knowledge Discovery in Databases en anglais) permet de construire des modèles prédictifs à partir de l'observation des données. L'exploitation des données se déroule en deux phases distinctes : (i) la première phase construit le modèle à l'aide d'algorithmes d'apprentissage, (ii) la deuxième phase exploite (ou déploie) ce modèle sur de nouvelles données. Dans cette thèse on s'intéresse plus particulièrement à la phase d'apprentissage. Traditionnellement la plupart des algorithmes d'apprentissage sont prévus pour travailler hors-ligne, c'est-à-dire que le modèle est construit sur les données disponibles au moment de la phase d'apprentissage puis le modèle est déployé sur les nouvelles données.

Le temps d'accès aux données est un critère important pour la rapidité de fonctionnement des algorithmes d'apprentissage. Ceux-ci profitent très largement d'un chargement des données en mémoire vive. Leur exécution sera beaucoup plus lente si le temps d'accès aux données est long comme, par exemple, sur un disque dur. Bien que le prix des mémoires à accès rapide ait fortement diminué, le prix d'acquisition de ces mémoires devient prohibitif pour des volumétries très importantes. Dans ces conditions, il est préférable d'accéder aux données à leur passage ce qui impose d'y accéder une seule fois et dans leur ordre d'arrivée. On parle alors d'un accès sous la forme d'un *flux de données*. Pour conserver les méthodes existantes hors-ligne, il faut alors s'orienter vers des méthodes d'échantillonnage, de résumés des données ou de calcul distribué. Cet accès aux données impose aux algorithmes d'apprentissage de traiter les exemples à la vitesse du flux. Ceci se traduit par des contraintes fortes en termes de temps de calcul. Les algorithmes satisfaisant les besoins liés aux flux de données sont qualifiés d'algorithme *en-ligne*.

Une manière de limiter le temps de calcul et les besoins en mémoire est de construire le modèle au fur et à mesure de l'arrivée des données en utilisant un algorithme d'*apprentissage incrémental*. Celui-ci est capable de mettre à jour son modèle à l'aide des nouvelles données sans avoir besoin de toutes les revoir. De nombreux algorithmes incrémentaux existent mais bien souvent ceux-ci ont des besoins en ressources mémoire et processeur qui croient non linéairement avec la taille des données. Un flux de données pouvant être de taille infinie, ces algorithmes ne peuvent être utilisés tels quels et doivent être adaptés pour prendre en compte les ressources disponibles.

De nombreux acteurs de l'informatique doivent faire face à ces problématiques nécessitant des techniques d'*apprentissage en-ligne sur flux de données*. Les plus connus sont Google et Yahoo avec le traitement des logs pour la publicité en-ligne, Facebook et Twitter qui modélisent les données provenant de leurs centaines de millions d'utilisateurs, de nombreux opérateurs téléphoniques comme Orange pour la gestion de réseaux de télécommunications. La volumétrie des données de ces applications continue de croître rapidement et la gestion des ressources né-

cessaires à les faire fonctionner devient critique. Les quantités de données générées ne sont plus compatibles avec l'utilisation de la plupart des algorithmes hors-ligne. Ceux-ci nécessiteraient, pour traiter ces volumétries, des ressources en mémoire et/ou en temps de calcul très important.

Dans cette thèse, les méthodes proposées prennent en compte : (i) un accès aux données sous la forme de flux limitant la visibilité des données à leur instant de passage, (ii) la gestion des ressources pour obtenir le meilleur compromis précision / mémoire / temps de calcul pour un apprentissage en-ligne. Ces contraintes imposent l'adaptation de l'ensemble des étapes du processus d'apprentissage et de déploiement du modèle. Les nouvelles méthodes que nous proposons sont incrémentales et possèdent une faible complexité en mémoire et en temps de calcul, ce qui leur permet de réaliser un *apprentissage en-ligne sur des flux de données.*

1.1 Cadre de la thèse

Dans ce manuscrit, on s'intéresse aux problèmes de classification supervisée sur des flux de données. Le terme supervisé est utilisé lorsque le modèle que l'on cherche à construire a pour but de prédire l'une des variables de l'espace de représentation des données. Si la variable à prédire est numérique, alors on a à faire à un problème de régression, si la variable est catégorielle alors il s'agit de classification. Les termes de *classe* ou *étiquette* sont aussi utilisés pour dénommer la variable catégorielle à prédire dans le cadre de la classification supervisée.

Le but des méthodes qui seront étudiées par la suite est de créer des modèles prédictifs, à partir de données étiquetés, grâce à un algorithme d'apprentissage. On appelle $X = \{X_1, X_2, ..., X_d\}$ le vecteur de d variables explicatives. La variable cible (classe à prédire) est toujours, dans cette thèse, catégorielle : $C = \{c_1, c_2, ..., c_K\}$. On appelle $x : x_1, x_2, ..., x_d, c_k$ un exemple. Les x_i correspondent aux d variables descriptives et c_k à la classe (étiquette) à prédire. Les variables explicatives peuvent être numériques : $x_i \in \mathbb{R}$, ou catégorielles $x_i \in \{v_1, v_2, ..., v_V\}$. Un classifieur est un modèle f qui détermine pour un exemple la classe qui doit lui être assignée :

$$f : X \to C$$

Les données du flux arrivent de manière continue selon un concept qui associe les données X à une classe de C, concept que le classifieur f cherche à apprendre. Cependant, au cours du temps ce concept peut changer : on parle alors de changement, ou dérive, de concept. La plupart des algorithmes d'apprentissage ne sont pas capables d'adapter le modèle directement si un changement se produit et il est alors nécessaire de reconstruire un nouveau modèle.

Applications chez Orange

Cette thèse est réalisée sous convention CIFRE chez Orange. Orange a de nombreux besoins en classification supervisée sur des données d'importante volumétrie qui peuvent nécessiter un apprentissage en-ligne sur flux. Orange possède le portail le plus visité de France et en optimise les revenus publicitaires à l'aide de millions de lignes de log collectés chaque jour. Ces logs arrivent en continu et sont nécessaires pour construire des modèles de prédiction de clics toutes les semaines ou tous les mois. La base clients d'Orange est constamment jointe avec les logs d'usage liés aux appels des clients. Ces données arrivent en continu et génèrent un flux de données. Ces données sont utilisées pour modéliser les profils des clients pour des besoins marketing comme le changement d'opérateur (churn), l'appétence pour un produit ou la montée en gamme (upselling). Ces deux cas d'utilisation représentent seulement une partie des problèmes d'*apprentissage automatique sur flux* chez Orange. On pourrait aussi citer : la détection de fraudes, l'analyse en-ligne des incidents réseaux, les recommandations aux clients suite à une visite de la boutique...

Pour l'instant la plupart de ces applications utilisent des méthodes hors-ligne sur une partie des données.

1.2 Apports de la thèse

Résumés en-ligne sur flux de données

L'un des problèmes majeur des flux de données est leur volumétrie qui rend impossible l'utilisation de la plupart des méthodes d'apprentissage hors-ligne. Dès que la quantité de données augmente de manière trop importante, ces méthodes ont des complexités spatiales et temporelles non linéaires incompatibles avec les ressources disponibles. Nous proposons de traiter ce problème de la volumétrie à l'aide de nouvelles techniques de résumés univariés pour flux de données. Ceux-ci permettent d'avoir des estimations de densité par variable avec une faible consommation des ressources. La méthode proposée est constituée de deux niveaux. Le premier niveau utilise des techniques incrémentales de résumé en-ligne pour les flux. Ceux-ci prennent en compte les ressources et possèdent des garanties en termes d'erreur. Le second niveau utilise les résumés de faible taille, issus du premier niveau, pour construire le résumé final à l'aide d'une méthode supervisée performante. Cette approche de discrétisation/groupage incrémentale supervisée constitue un pré-traitement. Ce pré-traitement permet, par la suite, la mise au point de classifieurs incrémentaux utilisant les estimations de densité fournies par nos résumés.

Classifieurs en-ligne sur flux de données

La plupart des classifieurs, prévus pour fonctionner hors-ligne, ne peuvent pas s'appliquer directement sur un flux de données. Parmi les solutions aux problèmes de l'apprentissage sur flux de données, les algorithmes d'apprentissage incrémentaux sont l'une des techniques les plus utilisées. Leur complexité spatiale et temporelle est bien souvent non linéaire avec la quantité de données à traiter et ne permet pas un traitement en-ligne des flux. Notre proposition consiste à adapter des méthodes existantes de classification supervisée hors-ligne en utilisant nos résumés dédiés aux flux. Ces résumés nous permettent de proposer de nouvelles versions du classifieur bayésien naïf et des arbres de décision fonctionnant en-ligne sur flux de données.

Détection de changement dans les flux

Les flux de données peuvent ne pas être stationnaires et comporter des changements de concept. Nous proposons une nouvelle méthode de détection de changement basée sur l'observation des variables du flux. Notre méthode utilise deux fenêtres et permet d'identifier si les données de ces deux fenêtres proviennent ou non de la même distribution. Elle est capable de détecter les changements brutaux ou progressifs sur la distribution des données conditionnellement ou non aux classes. Notre méthode a l'intérêt de n'avoir aucun *a priori* sur la distribution des données, ni sur le type de changement. De plus, à part la taille de la fenêtre, elle ne requiert aucun paramètre utilisateur.

1.3 Organisation du mémoire

Chapitre 2

Cette thèse traitant de l'apprentissage incrémental sur les flux de données, ce chapitre présente tout d'abord un état de l'art des méthodes incrémentales de classification supervisée, puis de leur adaptation afin qu'elles puissent traiter des flux. Ensuite, les différents types de changements de concept sont présentés ainsi que les différentes techniques pour les traiter. La dernière section de

ce chapitre répertorie les jeux de données et les méthodes d'évaluation utilisés dans le cadre des flux de données.

Chapitre 3

Dans ce chapitre, nous nous intéresserons aux méthodes de résumés de flux de données traitant le compromis précision / vitesse / mémoire. Notre objectif est d'avoir, pour les données numériques, une discrétisation supervisée avec une estimation de la densité par intervalle. Pour les données catégorielles, l'objectif est de grouper les modalités ayant une estimation de densité proche. Ce chapitre contient un état de l'art sur les méthodes existantes pour ce genre de problématique, puis nous proposons une méthode à deux niveaux pour répondre efficacement à ce problème. Notre méthode est évaluée sur des jeux de données synthétiques afin de montrer son intérêt et sa robustesse.

Chapitre 4

Ce chapitre exploite les résumés incrémentaux supervisés du chapitre précédent pour proposer des adaptations de classifieurs existants aux flux de données. Ces résumés permettent d'obtenir, avec peu de ressources, des estimations de densité par classe. Ces estimations peuvent être utilisées dans différents classifieurs, afin de les rendre incrémentaux et contrôler la mémoire qu'ils consomment. Nous proposons donc, dans ce chapitre, des modifications du classifieur bayésien naïf, de la version moyennée du classifieur bayésien naïf et des arbres de décisions. Ces modifications rendent ces classifieurs incrémentaux, en-lignes et capables de traiter les flux de données.

Chapitre 5

L'apprentissage sur les flux nécessite d'être capable de traiter les flux avec changement de concept. Ce chapitre propose une nouvelle méthode de détection des changements à partir de l'observation des variables du flux. Notre méthode, basée sur deux fenêtres, identifie des changements dans les distributions entre les fenêtres à l'aide d'une méthode supervisée. Des premières expérimentations, sans classifieur, sont menées pour montrer la capacité à détecter différents types de changement de notre nouvelle approche. Ensuite, nous proposons un nouvel algorithme, utilisant un classifieur et notre méthode, pour apprendre sur des flux non stationnaires. Ce nouvel algorithme est expérimenté sur des flux avec changement de concept provenant de générateurs de l'état de l'art.

Chapitre 2

Classification supervisée incrémentale : un état de l'art

Sommaire

Ce chapitre a fait l'objet d'une publication :

Classification incrémentale supervisée : un panel introductif [SL11a]

Plan du chapitre

La première section présente les familles d'algorithmes qui ont été développées pour répondre aux différents problèmes de volumétrie et d'accès aux données. La volumétrie varie de petites bases avec quelques centaines d'exemples à, potentiellement, une infinité d'exemples dans le cadre des flux de données. Pour l'accès aux données, celles-ci peuvent être toujours accessible dans le cas général, mais dans le cas des flux seulement à leur passage.

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

Lorsque les données ne sont plus stockables sur disque il est impératif de travailler directement sur le flux entrant des données. Dans ce cas une adaptation des algorithmes d'apprentissage incrémentaux est requise. Ces adaptations sont décrites dans la section 3 pour les principales techniques de classification.

Les flux de données peuvent contenir des changements. Différentes techniques permettent de traiter cette problématique afin d'adapter le classifieur. La section 4 présente les différentes approches de la littérature.

L'intérêt croissant de la communauté pour l'apprentissage incrémental ces dernières années (on peut en juger par le nombre croissant de workshops ou de conférences organisés sur ce sujet) a produit de nombreux algorithmes mais aussi différents indicateurs de comparaison entre algorithmes. Le but de la dernière section est de présenter les métriques et/ou indicateurs de comparaison les plus communément utilisés.

2.1 Les familles d'algorithmes d'apprentissage

Différentes familles d'algorithme d'apprentissage existent répondant à des problèmes différents. Historiquement l'apprentissage hors-ligne est le premier à apparaître pour traiter des données collectées auparavant. Puis les données arrivant de manière continue, il a alors fallu les traiter au fur et à mesure à l'aide d'algorithmes d'apprentissage en-ligne. L'apprentissage anytime répond aux besoins des systèmes ayant des contraintes fortes sur le temps d'apprentissage et/ou prédiction. Les sections suivantes présentent tout d'abord la problématique de la volumétrie et de l'accès aux données, puis les différentes familles d'apprentissage pour répondre à ces problèmes.

2.1.1 Préambule : volumétrie et accès aux données

Dans cette thèse, on s'intéresse aux modèles qui permettent de réaliser des tâches de classification supervisée. L'algorithme d'apprentissage utilise des exemples étiquettés pour construire le modèle. Mais la disponibilité de ces exemples varient : tous dans une base de données, tous en mémoire, partiellement en mémoire, un par un dans un flux... On trouve dans la littérature différents types d'algorithmes selon la disponibilité et la taille des données. La figure 2.1 présente ces différents types d'accès aux données.

Si les données peuvent être toutes chargées en mémoire alors l'algorithme a, à tout moment, accès à toutes les données. Cette hypothèse correspond au cas le plus simple sur laquelle repose la plupart des algorithmes. Mais parfois la quantité d'exemples peut être très importante, et il est impossible de tous les charger en mémoire. Il faut donc concevoir un algorithme qui puisse générer un modèle sans avoir besoin que tous les exemples soient en mémoire. On peut alors chercher à découper les données en plusieurs ensembles (chunks) de manière à pouvoir les traiter

les uns après les autres et ne pas avoir à tout charger en mémoire et / ou à utiliser des techniques de parallélisation de l'algorithme d'apprentissage. [PK99] a réalisé une étude sur les méthodes utilisées pour traiter cette problématique de volumétrie : parallélisation, partitionnement de l'espace...

Dans le pire des cas, les données sont très volumineuses et arrivent de manière continue, on parle alors de flux de données. Les exemples ne peuvent être vus qu'une seule fois et dans l'ordre dans lequel ils arrivent. L'algorithme doit réaliser l'apprentissage suffisamment rapidement pour suivre le flux de données. De plus comme l'indique [HSD01], l'apprentissage sur les flux nécessite d'être capable de traiter des flux dont la distribution des données peut changer au cours du temps. On parle alors de changement de concept. Celui-ci doit être pris en compte par les algorithmes d'apprentissage pour les flux. Ces changements peuvent être de différents types et seront détaillés dans la section 2.4.

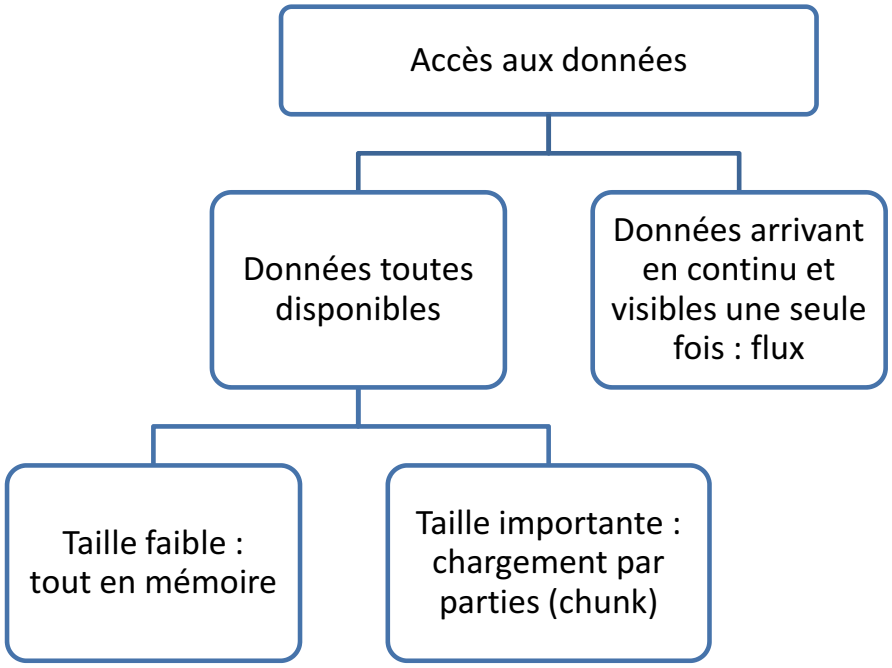


FIGURE 2.1 – Les différents accès aux données pour l'apprentissage.

2.1.2 Apprentissage hors-ligne

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

2.1.3 Apprentissage incrémental

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

2.1.4 Apprentissage en-ligne

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

2.1.5 Apprentissage anytime

La mise en place d'un modèle de prédiction se réalise en deux étapes : (i) une étape d'apprentissage du modèle, (ii) une étape de déploiement du modèle. Dans le cas d'un apprentissage non incrémental ces étapes sont réalisées l'une après l'autre mais dans le cas de l'apprentissage incrémental, on repasse en phase d'apprentissage dès qu'un nouvel exemple arrive. Cette phase peut être plus ou moins longue et il est parfois nécessaire de pouvoir en maîtriser la complexité calculatoire. [DB88] s'intéresse à cette problématique et définit le concept d'algorithme de prédiction anytime :

- capable d'être arrêté à tout moment et de fournir une prédiction
- la qualité de la prédiction est proportionnelle au temps consommé

On peut aller encore plus loin dans la qualification d'un algorithme d'anytime, comme le fait [SAK+09] :

- classification anytime : on utilise au mieux le temps que l'on a entre l'arrivée de deux exemples pour réaliser la classification. Plus on a de temps, meilleure est la classification.
- apprentissage anytime : l'idée est la même ici mais pour l'apprentissage. On apprend le mieux que l'on peut entre l'arrivée de deux exemples. Plus on a de temps, meilleur est l'apprentissage.

La famille des algorithmes par contrat est assez proche de celle des algorithmes anytime. [ZR96] proposent un algorithme s'adaptant aux ressources (temps / processeur / mémoire) qui lui sont passées en paramètres. Ils s'intéressent aussi à la notion d'algorithme interruptible, qui signifie qu'à tout moment la classification peut être arrêtée et que l'algorithme devra retourner un résultat.

2.2 Apprentissage incrémental

2.2.1 Introduction

De nombreux algorithmes d'apprentissage incrémentaux sont adaptés au problème de la classification supervisée. Sans tous les citer il existe : les Séparateurs à Vastes Marges (SVM), les réseaux de neurones, les méthodes des k plus proches voisins, les arbres de décision, la régression logistique, l'analyse discriminante linéaire, etc. L'utilisation de tel ou tel algorithme dépend fortement de la tâche à résoudre et du désir d'interprétabilité du modèle. Il n'est pas possible ici de tous les passer en revue. Les sections ci-dessous se concentrent sur les algorithmes les plus utilisés dans le cadre de l'apprentissage incrémental.

2.2.2 Arbre de décision

Un arbre de décision [BFOS84, Qui93] est un modèle de classification présenté sous la forme graphique d'un arbre. L'extrémité de chaque branche est une feuille qui contient le résultat obtenu en fonction des décisions prises à partir de la racine de l'arbre jusqu'à cette feuille. Les feuilles intermédiaires sont appelées des nœuds. Chaque nœud de l'arbre contient un test sur un attribut qui permet de distribuer les données dans les différents sous-arbres. Lors de la construction de l'arbre, un critère de pureté comme l'entropie (utilisé dans C4.5) ou Gini (utilisé dans CART) est utilisé pour transformer une feuille en nœud. L'objectif est de produire des groupes d'individus les plus homogènes possibles du point de vue de la variable à prédire (pour plus de détails voir par exemple [CM10] chapitre 13). En prédiction, un exemple à classer « descend » l'arbre depuis la racine jusqu'à une unique feuille. Son trajet dans l'arbre est entièrement déterminé par les valeurs de ses attributs. Il est alors affecté à la classe dominante de la feuille avec pour score la proportion d'individus dans la feuille qui appartiennent à cette classe.

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

Des versions incrémentales des arbres de décision sont assez rapidement apparues. [SF86] propose ID4 et [Utg89] propose ID5R qui sont basés sur ID3 mais dont la construction est incrémentale. ID5R garantit la construction d'un arbre similaire à ID3 alors qu'ID4 peut dans certains cas ne pas converger et dans d'autres cas avoir une prédiction médiocre. Plus récemment, [UBC97] propose ITI dont le fonctionnement est basé sur le maintien de statistiques dans les feuilles permettant une restructuration de l'arbre lors de l'arrivée des nouveaux exemples.

L'interprétabilité des arbres, ainsi que leur rapidité de classement, en font un choix très pertinent pour une utilisation sur d'importantes quantités de données. Cependant les arbres ne sont pas très adaptés aux changements de concept (voir section 2.4) car dans ce cas d'importantes parties de l'arbre doivent être élaguées et réapprises.

2.2.3 Séparateurs à Vaste Marge

Les bases des Séparateurs à Vaste Marge (SVM) datent de 1963 et ont été proposées par Vapnik, père de cette théorie. Cependant les premières publications pour la classification, basées sur ce procédé, sont apparues dans [BGV92, CV95]. L'idée de base est de trouver l'hyperplan qui maximise la distance (la marge) entre les exemples de classes différentes.

Des versions incrémentales des SVM ont été proposées. Parmi celles-ci [DG01] propose un découpage des données en partitions et quatre techniques différentes pour réaliser l'apprentissage incrémental :

- ED - Error Driven technique : lors de l'arrivée de nouveaux exemples, ceux qui sont mal classifiés sont conservés pour modifier le SVM.
- FP - Fixed Partition technique : un apprentissage sur chacune des partitions est réalisé et les vecteurs supports résultants sont agrégés [SLS99].
- EM - Exceeding-Margin technique : lors de l'arrivée de nouveaux exemples, ceux qui se situent dans la zone de marge de l'hyperplan sont conservés. Lorsqu'un nombre assez important de ces exemples est collecté le SVM est mis à jour.
- EM+E - Exceeding-margin+errors technique : utilisation de « ED » et « EM », les exemples qui se situent dans la zone de marge et ceux qui sont mal classifiés sont conservés.

[FTARR05b] propose un PSVM - Proximal SVM, qui au lieu de voir une frontière comme étant un plan, la voit comme plusieurs plans (un espace) à proximité du plan de frontière. Les exemples supportant les vecteurs supports, ainsi qu'un ensemble de points situés dans l'espace proche autour de l'hyperplan frontière, sont conservés. En faisant évoluer cet ensemble, on enlève certains exemples trop anciens, et on rajoute les nouveaux exemples, ce qui permet de rendre le SVM incrémental.

Plus récemment l'algorithme LASVM [BB05, BEWB05, LCB06] a été proposé. Il s'appuie sur une sélection active des points à intégrer dans la solution et donne des résultats très satisfaisants pour une utilisation en-ligne. L'apprentissage peut être interrompu à tout moment, avec une étape éventuelle de finalisation (qui correspond à éliminer les vecteurs supports devenus obsolètes). Cette étape de finalisation, faite régulièrement au cours de l'apprentissage en-ligne, permet de rester proche des solutions optimales. Du côté de la complexité calculatoire, il n'y a qu'une résolution analytique à chaque étape. Du côté mémoire, l'algorithme peut être paramétré pour règler le compromis entre le temps de calcul et l'espace mémoire utilisé.

2.2.4 Système à base de règles

En informatique, les systèmes à base de règles [BS84] sont utilisés comme un moyen de stocker et de manipuler des connaissances pour interpréter l'information de manière utile. Un exemple classique d'un système fondé sur des règles est le système expert d'un domaine spécifique qui utilise des règles pour faire des déductions ou des choix. Par exemple, un système expert peut aider un médecin à choisir le bon diagnostic en fonction d'un ensemble de symptômes. Un système à base de règles possède une liste de règles ou base de règles, qui constitue la base de connaissances. Cette base de connaissances est, soit donnée par un expert du domaine, soit élaborée (extraite) à partir d'un ensemble d'apprentissage. Si elle est extraite de manière automatique, il existe un certain nombre de critères qui permettent de juger de la qualité des règles. Nous invitons le lecteur à se reporter à [LTP07] pour une revue et une analyse d'un grand nombre de critères existants.

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

- STAGGER [SG86] est le premier système incrémental à avoir été conçu pour traiter le changement de concept. Il se base sur deux processus : le premier modifie le poids des attributs des règles et le deuxième ajoute de nouveaux attributs dans les règles.
- FLORA, FLORA3 [WK96] dont le principe de fonctionnement repose sur des fenêtres et un système de gestion des contextes pour les enregistrer et les réactiver si nécessaire.
- AQ-PM [MM00] est basé sur un système de conservation des exemples aux frontières des règles, et d'un mécanisme d'oubli lui permettant de faire face aux changements de concepts.

2.2.5 Classifieur Bayésien naïf

Dans cette section on s'intéresse au classifieur Bayesien naïf [LIT92]. La version naïve suppose que, sachant la classe cible, les variables explicatives sont indépendantes. Cette hypothèse réduit drastiquement les calculs nécessaires. Ce prédicteur s'est avéré très compétitif sur de nombreux jeux de données réelles [HY01].

Le principal avantage de ce classifieur est sa vitesse d'apprentissage et ses bonnes performances prédictives. Avec très peu de données, sa précision est bien souvent meilleure que celle d'autres algorithmes comme l'expérimente Domingos dans [DP97]. Cette qualité fait que le classifieur Bayésien naïf est assez souvent utilisé en combinaison avec d'autres algorithmes, comme par exemple les arbres de décision dans [Koh96] : NBTree. De plus, ce classifieur est naturellement incrémentale et peut être mis à jour sans qu'il soit nécessaire de tout recalculer. En effet, il suffit de mettre à jour les comptes permettant de calculer les probabilités conditionnelles univariées. Ces probabilités étant basées sur une estimation des densités, le problème réside dans l'estimation incrémentale des densités conditionnelles. [JL95] expérimentent deux méthodes d'estimation de densité : une gaussienne unique et des noyaux multiples. Il présente leurs complexités spatiales et temporelles, ainsi que l'incrémentalité naturelle de l'estimation par une gaussienne en mettant à jour la moyenne et la variance. [LYW06] propose IFFD, une méthode de discrétisation incrémentale permettant de ne pas réaliser une discrétisation complète à l'arrivée de chaque exemple et donc de ne pas recalculer toutes les probabilités conditionnelles. IFFD ne réalise que deux opérations : ajout d'une valeur à un intervalle ou éclatement d'un intervalle en deux.

2.2.6 Approche passive : plus proches voisins

Les méthodes passives (lazy learning [Aha97]), comme par exemple les k plus proches voisins (k-ppv), sont appelées passives car il n'y a pas réellement de phase d'apprentissage mais simplement une conservation de certains exemples. A proprement parler, on ne construit jamais de modèle « global », on se contente d'un modèle local construit à la demande au moment de l'apparition d'un nouveau motif. Rendre ces méthodes incrémentales est donc assez simple car il suffit de mettre à jour la base d'exemples conservée voire de n'en garder qu'une partie comme le propose [BM02]. Tout le traitement a lieu pendant la phase de classification, lors de la recherche du plus proche voisin. L'approche suivie par ce type de méthodes est de trouver, au sein de l'ensemble des exemples d'apprentissage conservés, ceux qui sont les plus proches de la donnée que l'on souhaite étiqueter. La(es) classe(s) des exemples proches trouvée(s) donne une bonne indication de la classe à prédire pour la donnée présentée.

La littérature s'est consacrée en grande partie d'une part (i) à accélérer la recherche des k-ppv [MSMO02, VLC00] et d'autre part (ii) à l'apprentissage de métrique [KR03, GR05, WS09]. On trouvera aussi dans [SSV07] un travail très intéressant dont une comparaison « incrémental versus non incrémental » d'une méthode à k-ppv.

2.3 Apprentissage incrémental sur flux

2.3.1 Introduction

Depuis les années 2000, le volume de données à traiter a fortement augmenté avec notamment l'essor d'Internet et plus récemment des réseaux sociaux. Ces données arrivent séquentiellement et en continu et ne sont accessibles qu'au moment de leur passage. Celles-ci sont qualifiées de flux de données. Des algorithmes spécifiques pour traiter l'apprentissage supervisé dans ces conditions ont été proposés. Cette partie est une présentation des principales méthodes de la littérature. La

partie sur les arbres de décision est plus détaillée, ceci est dû au nombre plus important d'articles dédiés à cette technique.

On peut se demander en premier lieu quelles sont les propriétés désirées d'un algorithme de classification incrémentale sur les flux. Domingos dans un article de réflexion sur l'apprentissage sur les flux [DH01] propose les critères suivants :

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

Des critères assez similaires avaient déjà été abordés par [FPSSU96] dans le cadre des bases de données de taille importante. Plus récemment, [SCZ05] propose huit propriétés d'un bon algorithme de traitement temps réel des flux. Cependant, il se situe plutôt dans le cadre de la base de données que dans celui de l'apprentissage. Le tableau 2.1 présente une comparaison des différentes propriétés désirées selon les auteurs.

	[FPSSU96]	[HSD01]	[SCZ05]
itératif/incrémental	x	x	
lecture une seule fois des données	x	x	x
gestion de la mémoire/ressources	x	x	x
anytime	x	x	x
gestion de la dérive de concept		x	

TABLE 2.1 – Propriétés d'un bon algorithme d'apprentissage sur flux de données.

Dans un deuxième temps, on s'intéresse aux éléments de comparaison des différents algorithmes. Le tableau 2.2 compare les différents algorithmes par rapport aux critères généralement utilisés [ZR00] pour évaluer un algorithme d'apprentissage. La complexité en déploiement, donnée dans ce tableau, correspond à la complexité du pire cas, d'un problème à deux classes, pour obtenir l'une des probabilités conditionnelles ($P(C_k|X)$) pour un exemple en test. On notera que cette complexité maximale est rarement atteinte pour certains classifieurs : à titre d'exemple pour un arbre elle est de fait (si l'arbre n'est pas complètement déséquilibré) en $O(h)$ où h désigne la hauteur de l'arbre.

Finalement les différences entre apprentissage incrémental et apprentissage incrémental pour flux de données sont présentées dans le tableau 2.3. On appelle phase de « Post-optimisation » une phase qui aurait pour but d'améliorer la solution trouvée après la première passe sur les exemples. Cette passe de post-optimisation peut chercher à améliorer le modèle obtenu sans nécessairement avoir besoin de relire les exemples.

La suite de cette section propose des algorithmes qui non seulement sont incrémentaux mais semblent aussi adaptés aux flux de données : capable d'apprendre sans ralentir le flux (complexité plus basse que ceux de la section précédente), capable d'être réglés pour aller vers un compromis temps/mémoire/précision ; et s'attaquant au problème de la dérive de concept. Ils remplissent tout ou partie des éléments du tableau 2.3.

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

TABLE 2.2 – Comparatif des principaux algorithmes. On note dans ce tableau : n le nombre d'exemples ; d le nombre d'attributs ; a le nombre de règles ; b le nombre moyen de prémices par règle et s le nombre de vecteurs supports.

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

TABLE 2.3 – Propriétés de la classification incrémentale vis-à-vis de la classification incrémentale sur flux (oui=requis, non=non requis).

2.3.2 Arbre de décision

2.3.2.1 Préambule

Limitation des anciens algorithmes : SLIQ [MAR96], SPRINT [SAM96], RAINFOREST [GRG00] sont des algorithmes spécialement conçus pour fonctionner sur des bases de données de taille importante. Cependant il leur est nécessaire de voir plusieurs fois les données et ils ne satisfont donc pas les contraintes des flux. Certains algorithmes n'ont besoin de voir les données qu'une seule fois comme ID5R [Utg89] ou son successeur ITI [UBC97]. Cependant l'effort pour mettre à jour le modèle est parfois plus conséquent que celui pour construire un modèle à partir de toutes les données. Ces algorithmes ne peuvent alors plus être considérés comme « en-ligne » ou « anytime ».

Taille des arbres : De par sa méthode de construction, la taille d'un arbre de décision croît avec l'arrivée de nouvelles données [OJ97] sans pour autant toujours améliorer son taux de bonne

prédiction ([ZR00] - p203). Dans le cas des flux, l'arbre n'arrêtera donc pas de grandir si aucun traitement n'est prévu pour limiter sa croissance. De nouveaux algorithmes ont été développés afin de traiter le cas de l'apprentissage sur les flux.

Dérive de concept : Dans le cas où une dérive de concept (section 2.4) apparaît dans le flux de données, l'arbre doit être élagé ou si possible restructuré [Utg89, UBC97].

Borne de Hoeffding : De très nombreux algorithmes d'apprentissage incrémentaux sur les flux utilisent la borne de Hoeffding [Hoe63, DH00]. Elle est définie de la manière suivante. Soit une variable aléatoire X dont les valeurs sont comprises dans un intervalle de longueur R et $\bar{X_n}$ sa moyenne empirique après n observations indépendantes. La borne de Hoeffding assure, avec une probabilité de $1-\delta$, que la différence entre la moyenne empirique $\bar{X_n}$ et l'espérance de X ne sera pas supérieure à ϵ avec $\epsilon = \sqrt{\frac{R^2}{2n} ln(\frac{1}{\delta})}$. L'intérêt de cette borne est qu'elle ne dépend pas de la distribution des valeurs mais seulement de : (i) de la longueur de l'intervalle R, (ii) du nombre d'observations n, (iii) de la confiance désirée δ. Par contre cette borne est plus conservatrice que des bornes prenant en compte la distribution des valeurs.

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 (index de Gini, information gain...). Le meilleur attribut a est considéré définitivement meilleur que le deuxième meilleur attribut b si $\bar{G_a} - \bar{G_b} > \epsilon$.

2.3.2.2 Les principaux algorithmes rencontrés

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

• **VFDT** [DH00] est considéré comme un article de référence de l'apprentissage sur flux de données sachant gérer plusieurs millions d'exemples. Il est très largement cité et comparé aux nouvelles approches proposées depuis l'année 2000 sur la même problématique. Dans VFDT, la création de l'arbre est incrémentale et aucun exemple n'est conservé. Le taux d'erreur de l'algorithme est plus important en début d'apprentissage qu'un algorithme comme C4.5. Cependant après avoir appris plusieurs centaines de milliers d'exemples, ce taux d'erreur devient plus faible car C4.5 n'est pas capable de travailler avec des millions d'exemples et doit donc n'en utiliser qu'une partie. La figure 2.2, extraite de l'article de VFDT, montre ce comportement et l'intérêt de VFDT par rapport à C4.5. De plus, Domingos et Hulten ont prouvé que leurs « arbres de Hoeffding » sont proches de l'arbre qui aurait été généré par un algorithme hors-ligne. Afin de pouvoir mieux répondre à la problématique des flux, VFDT peut être paramétré. Les deux principaux paramètres sont : (i) la quantité maximale de mémoire à utiliser, (ii) le nombre minimal d'exemples à voir avant de réaliser le calcul du critère. Cette version des arbres de Hoeffding n'est capable de traiter que des attributs catégoriels.

• **CVFDT** [HSD01] est une extension de VFDT pour gérer les changements de concept. Des sous arbres alternatifs sont construits si un changement de concept est détecté. Ces sous-arbres alternatifs remplacent le sous arbre original quand leurs taux d'erreurs deviennent plus faibles. Pour limiter l'utilisation de la mémoire seuls les sous arbres les plus prometteurs sont conservés. Malgré l'utilisation d'une fenêtre temporelle sur les exemples, la complexité reste en $O(1)$ car toutes les données de la fenêtre ne sont pas à nouveau parcourues mais seulement la nouvelle donnée arrivée. D'après les expérimentations de [HSD01] sur le jeu de données des « hyperplans

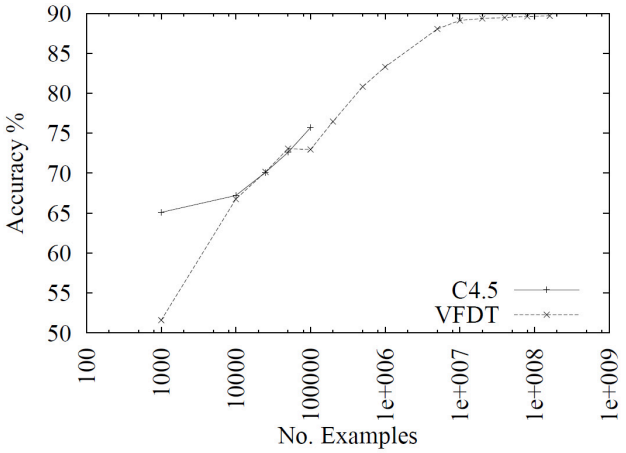


FIGURE 2.2 – Comparaison de C4.5 avec VFDT (extraite de [DH00]).

en mouvement » (voir Tableau 2.5), on constate que le modèle contient quatre fois moins de nœuds mais consomme cinq fois plus de temps comparativement à VFDT.

• **VFDTc** [GRM03] est une extension de VFDT qui gère les attributs numériques continus et non plus seulement catégoriels. Dans chaque feuille on conserve par attribut les comptes des valeurs numériques qui ont atteint cette feuille. Cela permet par la suite de trouver la meilleure valeur de coupure pour cet attribut pour transformer un nœud en feuille. De plus un classifieur Bayesien naïf est ajouté dans les feuilles afin d'améliorer la prédiction. [GRM03] observe qu'il faut de 100 à 1000 exemples avant de transformer une feuille en nœud. Ces exemples dans les feuilles ne sont pas utilisés pour améliorer le modèle tant que la feuille n'est pas transformée en nœud. VFDTc propose d'utiliser ces exemples en ajoutant dans chaque feuille un modèle local. Le classifieur Bayesien naïf est connu pour apprendre bien et vite sur peu de données (voir section 2.2.5). C'est donc le classifieur choisi pour être utilisé dans les feuilles. Cette modification améliore la prédiction de l'arbre sur les jeux de tests WaveForm et LED de l'UCI [BM98].

• Dans **IADEM** [RJdCAMB06] et **IADEMc** [dCARJGMB06] la construction de l'arbre est aussi basée sur la borne de Hoeffding. La croissance de l'arbre est gérée à l'aide du taux d'erreurs de celui-ci. L'arbre se développe jusqu'à arriver au taux d'erreurs maximum passé en paramètre de l'algorithme. IADEMc est une extension qui permet de gérer les attributs continus et qui possède un classifieur Bayesien naïf dans les feuilles (comme pour VFDTc). Les expérimentations réalisées sur les jeux de tests WaveForm et LED montrent les différences suivantes par rapport à VFDTc : une prédiction légèrement moins bonne mais des arbres dont la taille grandit beaucoup moins.

• Enfin, Kirkby dans [Kir08] réalise une étude sur les « arbres de Hoeffding » et propose des améliorations à l'aide **d'ensemble « d'arbres de Hoeffding »**. Sa première modification consiste à avoir dans les feuilles soit un modèle basé sur la classe majoritaire soit un modèle Bayesien naïf. Ses expériences ont montré que le Bayesien naïf n'était pas toujours le meilleur choix. Le classifieur qui a la plus faible erreur est conservé comme classifieur dans les feuilles. Sa deuxième modification aboutit à la proposition des « Hoeffding Option Trees » (inspiré de [KK97]) qui sont une variante des « Hoeffding trees » possédant plusieurs sous-arbres dans chaque nœud. Les mêmes exemples d'apprentissage peuvent mettre à jour plusieurs arbres en même temps. La prédiction se fait par un vote à la majorité, qui détermine la classe prédite. Les techniques de bagging et de boosting sont aussi expérimentées : le bagging permet une amélioration de la prédiction mais pour le boosting le résultat est plus mitigé.

2.3.2.3 Discussion

Du fait de la structure en arbre, le premier nœud a une très grande importance. Si un changement de concept se produit sur ce nœud ou sur des nœuds assez hauts, l'évolution de l'arbre est plus difficile, voire une reconstruction complète peut être nécessaire comme l'indique [WFYH03]. Dans ce cas particulier, l'utilisation des arbres pour la classification sur un flux non stationnaire peut s'avérer coûteuse. La capacité à se restructurer d'un arbre construit de manière incrémentale est donc certainement un élément clef de réussite.

Du coté « anytime », [SAK+09] propose l'utilisation d'un arbre Bayésien contenant dans ses nœuds des modèles. Ainsi s'il n'est pas possible d'arriver à une feuille de l'arbre on met à jour le modèle dans les nœuds.

2.3.3 Séparateurs à Vaste Marge

Les SVMs sont assez peu abordés dans la littérature comme classifieur pour les flux de données. Dans [TKC06], une version des SVMs pour apprendre sur d'importantes quantités d'exemples est présentée. Elle est basée sur une approximation de la solution optimale à l'aide de la résolution de problèmes de type « MEB - Minimum Enclosing Balls ». Sa complexité spatiale est indépendante de la taille de l'ensemble d'apprentissage. Un paramètre permet de régler la préférence pour la rapidité ou pour la précision du classifieur.

[DKS05] présentent une optimisation des méthodes existantes afin d'augmenter la capacité des SVMs. Cette optimisation utilise la technique « diviser pour régner », en séparant le problème en sous-problèmes, et en trouvant et en éliminant rapidement les exemples n'étant pas des vecteurs supports.

On pourrait aussi utiliser LASVM [BB05, BEWB05, LCB06] sur chaque exemple du flux. Soit cet exemple est éligible pour devenir un vecteur support et alors on met à jour la solution courante, soit il n'est pas éligible (loin des frontières courantes indépendamment de son étiquette), on ne fait rien. Au pire des cas, lorsqu'un exemple est éligible, la complexité est au mieux en $O(s^2)$ avec s le nombre courant de vecteur supports. Dans les deux cas (éligible ou non), le calcul des éléments du noyau pour ce point et le calcul des vecteur supports doivent être réalisés. On s'aperçoit donc que la complexité n'est pas négligeable, ce qui explique peut être que d'un point de vue applicatif on trouve peu de «LASVM» appliqués aux flux de données (comparé à la complexité des arbres incrémentaux par exemple). Un article récent présente les garanties des algorithmes en-ligne du type LASVM [UPB10].

L'une des voies d'amélioration est l'utilisation de SVM linéaire et la parallélisation des calculs comme proposé par Poulet et al. [DNP09]. L'utilisation de ces SVMs sur GPUs permet d'apprendre sur des flux rapides et d'obtenir des améliorations, en termes de rapidité, supérieures à un facteur 100.

2.3.4 Système à base de règles

On trouve peu d'articles concernant l'approche à base de règles appliquée aux flux. Des algorithmes incrémentaux, basés sur les systèmes à bases de règles, existent mais ils ne sont pas prévus pour des flux de données. On trouve néanmoins une série d'articles de Ferrer et al. sur ce sujet [FTARR05a, FTARR06]. Ces derniers proposent l'algorithme FACIL, basé sur AQ-PM [MM00], qui est la première approche à base de règles incrémentale sur flux de données.

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

- Lors de l'arrivée d'un nouvel exemple, on recherche pour toutes les règles ayant son étiquette celles qui le couvrent, et on incrémente leur support positif.

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

2.3.5 Classifieur Bayésien naïf

L'apprentissage d'un classifieur Bayésien naïf consiste dans un premier temps à réaliser une estimation des probabilités conditionnelles aux classes (voir section 2.2.5). Cette estimation est très souvent réalisée après une étape de discrétisation des variables explicatives numériques. Dans le cadre des flux de données, seule cette première étape est modifiée pour satisfaire la contrainte du flux. Dans l'état de l'art on trouve deux types de publications concernant la discrétisation incrémentale. Celles qui concernent l'apprentissage et celles relatives aux systèmes de gestion de bases de données qui conservent des statistiques sous forme d'histogramme. Le chapitre suivant, dans sa section 3.1, contient un état de l'art détaillé de ces différentes méthodes de discrétisation incrémentale sur flux.

2.3.6 Approche passive : plus proches voisins -

Les algorithmes des plus proches voisins sont facilement incrémentaux (voir section 2.2.6). Dans le cadre de la problématique d'apprentissage sur flux, les solutions consistent à trouver la meilleure base d'exemples à conserver pour une faible mémoire en (i) oubliant des exemples les plus anciens, (ii) agrégeant les exemples proches. Dans la littérature on trouve différentes méthodes basées sur les plus proches voisins pour les flux. [LZ05] utilisent une technique de discrétisation de l'espace d'entrée afin de limiter le stockage des exemples. [BH07] conservent les exemples les plus récents et gèrent une fenêtre d'oubli.

2.4 Changement de concept

2.4.1 Préambule

Supposons un problème de classification supervisée où l'algorithme d'apprentissage observe une séquence d'exemples munis de leurs classes. La classe d'appartenance des exemples suit une loi de probabilité notée $P(C)$. On appelle concept cible la probabilité jointe $P(X, C) = P(C|X)P(X)$. Si le processus sous-jacent, qui « génère » les données, n'est pas stationnaire ce concept cible à apprendre peut varier au cours du temps. L'algorithme d'apprentissage doit donc être capable d'adapter le modèle de classification supervisée en conséquence. Le caractère non stationnaire du problème à résoudre peut être présent sous deux principales formes de dérives :

1. **Concept drift/shift**. Le concept cible n'est pas toujours constant dans le temps, parfois il se modifie, on parle alors de dérive de concept [MMHL86]. Le passage d'un concept à un autre peut se réaliser de diverses manières (figure 2.3). Le changement peut être *soudain* (sous-figure 2.3a) car à un moment donné le concept change complétement. Il peut être *graduel* (sous-figure 2.3b) si pendant une certaine période de temps l'ancien et le nouveau concepts coexistent puis le nouveau concept reste le seul présent. Dans ce cas le début et

la fin du changement sont très facilement assimilable à du bruit. Si le passage de l'ancien concept au nouveau se réalise d'une manière progressive, on parle alors de changement *incrémental* (sous-figure 2.3c). Un dernier type correspond à la réapparition d'un concept déjà rencontré, on parle alors de *réoccurence de concept* (sous-figure 2.3d). [Ž10] détaillent plus précisément ces différents types de changement ainsi que les techniques pour réaliser un apprentissage dans ces différentes conditions.

2. **Covariate shift**. La distribution des données $P(X)$ peut varier au cours du temps sans modification des $P(C|X)$, on parle alors de covariate shift [QCSSL09]. Le « covariate shift » apparaît aussi dans le cas où l'on effectue une sélection d'exemples non i.i.d., comme par exemple une sélection de données d'apprentissage artificiellement équilibrées (mais non équilibrées dans le jeu de test). Il existe un débat sur cette notion de covariate shift, on peut en effet supposer que la distribution sous-jacente des exemples ne change pas, mais que ce sont que les exemples que l'on observe effectivement qui changent. Le sujet du covariate shift est largement approfondi dans les deux livres suivants : [QCSSL09] et [CM10].

La question de la détection de nouveautés abordée dans [Mar03], ne sera pas traitée dans ce manuscrit. La suite de cette section est organisée en trois parties. La première partie présente différentes techniques de gestion des instances afin de suivre les changements. La deuxième partie décrit quelques méthodes de détection de la dérive de concept. La dernière partie présente les méthodes à base d'ensemble de classifieurs adaptées aux changements de concept. Ces méthodes peuvent être à la fois utilisées ensemble ou de manière indépendante.

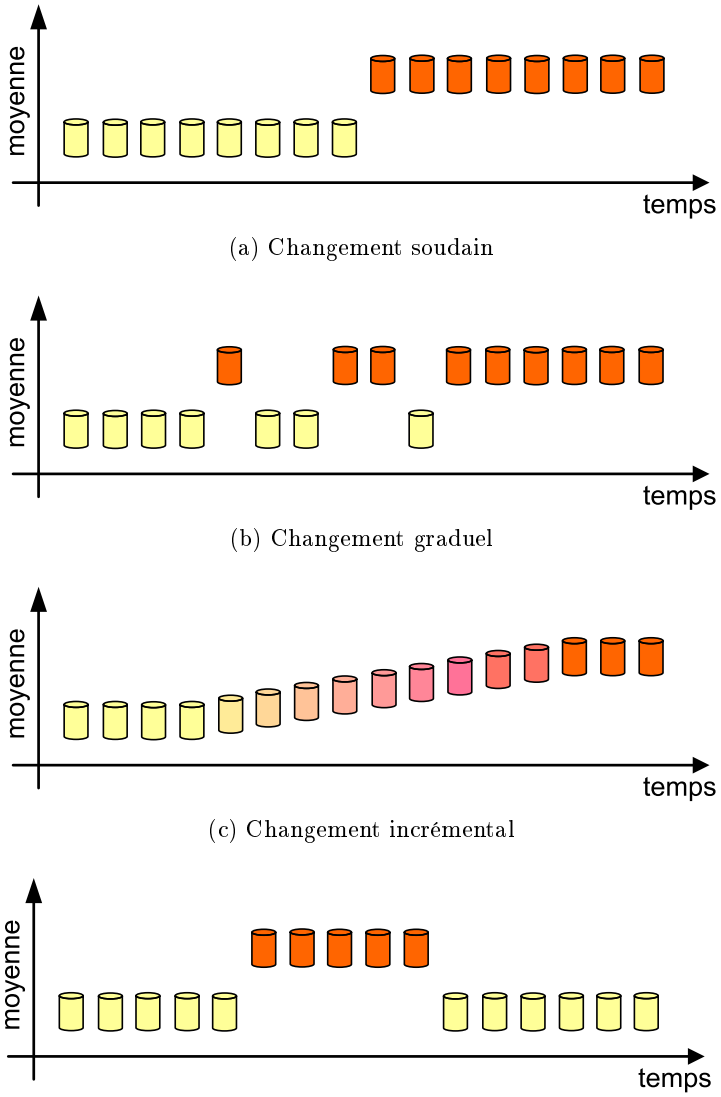
2.4.2 Gestion des instances

Une manière de traiter les changements de concept est de ne garder qu'une partie des données à apprendre, idéalement seulement la partie correspondante au concept actuel. Pour cela différentes techniques existent :

Fenêtres Les fenêtres temporelles sont l'une des manières les plus simples de gérer les changements de concept. Elles consistent à ne garder qu'une partie des données et à réaliser l'apprentissage seulement sur celles-ci. Cependant, il faut connaître la « taille » du concept afin de pouvoir fixer celle de la fenêtre. Une fenêtre trop grande induira une réaction lente au changement de concept, et une fenêtre trop petite limitera la qualité du modèle. Si la taille du concept est connue *a priori,* alors il suffit de l'utiliser pour paramétrer la taille de la fenêtre. Si ce n'est pas le cas ou si cette taille varie dans le temps, alors différentes techniques sont possibles. [LVB04] propose d'utiliser des fenêtres de différentes tailles, et de n'utiliser que la fenêtre qui donne les meilleurs performances. L'algorithme FLORA de [WK92, WK96] utilise lui aussi une fenêtre, mais sa taille est dynamique : si un changement est détecté alors la taille de la fenêtre est diminuée afin de réagir rapidement, sinon la fenêtre grandit afin d'améliorer les performances du classifieur. Cette approche se retouve aussi dans les articles [SK07, KM03, BG07].

Pondération Une technique similaire aux fenêtres est la pondération des instances (une fenêtre correspond à une pondération binaire : 0 ou 1). Les exemples qui servent à apprendre le modèle sont pondérés au cours du temps. Différentes fonctions de pondération peuvent être utilisées : oubli linéaire, exponentiel, logarithmique... Bien souvent la pondération se fait en fonction du temps. [Sal97] propose différentes techniques de pondération :

– TWF (Time-Weighted Forgetting) : plus l'exemple est ancien plus son poids est faible ;



(a) Changement soudain

(b) Changement graduel

(c) Changement incrémental

(d) Réoccurance de concept

FIGURE 2.3 – Les différents types de changement de concept.

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

2.4.3 Détection de la dérive de concept

On considère dans cette section uniquement la partie détection de la dérive sans s'occuper de la gestion qui en sera faite. La détection de la dérive de concept peut être réalisée principalement à l'aide de trois voies de surveillance :

1. performances du modèle
2. distribution des exemples
3. évolution du modèle

Ces différentes méthodes de détection peuvent être utilisées soit seules, soit combinées les unes avec les autres afin de collecter plus d'informations.

Performance du modèle [WK96] proposent de détecter le changement de concept en analysant le taux d'erreurs de classification. A partir d'une certaine variation, il diminue la taille de la fenêtre d'apprentissage sinon il la laisse croître afin d'avoir plus d'exemples pour réaliser un meilleur apprentissage. Ce fut l'un des premiers travaux sur ce sujet et sur lequel s'appuient beaucoup d'autres auteurs. Gama dans [GRSR09] propose d'utiliser le test statistique de Page-Hinkley [Pag54]. Il le modifie en y ajoutant un facteur d'oubli. Ce test est basé sur le calcul des écarts par rapport à la moyenne d'une variable. Son intérêt réside dans le fait qu'il est à la fois performant, simple à calculer et incrémental.

[KR98] a réalisé une étude sur les performances en terme de précision, taux d'erreur et rappel de nombreux classifieurs (Naïve Bayes, SVM, kNN, C4.5...) dans le cadre des changements de concept.

Distribution des exemples La distribution des exemples peut s'observer à partir de différentes méthodes statistiques : moyenne, variance, quantiles, etc. On peut utiliser des méthodes paramétriques ou des tests statistiques, mais ceux-ci présupposent souvent de ce que l'on veut détecter. Ces méthodes sont *a priori* imbattables quand le type de rupture correspond exactement à leur hypothèse. Cependant, si on cherche n'importe quel type de rupture, ces méthodes ne sont pas adaptées et ne vont pas être capables de détecter une partie des changements.

L'arbre de VFDTc proposé dans [GMR05] utilise une méthode spécifique pour détecter des changements dans la distribution. L'arbre est construit incrémentalement en-ligne, par conséquent les nœuds les plus hauts de l'arbre ont été créés avec les données les plus anciennes, et les feuilles avec les données les plus récentes. En gardant des statistiques dans tous les nœuds et les feuilles, puis en les comparant, il est possible d'observer les changements de distribution. Si un changement est détecté alors une partie de l'arbre est élaguée et, avec l'arrivée des nouveaux exemples, de nouvelles feuilles apparaissent correspondant au nouveau concept.

Evolution du modèle Dans [WK96] Widmer et Kubat, en plus de l'observation de la performance du modèle, s'intéressent aux modifications qui se produisent dans les structures de l'algorithme d'apprentissage. L'ajout de nouvelles définitions dans les règles de son modèle déclenche une diminution de la fenêtre d'exemples. [DR09] utilisent la distance à l'hyperplan d'un SVM afin de détecter les changements. De plus, leur méthode est aussi basée sur les deux autres voies de surveillance vu précédemment.

2.4.4 Méthodes adaptatives

Les méthodes adaptatives travaillent directement sur le modèle pour, qu'en cas de changement, celui-ci puisse rapidement s'adapter au nouveau concept. Cette adaptation peut se faire à

l'aide de détecteurs (voir section 2.4.3) qui vont déclencher une action sur le modèle. Certaines méthodes sont dites « aveugles » car elles n'utilisent pas de détecteurs, mais ne font que mettre régulièrement à jour le modèle.

En cas de changement de concept, les modèles globaux comme le Bayésien naïf ou les SVMs doivent être entiérement reconstruits. D'autres modèles, comme les arbres ou règles de décision, peuvent s'adapter sans qu'ils faillent nécessairement reconstruire tout le modèle (élagage des arbres, suppressions de règles, etc.). La plupart des méthodes adaptatives cherchent à être plus générales. Pour cela, elles se basent sur un ensemble de classifieurs dont l'adaptivité vient de la possibilité d'ajouter ou de retirer des classifieurs de l'ensemble au cours du temps.

Les deux paragraphes suivants présentent des exemples d'adaptation de modèles simples et d'ensembles de modèles.

Adaptation d'un modèle simple CVFDT est un exemple d'arbre de décision construit en-ligne et capable de s'adapter aux changements de concept. Son principe est expliqué dans la section 2.3.2.2. L'idée est de construire plusieurs sous arbres dans un nœud au cours du temps, et de ne garder que les meilleurs ou les plus prometteurs. CVFDT peut remplacer un sous arbre qui n'est plus valide pour un sous arbre, plus récent, correspondant au nouveau concept.

Adaptation avec un ensemble de modèles L'idée de départ provient du « bagging » [Bre96] qui consiste à utiliser plusieurs classifieurs en même temps afin d'améliorer les capacités de prédiction de ceux-ci. Différentes approches existent pour maintenir cet ensemble.

- [SK01] proposent l'algorithme SEA : « Streaming Ensemble Algorithm » qui utilise un ensemble de classifieurs sur les flux. Le flux remplit un tampon de taille définie et dès que le tampon est plein l'algorithme C4.5 est lancé dessus. On obtient une suite de classifieurs que l'on met dans un pool. Une fois le pool plein, si le nouveau classifieur améliore la prédiction du pool alors il est conservé, sinon il est rejeté. Dans [WFYH03] la même technique est utilisée mais différents types de classifieurs composent l'ensemble : bayésien naïf, C4.5, RIPPER...
- Dans [KM03], un poids est affecté à chaque classifieur, les poids sont mis à jour uniquement lors de la mauvaise prédiction du nouvel exemple. Dans ce cas, un nouveau classifieur est ajouté et les poids des autres classifieurs sont diminués. Les classifieurs ayant un poids trop faible sont retirés de l'ensemble.
- [BG07, BHP+09] proposent aussi une méthode pour mettre à jour l'ensemble : ADWIN (ADaptative WINdowing). Elle est basée sur la détection des changements de concept à l'aide d'un réservoir de taille variable W. Le réservoir s'agrandit quand il n'y a pas de changements de concept, et diminue lorsqu'il en rencontre un. Cette approche ne consomme comme mémoire que $log(W)$. Quand un changement est détecté, le plus mauvais classifieur est retiré et un nouveau classifieur est ajouté à l'ensemble.
- [BMS10, GK09] conservent un pool de classifieurs potentiels, mais les classifieurs sont utilisés un à un. Le choix du classifieur à utiliser se fait à l'aide d'un détecteur de changement. Si un changement est détecté un nouveau classifieur est appris, et les anciens classifieurs sont évalués sur les nouvelles données. Ensuite, selon les performances observées, soit le nouveau soit l'un des anciens classifieurs est utilisé comme classifieur actuel.

2.4.5 Conclusion

La non stationnarité des données peut se traiter de différentes manières. Le plus simple est bien souvent d'utiliser une sélection des instances par des fenêtres temporelles et de continuer

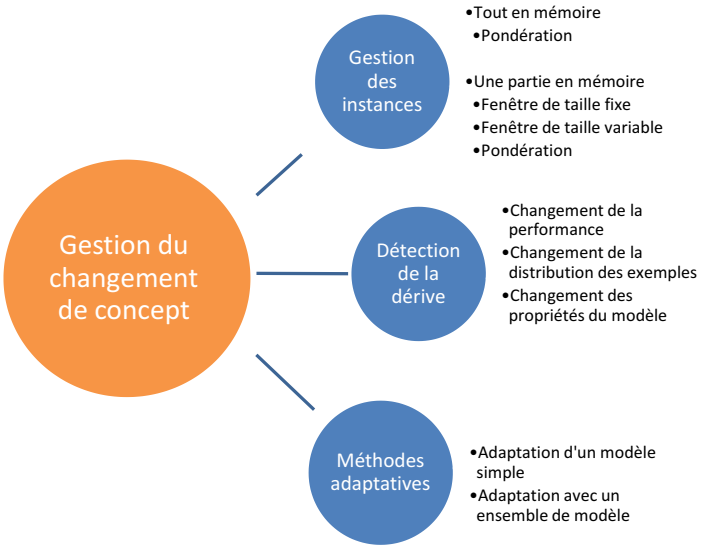


FIGURE 2.4 – Les différentes méthodes de gestion du changement de concept.

à utiliser les méthodes « classiques ». On peut aussi chercher à détecter quand le changement de concept a lieu et ensuite décider des changements à effectuer. Une dernière possibilité est d'utiliser des méthodes adaptatives qui grâce à la modification du modèle ou l'utilisation d'ensemble permettent assez facilement de s'adapter aux changements. Toutes ces méthodes sont complémentaires et ont la possibilité d'être utilisées ensemble.

2.5 Évaluation des algorithmes

Récemment de nombreux algorithmes ont été publiés pour l'apprentissage sur les flux de données. Cependant il est difficile de réaliser une comparaison de ces algorithmes entre eux. En effet les différents auteurs n'utilisent pas toujours les mêmes méthodes, ni les mêmes jeux de données. Gama dans [GRSR09] s'intéresse à cette problématique et propose une méthode d'expérimentation pour l'évaluation et la comparaison des algorithmes d'apprentissage sur les flux.

Dans cette section, nous allons tout d'abord nous intéresser à la mesure du taux de prédiction puis, par la suite, aux jeux de données utilisés dans la littérature. Finalement, nous discuterons plus généralement des méthodologies utilisées pour les expérimentations.

2.5.1 Mesures de précision et ensemble de test

Pour l'apprentissage « hors-ligne », la méthode la plus utilisée est la validation croisée en « k folds » (avec k=10). Les données sont découpées en 10 ensembles de taille identique. Le premier ensemble sert comme jeu de test, et les exemples des autres ensembles pour l'apprentissage. Ensuite, le deuxième ensemble sert comme jeu de test et les autres pour l'apprentissage. Ce processus est réalisé, ainsi de suite, dix fois pour utiliser tous les jeux de tests différents. Les données d'un flux, de par leur quantité et leur disponibilité, ne s'accommodent pas facilement

de ce genre de méthodes. Selon que le flux soit stationnaire ou non-stationnaire (présence de changement de concept), plusieurs méthodes d'évaluation sont envisageables. Elles sont basées sur des jeux de données pour le test construits différemment :

- un jeu de données indépendant pour réaliser le test et que l'on garde tout au long de l'expérimentation
- un jeu de données remis à jour régulièrement en tirant au hasard des exemples du flux qui n'ont pas encore été utilisés pour l'apprentissage
- pas de jeu de données stocké mais les exemples d'apprentissage sont utilisés pour l'évaluation avant qu'ils ne soient appris (aussi appelé « tester puis apprendre » [Kir08] ou « préquentiel » [GRSR09]). L'indicateur de précision est alors calculé de manière incrémentale. Cette méthode est plus pessimiste que celle basée sur un jeu de données indépendant. L'utilisation de fenêtre ou de facteur d'oubli permet de rendre cet évaluateur moins pessimiste comme le propose [GRSR09]. De cette manière les mauvaises prédictions du passé n'influencent plus (ou moins) les nouvelles mesures de la prédiction de l'algorithme. La figure 2.5 compare les erreurs de prédiction pour l'algorithme VFDT sur le jeu de test WaveForm entre l'approche « jeu de données indépendant : holdout », « préquentiel », fenêtre et facteur d'oubli.

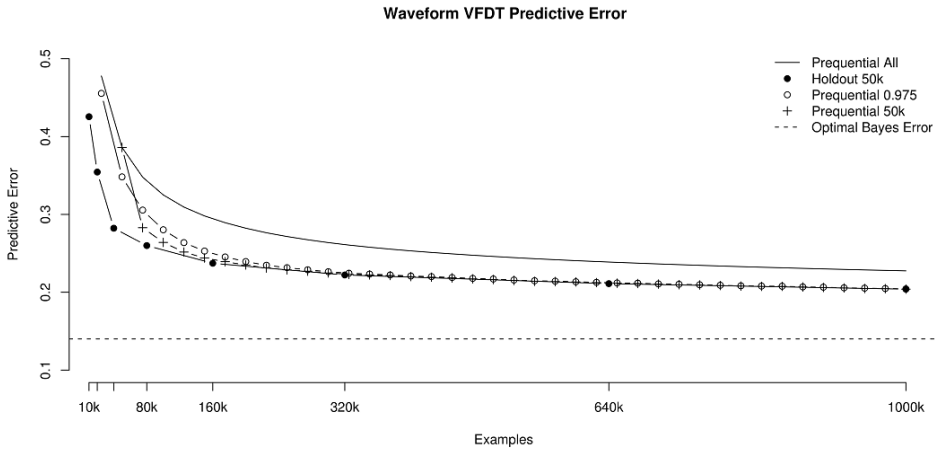


FIGURE 2.5 – Comparaison entre l'erreur avec jeu de test indépendant et préquentiel avec des facteurs d'oubli.

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

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

2.5.2 Jeux de données

Bien souvent dans la littérature, une première évaluation des algorithmes est réalisée sur les bases UCI [BM98]. Elles permettent d'avoir une première idée des performances de l'algorithme bien qu'un algorithme sur les flux ne soit pas forcément prévu pour bien fonctionner sur des petits jeux de données. Ensuite pour simuler les flux, plusieurs approches sont possibles selon que l'on souhaite avoir un flux avec ou sans dérive de concept. Le tableau 2.4 présente une synthèse des jeux utilisés dans les différentes publications de l'état de l'art.

2.5.2.1 Générateur de flux sans présence de dérive de concept

Afin d'avoir assez de données pour tester les algorithmes d'apprentissage sur les flux, des générateurs artificiels existent pour pouvoir générer le nombre d'exemples désirés. Un état de l'art de ces générateurs est présenté dans [Kir08]. Les principaux générateurs sont :

- Random RBF Generator (Radial Basis Function) : basé sur des sphères de différentes tailles à différentes positions dans l'espace. On affecte une classe à chaque sphère. Les exemples sont générées à partir de ces sphères.
- Random Tree Generator : génération d'un arbre avec certains paramètres et ensuite les données sont générées à partir de l'arbre. Ce générateur favorise les algorithmes d'apprentissage basés sur les arbres.
- LED Generator : prédiction pour un afficheur LED.
- Waveform Generator : problème à trois classes généré à partir de fonctions en forme d'onde différentes et combinées.
- Function Generator : les exemples sont générés à partir de règles sur les attributs qui déterminent la classe.

Le challenge « Large Scale Learning [1] [2] », proposé par le réseau d'excellence PASCAL, fournit un ensemble de bases d'apprentissage de grandes tailles. Toutes ces bases contiennent 500 000 exemples étiquetés, ce qui peut être suffisant pour évaluer un algorithme en-ligne.

2.5.2.2 Générateur de flux avec présence de dérive de concept

Afin de pouvoir tester les performances de leurs algorithmes sur des flux de données avec des changements de concept, des auteurs (voir Tableau 2.5) ont proposé des générateurs de flux pour lesquels on peut paramétrer la position et la vitesse du changement (voir aussi [Gam10] page 209). Il faut aussi noter que les générateurs, présentés section précédente, peuvent être facilement modifiés pour contenir des dérives de concept.

2.5.2.3 Jeux de données réelles

Les jeux de données réels « Forest Covertype » et « Poker Hand » de l'UCI, contiennent plusieurs centaines de milliers d'exemples. Dans ces deux jeux, l'existence et la position d'un éventuel changement de concept ne sont pas connues. De part leur taille assez importante, ils sont utilisés pour tester les algorithmes dédiés aux flus de données. Les auteurs ci-dessous proposent des données qui leurs sont propres :

- proxy web de l'Université de Washington pour Domingos dans [DH00, HSD01]
- consommation électrique en Australie pour Gama dans [GMR05]

1. `http://largescale.ml.tu-berlin.de/about/`
2. `http://jmlr.csail.mit.edu/papers/topic/large_scale_learning.html`

Nom de l'algorithme	Réf	Auteur	Année	Source des données	Type source	Bruit	Apprentissage	Test
VFDT	[DH00]	Domingos	2000	14 jeux synthétiques privée : proxy Web	artificielle réelle	0 à 30% ?	100M 4M	50K 267k
CVFDT	[HSD01]	Domingos	2001	Hyperplan en rotation privée : proxy Web	artificielle réelle	5% ?	1M 4M	50k 267k
SEA	[SK01]	Street	2001	Adulte SEER Navigation web SEA Concept	UCI réelle réelle UCI réelle artificielle	?	45K 38K 33K 50K	5 CV 5 CV 5 CV 10k
VFDTc	[GRM03]	Gama	2003	Waveform : (21 et 41 attributs - 3 classes) LED (24 attributs - 10 classes) Forest CoverType	UCI artificielle UCI KDD	14% 26% 30%	1M 566K	250K 15K
UFFT	[GMR05]	Gama	2005	Balance : (4 attributs - 3 classes) Waveform : (21 et 41 attributs - 3 classes) LED (24 attributs - 10 classes)	UCI artificielle	? 14% 26%	1M	100k
FACIL	[FTARR05a]	Ferrer	2005	Balance S, Breast C, Glass, Heart S, Ionospshere, Iris, Prima D, Sonar, Vehicle, Vowel, Waveform, Wine Hyperplan en mouvement	UCI artificielle	- 5%	x 45K	x 5K
Ensemble de Hoeffding Tree	[Kir08]	Kirkby	2008	RTS, RTSN, RTC, RTCN, RRBFS, RRBFC, LED, Waveform (21 et 41 attributs), GenFx (x allant de 1 à 10)	artificielle	0 à 26%	1M à plusieurs G	1M

TABLE 2.4 – Comparatif des algorithmes d'apprentissage sur les flux

Nom	Proposé par	Utilisé dans	Type	Taille
STAGGER	[SG86]	[BH07, BK09, KM03, BHP+09]	Artificiel	infini
SEA Concept	[SK01]	[GMR05]	Artificiel	infini
Hyperplan en mouvement	[HSD01]	[BHP+09, BH07, WFYH03, FTARR05a, NK07]	Artificiel	infini
Forest Covertype	UCI	[GRM03, LZ05, BHP+09]	Réel	581K
Poker Hand	UCI	[BK09]	Réel	1M

TABLE 2.5 – Flux de données pour les algorithmes pour les flux.

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

2.5.3 Comparaison des algorithmes

On trouve très majoritairement la même méthodologie d'évaluation dans la littérature. Dans un premier temps, les auteurs d'un nouvel algorithme effectuent une comparaison avec des algorithmes connus mais non prévus pour fonctionner sur des flux de données comme : C4.5, ID3, Bayésien naïf, forêt d'arbres... L'idée est de voir les performances sur de petites quantités de données contre des algorithmes connus. Puis dans un second temps, ils se confrontent aux autres algorithmes déjà présents dans la littérature de l'apprentissage sur les flux. Le lecteur pourra trouver dans [Kir08] ainsi que dans [GRSR09] un tableau comparatif des méthodes d'évaluation entre certains algorithmes cités dans cette partie.

Environnement d'évaluation : L'un des problèmes lorsque l'on veut réaliser une comparaison entre divers algorithmes de classification est de réaliser facilement des expérimentations. En effet, bien souvent il est difficile d'obtenir le code source de l'algorithme. Parfois les formats d'entrée et de sortie sont différents entre les divers algorithmes. Pour l'apprentissage hors-ligne l'environnement WEKA [WF05], proposé par l'université de Waikato, permet de réaliser rapidement des expérimentations. Cette même université a proposé en 2009 une boîte à outils d'expérimentation pour les flux : MOA [BK09]. On y retrouve la majorité des générateurs vus précédemment, ainsi qu'un nombre important d'algorithmes d'apprentissage sur les flux basés sur les arbres de Hoeffding. Kirkby [Kir08] utilise MOA pour réaliser ses expérimentations sous différentes contraintes de mémoire simulant divers environnements : réseau de capteurs (100 Ko de mémoire vive), PDA (32 Mo), serveur (400Mo). On peut donc comparer les différents algorithmes par rapport à l'environnement dans lequel il serait exécuté.

Autres points de comparaison : L'AUC [Faw04] est une mesure très souvent utilisée dans le cadre de la classification supervisée. Mais celle-ci n'est pas facilement calculable en-ligne, d'où l'utilisation assez générale du taux d'erreur (voir section 2.5.1). Dans le cadre des flux de données, d'autres mesures non corrélés à la qualité du modèle sont parfois utilisées en complément :

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

La comparaison de la vitesse d'apprentissage de deux algorithmes peut poser des problèmes, car il suppose que le code/logiciel est disponible publiquement. De plus, on n'évalue pas seulement la vitesse d'apprentissage, mais aussi la plate-forme sur laquelle tourne l'algorithme et la qualité de

l'implémentation. Ces mesures sont utiles pour donner un ordre d'idée, mais doivent être prises avec précautions et non pas comme des éléments absolus de comparaison.

2.5.4 Conclusion

Des techniques d'évaluation ont été récemment proposées par la communauté et une recherche de consensus est en cours [GRSR09]. Il faut donc trouver des critères, des jeux de données et des plateformes communs afin de pouvoir réaliser une comparaison aussi impartiale que possible des différentes méthodes proposées.

2.6 Conclusion

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

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

Au niveau des verrous à lever, le compromis précision / vitesse / mémoire semble une voie intéressante si ce compromis peut être directement écrit comme un critère à optimiser lors de l'apprentissage du classifieur. Les chapitres 3 et 4 aborderont cette problématique en proposant de nouveaux résumés (chapitre 3) et de nouvelles adaptations de classifieurs existants (chapitre 4). L'adaptation aux changements de concept est l'un des challenges les plus importants et difficiles à résoudre. En effet, il faut d'une part trouver quand le changement a lieu puis modifier le modèle en conséquence et ceci avec le moins de fausses détections possibles. Le chapitre 5 propose une nouvelle méthode de détection de changement pour les flux de données.

Chapitre 3

Discrétisation et groupage incrémentaux en-ligne

Sommaire

Ce chapitre a fait l'objet des publications suivantes :

A two layers incremental discretization based on order statistics (version courte) [SL11b]
A two layers incremental discretization based on order statistics (version longue) [SL13]

Introduction

Beaucoup d'applications génèrent d'importantes quantités d'informations : logs d'équipements réseaux, données du Web, réseaux sociaux... Le débit de ces données font qu'elles ne peuvent être exploitées qu'à leur passage. On parle alors de flux de données dont les caractéristiques sont les suivantes : (i) les données ne sont visibles qu'une fois, (ii) et dans leur ordre d'arrivée (voir section 2.3). Comme ces données ne peuvent pas être relues par la suite et que l'on se positionne, dans cette thèse, dans un environnement contraint en mémoire et en capacité de calcul, une grande partie des techniques habituelles d'apprentissage ne peuvent s'appliquer ici. Nous proposons de traiter cette problématique à l'aide de résumés univariés (un résumé par variable) incrémentaux et en-ligne. Étant donné que cette thèse considère des problèmes de classification supervisée, ces nouveaux résumés ont pour but l'estimation en-ligne de la densité conditionnelle aux classes à des fins de classification. Nos résumés permettent d'obtenir des estimations fiables de densité conditionnelle avec une faible consommation de mémoire et de temps de calcul.

Notre objectif est d'avoir, pour les données numériques, une discrétisation supervisée des données avec une estimation de la densité conditionnelle aux classes par intervalle. Pour les données catégorielles, l'objectif est de grouper les modalités ayant une estimation conditionnelle de densité proche. Notre solution est basée sur une méthode à deux niveaux utilisant des statistiques d'ordre pour les variables numériques et des comptes pour les variables catégorielles. Le premier niveau utilise une technique incrémentale de résumé en-ligne pour les flux qui prend en compte les ressources mémoires et processeurs et possède des garanties en termes d'erreur. Le second niveau utilise le résumé de faible taille issu du premier niveau pour construire la discrétisation finale. La faible taille du premier niveau et sa capacité à fournir des comptes permettent l'utilisation de méthodes supervisées et régularisées performantes tout en conservant une faible complexité temporelle.

Cette approche de discrétisation/groupage incrémentale supervisée constitue un prétraitement qui par la suite permettra de réaliser un apprentissage avec différentes techniques : classifieur bayésien naïf, arbres de décision... (voir chapitre 4) pouvant utiliser, soit directement les comptes, soit une estimation de densité fournie à partir des comptes.

Ces résumés sont prévus pour fonctionner sur un flux stationnaire. En cas de flux non stationnaire, les résumés se mettront à jour et fourniront des statistiques sur l'ensemble de la période observée. Une méthode de détection de changement de concept peut être ajoutée afin que de nouveaux résumés soient construits dès qu'un changement de concept est détecté (voir chapitre 5).

Plan du chapitre

Ce chapitre débute par un état de l'art des méthodes de discrétisation/groupage en-ligne et hors-ligne afin d'étudier les meilleurs choix possibles pour nos deux niveaux. La première section s'intéresse plus particulièrement aux variables numériques et à la discrétisation. La deuxième section effectue ce même état de l'art mais pour les méthodes de comptage/groupage dans le cadre des variables catégorielles. La troisième section présente de manière détaillée notre méthode à deux niveaux ainsi que les choix des méthodes pour les deux niveaux. Les méthodes choisies correspondent à une sélection des meilleures méthodes de l'état de l'art. Cependant notre positionnement se situant sur la classification supervisée nous proposons quatre nouvelles méthodes de résumé en-ligne et incrémental pour les données numériques dans le cadre supervisé. Les sections 4 et 5 correspondent aux expérimentations sur des jeux de données synthétiques de nos méthodes. La section 4 s'intéresse aux données numériques et la section 5 aux données catégo-

rielles. La dernière section conclut ce chapitre en résumant les apports de notre méthode à deux niveaux basée sur les comptes.

3.1 État de l'art des méthodes de discrétisation

3.1.1 Préambule

De nombreuses techniques de discrétisation existent dans la littérature. Des critères ont été proposés afin de réaliser une typologie des différentes méthodes existantes. On relèvera par exemple les travaux de Dougherty et al. [DKS95] et Gama [GP06] qui proposent les critères suivants :

- *globale/locale* : ce critère correspond à la manière dont les données sont utilisées pour trouver les intervalles. Une méthode utilisant toutes les données pour construire les intervalles est considérée comme *globale*. Une méthode n'utilisant qu'une partie des données et prenant des décisions localement est considérée comme *locale*.
- *supervisée* : la méthode utilise l'information sur les classes pour réaliser la discrétisation.
- *paramétrique* : une méthode non paramétrique est capable de trouver par elle-même le nombre d'intervalles. On peut citer l'approche MDL « Minimum Description Length » comme exemple de méthodes permettant de trouver le nombre d'intervalles.

Comme dans ce mémoire nous cherchons à évaluer la capacité à travailler en-ligne (voir section 2.1.4), le critère *« en-ligne »* sera ajouté.

Un état de l'art des différentes méthodes de discrétisation est présenté dans cette section. Celle-ci sera conclue par un tableau présentant le positionnement de ces méthodes selon les critères énoncés ci-dessus. L'état de l'art concernant la discrétisation incrémentale ici présenté est issu principalement de deux domaines :

- le domaine de l'apprentissage : les publications sont très nombreuses dans le cas de l'apprentissage statique (voir section 2.1.2), mais plus limitées pour l'apprentissage incrémental (voir section 2.1.3).
- le domaine des base de données : de nombreuses publications abordent le problème du maintien de statistiques en-ligne pour les données contenues dans les bases de données. Nous nous intéressons plus particulièrement à l'estimation de quantiles en-ligne car elle peut être utilisée comme une méthode de discrétisation incrémentale.

3.1.2 Apprentissage

Cette section s'intéresse à la discrétisation comme méthode de prétraitement. Le but est de réaliser par la suite l'apprentissage d'un classifieur supervisé utilisant ce prétraitement. Certains algorithmes sont capables de travailler directement sur des attributs numériques mais peuvent cependant profiter de la discrétisation [DKS95]. En effet, une première phase de discrétisation peut permettre une estimation plus fiable de la densité conditionnelle aux classes par intervalle. De manière générale une sur-discrétisation ne permet pas d'avoir des estimations fiables et peut engendrer des problèmes de sur-apprentissage, alors qu'une sous-discrétisation peut nuire aux performances du classifieur les utilisant.

Dans un premier temps les méthodes « hors-ligne » les plus connues de la discrétisation sont présentées puis les méthodes « en-ligne ».

3.1.2.1 Méthodes « hors-ligne »

Toutes les méthodes « hors-ligne » suivantes supposent que l'ensemble des données soit chargé en mémoire (pour certaines méthodes les performances seraient directement affectées s'il fallait faire de multiples chargements depuis le disque).

Equal Width Cette méthode découpe les données en I intervalles de même taille. Pour cela on a seulement besoin de connaître la valeur minimale min et maximale max des données à discrétiser. La taille d'un intervalle est ensuite définie par : $\frac{(max-min)}{I}$. Cette technique s'adapte mal aux flux de données car min et max sont inconnus et nécessiteraient une lecture préalable du flux pour être connu. De plus cette méthode n'est pas robuste aux « outliers » et si la distribution est non uniforme elle ne permet pas une bonne estimation de la densité.

Equal Frequency Cette méthode découpe les données en intervalles contenant la même quantité de données (soit le même nombre, soit la même somme de poids si les données sont pondérées). La méthode naturelle utilisée pour réaliser cette discrétisation consiste à trier toutes les valeurs et ensuite à trouver les points de coupure en relisant les données triées. Cette technique s'adapte mal aux flux de données car on ne peut pas conserver toutes les données et donc trouver les points de coupure exacts.

MDLP ([FI93]) Cette méthode est supervisée et sans paramètre. Elle utilise les comptes par classe et évalue récursivement les découpages binaires possibles. Pour chaque intervalle sa valeur d'entropie est calculée et le découpage qui permet d'avoir le plus grand gain en entropie est celui choisi. La notion de taille minimale de description est utilisée pour arrêter la discrétisation (approche MDL : Minimum Description Length). Cette technique s'adapte mal aux flux de données car on ne peut pas conserver toutes les données et de plus elle nécessite de nombreuses lectures des données.

MODL ([Bou06a]) Cette méthode est assez similaire à l'approche précédente : elle est supervisée, sans paramètre et utilise une approche MDL. La différence se situe au niveau de la méthode de découpage qui n'est plus basée sur l'entropie mais sur une approche bayésienne d'évaluation de la qualité du modèle. Son but est de trouver les meilleurs paramètres de la discrétisation : nombre d'intervalles, bornes des intervalles et répartition des classes dans les intervalles au sens bayésien. Les limitations sont similaires à l'approche MDLP : les données doivent être relues plusieurs fois. Les expérimentations menées dans [Bou06a] montrent que la méthode MODL nécessite moins de données pour trouver des motifs et est plus robuste au bruit que la méthode MDLP.

3.1.2.2 Méthodes incrémentales « en-ligne »

Contrairement aux méthodes précédentes qui sont prévues pour fonctionner « hors-ligne » et avec toutes les données en mémoire, cette section traite des méthodes de discrétisation capables de travailler « en-ligne » et de manière incrémentale.

Approximation par une gaussienne par classe Cette méthode suppose que la distribution des données se rapproche d'une loi normale que l'on va chercher à approximer. Pour cela il suffit de ne conserver que les trois valeurs par classe : la moyenne μ, l'écart type (ou la variance σ) et le nombre n d'éléments qui définissent cette gaussienne. Le maintien de ces trois valeurs

peut se faire de manière incrémentale et donc cette méthode est parfaitement adaptée à une utilisation en-ligne ou sur les flux. Dans [PHK08] l'adaptation complète de cette méthode pour les flux de données est présentée. Afin d'améliorer la recherche des points de coupure le minimum min et le maximum max sont stockés. La consommation mémoire est constante quelle que soit la nature du flux observé et la méthode conserve par classe un tuple de la forme suivante : $< \mu, \sigma, n, min, max >$ comme résumé. Pfahringer et al. [PHK08] recommandent l'utilisation de cette méthode dans le cadre des résumés pour les arbres d'Hoeffding car ils jugent celle-ci plus performante sur leurs jeux de données.

Incremental Flexible Frequency Discretization (IFFD) La méthode IFFD (Incremental Flexible Frequency Discretization) a été développée afin d'avoir une discrétisation incrémentale pour le classifieur bayésien naïf. Elle est basée sur deux paramètres $minIntervalleSize$ et $maxIntervalleSize$ qui sont les nombres minimaux et maximaux de valeurs que peut contenir un intervalle. Si un intervalle est trop grand alors il est découpé en deux intervalles. Toutes les valeurs étant conservées, aucune erreur n'est introduite mais ce stockage rend cette méthode inutilisable sur les flux. De plus le nombre d'intervalles va devenir très important au fur et à mesure de la discrétisation sur un grand volume de données. Les expérimentations menées sur IFFD se trouvent dans [LYW06] et une version plus détaillée est disponible dans [YW08]. Cependant la comparaison est réalisée avec les méthodes EqualWidth et EqualFrequency mais limitée à 10 intervalles alors que la méthode IFFD peut utiliser un plus grand nombre d'intervalles. On peut dès lors se demander si le fait d'avoir plus d'intervalles ne serait pas la seule contribution aux meilleurs résultats.

Construction d'histogramme en-ligne (SPDT) Ben-Haim et al [BHTT10] cherchent à réaliser des arbres de décision qui se construisent en-ligne. Une discrétisation en-ligne et incrémentale est donc nécessaire afin de pouvoir trouver les points de coupure des variables numériques. Ils proposent une méthode en-ligne basée sur les opérations suivantes :

- Update : ajout d'un exemple dans un intervalle si sa valeur existe déjà, sinon création d'un nouvel intervalle et réalisation d'un Merge par la suite ;
- Merge : fusion de 2 intervalles ;
- Sum : estimation du nombre d'exemples entre les valeurs [a,b] grâce à une extrapolation de la densité par la méthode des trapèzes.

Une dernière étape peut être réalisée afin de « lisser » la discrétisation pour se rapprocher de la méthode hors-ligne « Equal Frequency » :

- Uniform : par la méthode des trapèzes on recalcule toutes les bornes des intervalles de manière à avoir le même nombre d'exemples par intervalle

Cette méthode a été évaluée sur différents types de distribution : normale, exponentielle, uniforme. Elle a de bonnes performances empiriques bien que sa borne d'erreur théorique soit assez élevée.

Arbres binaires exhaustifs Gama et al. utilisent cette méthode pour leur arbre VFDTc [GRM03]. 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. Si le nombre de valeurs différentes devient important alors le temps d'insertion/consultation et la mémoire utilisée par le résumé devient conséquent. Cette méthode ne

peut plus être considérée comme « en-ligne » dans ces cas là car son caractère exhaustif montre ses limites.

Partition Incremental Discretization (PiD - [GP06]) Gama et Pinto dans [GP06] proposent une solution basée sur deux niveaux qui est partiellement incrémentale. 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 qui doit contenir plus d'intervalles que le niveau 2. Le deuxième niveau n'est pas incrémental et utilise la discrétisation réalisée au niveau 1 pour effectuer la discrétisation finale. Celle-ci peut être de plusieurs types : « Equal Width », « Equal Frequency », K-means, MDLP, proportional discretization.

La qualité de la discrétisation dépend essentiellement de la discrétisation de niveau 1. Si celle-ci ne contient pas assez d'intervalles alors l'information est définitivement perdue pour le niveau 2. Par rapport à une méthode conservant toutes les données, les erreurs introduites par PiD sont les suivantes :

- sur les bornes de coupure car toutes les valeurs ne sont pas gardées ;
- sur les comptes car les intervalles sont redivisés en faisant l'hypothèse de répartition uniforme des données dans cet intervalle.

La quantité de mémoire nécessaire pour le niveau 1 de cette méthode n'est pas bornée. Par exemple, si la distribution des données est exponentielle on peut se retrouver avec un nombre d'intervalles beaucoup plus important que celui paramétré à l'initialisation de la méthode.

3.1.3 Base de données

Les estimations de la répartition des données se retrouvent aussi dans les systèmes de bases de données (SGBD). Ils sont utilisés pour avoir des statistiques sur les données afin de créer les meilleurs plans d'exécution et savoir quelles optimisations peuvent être utilisées.

3.1.3.1 Réservoir et histogrammes (RH)

Gibbons et al [GMP02] proposent des techniques incrémentales de mise à jour des histogrammes pour les systèmes de bases de données. Ces histogrammes permettent de connaître la distribution des valeurs et donc de pouvoir réaliser de bons plans d'exécution des requêtes sur la base de données. Leur approche est basée sur deux structures :

- un réservoir de données contenant des données représentatives des données réelles (basé sur la technique du « Reservoir sampling » [Vit85]) : par exemple 100 Ko - sur disque
- des histogrammes « EqualFrequency » résumant les données : par exemple 2 Ko - en mémoire vive

Seul l'histogramme en mémoire est utilisé pour la réalisation des plans d'exécution. Dans un premier temps, la mise à jour des histogrammes est réalisée au maximum en mémoire sans avoir accès ni au réservoir ni aux tables de la base de données sous jacente. Dans un second temps quand l'erreur estimée est trop importante, le réservoir sur disque peut être utilisé pour mettre à jour les histogrammes. L'écart entre l'histogramme réel et son estimation possède des bornes calculables qui sont présentées dans la publication. L'approximation de l'histogramme peut produire deux types d'erreurs sur :

- les bornes des intervalles
- les comptes dans un intervalle

L'utilisation de la mémoire et les garanties en termes d'erreurs de cette méthode sont moins intéressantes que les méthodes suivantes basées sur des résumés de quantiles.

3.1.3.2 Résumé de quantiles

Les quantiles sont équivalents à une discrétisation en « Equal Frequency » pour laquelle le nombre d'intervalles est le nombre de quantiles. Cependant, une discrétisation « Equal Frequency » en-ligne n'est pas réalisable car elle nécessiterait de relire plusieurs fois les données [MP78]. Les résumés de quantiles présentés ici permettent d'obtenir une discrétisation de type « Equal Frequency » sur un flux de données avec une erreur bornée en fonction de la taille mémoire.

Les quantiles correspondent à 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 $\lceil (\phi + \epsilon) N \rceil$. Les deux algorithmes, présentés ci-dessous, sont insensibles à l'ordre d'arrivée des données.

MLR [MRL98] proposent une solution basée sur un ensemble de buffers. Cette solution est basée sur trois opérations :

1. New : prend en entrée un buffer vide et le remplit avec les nouvelles valeurs du flux.
2. Collapse : quand tous les buffers sont remplis, il faut en fusionner au moins deux afin de récupérer des buffers vides. Cette opération prend au moins deux buffers et les fusionnent afin qu'ils n'en forment plus qu'un.
3. Output : cette opération correspond au calcul du quantile passé en paramètre à partir des différents buffers.

Le besoin en mémoire de cette approche est en $O(\frac{1}{\epsilon} log^2(\epsilon N))$.

GK : Greenwald et Khanna [GK01] proposent une amélioration de la méthode MLR en abaissant la consommation de mémoire : $O(\frac{1}{\epsilon} log(\epsilon N))$ dans le pire des cas. De plus cette méthode, contrairement à MLR, ne nécessite pas de connaître la taille des données à l'avance. Un des avantages de cette méthode est que, selon le besoin, on peut soit définir une borne d'erreur maximale, soit définir une borne de mémoire maximale à utiliser. Cette méthode est présentée de manière détaillée dans la section 3.3.2.

3.1.4 Bilan

Un résumé synthétique des méthodes vues précédemment en fonction des critères proposés dans le préambule de ce chapitre (Section 3.1.1) est présenté dans le tableau 3.1

Les méthodes présentées dans cette section correspondent à des références de l'état l'art. Il n'est pas possible de fournir une liste exhaustive de toutes les méthodes. Certaines de ces méthodes ont été modifiées afin de répondre à des problèmes particuliers. On peut par exemple noter que la méthode GK a été adaptée ([ZW07]) afin d'être plus rapide en temps de calcul mais au prix d'une consommation mémoire supérieure. Nous verrons par la suite que l'approche proposée dans ce chapitre est indépendante du choix de la méthode de discrétisation. Le choix des méthodes peut donc se faire selon les contextes d'utilisation.

Dans le cadre de notre approche présentée par la suite le résumé GK pourra être utilisé pour les besoins « en-ligne », typiquement pour réaliser un résumé de quantiles sur un flux. La

Méthodes	globale / locale	supervisée	paramétrique	en-ligne
Equal Width	globale	non	oui	non
Equal Frequency	globale	non	oui	non
MODL	globale	**oui**	**non**	non
MDLP	locale	**oui**	**non**	non
Gaussienne	globale	non	**non**	**oui**
IFFD	globale	non	oui	non
SPDT	globale	non	oui	**oui**
Arbre binaire	globale	non	**non**	non
PiD (1ère couche)	globale	non	oui	**oui**
RH	globale	non	oui	**oui**
MLR	globale	non	oui	**oui**
GK	globale	non	oui	**oui**

TABLE 3.1 – Comparaison des méthodes de discrétisation.

discrétisation MODL sera quant à elle réservée pour les usages sur une petite quantité de données comme par exemple un résumé.

Suite à cet état de l'art nous considérons ultérieurement comme :

Méthodes performantes « en-ligne », le résumé de quantiles GK qui semble être un bon compromis entre la consommation mémoire et le temps de calcul. De plus il possède des garanties en termes d'erreurs et aucun *a priori* sur la distribution des données.

Méthodes performantes « hors-ligne », la discrétisation MODL qui est supervisée, sans paramètre et régularisée. Son utilisation en-ligne n'est envisageable que sur de petites quantités de données ou sur un résumé de petite taille.

3.2 État de l'art sur les méthodes de comptage/groupage de valeurs

3.2.1 Préambule

Les méthodes de résumé utilisées pour la discrétisation ne peuvent être identiques à celles utilisées pour le groupage. En effet, la discrétisation porte sur une variable continue dont les valeurs sont ordonnées contrairement aux valeurs d'une variable catégorielle. Dans la suite de ce chapitre, une première section présente des méthodes de comptage non supervisées adaptées à une utilisation en-ligne. Une deuxième section s'intéresse aux méthodes de groupage supervisées prévues pour fonctionner hors-ligne.

3.2.2 Méthodes non supervisées

3.2.2.1 Simple comptage

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.

3.2.2.2 Count sketch (CS)

Le but du Count Sketch [Cha04] est de trouver les valeurs qui apparaissent le plus souvent dans un flux de données avec une erreur maximale ϵN. Ceci correspond à trouver la liste de d éléments les plus fréquents respectant la contrainte $n_i > (1-\epsilon)n_d$ où n_i est le nombre d'apparitions de l'élément d'indice i dans la liste des éléments triés par fréquence. Ce résumé est stocké sous la forme d'une matrice de comptage de taille $t \times b$. Il utilise t fonctions de hachage s_i dans $\{+1, -1\}$ et t fonctions de hachage h_i dans $\{1, \ldots, b\}$. La matrice est mise à jour à l'arrivée d'un nouvel élément x de la manière suivante :

$$\forall i = 1, \ldots, t \quad b[i, h_i(x)] \leftarrow b[i, h_i(x)] + s_i(x)$$

La fréquence d'un élément est estimée par la médiane de $h_i(x) \times s_i(x)$. La matrice ne contient que des comptes, si l'association valeur/compte doit être conservée alors une liste de ceux-ci doit être maintenue en parallèle. Cette liste ne peut être exhaustive et elle ne conserve donc que les couples valeurs/fréquences dont les fréquences sont les plus importantes. La consommation mémoire du « count sketch » est en $O(1/\epsilon^2)$.

3.2.2.3 Count-min Sketch (CMS)

Le Count-min Sketch [CM05] est similaire au Count sketch vu précédemment mais propose des améliorations sur différents points :

1. la consommation mémoire est en $O(1/\epsilon)$
2. la mise à jour du sketch est de complexité sublinéaire par rapport à la taille du sketch $O(log(1/\epsilon))$

Comme auparavant une matrice de comptage de taille $t \times b$ est utilisée pour le stockage. La différence provient de l'absence des fonctions s_i qui sont remplacées par une mise à jour par simple incrémentation :

$$\forall i = 1, \ldots, t \quad b[i, h_i(x)] \leftarrow b[i, h_i(x)] + 1$$

Le choix de t et b se fait à l'aide de deux paramètres δ et ϵ. Si l'on veut que l'estimation du nombre d'apparitions $\hat{c}$ d'un item n'ait pas une erreur supérieure à ϵn avec une probabilité d'au moins $1-\delta$ alors il faut que $t = \lceil ln\frac{1}{\delta} \rceil$ et $b = \lceil \frac{e}{\epsilon} \rceil$ ($e = 2,71828$). La fréquence d'un élément x est estimée par le minimum de $h_i(x)$.

$$\hat{c} = \underset{i}{argmin}(b[i, h_i(x)])$$

3.2.2.4 Résumé multi-dimensionel : ASP

Hershberger et al. dans [HSST06] s'intéressent aux résumés de données pour le cas multi-dimensionnel. Ils proposent la méthode ASP (Adaptive Spacial Partitionning) qui peut à la fois travailler sur les variables catégorielles et continues. Dans le cas d'un jeu de données n'ayant qu'une dimension cette méthode se comporte bien et a des résultats assez proches d'autres méthodes comme GK. Par contre son extension au cas multidimensionnel pose problème car il

se fait en $O(2^d)$ où d est le nombre de dimensions (nombre d'attributs). La complexité de cette méthode devient rapidement trop importante pour les cas multivariés ce qui limite son intérêt.

3.2.3 Méthodes supervisées

Les méthodes supervisées de groupement de valeurs sont nettement moins étudiées dans la littérature que la discrétisation supervisée. Cependant dans le cas de l'apprentissage et plus particulièrement dans le cas des arbres de décision le regroupement cherche à éviter une trop grande fragmentation des données. Pour les méthodes nécessitant une traduction des valeurs en un codage disjonctif complet (réseaux de neurones, réseaux bayésien ou régression logistique) le but est de limiter le nombre de points d'entrée. Cette section présente les principales méthodes de groupement de valeurs dont aucune n'est adaptée à un usage en-ligne.

3.2.3.1 MODL

La méthode MODL [Bou05] pour le groupage de valeurs est similaire à la discrétisation MODL. Cette méthode est basée sur le formalisme bayésien pour trouver le groupage le plus probable par rapport aux données observées. Son but est de trouver les meilleurs paramètres du groupage : nombre de groupes, choix de la partition des valeurs explicatives et la répartition des classes dans les groupes au sens bayésien. L'algorithme MODL utilise une méthode ascendante gloutonne pour optimiser le critère.

3.2.3.2 ChAID

La méthode ChAID (CHi-squared Automatic Interaction Detector) [Kas80] est basée sur l'utilisation du critère du $\mathcal{X}^2$ pour établir si deux intervalles sont statistiquement différents. L'algorithme se réalise en 3 étapes :

1. Calcul pour toutes les paires de groupement possibles de leurs valeurs du $\mathcal{X}^2$.
2. On fusionne la paire ayant le plus faible $\mathcal{X}^2$ si celui-ci est inférieur au seuil α fixé. On retourne à l'étape 1 tant que l'on peut faire des fusions.
3. Une fois toutes les fusions de paires possible réalisées, on essaye de segmenter les groupes créés. Étant donné que l'étape 1 fusionne les groupes 2 par 2, tous les groupements possibles ne sont pas testés. Le but de cette étape est de chercher à diviser un groupement de plus de 2 groupes qui soit supérieur au seuil α.

L'algorithme ChAID s'apparente fortement aux méthodes ascendantes Chi-Merge [Ker92] et Fusinter [ZRR98]. Seule l'étape 3 est spécifique à ChAID et l'algorithme effectue rarement un redécoupage (étape 3) après les regroupements (étape 2) [ZR00].

3.2.3.3 CART

Le critère utilisé dans l'approche CART [BFOS84] se base sur l'indice d'impureté de Gini. Pour un problème à deux classes, on ordonne les valeurs par ordre croissant des probabilités d'apparition d'une classe donnée pour tous les groupes. Ensuite on cherche la bipartition qui donne le plus grand gain de l'indice de Gini sur cet ordre pré-établi. Pour un problème multi-classe, on peut utiliser l'indice de Gini pour plusieurs classes mais l'ordonnancement des données n'est plus suffisant pour garantir la solution optimale. Toutes les bipartitions possibles doivent alors être testées, ce qui n'est pas envisageable si le nombre de valeurs est très important.

3.2.3.4 Gain informationnel

L'algorithme C4.5 [Qui93] utilise la gain informationnel basé sur l'entropie de Shannon pour évaluer les variables catégorielles mais sans réaliser de groupage. Ce critère ayant tendance à favoriser les attributs avec beaucoup de modalités, Quinlan propose dans la méthode C4.5 un correctif qui consiste à diviser le gain informationnel par la quantité d'information contenue dans la variable catégorielle.

3.2.4 Bilan

Un résumé synthétique des méthodes vues précédemment est présenté dans le tableau 3.2. Les critères utilisés sont un sous-ensemble de ceux présentés dans le préambule de ce chapitre (voir section 3.1.1).

Méthodes	supervisée	paramétrique	en-ligne
Comptage	non	**non**	**oui** (mais mémoire non limitée)
CS	non	oui	**oui**
CMS	non	oui	**oui**
ADP	non	oui	non
MODL	**oui**	**non**	non
ChAID	**oui**	oui	non
Cart	**oui**	**non**	non
Gain informationnel	**oui**	**non**	non

TABLE 3.2 – Comparaison des méthodes de groupage.

Les références de l'état l'art en matière de comptage/groupage ont été présentées dans cette section. Bien que le nombre de publications relatives à cette thématique soit bien moins important que celles concernant la discrétisation, il n'est pas possible d'être exhaustif dans ce chapitre de thèse. Le choix de la méthode de groupage est indépendant de l'approche à deux niveaux proposée dans la section suivante. Ce choix peut donc se faire selon les contextes d'utilisation. Des études existent pour aider ce choix. On peut par exemple citer, pour le comptage des items fréquents, l'étude de Cormode *et al.* [CH08] sur divers algorithmes avec plusieurs distributions de données.

Comme pour la discrétisation nous pouvons distinguer deux cas d'utilisation. Pour le comptage « en-ligne » d'un nombre important de données nous proposons d'utiliser le résumé CMS. Pour le groupage sur une petite quantité de données, comme par exemple un résumé, notre choix se porte sur la méthode MODL. Celle-ci a été comparée aux méthodes concurrentes ChAID et « Gain ratio » dans [Bou05] et apparaît comme étant la plus performante en terme de nombre de groupes retrouvés et de robustesse au bruit.

Suite à cet état de l'art nous considérons ultérieurement comme :

Méthodes performantes « en-ligne », le Count-min Sketch (CMS) qui possède les meilleures propriétés en termes de consommation mémoire, de temps de calcul et d'erreur. Les approches que nous allons utiliser par la suite ne nécessitent pas de connaître *a posteriori* les valeurs des variables nominales qui ont servi à construire le résumé. L'utilisation de méthodes basées sur une projection par hachage ne pose donc aucun problème du moment qu'elles permettent de retourner des comptes pour la valeur fournie en entrée.

Méthodes performantes « hors-ligne », le groupage MODL est supervisé, sans paramètre et régularisé. Son utilisation en-ligne n'est envisageable que sur de petites quantités de données ou sur un résumé de petite taille.

3.3 Une méthode à deux niveaux

3.3.1 Présentation de l'approche

Plusieurs méthodes de discrétisation/groupage incrémentale pour les flux ont été présentées précédemment. Certaines méthodes sont dédiées aux flux mais ne sont pas prévues pour réaliser une discrétisation supervisée. D'autres méthodes sont supervisées et non paramétriques mais ne sont quant à elles pas adaptées aux flux. Cette section propose une méthode de discrétisation/groupage supervisée à deux niveaux inspirée des travaux de Gama avec sa méthode PiD (voir 3.1.2.2). Le premier niveau construit « en-ligne » un résumé de faible taille. Cette faible taille permet l'utilisation d'une méthode « hors-ligne » pour le deuxième niveau. Cette approche permet donc de combiner les méthodes dédiées à une utilisation en-ligne avec des méthodes plus performantes mais qui ne peuvent satisfaire les contraintes d'une utilisation en-ligne.

Pour le **premier niveau** notre choix se porte sur des méthodes capables de **contrôler au mieux l'erreur pour une quantité de mémoire donnée**. Les méthodes GK et CMS possèdent ces propriétés et seront utilisées comme point de départ pour notre premier niveau. Les autres méthodes de l'état de l'art possédant une faible consommation mémoire comme le premier niveau de PiD et le résumé par une gaussienne seront utilisés à titre de comparaison.

Pour le **deuxième niveau** les méthodes **supervisées**, **régularisées** et **sans paramètre** nous apparaissent comme un choix naturel. La méthode MODL, possèdant toutes ces propriétés, est un choix pertinent pour notre deuxième niveau. De plus celle-ci s'applique à la fois aux variables numériques et catégorielles.

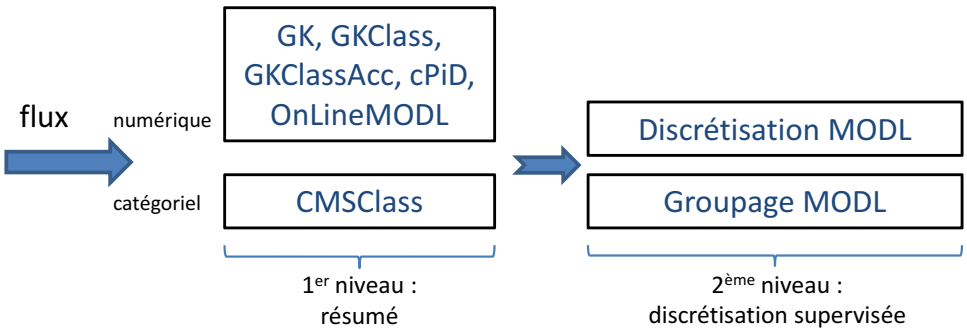


FIGURE 3.1 – Une nouvelle méthode à deux niveaux.

La figure 3.1 illustre notre nouvelle méthode et indique les différents résumés qui seront utilisés en fonction du niveau et du type de variables. Pour le premier niveau numérique, 4 nouvelles méthodes de résumé qui se construisent en-ligne de manière incrémentale sont proposées et seront détaillées dans la suite de ce chapitre. Notre première proposition (GKClass) reprend le résumé GK mais l'adapte en ajoutant les comptes par classe dans les tuples afin d'améliorer ses performances en classification. Notre deuxième proposition (GKClassAcc) est une modification de l'algorithme GKClass afin d'augmenter la vitesse d'insertion dans le résumé. La troisième proposition (cPiD) est une modification de l'algorithme de résumé PiD qui fonctionne à mémoire constante ; ceci afin de permettre la comparaison du premier niveau de la méthode PiD de Gama avec les autres

méthodes. La dernière proposition (OnLineMODL) est une méthode de résumé en-ligne supervisée basée sur le critère MODL.

3.3.2 Premier niveau

Pour notre premier niveau nous avons choisi des méthodes en-ligne qui contrôlent la quantité d'erreur par rapport à la mémoire qui leur est allouée. Pour les variables numériques le choix de départ se portent sur le résumé de quantiles GK. Pour les variables catégorielles le Count-min Sketch (CMS) apporte les garanties recherchées. Ces deux méthodes sont considérées comme parmi les plus performantes de l'état de l'art. De plus elles travaillent directement avec des comptes qui sont nécessaires au deuxième niveau.

Ces résumés ne sont pas originellement prévus pour des tâches de classification et certaines améliorations semblent possibles. Après avoir détaillé plus précisément le fonctionnement de l'algorithme GK, nous proposons 4 nouvelles méthodes de résumé numérique dans la suite de cette section. Pour les variables catégorielles une version adaptée du CMS est proposée.

3.3.2.1 GK : résumé de quantiles de Greenwald et Khanna

Ce résumé de quantiles a pour intérêt de gérer le compromis consommation mémoire versus nombre d'erreurs. Il a été succinctement présenté dans la section 3.1.3.2 et est basé sur un tableau T de taille maximale M. Un tuple t_i de ce tableau T est défini par le triplet $< v_i, g_i, \Delta_i >$ où

- v_i est une valeur du flux de données
- g_i correspond au nombre d'individus vus entre v_{i-1} et v_i
- Δ_i est l'erreur maximale possible sur g_i

La figure 3.2 est une illustration de résumé GK avec plusieurs tuples. Deux versions de ce résumé existent. Dans la première version l'erreur maximale est fixée et le résumé consomme autant de mémoire que nécessaire pour que l'erreur maximale ne soit pas dépassée. La deuxième version fonctionne de la manière inverse : on fixe une quantité de mémoire maximale et on l'utilise au mieux pour minimiser l'erreur. On s'intéresse par la suite à cette deuxième version qui supprime un tuple après chaque insertion afin d'avoir un résumé de taille constante comme le montrent les algorithmes 1 et 2.

Entrées : T, x, M
$position \leftarrow$ `TrouverIntervalInférieurOuEgal` (T, x)
$fullNess \leftarrow T[position].g + T[position].\Delta$
`InsérerTupleDansTableau` $(T, position, < v = x, g = 1, \Delta = fullNess - 1 >)$
si `Taille` $(T) > M$ **alors**
| `Réduire` (T, M)
fin

Algorithme 1 – Insertion d'un nouvel individu dans le résumé GK.

3.3.2.2 GKClass : adaptation du résumé GK pour la classification

Le résumé GK n'est pas prévu au départ pour être utilisé pour des tâches de classification supervisée mais pour déterminer des quantiles et la répartition des données. Ce résumé a été évalué tel quel dans [PHK08], c'est-à-dire en conservant un résumé par classe et par attribut. Il est évalué comme étant moins performant que le résumé par une gaussienne ou la méthode

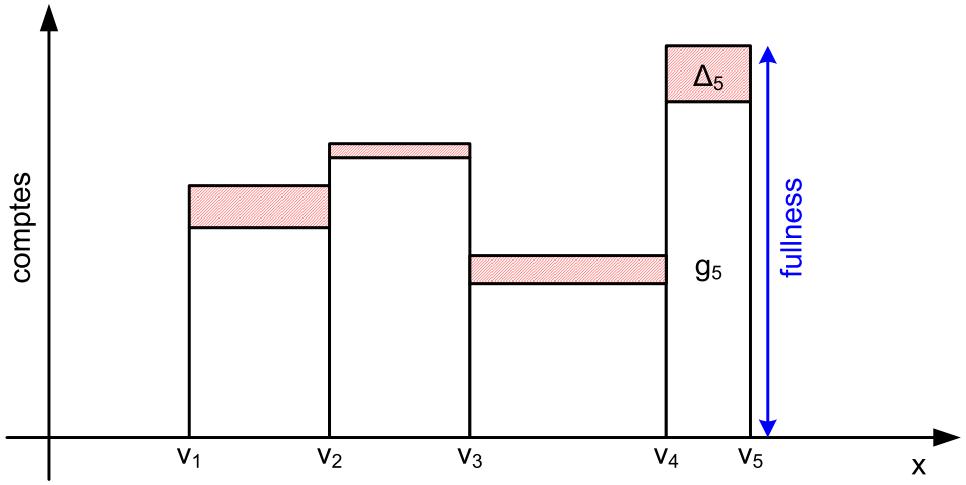


FIGURE 3.2 – Illustration du résumé GK (t_5 est le seul tuple dont les valeurs $< v_5, g_5, \Delta_5 >$ sont indiquées).

Entrées : T, M
Sorties : $positionFusion$
$smallestFullness \leftarrow \infty$
$positionFusion \leftarrow 0$
pour $i \leftarrow 1$ **a** *M-1* **faire**
  $fullnessAfterMerge \leftarrow T[i].g + T[i+1].g + T[i+1].\Delta$
  si $(T[i].\Delta >= T[i+1].\Delta) et (fullnessAfterMerge < smallestFullness)$ **alors**
    $smallestFullness \leftarrow fullnessAfterMerge$
    $positionFusion \leftarrow i$
  fin
fin
`FusionnerTuples` $(T, smallestFullnessIndex, smallestFullnessIndex + 1)$
retourner $positionFusion$

Algorithme 2 – Fonction `Réduire` : réduction de la taille du résumé GK.

VFML (méthode qui choisit les intervalles en utilisant les premières valeurs vues comme points de coupure). Étant données les qualités de l'approche GK vues précédemment ces résultats paraissent surprenants. Cependant étant donné que le résumé évalué dans [PHK08] conserve un résumé par classe et par attribut, les points de coupure ne sont pas les mêmes selon les classes. Le nombre de valeurs avant un point de coupure est la somme des comptes stockés dans chaque quantile strictement inférieur au point de coupure. Leur méthode introduit donc des erreurs qui sur certains jeux de données rend leur résumé GK très peu précis. Leurs expérimentations mettent particulièrement en avant ce comportement avec le générateur « Random Tree » (voir la Figure 2 dans [PHK08]).

Suite à ce bilan nous avons adapté le résumé GK afin de n'avoir qu'un résumé par attribut mais dans lequel chaque tuple contient les comptes par classe. Le tuple d'origine qui était composé de 3 valeurs : v_i, g_i, Δ_i devient le tuple $v_i, \Delta_i, c_{i1}, c_{i2}, ..., c_{ik}$, où $c_{i1}, c_{i2}, ..., c_{ik}$ correspondent aux comptes par classe des individus de l'intervalle i. Afin de valider cette approche nous avons mené une expérimentation similaire à celle de Pfahringer et al. [PHK08]. Celle-ci compare les différentes méthodes pour une utilisation des résumés dans les arbres de Hoeffding. Afin que nos

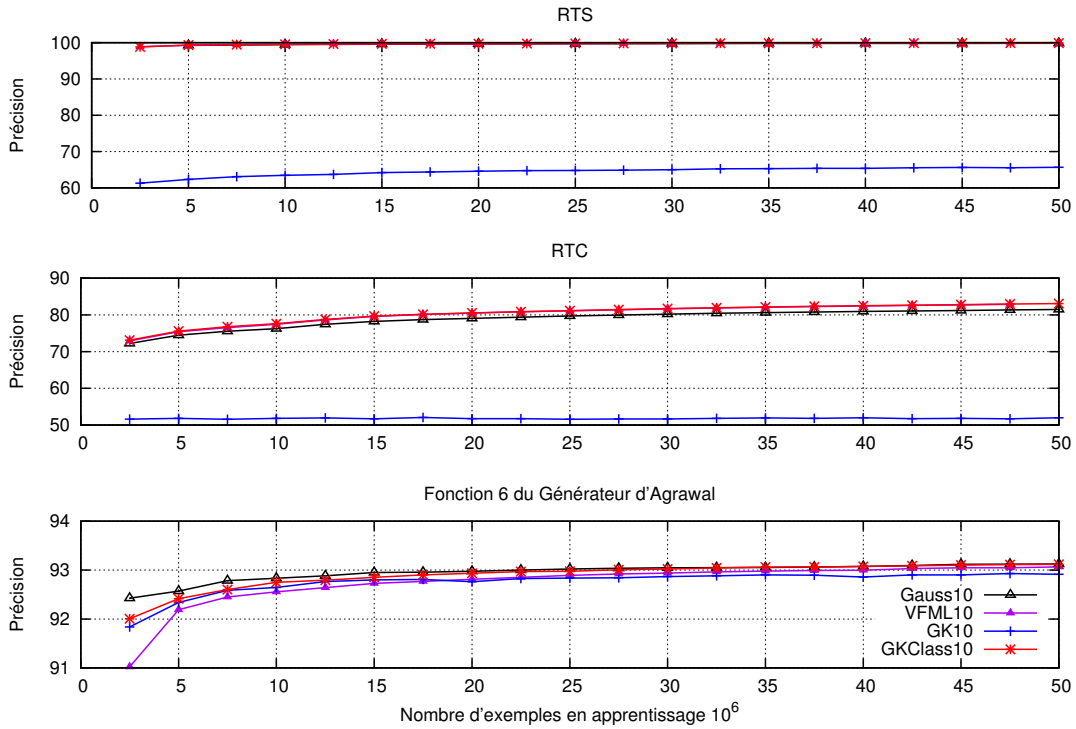


FIGURE 3.3 – Comparaison de différentes méthodes de discrétisation en-ligne sur un arbre de Hoeffding.

résultats soient comparables avec l'article [PHK08] nous reprenons leurs algorithmes, notations et paramètres d'expérimentations (10 intervalles). Les 4 méthodes suivantes ont été évaluées :

- Gauss10 : résumé par une gaussienne avec l'évaluation de 10 points de coupure ;
- VFML10 : méthode VFML avec 10 intervalles (10 premières valeurs vues comme frontière des intervalles) ;
- GK10 : résumé GK construit pour chaque attribut et chaque classe avec 10 tuples ;
- GKClass10 : notre résumé GK construit seulement par attribut et contenant les comptes des classes dans chaque tuple avec 10 tuples ;

Trois jeux de données identiques à ceux utilisés dans [PHK08] ont été produits (les détails de ces générateurs se trouvent dans [Kir08]). RTS (741 nœuds) et RTC (127 837 nœuds) sont des générateurs basés sur un modèle qui est un arbre. La fonction F6 d'Agrawal est basée sur un modèle à base de règles.

	RTS	RTC	F6 Agrawal
Nombre d'attributs numériques	10	50	6
Nombre d'attributs catégoriels	10	50	3
Profondeur maximale de l'arbre	5	10	-
Taux de bruit	0%	0%	5%

La figure 3.3 montre que notre modification (GKClass) s'avère très fructueuse puisque les performances arrivent au même niveau que les meilleures méthodes concurrentes. La méthode VFML10 ressort aussi comme une bonne méthode, mais de part sa méthode de construction, elle est extrêmement sensible à l'ordre d'arrivée des exemples. Dans [Kir08] une expérience sur un

jeu de données trié montre les limites de la méthode. Les méthodes basées sur GK et sur une gaussienne ne sont pas sensibles à l'ordre d'arrivée et sont donc les méthodes utilisées pour les évaluations futures.

3.3.2.3 GKClassAcc : accélération des insertions dans GKClass

GKClassAcc est une version modifiée de l'algorithme GK qui a pour but d'accélérer les insertions dans le résumé. Elle repart de l'algorithme GK d'origine qui supprime un tuple après chaque insertion afin d'avoir un résumé de taille constante. La différence porte au niveau des insertions qui se font directement dans le tuple si possible. En effet l'algorithme proposé par Greenwald et Khanna insère toujours un nouveau tuple quand une valeur arrive puis réalise une opération de fusion. Notre idée avec GKClassAcc est d'éviter si possible cette étape d'insertion puis de fusion en permettant une insertion directement dans le tuple. Pour cela la pire valeur de remplissage max_F (« fullness » : $g_i + \Delta_i$ - voir figure 3.2) est conservée. Si l'insertion directe dans le tuple ne conduit pas à un tuple ayant un remplissage supérieur à max_F alors l'insertion peut se faire directement. Sinon le mode de fonctionnement de l'algorithme de départ, comme il est proposé par Greenwald et Khanna, est repris. Notre nouvelle version contrôle elle aussi son erreur mais ne possède plus les garanties théoriques d'erreurs pour une taille de mémoire donnée.

```
// max_F est initialisé à 0 à la création du résumé
Entrées : T, x, M, max_F
position ← TrouverIntervalInférieurOuEgal (T, x)
fullNess ← T[position].g + T[position].Δ
si Taille (T) < M alors
    // tant que le résumé n'est pas plein on insère des tuples
    InsérerTupleDansTableau (T, position, < v = x, g = 1, Δ = fullNess − 1 >)
sinon
    si (T[position].g + T[position].Δ < max_F) alors
        // ce tuple n'est pas le plus rempli : on peut directement insérer dedans
        T[position].g ← T[position].g + 1
    sinon
        // tuple trop rempli : on insère normalement...
        InsérerTupleDansTableau (T, position, < v = x, g = 1, Δ = fullNess − 1 >)
        // ...et on récupère la mémoire en faisant une fusion
        positionFusion ← Réduire (T, M)
        max_F ← max(max_F, T[positionFusion].g + T[positionFusion].Δ)
    fin
fin
```

Algorithme 3 – Insertion d'un nouvel individu dans le résumé GKClassAcc (la classe a été retirée pour plus de clarté et facilité la comparaison avec l'algorithme 1 (GK)).

Afin de comparer les performances en termes de vitesse d'insertion de ce nouvel algorithme nous avons réalisé une étude qui porte sur deux axes : le nombre d'insertions et la taille du résumé. Le nombre d'insertions varient de 10^3 à 10^7 et la taille du résumé de 10 à 10^5 tuples. Le tableau 3.3 présente les résultats en terme d'amélioration du temps d'insertion dans le résumé. On remarque que l'algorithme fonctionne toujours aussi bien mais dès qu'il y a plus d'insertions que la taille du résumé alors il devient meilleur. La phase d'insertion est identique dans les deux algorithmes ce qui explique que l'amélioration n'intervient qu'après. La figure 3.4 montre aussi

nombre d'insertions	10 tuples	50 tuples	100 tuples	1000 tuples	10000 tuples
10^3	21%	153%	187%	0%	0%
10^4	67%	200%	400%	691%	0%
10^5	150%	250%	400%	2331%	817%
10^6	92%	233%	428%	3965%	4025%
10^7	87%	275%	413%	5101%	16343%

TABLE 3.3 – Pourcentage d'amélioration du temps d'insertion dans le résumé GKClass accéléré par rapport à GKClass.

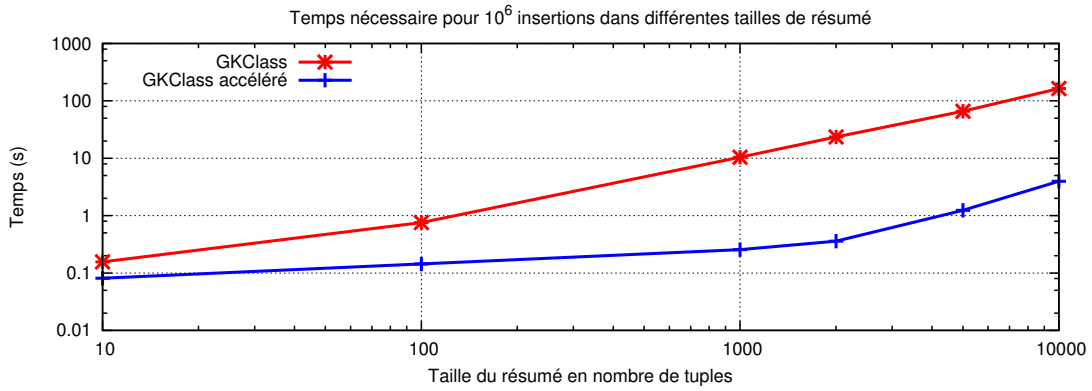


FIGURE 3.4 – Comparaison des temps d'insertion des méthodes GKClass et GKClass accéléré.

que sa complexité croit linéairement mais avec un ordre de magnitude de moins que la version d'origine. Cette différence provient de l'insertion directe dans le tuple quand cela est possible. Si l'insertion n'est pas possible directement alors on repasse au mode de fonctionnement d'origine qui consiste à réaliser la fusion des deux tuples qui minimisent le « fullness ».

Nous ne nous intéressons ici qu'à la vitesse d'insertion. La qualité de la discrétisation sera étudiée dans la section suivante.

3.3.2.4 cPiD : PiD à mémoire constante

PiD [GP06] est la méthode dont nous nous inspirons pour notre méthode à deux niveaux. Gama fournit l'algorithme pour son niveau 1 mais celui-ci ne fonctionne pas à mémoire constante. Nous proposons donc une modification afin d'avoir une mémoire constante et pouvoir réaliser une comparaison équitable entre les différentes méthodes. L'augmentation de la consommation de mémoire de la méthode PiD est due à la création de nouveaux intervalles. En effet si un intervalle devient trop peuplé alors il est divisé en deux intervalles contenant chacun la moitié des individus. Nous proposons une modification de l'algorithme de Gama afin de conserver une consommation mémoire constante. Cette modification consiste, suite à la division d'un intervalle, à fusionner les deux intervalles consécutifs dont la somme des comptes est la plus faible. Ainsi le nombre d'intervalles stockés reste toujours le même.

L'algorithme utilise une structure de données basée sur un tableau T de taille maximale M possédant des intervalles

$$t_i < v_i, c_{i1}, c_{i2}, ..., c_{iK} >$$

où :

- v_i est la valeur minimale de l'intervalle i
- $c_{i1}, c_{i2}, ..., c_{iK}$ correspondent aux comptes des individus dans l'intervalle i

Aucune comparaison n'est réalisée entre PiD et cPiD car notre intérêt se porte sur une utilisation à mémoire constante des méthodes. Le nombre de tuples utilisés par PiD peut s'accroître au fil des insertions ce qui empêche de comparer ces deux méthodes équitablement. Cependant si PiD n'a pas besoin d'allouer plus de mémoire alors cPiD et PiD auront les mêmes performances car les algorithmes sont les mêmes dans ce cas.

3.3.2.5 OnLineMODL : un premier niveau basé sur le critère MODL

Présentation

Toutes les approches précédentes utilisent deux niveaux dont le premier n'est pas supervisé et le second l'est. Dans cette section nous proposons une méthode basée sur le critère MODL pour la discrétisation du premier niveau. L'intérêt de cette approche est de rendre le premier niveau supervisé et d'être encore plus cohérent et performant avec le second niveau basé sur la discrétisation MODL. Le fait de rendre le premier niveau supervisé doit permettre de le rendre capable d'allouer plus de mémoire aux régions contenant plus d'informations et moins à celle en contenant peu. L'inconvénient principal de cette méthode est que la qualité de la discrétisation dépend de l'ordre d'arrivée des données car les fusions une fois effectuées ne peuvent être remises en cause.

La méthode de découpage de l'approche MODL est basée sur une approche bayésienne d'évaluation de la qualité du modèle. Son but est de trouver les meilleurs paramètres de la discrétisation : nombre d'intervalles, bornes des intervalles et répartition des classes dans les intervalles au sens bayésien. Dans le cas des variables numériques on cherche à minimiser le critère C_{disc} suivant :

$$C_{disc} = logN + \tag{3.1}$$

$$log\binom{N+I-1}{I-1} + \tag{3.2}$$

$$\sum_{i=1}^{I} log\binom{c_i+K-1}{K-1} + \tag{3.3}$$

$$\sum_{i=1}^{I} log\frac{c_i!}{c_{i1}!c_{i2}!\ldots c_{iK}!} \tag{3.4}$$

avec :

- N : taille des données (nombre d'individus)
- K : nombre de classes (connu)
- I : nombre d'intervalles (inconnu)
- c_i : nombre d'individus dans l'intervalle i
- c_{ik} : nombre d'individus dans l'intervalle i pour la classe k

Ce critère se décompose en quatre termes. Les trois premiers (3.1, 3.2 et 3.3) correspondent à l'*a priori* et le quatrième (3.4) à la vraisemblance. Le premier terme (3.1) correspond au choix du nombre d'intervalles et le second terme (3.2) au choix de la partition en intervalles des valeurs explicatives. Le terme (3.3) représente le choix des distributions des classes dans

chaque intervalle. Le dernier terme (3.4) exprime la probabilité d'observer ces distributions dans les données (vraisemblance). Tous les détails sur cette approche sont disponibles dans [Bou06a, Bou07a].

Comme nous travaillons en-ligne à un moment donné, notre résumé contient un nombre N d'exemples et donc le premier terme (3.1) aura toujours la même valeur. Le premier niveau consomme une mémoire fixe qui permet le stockage d'un certain nombre d'intervalles. Cette quantité de mémoire définit le nombre de tuples que les résumés peuvent contenir. Les méthodes vues précédemment ont aussi leurs nombres de tuples/intervalles définis par la quantité de mémoire. Ce nombre d'intervalles I est fixe et est compté comme un coût *a priori* dans le terme (3.2) du critère MODL. Ce coût est le même quelque soit le modèle de discrétisation choisi. Les termes (3.3) et (3.4) quant à eux dépendent des données et le but de notre premier niveau est donc de chercher à minimiser leur somme. Soit c_{t_i} le coût d'un intervalle :

$$c_{t_i} = log\binom{c_{i.} + K - 1}{K - 1} + log\frac{c_{i.}!}{c_{i1}!c_{i2}!\ldots c_{iK}!} \tag{3.5}$$

avec :
- K : nombre de classes
- c_i : nombre d'individus dans l'intervalle i
- c_{ik} : nombre d'individus dans l'intervalle i pour la classe k

Algorithme OnLineMODL

L'algorithme OnLineMODL utilise une structure de données basée sur un tableau T d'intervalles de taille M

$$t_i < min_i, max_i, c_{i1}, c_{i2}, ..., c_{ik} >$$

où :
- min_i, max_i sont le minimum et le maximum des valeurs vues dans l'intervalle i
- $c_{i1}, c_{i2}, ..., c_{ik}$ correspondent aux comptes des individus dans l'intervalle i

Le fonctionnement de l'insertion est décrit dans l'algorithme 4, il possède deux cas :

1. si la valeur à insérer tombe dans un intervalle qui existe déjà alors celle-ci est directement intégrée dedans ;
2. si la valeur tombe entre deux intervalles alors un nouvel intervalle est créé. Si la taille maximale est dépassée suite à cette insertion alors une opération de fusion est déclenchée.

L'opération de fusion (algorithme 5) agrège deux intervalles de manière à ce que le critère MODL de notre résumé soit minimal (la somme du terme 3.5 soit minimale). Le choix des deux meilleurs intervalles à fusionner correspond à trouver les deux intervalles qui minimisent le critère après fusion. Le critère étant additif, il suffit de soustraire la somme des critères des deux intervalles avant fusion avec la valeur du critère du nouvel intervalle après fusion. La meilleure fusion est celle dont la différence est minimale. En cas d'égalité le choix se porte sur la fusion des intervalles les plus proches, c'est-à-dire pour lesquels $t_{i+1}.min - t_i.max$ est minimal.

Soit $c_{i.} = \sum_{k=1}^{K} c_{ik}$. Explicitons le critère c_{t_i} pour le tuple t_i de notre résumé. D'après la formule du coefficient binomial $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ on peut transformer la formule (3.5) en la formule suivante :

$$\begin{aligned} c_{t_i} &= log\left(\frac{(c_{i.}+K-1)!}{(K-1)!c_{i.}!}\right) + log\frac{c_{i.}!}{c_{i1}!c_{i2}!\ldots c_{iK}!} \\ &= log\left((c_{i.}+K-1)!\right) - log((K-1)!) - log(c_{i.}!) + log(c_{i.}!) \\ &\quad -log(c_{i1}!) - log(c_{i2}!) - ... - log(c_{iK}!)) \\ &= log\left((c_{i.}+K-1)!\right) - log((K-1)!) - log(c_{i1}!) - log(c_{i2}!) - ... - log(c_{iK}!) \end{aligned}$$

Ce qui donne dans le cas binaire ($K = 2$) :

$$c_{t_i} = log\left((c_{i1}+c_{i2})!\right) - log(c_{i1}!) - log(c_{i2}!)$$

Si nous prenons l'exemple de trois tuples :

$$\begin{aligned} t_1 < min_1 = 0, max_1 = 1, c_{11} = 2, c_{12} = 10 >,\ c_{t_1} = 6,75 \\ t_2 < min_2 = 2, max_2 = 5, c_{21} = 5, c_{22} = 1 >,\ c_{t_2} = 3,74 \\ t_3 < min_3 = 7, max_3 = 9, c_{31} = 8, c_{32} = 2 >,\ c_{t_3} = 6,20 \end{aligned}$$

Essayons les différentes fusions possibles et regardons les pertes en valeurs de critère :

$$\begin{aligned} p_{t_1 \cup t_2} = c_{t_1 \cup t_2} - c_{t_1} - c_{t_2} = 13,31 - 6,75 - 3,74 = 2,82 \\ p_{t_2 \cup t_3} = c_{t_2 \cup t_3} - c_{t_2} - c_{t_3} = 9,16 - 3,74 - 6,20 = -0,78 \end{aligned}$$

Le but étant de minimiser le critère, la fusion de t_2 avec t_3 donne la plus grande diminution du critère et est donc la fusion qui est choisie. Ce choix est logique car les distributions des tuples t_2 et t_3 sont plus proches que celles de t_1 et t_2.

Entrées : $T, x, classe, M$
$position \leftarrow$ `TrouverIntervalInférieurOuEgal` (T, x)
si $x \in T[position]$ **alors**
| $T[position].c[classe] \leftarrow T[position].c[classe] + 1$
sinon
| `Insérer` $(T, position, x, classe)$
fin
si `Taille` $(T) > M$ **alors**
| `RéduireOnLineMODL` (T, M)
fin

Algorithme 4 – Insertion d'un nouvel individu (OnLineMODL).

3.3.2.6 CMSClass : Count-min Sketch adapté à la classification

L'utilisation du CMS tel qu'il a été proposé par Cormode et Muthukrishnan permet de trouver le nombre d'apparitions d'une valeur dans un flux de données. Ce résumé n'est pas directement adapté aux tâches de classification car il ne stocke pas l'information sur les classes. Cependant nous l'avons adapté d'une manière similaire au résumé GK en stockant le compte c_k par classe dans

Entrées : T, M
$minLost \leftarrow \infty$
$distance \leftarrow \infty$
pour $i \leftarrow 1$ **a** $M-1$ **faire**
  $criterionAfterMerge \leftarrow$ `MODLCriterion` $(T[i], T[i+1])$
  $lost \leftarrow criterionAfterMerge-($`MODLCriterion` $(T[i])$ + `MODLCriterion` $(T[i+1]))$
  si $lost = minLost$ **alors**
    $newDistance \leftarrow T[i+1].min - T[i].max$
    si $newDistance < distance$ **alors**
      $posMerge \leftarrow i$
      $distance \leftarrow newDistance$
    fin
  fin
  si $lost < minLost$ **alors**
    $minLost \leftarrow lost$
    $pos \leftarrow i$
    $distance = T[i+1].min - T[i].max$
  fin
fin
`FusionnerTuples` $(T, pos, pos+1)$;

Algorithme 5 – Fonction `RéduireOnLineMODL` : fusion des 2 intervalles minimisant l'augmentation du critère (OnLineMODL).

le tableau de projection b. Les t tableaux b de projection contiennent maintenant l'information concernant les k classes. La mise à jour du résumé à l'arrivée d'un nouvel exemple x est illustrée dans la figure 3.5 et correspond à une incrémentation des comptes de la manière suivante :

$$\forall i = 1, \ldots, t \quad b[i, [h_i(x), c_k]] \leftarrow b[i, [h_i(x), c_k]] + 1$$

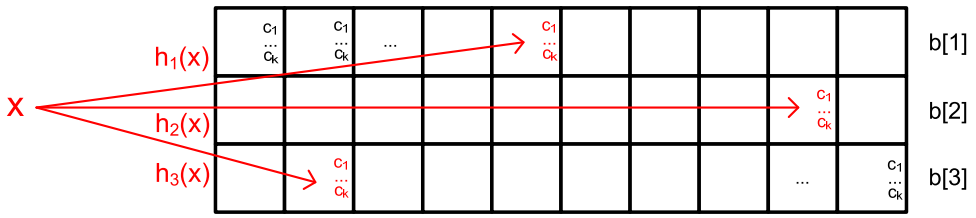


FIGURE 3.5 – Insertion d'un élément x dans un résumé CMSClass avec $t = 3$.

Le compte retourné par le CMSClass pour une valeur v est celui ayant la somme minimale des comptes de toutes les classes parmi les t tableaux b (identique au CMS d'origine sauf que l'on prend la somme des comptes pour toutes les classes au lieu du compte) :

$$\hat{c}_v = \underset{i}{argmin}(\sum_{k=1}^{K} b[i, [h_i(x), c_k]]) \tag{3.6}$$

On rappelle (voir 3.2.4) que l'utilisation des résumés par la suite ne nécessite pas de connaitre *a posteriori* les valeurs des variables. Les méthodes basées sur une projection par hachage qui ne

stockent pas ces valeurs ne posent donc pas de problème car elles permettent de retourner des comptes pour n'importe quelle valeur fournie en entrée.

Fonctionnement avec le deuxième niveau. Chaque tableau $b[i]$ est indépendant et par conséquent le groupage du niveau 2 s'applique ligne par ligne à chacun d'entre eux, in fine t groupages sont lancés. Le groupage retourné par le niveau 2 permet de savoir quelles « cellules » du tableau $b[i]$ agréger. Cette agrégation permet d'obtenir une estimation de densité plus robuste que l'utilisation directe des comptes du niveau 1. L'index i du tableau $b[i]$, utilisé par le niveau 2, est le même que pour le niveau 1, c'est-à-dire celui qui a le compte minimum pour la valeur x demandée (voir équation 3.6).

3.3.3 Deuxième niveau

Les meilleures méthodes de discrétisation/groupage sont supervisées, régularisées et sans paramètres. Parmi celles-ci la méthode MODL est performante et a l'avantage de s'appliquer à la fois sur les variables numériques (discrétisation) et catégorielles (groupage). Cette méthode est robuste dans le sens où elle n'a pas d'hypothèse sur la distribution des données et a une faible sensibilité au bruit. Cette méthode est celle sélectionnée comme deuxième niveau de notre approche et nécessite en entrée des comptes.

3.3.3.1 Discrétisation

L'approche et le critère MODL pour la discrétisation [Bou06a] a été décrit de manière détaillée lors de la présentation de la méthode OnLineMODL dans la section 3.3.2.5.

3.3.3.2 Groupage

L'approche utilisée pour le groupage [Bou05] est la même mais le critère diffère légèrement. En effet, pour les variables numériques les intervalles doivent se suivre mais pour les variables catégorielles il n'y a pas de notion d'ordre. Ceci nous donne le critère C_{group} suivant qui doit lui aussi être minimisé :

$$C_{group} = logV + \tag{3.7}$$

$$logB(V,G) + \tag{3.8}$$

$$\sum_{g=1}^{G} log\binom{c_g + K - 1}{K - 1} + \tag{3.9}$$

$$\sum_{g=1}^{G} log\frac{c_g!}{c_{g1}!c_{g2}!\ldots c_{gK}!} \tag{3.10}$$

avec comme notations supplémentaires :
- V : nombre de valeurs
- G : nombre de groupes (inconnu)
- c_g : nombre d'individus du groupe g
- c_{gk} : nombre d'individus dans le groupe g pour la classe k

– $B(V,G)$ correspond à une somme de nombre de Stirling de deuxième espèce (nombre de partitions de V valeurs en groupes non vides).

Le premier terme (3.7) correspond au choix du nombre de groupes et le second terme (3.8) au choix de la partition en groupes des valeurs explicatives. Le terme (3.9) représente le choix des distributions dans chaque groupe. Le dernier terme (3.10) exprime la probabilité d'observer ces distributions dans les données (vraisemblance).

3.3.4 Cohérence et apport de l'approche par rapport à PiD

Notre approche est similaire à la méthode PiD de Gama mais nos deux niveaux sont basés sur des méthodes utilisant des comptes. Notre premier niveau contrôle son erreur et fonctionne à mémoire constante ce qui n'est pas le cas de la méthode PiD. De plus notre approche s'applique aussi bien aux variables numériques que catégorielles alors que PiD n'est prévu que pour les variables numériques.

La mémoire consommée par le premier niveau de PiD dépend de la distribution des données. Si les données arrivent suffisamment bien mélangées et si elles ont une distribution uniforme alors la consommation mémoire sera constante. Dans les autres cas, des intervalles supplémentaires vont être ajoutés. En effet la méthode découpe en deux intervalles un intervalle si elle évalue que celui-ci est trop rempli. Cette évaluation se fait à l'aide d'un paramètre α qui correspond à la proportion $(n \times \alpha)$ maximale d'exemples qu'un intervalle peut contenir. Un deuxième paramètre : *step* définit la distance des frontières entre les intervalles. Si le minimum et le maximum sont connus on peut trouver le paramètre *step*, sinon on en fixe un par défaut. Si la valeur par défaut est trop grande alors on va se retrouver avec de nombreux intervalles fortement peuplés qui devront être redécoupés. Ceci entraine une augmentation du nombre d'intervalles et donc de la consommation mémoire. Notre méthode basée sur les quantiles ne souffre pas de ce genre de problèmes car le nombre d'intervalles est constant et les frontières sont définies par des valeurs du flux qui sont mises à jour si nécessaire. De plus le résumé GK et CMS conservent leurs garanties quel que soit la distribution et l'ordre d'arrivée des données.

3.4 Expérimentations sur la discrétisation

3.4.1 Plan d'expérience

Le but de cette section est d'évaluer la qualité de notre approche dans le cadre de problèmes artificiels et d'observer son comportement dans les cas-limites. Les données utilisées proviennent de générateurs de données basés sur différents motifs. Cette étude s'intéresse tout d'abord aux erreurs introduites par le premier niveau non supervisé. On s'intéresse ensuite au deuxième niveau supervisé pour évaluer sa capacité à retrouver les motifs issus des générateurs en se basant sur les comptes obtenus par le premier niveau. On étudiera par la même occasion la robustesse de l'approche en ajoutant 10% de bruit aux données.

Les données synthétiques générées sont des problèmes de classification binaire. Les expérimentations sont lancées 1000 fois afin d'obtenir une moyenne et un écart type sur les résultats. L'espace de valeurs des paramètres étudiés est :

n	taille des données à discrétiser : 1 million d' exemples
$\|S\|$	taille du résumé en octets : 192, 384, 576, 768, 960, 1536, 3072, 9216
bruit	quantité de bruit ajouté dans les données : 0%, 10%
J	nombre d'expérimentations : 1000

3.4.1.1 Lois utilisées par les générateurs

Les valeurs de x sont tirées aléatoirement, soit dans un intervalle pour les lois en créneaux, soit dans $\mathbb{R}$ pour la loi gaussienne. La probabilité de la classe associée à x est définie selon les lois suivantes :

Loi en créneaux Le motif « créneaux » est repris des expérimentations menées sur la discrétisation MODL [Bou06a]. Le motif étudié comporte 15 intervalles définis pour $x \in [a, b[$ avec $a = 0$ et $b = 15$ par exemple, et est illustré graphiquement par la figure 3.6. Chaque intervalle contient uniquement des exemples d'une seule et même classe.

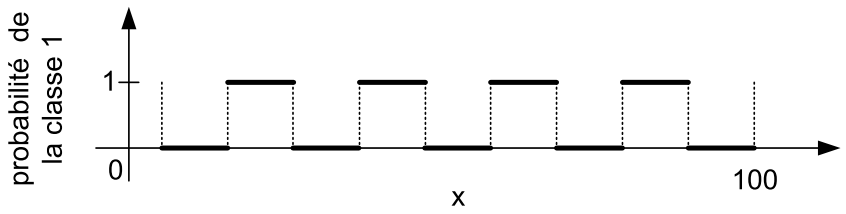


FIGURE 3.6 – Exemple de motif en créneaux sur la variable x.

Loi en créneaux avec plateau Cette loi est définie pour $x \in [a, b]$. Ce motif diffère du premier motif en créneaux car l'espace ne contient pas que des créneaux répartis uniformément : une partie des créneaux est remplacé par un plateau. Le but de ce motif est d'étudier le comportement des méthodes supervisées et de voir si elles sont capables d'avoir une utilisation « intelligente » de la mémoire. En effet on peut s'attendre à ce qu'elles puissent être capable d'utiliser peu de mémoire pour le plateau et de réaffecter la mémoire pour les parties plus complexe du motif (les créneaux). Le motif choisi est composé de 16 intervalles : 8 intervalles suivi d'un plateau puis 7 intervalles comme le montre la figure 3.7.

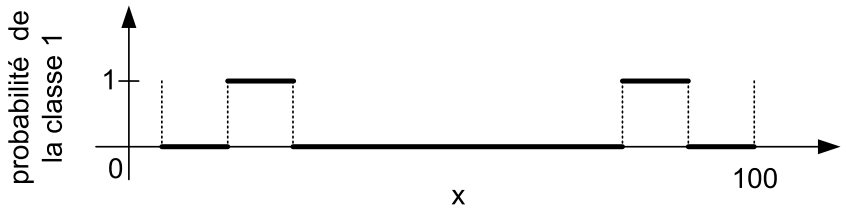


FIGURE 3.7 – Exemple de motif en créneaux avec un plateau sur la variable x.

Loi gaussienne Pour ce motif la distribution de la variable explicative des exemples de la classe 0 suit une gaussienne de paramètre $(0, 1)$ et ceux de la classe 1 une gaussienne de paramètre $(2, 1)$. La figure 3.8 illustre cette distribution. Le but de ce motif est d'étudier le comportement des méthodes quand les classes ne sont pas disctinctes mais mélangées.

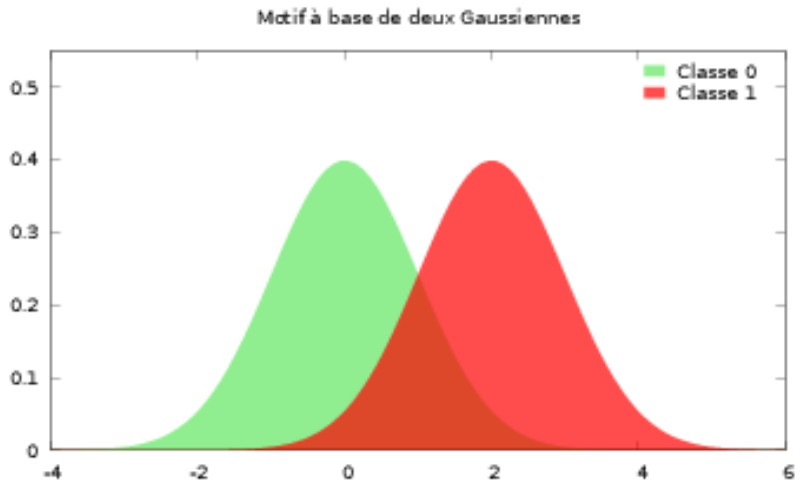


FIGURE 3.8 – Motif basé sur deux gaussiennes de paramètres $(0, 1)$ et $(2, 1)$.

3.4.1.2 Algorithmes comparés

La comparaison des algorithmes se fait entre trois nouvelles méthodes que nous proposons et deux méthodes de l'état de l'art.

1. Nouvelles méthodes :
   (a) GKClass : algorithme GK adapté à la classification supervisée (section 3.3.2.2)
   (b) GKClassAcc : version « accélérée » de GKClass (section 3.3.2.3)
   (c) OnLineMODL : discrétisation en-ligne basée sur le critère MODL (section 3.3.2.5)
2. Méthodes de l'état de l'art :
   (a) cPiD : méthode PiD de Gama modifiée pour avoir une consommation mémoire constante (section 3.3.2.4)
   (b) Gaussienne (section 3.1.2.2)

Mémoire consommée par les différents algorithmes Nous nous plaçons dans le cadre d'un problème de classification à deux classes. Le tableau 3.4 donne en fonction de la mémoire consommée le nombre de tuples que l'algorithme peut utiliser. La méthode basée sur une gaussienne consomme toujours 40 octets par classe et n'apparait donc pas dans ce tableau. Ces 40 octets proviennent du stockage de cinq numériques : $< \mu, \sigma, n, min, max >$; soit dans le cas d'un résumé à deux classes 80 octets. La méthode GKClass nécessite 4 numériques soit 32 octets : $< v_i, \Delta_i, c_{i0}, c_{i1} >$. Pour la méthode OnLineMODL il faut aussi 4 numériques : $< min_i, max_i, c_{i0}, c_{i1} >$. PiD ne nécessite que 3 numériques : $< v_i, c_{i0}, c_{i1} >$ soit 24 octets par tuple.

3.4.1.3 Bruit

Les expérimentations sont menées dans un premier temps avec des motifs non bruités puis avec un niveau de bruit de 10%. Le bruit ajouté est un bruit de classe qui consiste à changer, avec une probabilité de 0,1, l'étiquette d'un exemple par une autre étiquette que celle du motif à apprendre.

Taille mémoire en octets	GKClass / OnLineMODL	PiD
192	6	8
384	12	16
576	18	24
768	24	32
960	30	40
1536	48	64
3072	96	128
9216	288	384

TABLE 3.4 – Nombre de tuples versus quantité de mémoire nécessaire selon les méthodes pour un résumé à deux classes.

3.4.2 Résultats pour le premier niveau

Le résumé de premier niveau introduit des erreurs par rapport aux données du flux. Une première expérimentation est lancée afin d'étudier selon les résumés et la mémoire utilisée la quantité d'erreur introduite.

3.4.2.1 Mesure utilisée pour la comparaison

L'objectif de notre premier niveau de résumé est d'être capable pour une valeur x donnée d'obtenir la probabilités des classes. Ces expérimentations ont pour but de mesurer la différence entre les densités obtenues par nos résumés et les densités réelles des motifs. Étant donnés les motifs en créneaux utilisés, la densité est soit de 0 ou 1. Dans ce cas, comparer par intervalle la classe prédite par le résumé à la classe de densité 1 revient à comparer les estimations de densité. Les classes du motif basé sur des gaussiennes étant suffisamment séparées, ce protocole lui sera aussi appliqué.

On construit donc un classifieur univarié à partir des comptes fournis par le résumé en prenant la classe majoritaire par intervalle. La performance du résumé est évalué au travers la taux d'erreur de ce classifieur sur un jeu de test. Soit x un exemple du jeu de test appartenant à l'intervalle i du résumé et k_x sa classe réelle. L'erreur pour cet exemple est :

$$erreur(x) = \begin{cases} 0 & \text{si } \underset{k}{argmax}(c_{ik}) = k_x \\ 1 & sinon \end{cases}$$

Le nombre d'erreurs est calculé sur un jeu de données de test d'un million d'exemples et nous utilisons le taux d'erreur comme mesure de comparaison.

3.4.2.2 Discussion des résultats

Les résultats sur le comportement des différentes méthodes de discrétisation en-ligne sont présentés par générateurs. Pour chaque générateur les courbes présentent le taux d'erreurs en fonction de la mémoire utilisée par les différents résumés. La figure 3.9 présente les résultats pour des données générées sans bruit et la figure 3.10 les mêmes résultats pour des données bruitées à 10%. Les expérimentations ont été lancées 1000 fois afin d'obtenir une moyenne et un écart type sur les courbes.

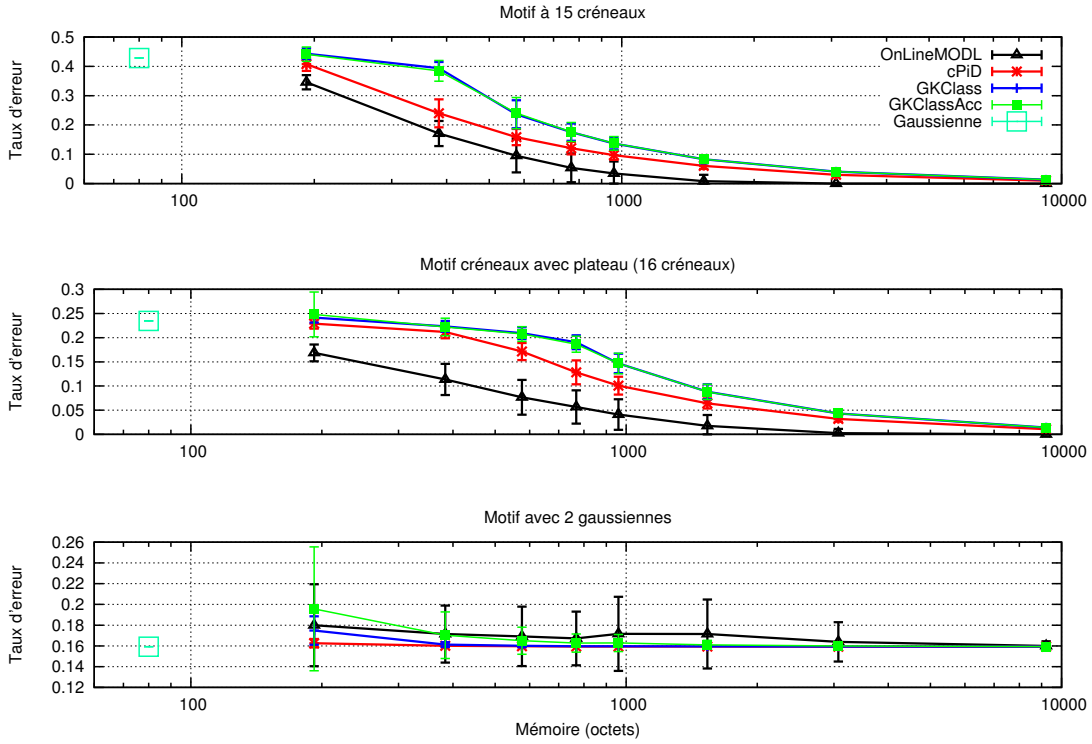


FIGURE 3.9 – Comparaison des taux d'erreurs des différents résumés en fonction de la mémoire utilisée pour 1 million d'insertion.

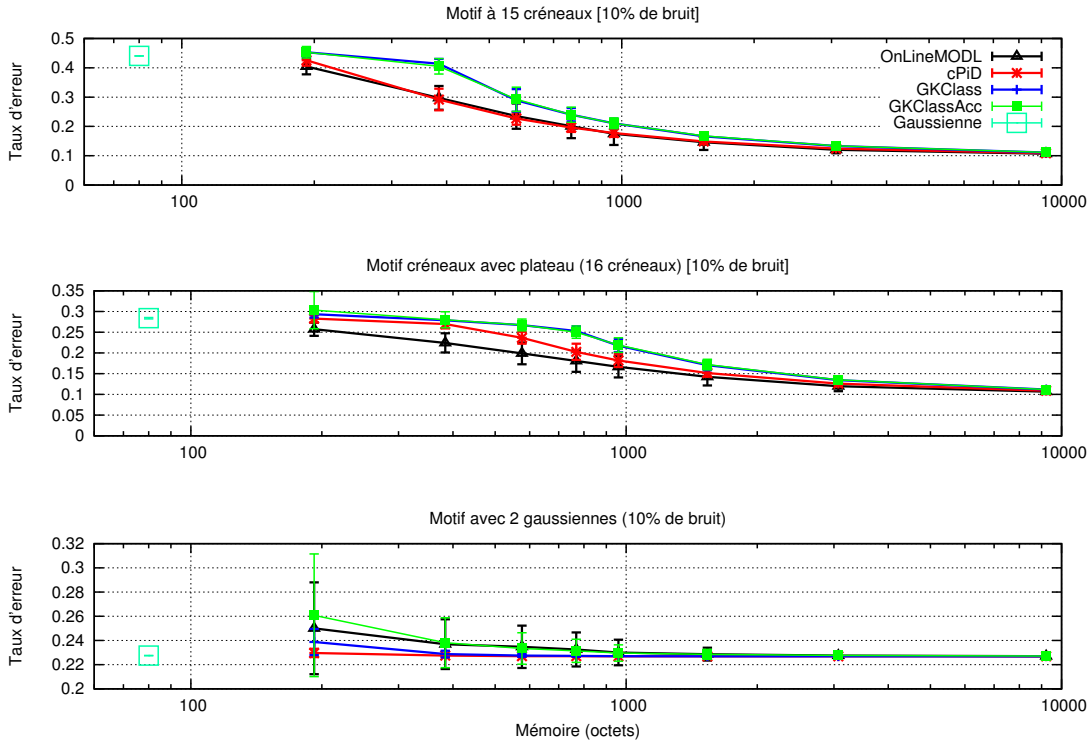


FIGURE 3.10 – Comparaison des taux d'erreurs des différents résumés en fonction de la mémoire utilisée pour 1 million d'insertion avec 10% de bruit.

Le résumé basé sur une gaussienne bien qu'étant la méthode consommant le moins de mémoire n'est pas du tout adaptée aux motifs en créneaux. Par contre elle fonctionne très bien sur un motif basé sur une gaussienne. Ces résultats n'ont rien de surprenant, si on a un *a priori* sur la distribution des données nous informant que celles-ci suivent une loi gaussienne, alors ce résumé est le plus adapté.

La méthode OnLineMODL se distingue particulièrement avec peu de mémoire car elle est supervisée et est donc capable de répartir sa mémoire pour essayer d'apprendre le motif au mieux. Le motif avec un plateau est le meilleur exemple de ce comportement. Sur ce motif la méthode utilise peu de mémoire pour le plateau et essaye de capturer le motif en créneaux présent aux extrémités avec le maximum de mémoire. Sa variance est globalement supérieure aux autres méthodes. Cela est dû à l'impossibilité de pouvoir redécouper un intervalle. En effet si deux (ou plus) créneaux ont été fusionnés alors on ne pourra plus jamais les distinguer. Par conséquent dès que ce cas se produit on obtient une forte augmentation de l'erreur à comparer à une discrétisation quasi parfaite (voir section 3.4.2.3), d'où une très grande variance. Malgré tout cette méthode reste meilleure même dans le pire des cas sur les données en créneaux. Sur les données du motif gaussien ses résultats sont légèrement moins bons que ceux des autres méthodes.

Les méthodes GKClass et GKClassAcc ont des performances similaires sur les jeux de données de notre expérimentation. Elles se comportent un peu mieux que la méthode OnLineMODL sur les données issues de la gaussienne mais moins bien sur les créneaux. Ces méthodes ne sont pas supervisées et contrôlent leur erreur ce qui les rend plus stables de manière générale.

La méthode cPiD issue de la méthode PiD de Gama modifiée pour avoir une consommation mémoire constante se comporte bien sur les trois jeux de données de l'expérimentation. Elle est meilleure que les méthodes à base de résumés de quantiles GKClass. Une partie de cette différence est due au fait qu'elle ne stocke pas d'information concernant l'erreur contrairement aux résumés GK. Il en résulte qu'elle peut conserver plus de tuples pour une mémoire équivalente à GKClass comme l'indique le tableau 3.4. Afin d'étudier si cette différence du taux d'erreur est directement lié aux nombre de tuples on réalise une comparaison avec des résumés utilisant 768 octets. Ces résultats sont présentés dans le tableau 3.5. On constate que pour un même consommation mémoire le résumé cPiD est meilleur. Cette différence est sans doute due à l'hypothèse de distribution uniforme des données de PiD qui est utilisée pour diviser un intervalle en deux. Les méthodes GKClass n'ont pas d'*a priori* et contrôlent l'erreur en prenant comme hypothèse le pire cas.

La méthode OnLineMODL est celle qu'il est la plus sensible à l'ajout de bruit mais reste meilleure si les données contiennent des zones avec des densités conditionnelles constantes (comme pour le motif avec un plateau). Cela est due à son incapacité à rediviser un intervalle et à sa méthode de construction supervisée qui agrége rapidement deux intervalles bruités. Par contre les autres méthodes non supervisées sont peu sensibles à l'ajout de bruit.

Avec 10 Ko de mémoire toutes les méthodes (sauf celle basée sur la gaussienne) ont un très faible taux d'erreurs. Ces 10 Ko paraissent faibles étant donné les quantités de mémoire utilisées actuellement (plusieurs Go). Cependant il faut compter un résumé par variable et la mémoire consommée par les résumés reste très importante dans bien des cas :

- Jeu de données possédant des milliers de variables.
- Environnement à mémoire limitée comme les capteurs, systèmes embarqués, équipements réseaux...
- Modèle multivarié nécessitant des résumés par partie de l'espace. Les arbres de Hoeffding (voir section 2.3.2) en sont un très bon exemple car ils utilisent un résumé par variable et par feuille.

	Créneaux	Créneaux avec plateau	Gaussienne
GKClass	17,53% ± 2,95%	19,06% ± 1,44%	15,96% ± 0,06%
GKClassAcc	17,60% ± 3,25%	18,64% ± 1,65%	16,28% ± 0,87%
OnLineMODL	5,36% ± 4,92%	5,66% ± 3,45%	16,72% ± 2,59%
cPiD	12,09% ± 2,13%	12,82% ± 2,47%	15,94% ± 0,04%
cPiD (24 tuples)	15,86% ± 2,76%	17,13% ± 1,78%	15,96% ± 0,07%

TABLE 3.5 – Comparaison des taux d'erreur des résumés avec 768 octets après 1 million d'insertion (bruit de 0%)
(cPiD avec 24 tuples est présent à titre d'information car il possède le même nombre de tuples que les autres méthodes).

3.4.2.3 Créneaux retrouvés par le premier niveau

Afin de mieux visualiser les types d'erreurs introduites dans les résumés lors de la discrétisation du premier niveau du flux des expériences sur un motif « Créneaux avec plateau » ont été réalisées. Le motif est constitué de 6 créneaux avec un plateau au milieu des créneaux. Un million d'exemples ont été insérés dans le résumé puis les frontières entre les tuples et l'estimation de la densité ont été tracées sur la figure 3.11. Comme on peut s'y attendre le résumé par une gaussienne n'est pas capable de capturer ce genre de motif. Les méthodes cPiD et GKClass n'arrivent pas exactement à retrouver les bornes du motif car elles ne sont pas supervisées. Elles essayent simplement de mieux répartir les comptes cumulés (somme des comptes de la classe 0 et de la classe 1) par intervalle. Quant à la méthode supervisée OnLineMODL, elle trouve la plupart du temps parfaitement les frontières. Il lui arrive cependant de rater des frontières et de fusionner un créneau entier comme dans la dernière sous-figure. Dans ce cas les erreurs sont de suite plus importantes que les méthodes cPiD ou GKClass qui ont leurs erreurs réparties sur tout le motif. Ce comportement explique la plus grande variance de la méthode OnLineMODL.

3.4.3 Résultats pour le deuxième niveau

3.4.3.1 Créneaux retrouvés par la discrétisation MODL au niveau 2 : présentation

Cette sous-section présente les résultats en termes de créneaux retrouvés après le passage par les deux niveaux. Le but est d'étudier les différentes méthodes du premier niveau et d'observer en fonction de la mémoire consommée combien de motifs sont retrouvés par la discrétisation MODL du second niveau. La discrétisation MODL est reconnue comme robuste (pas d'hypothèse sur la distribution des données et faible sensibilité au bruit) et peu sensible à la « surdiscrétisation ». Par conséquent la différence entre le nombre de motifs retrouvés et le nombre de motifs réels est un bon indicateur de l'adéquation du résumé du premier niveau avec le second. Les données utilisées proviennent des mêmes générateurs qu'auparavant. Les courbes de la figure 3.12 présentent les résultats pour des données générées sans bruit et la figure 3.13 les mêmes résultats pour des données bruitées à 10%.

3.4.3.2 Discussion des résultats

La discussion suivante s'intéresse plus particulièrement aux motifs « Créneaux » et « Créneaux avec plateau » car le nombre de motifs à retrouver est connu. Pour le motif « gaussien » le nombre de motif retrouvé n'est pas un bon critère pour évaluer la qualité de la discrétisation car on ne connait pas la valeur idéale. On pourrait pour cela lancer la discrétisation MODL sur toutes les

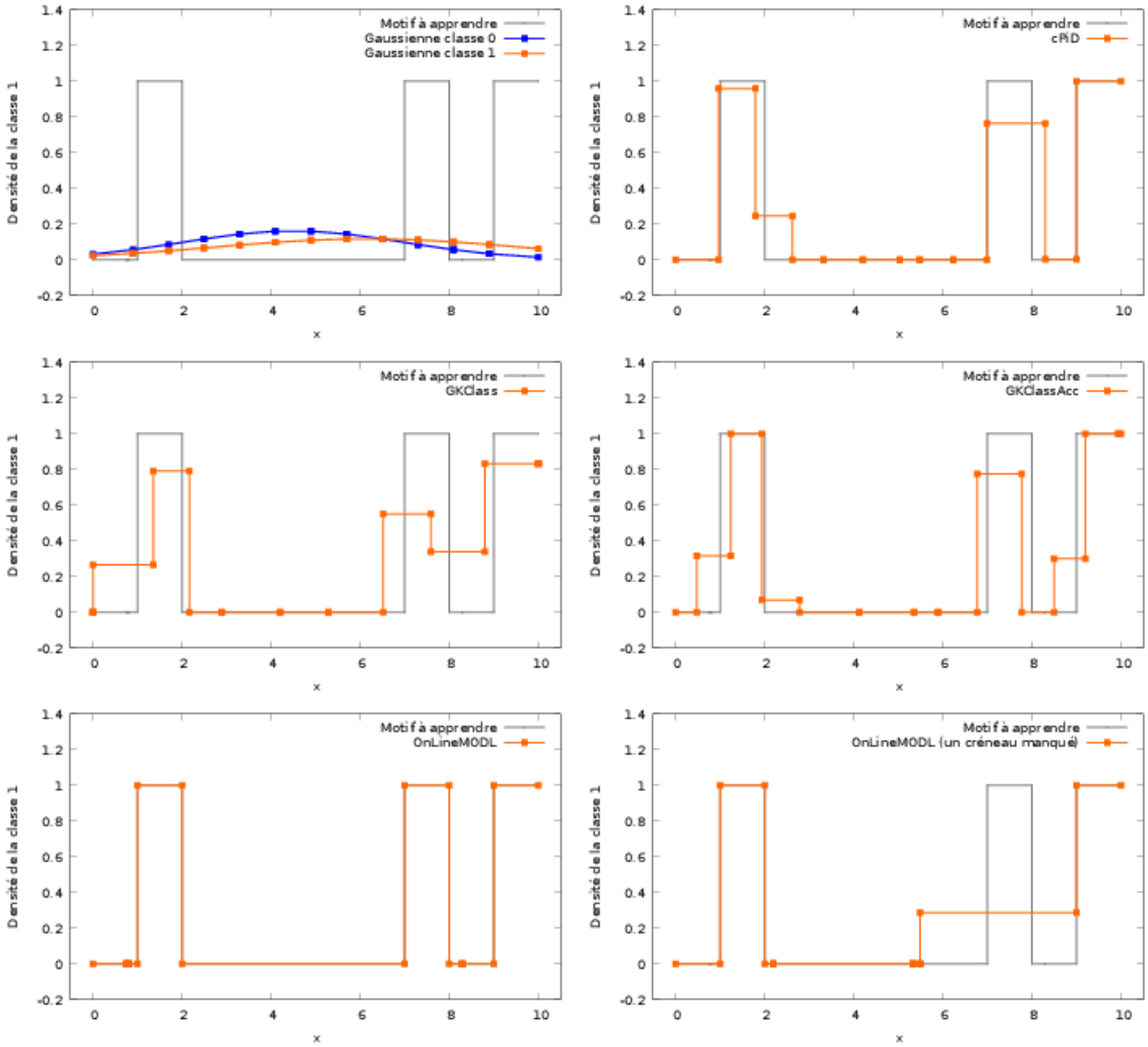


FIGURE 3.11 – Comparaison sur un motif « créneaux avec plateau » des différents résumés étudiés (12 tuples). Le résumé OnLineMODL est présent deux fois pour montrer le type d'erreur qu'il introduit quand il ne retrouve pas le motif original.

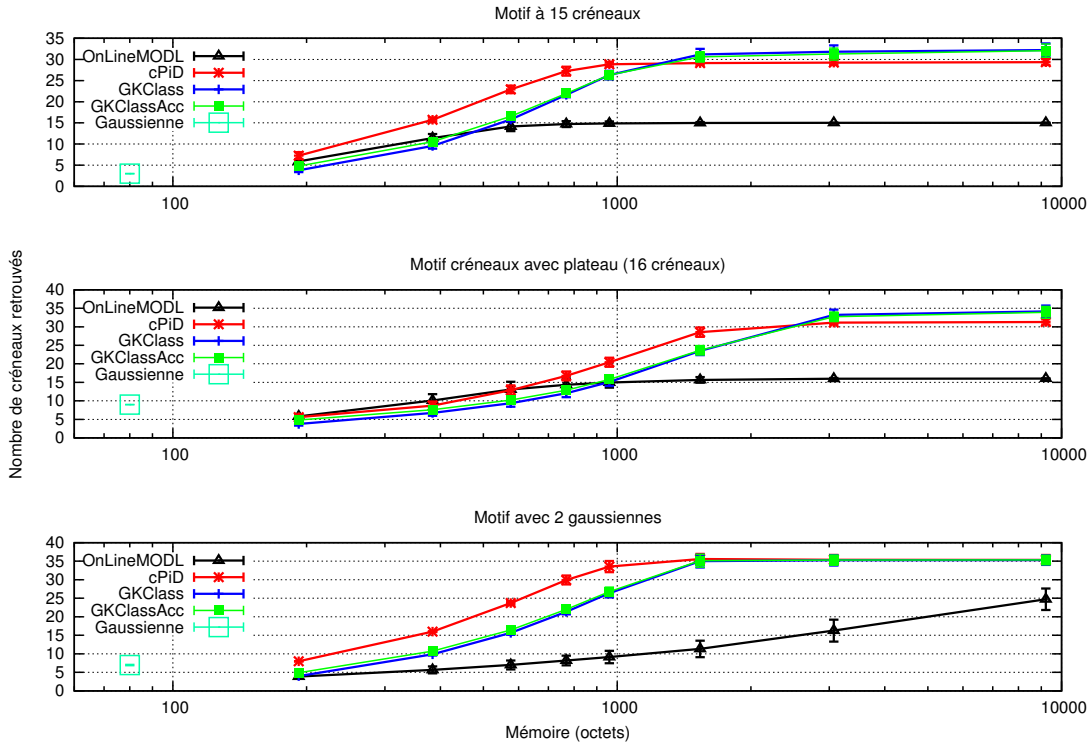


FIGURE 3.12 – Comparaison des taux d'erreurs des différents résumés en fonction de la mémoire utilisée pour 1 millions d'insertion.

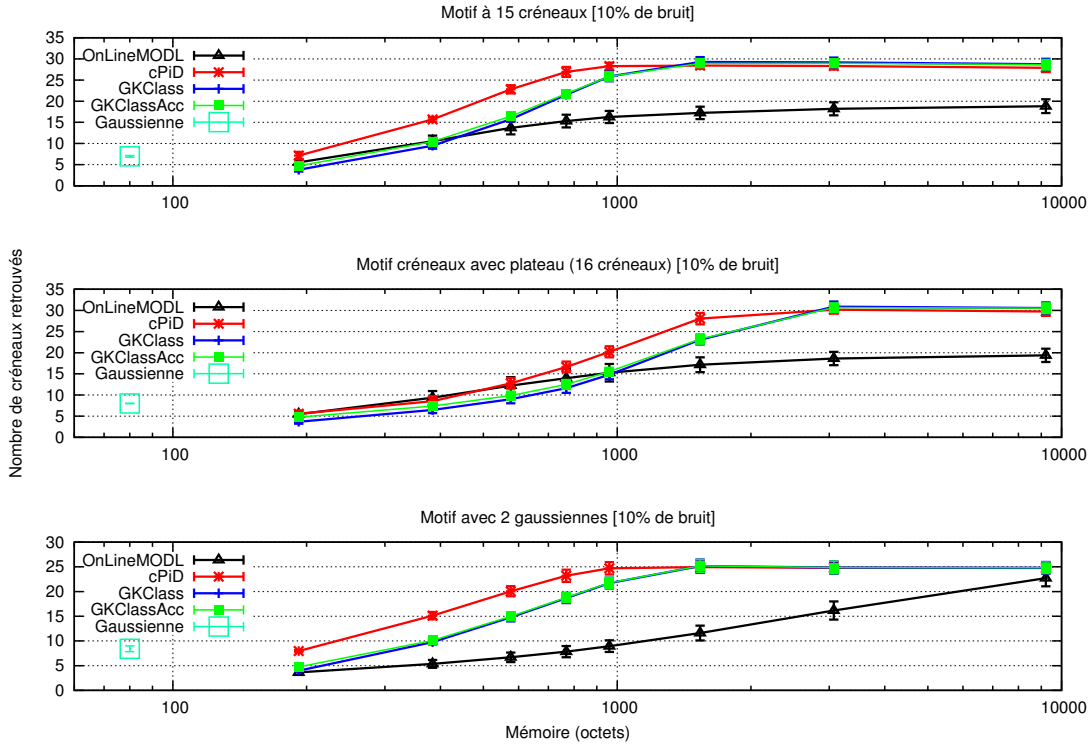


FIGURE 3.13 – Comparaison des taux d'erreurs des différents résumés en fonction de la mémoire utilisée pour 1 millions d'insertion avec 10% de bruit.

données mais même dans ce cas le nombre de créneaux retrouvés serait moins interprétable que pour les motifs en créneaux. Notre discussion ne s'intéresse donc par la suite qu'aux deux motifs ayant des créneaux.

De manière générale le résumé OnLineMODL est celui qui se rapproche le plus du nombre de créneaux à retrouver. Cette méthode est supervisée et comme expliqué dans la section 3.4.2.3 OnLineMODL arrive très souvent à retrouver parfaitement les frontières du motif à apprendre. Par conséquent on obtient un premier niveau avec des créneaux purs. Ces créneaux purs sont fusionnés par la discrétisation MODL du niveau 2 s'ils sont adjacents et de même classe car ils sont considérés comme ayant la même distribution. Le motif est donc parfaitement retrouvé.

Pour les autres méthodes : cPiD, GKClass et GKClassAcc, le nombre de créneaux retrouvés est supérieur au nombre du motif car ces méthodes ne sont pas supervisées. Ces méthodes ne cherchent pas à trouver des créneaux purs mais à répartir au mieux le nombre d'observations par intervalle. Les points de coupure du résumé ne correspondent donc pas à ceux du motif comme on peut l'observer sur la figure 3.11. On obtient des intervalles non purs que la discrétisation MODL du niveau 2 considère de deux manières : (i) la distribution est suffisamment similaire à celle d'un intervalle adjacent et les deux intervalles sont fusionnés ; (ii) la distribution est considérée différente et cet intervalle est conservé comme intervalle à part entière. Le cas (ii) entraîne une augmentation du nombre d'intervalles par rapport au motif initial et explique que dès qu'il y a assez de tuples (plus de 1000 octets de résumé dans le cas des motifs présentés ici) on observe approximativement un doublement du nombre de motifs retrouvés.

Globalement l'ajout de bruit change peu le nombre de créneaux retrouvés. Les conséquences sont différentes selon les méthodes du premier niveau. La méthode OnLineMODL retrouve plus de créneaux car le bruit fait apparaitre de nouveaux créneaux dans le niveau 1. Le niveau 2 juge ceux-ci suffisamment différents des créneaux adjacents et donc ne les fusionnent pas. Pour les autres méthodes l'effet inverse est observé. Ces autres méthodes n'étant pas supervisées leurs intervalles sont déjà bruités quand les motifs ne le sont pas. L'ajout de bruit rend certains créneaux plus « semblables » et ceux-ci sont donc fusionnés par la discrétisation MODL du deuxième niveau. Par conséquent le nombre de créneaux retrouvés est plus faible pour ces méthode non supervisées.

On observe que la méthode cPiD trouve des créneaux plus tôt que les méthodes GKClass et GKClassAcc. Cette différence s'explique par la plus faible consommation mémoire par tuple de la méthode cPiD qui lui permet à mémoire équivalente de conserver plus de tuples comme l'indique le tableau 3.4.

3.5 Expérimentations sur le groupage

3.5.1 Plan d'expérience

Les expérimentations sur la discrétisation et le groupage ne peuvent être identiques car les motifs à apprendre dans les deux cas sont différents. En effet la discrétisation porte sur une variable continue et les motifs à apprendre des jeux de tests utilisent cette continuité : créneaux, gaussienne... alors que pour les variables catégorielles les valeurs sont discrètes. Les expérimentations pour les variables catégorielles consistent à évaluer les erreurs dues aux estimations des comptes du niveau 1 et à présenter le nombre de groupes retrouvés par le niveau 2. Les expérimentations sont lancées 1000 fois afin d'obtenir une moyenne et un écart type sur les résultats.

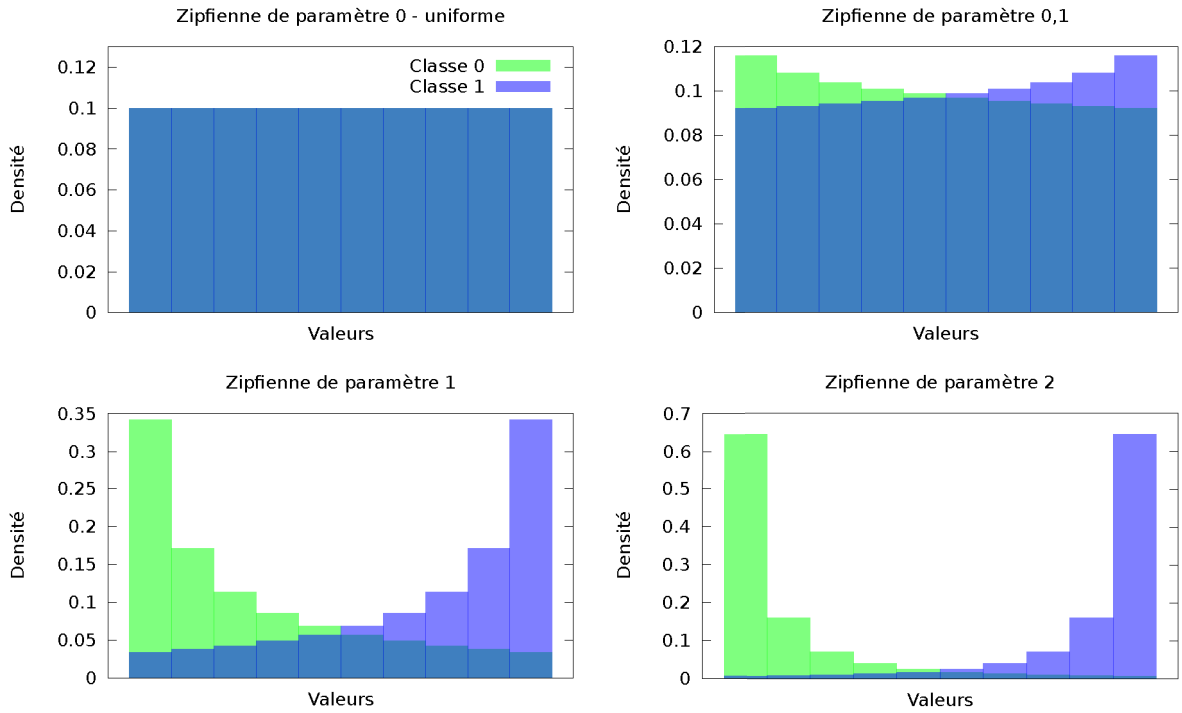


FIGURE 3.14 – Visualisation des densités pour différentes distributions zipfiennes avec deux classes et 10 valeurs distinctes.

n	taille des données à grouper : 1 million d' exemples
$\|S\|$	taille du résumé en octets : de 10 octets à 20 000 octets
bruit	quantité de bruit ajouté dans les données : 0%, 10%
V	nombre de valeurs différentes : 1000
J	nombre d'expérimentations : 1000

3.5.1.1 Générateurs

Pour les données catégorielles, un générateur zipfien de données est utilisé avec des valeurs allant de 0,1 (quasi-uniforme) à 2 (très déséquilibré). Nous n'étudions pas une zipfienne de paramètre 0 (uniforme) car ses densités sont toutes identiques et on obtient donc un taux d'erreurs de 0% pour le premier niveau et un seul groupe pour le deuxième niveau. Une zipfienne par classe est utilisée et les variables catégorielles ont chacune 1000 valeurs différentes. Les différentes valeurs de paramètre de la zipfienne que nous allons expérimenter sont de 0,1 - 1 et 2. La figure 3.14 illustre les différentes densités obtenues pour ces valeurs dans le cas d'une variable catégorielle à 10 valeurs différentes (contre 1000 dans les expérimentations).

3.5.1.2 Algorithmes comparés

L'algorithme de niveau 1 de notre méthode à deux niveaux que nous expérimentons pour le comptage est le Count-min Sketch adapté pour la classification (CMSClass). Le Count Sketch aurait aussi pu être utilisé mais étant donné qu'il a un comportement assez similaire (projection des valeurs) mais des garanties moins intéressantes nous avons choisi de n'étudier que le CMSClass. Les expérimentations sont lancées avec un nombre de fonction de hachage t de 1, 2, 3 ou 4 et

une taille b du tableau de projection allant de 10 à 5 000 entiers. La consommation mémoire est proportionnelle à $t \times b$.

3.5.2 Résultats pour le premier niveau

3.5.2.1 Mesure utilisée pour la comparaison

La mesure utilisée est la même que pour la discrétisation (voir 3.4.2.1). Cette mesure permet de savoir si les résumés sont capables d'identifier la classe majoritaire pour les différentes valeurs d'une variable catégorielle. Les mêmes valeurs pour les variables catégorielles sont utilisées en apprentissage et en test. Soit x un exemple du jeu de test appartenant au groupe v du résumé et k_x sa classe réelle. L'erreur pour un cet exemple est :

$$erreur(x) = \begin{cases} 0 & argmax_k(c_{vk}) = k_x \\ 1 & sinon \end{cases}$$

3.5.2.2 Résultats

Les résultats sont présentés de manière synthétique par les courbes de la figure 3.15. Cette figure contient 3 sous-figures correspondantes aux 3 valeurs (0,1 - 1 - 2) du générateur de données zipfien. Ces courbes tracent la moyenne des erreurs en fonction de la taille du tableau de projection b (de 10 à 5000) et ce pour un nombre de fonctions de hachage allant de 1 à 4. La figure 3.16 est identique à la figure précédente mais les données ont été bruitées à 10%.

De manière générale on observe que le CMSClass fonctionne mieux sur des données « déséquilibrées » (paramètre élevé pour la zipfienne) et avec plusieurs fonctions de hachage.

Sur les données réparties quasi uniformément (zipfienne de paramètre 0,1 avec un mélange initial de 46,3%), le CMSClass fonctionne moins bien. Ce résultat était prévisible car le CMS est prévu pour trouver les comptes les plus importants. Si les données sont relativement équiréparties la projection par hachage puis la recherche d'un minimum qui permettent d'isoler les « grands comptes » sont moins efficaces. Si on a comme *a priori* que les données sont toutes équiréparties l'utilisation d'un résumé comme le CMSClass est inutile et conserver une moyenne consommera moins de mémoire et sera plus performant. Plus la répartition est déséquilibrée (augmentation du paramètre de la zipfienne) meilleure est la convergence du résumé car les différentes projections permettent d'isoler efficacement les grands comptes et limitent les collisions [CM05].

La taille b du tableau de projection influence grandement le taux d'erreur car il permet de limiter les collisions. L'augmentation du nombre de fonctions de hachage fait aussi fortement diminuer le taux d'erreur car il y a peu de chance que deux fonctions différentes produisent la même collision. L'intérêt le plus important de l'augmentation du nombre de fonctions de hachage est surtout qu'il diminue très fortement (voir 3.3.2.6) la probabilité d'erreur sur un compte. Ceci se traduit directement par une plus faible variance des résultats et s'observe particulièrement bien sur les courbes pour la zipfienne de paramètre 2.

L'ajout de 10% de bruit n'influence que très peu les résultats pour les données équiréparties (zipfienne de paramètre 0,1) car les données sont déjà très mélangées. Pour des données moins bien réparties (paramètre 1 et 2) le CMSClass est robuste et la convergence vers le taux d'erreurs minimal est identique au cas des données sans bruit.

Les résultats obtenus pour ce premier niveau correspondent à ceux auxquels on pouvait s'attendre. L'amélioration des résultats est proportionnelle à l'augmentation à la taille 'b' du tableau de projection et au nombre de fonctions de hachage. Les données les plus disproportionnées sont

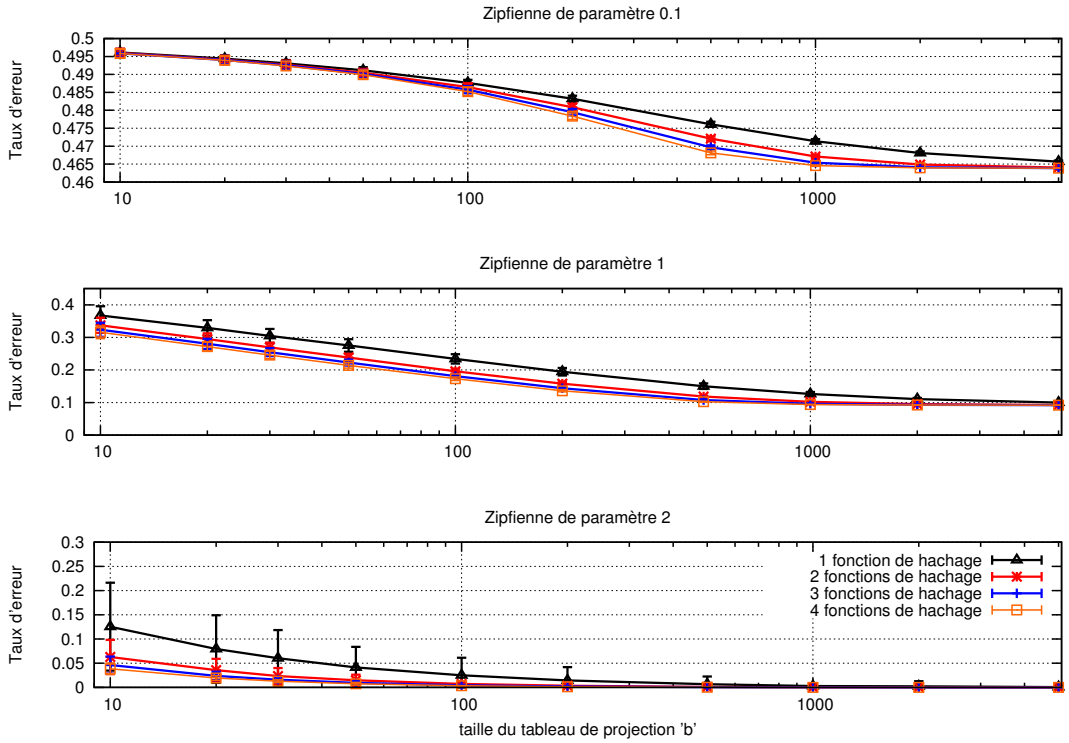


FIGURE 3.15 – Comparaison des moyennes des taux d'erreurs pour le résumé CMSClass après 1 millions d'insertion avec 'b' allant de 10 à 5000.

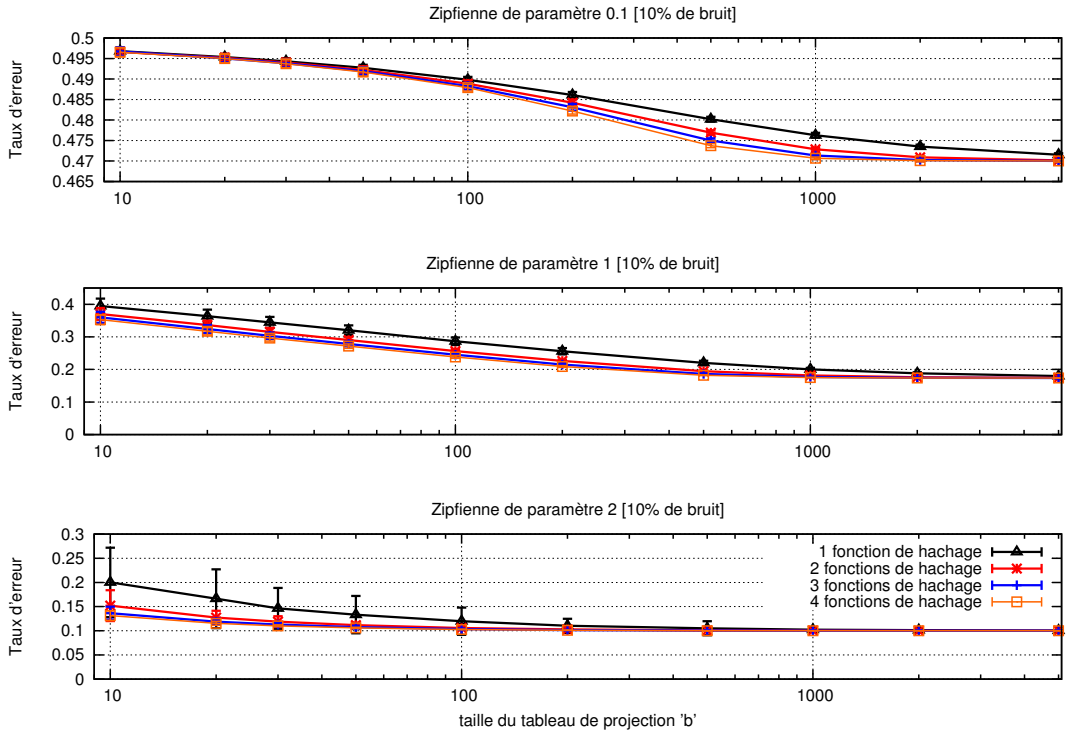


FIGURE 3.16 – Comparaison des moyennes des taux d'erreurs pour le résumé CMSClass après 1 millions d'insertion avec 'b' allant de 10 à 5000 après ajout de 10% de bruit.

les plus facilement résumables par le CMS. Nos résultats se placent dans le cadre d'une utilisation en classification supervisée du CMS mais d'autres études basées simplement sur les comptes des valeurs ayant les plus grands comptes (« heavy hitters ») existent aussi [CH08].

3.5.3 Résultats pour le deuxième niveau

3.5.3.1 Groupes retrouvés par le groupage MODL au niveau 2

Cette sous-section présente les résultats en termes de groupes retrouvés après le passage par les deux niveaux. Le but est d'étudier le nombre de groupes retrouvés par le groupage MODL du second niveau. Les données utilisées proviennent des mêmes générateurs qu'auparavant. Les courbes de la figure 3.12 présentent les résultats pour des données générées sans bruit et la figure 3.13 les mêmes résultats pour des données bruitées à 10%.

3.5.3.2 Discussion des résultats

Notre deuxième niveau retrouve confirme du groupage MODL qui est d'éviter de faire presque autant de groupes que de valeurs ayant des densités différentes. Bien des méthodes supervisées ont tendance à voir leur nombre de groupes augmenter lorsque le nombre de valeurs différentes augmente [Bou05].

Le nombre de fonctions de hachage n'influence pas le nombre de groupes retrouvés mais la variance sur le nombre moyen de groupes retrouvés est plus faible si le nombre de fonctions de hachage est plus important. Cela est dû à l'application du groupage non pas en une fois à tout le résumé CMSClass mais t fois : une fois par tableau b de projection (voir 3.3.2.6). Ceci explique par la même occasion que le nombre de fonctions de hachage n'ait pas d'influence sur le nombre de groupes retrouvés.

L'ajout de bruit dans les données a pour conséquence de diminuer le nombre de groupes retrouvés. Ce comportement est tout à fait normal car le bruit rend indistinguable une partie des groupes qui pouvaient être distingués auparavant.

On observe que le résumé CMSClass avec une taille de tableau b de projection supérieure à 1000 est très proche du nombre de groupes retrouvés par le groupage MODL appliqué sur toutes les données sans qu'elles soient résumées. Ceci confirme l'intérêt d'utiliser une méthode à deux niveaux afin de pouvoir utiliser le groupage MODL sur un résumé de bien plus faible taille que le flux de données. Notre approche à deux niveaux permet à la fois de travailler en-ligne et d'être capable d'utiliser une méthode performante hors-ligne.

3.6 Conclusion

Ce chapitre présente notre méthode à deux niveaux utilisant des comptes. Elle s'inspire de la méthode PiD de Gama mais propose des choix différents pour les deux niveaux. Ces choix permettent une utilisation du résumé aussi bien sur des variables numériques que catégorielles.

Ce chapitre débute par un état de l'art des méthodes de résumé pour les variables numériques et catégorielles qui permet d'identifier les meilleures méthodes disponibles pour nos deux niveaux.

Pour le premier niveau numérique nous avons choisi des méthodes contrôlant leur erreur comme le résumé de quantiles de Greenwald et Khanna (GK). Ce résumé n'est pas directement adapté à la classification supervisée et nous l'avons adapté en ajoutant les comptes par classe (GKClass) dans les tuples afin d'améliorer ses performances en classification. Nous avons aussi proposé une autre modification de l'algorithme GK d'origine permettant d'augmenter jusqu'à

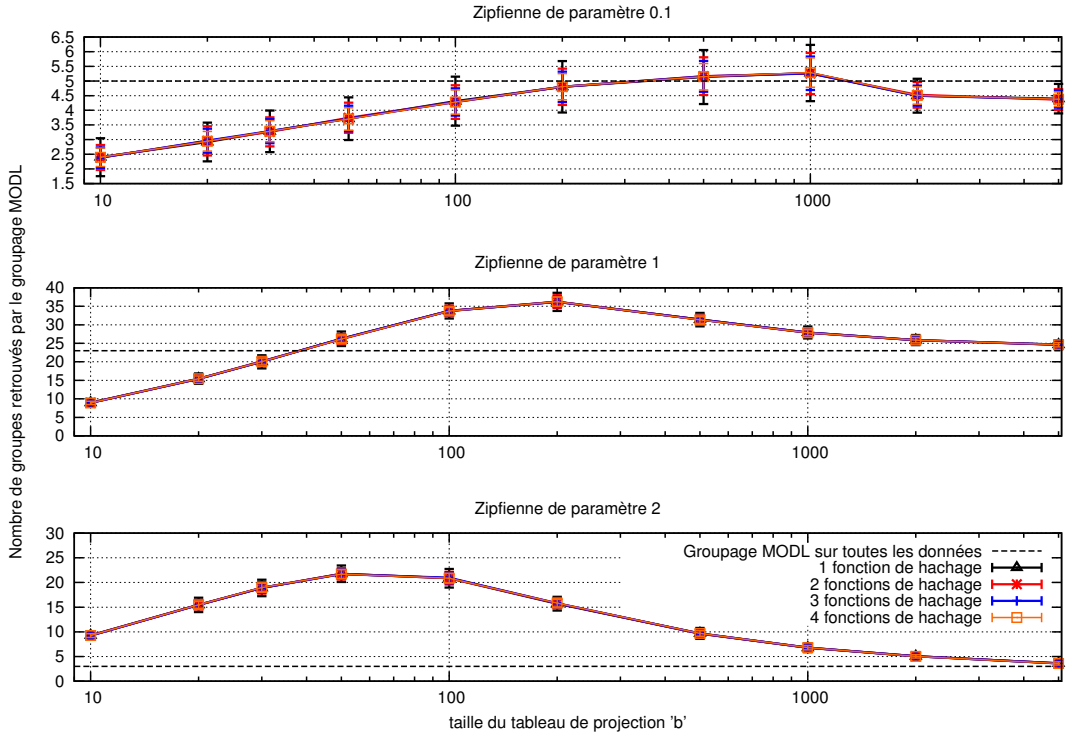


FIGURE 3.17 – Comparaison du nombre de groupes pour le résumé CMSClass après 1 millions d'insertion avec 'b' allant de 10 à 5000.

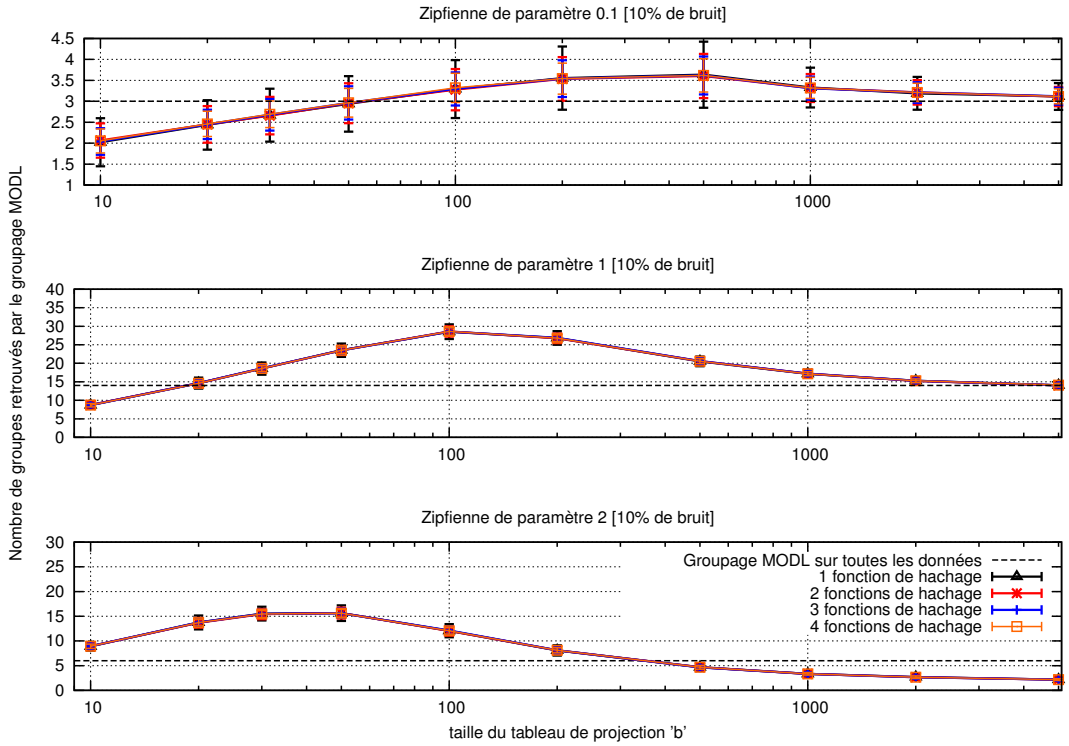


FIGURE 3.18 – Comparaison du nombre de groupes pour le résumé CMSClass après 1 millions d'insertion avec 'b' allant de 10 à 5000 après ajout de 10% de bruit.

un ordre de grandeur la vitesse d'insertion dans le résumé (GKClassAcc). Nous avons étudié le critère hors-ligne MODL de discrétisation supervisée et nous l'avons adapté (OnLineMODL) afin de pouvoir l'utiliser dans un environnement en-ligne à mémoire limitée. Ce résumé étant supervisé est capable d'allouer plus de mémoire aux régions de l'espace les plus informatives et moins aux autres. Notre dernière proposition de résumé (cPiD) repart de la première couche de PiD mais en l'adaptant afin qu'elle utilise une quantité de mémoire fixe. Les expérimentations sur ce premier niveau montre que la méthode OnLineMODL se comporte bien mieux que les autres sur des données à base de créneaux et plus particulièrement quand l'espace contient des zones « vides » (plateaux). Les autres méthodes sont relativement proches bien que cPiD soit un peu meilleure à mémoire identique car ses tuples consomment moins de mémoire. Pour le premier niveau catégoriel notre choix s'est porté sur le Count-min Sketch qui possède comme le résumé GK des garanties théoriques sur le taux d'erreur en fonction de la mémoire utilisée. Nous l'avons adapté (CMSClass) à la classification en ajoutant les comptes par classe dans son tableau de stockage.

Le deuxième niveau utilise l'approche MODL qui fonctionne aussi bien pour les variables numériques que catégorielles et prend en entrée les comptes fournis par les résumés de notre premier niveau. Elle est supervisée, régularisée et sans paramètre. Le résumé du premier niveau est de faible taille ce qui permet une exécution rapide des méthodes MODL bien qu'elles soient prévues pour fonctionner hors-ligne.

Finalement notre approche à deux niveaux est expérimentée sur différents jeux de données afin de montrer la validité de l'approche. Ces expérimentations sont conduites dans un premier temps sur des variables numériques puis dans un second temps sur des variables catégorielles. A ce stade aucun choix n'est effectué pour les algorithmes du premier niveau. Ils seront tous expérimentés dans le chapitre suivant.

Le prochain chapitre utilise nos résumés à deux niveaux pour construire des classifieurs en-ligne incrémentaux. Le premier classifieur expérimenté est le classifieur bayésien naïf qui prend en entrée les probabilités conditionnelles de notre résumé à deux niveaux. Une version moyennée en-ligne de ce classifieur est aussi proposée. Le deuxième classifieur adapté est un arbre en-ligne de type arbre de Hoeffding. L'utilisation de résumés à la place du stockage de toutes les données permet une plus faible consommation de mémoire et donc la possibilité d'avoir des arbres de plus grande taille.

Chapitre 4

Classifieurs incrémentaux en-ligne

Sommaire

.

Ce chapitre a fait l'objet des publications suivantes :

Arbres en ligne basés sur des statistiques d'ordre [SL12a]
Incremental Decision Tree based on order statistics [SL12b]
Stumping along a Summary for Exploration & Exploitation Challenge 2011 [SU12]

4.1 Introduction

Pour les données hors-ligne, des méthodes d'extractions de connaissance performantes et éprouvées depuis plusieurs années existent. Différents types de classifieurs ont été proposés : plus proches voisins, bayésien naïf, SVM, arbre de décision, système à base de règles... Mais avec l'apparition de nouvelles applications comme les réseaux sociaux, la publicité en-ligne, les données du Web... la quantité de données et leurs disponibilités ont changé. Les données auparavant facilement disponibles et pouvant tenir en mémoire (données hors-ligne) sont devenus massives et visibles une seule fois (flux de données). La plupart des classifieurs, prévus pour fonctionner hors-ligne, ne peuvent généralement pas s'appliquer directement sur un flux de données.

Depuis les années 2000, l'extraction de connaissances sur flux de données est devenue un sujet de recherche à part entière. De nombreux travaux adressant cette nouvelle problématique ont été proposés (voir section 2.3). Parmi les solutions aux problèmes de l'apprentissage en-ligne sur flux de données, les algorithmes d'apprentissage incrémentaux sont l'une des techniques les plus utilisées (voir section 2.2). Ces algorithmes sont capables de mettre à jour leur modèle à partir d'un seul nouvel exemple. Cependant la plupart d'entre eux, bien qu'étant incrémentaux, ne sont pas capables de traiter des flux de données car leur complexité n'est pas linéaire. Notre proposition consiste à adapter des méthodes existantes de classification supervisée hors-ligne en utilisant les résumés présentés dans le chapitre précédent afin qu'elles puissent traiter les flux de données en-ligne.

Les résumés présentés précédemment permettent d'obtenir, avec peu de mémoire, des estimations robustes des probabilités conditionnelles aux classes $P(X_i|C)$ pour chaque variable X_i. Ces estimations peuvent être utilisées dans différents classifieurs afin de les rendre incrémentaux et contrôler la mémoire qu'ils consomment. Parmi les méthodes performantes de l'état de l'art répondant aux contraintes du flux de données (voir section 2.3) on retrouve le classifieur bayésien naïf et les arbres de décision en-ligne (arbre de Hoeffding - voir section 2.3.2). Ces deux classifieurs nécessitent des probabilités conditionnelles aux classes pour réaliser leurs prédictions (bayésien naïf) ou pour se construire (calcul de l'entropie pour réaliser les coupures dans les arbres). Ce chapitre décrit comment ces classifieurs ont été adaptés afin de tirer partie de nos résumés à deux niveaux. Les modifications proposées rendent ces classifieurs incrémentaux, en-ligne et capable de traiter les flux de données.

Plan du chapitre

La deuxième section de ce chapitre présente l'adaptation du classifieur bayésien naïf. Celui-ci utilise directement les probabilités $P(X_i|C)$ fournies par les résumés. Le classifieur en lui-même ne nécessite aucune modification car sa prédiction consiste en une simple multiplication des probabilités conditionnelles. Le classifieur bayésien naïf peut être grandement amélioré en pondérant ses variables. Cette pondération s'effectue habituellement hors-ligne à l'aide d'ensembles de validation. Nous proposons une méthode basée sur une descente de gradient permettant d'optimiser ces poids de manière incrémentale et en-ligne. Nos modifications sont validées sur les données du « Large Scale Learning Challenge » et comparées aux méthodes hors-ligne. Dans la section 3, nous proposons d'améliorer les arbres de Hoeffding qui sont très présents dans l'état de l'art de la classification incrémentale et reconnus comme étant une méthode performante pour traiter ces problèmes. Nos améliorations portent sur les différents éléments qui les composent, en proposant : (i) de nouveaux résumés dans les feuilles, (ii) un nouveau critère de coupure, (iii) un modèle local plus robuste. Les générateurs de flux de données de la littérature sont utilisés pour valider notre approche. La dernière section conclut ce chapitre en résumant les modifications apportées aux classifieurs existants afin de les rendre adaptés aux flux.

4.2 Classifieur bayésien naïf moyenné en-ligne

Le classifieur bayésien naïf est souvent utilisé pour l'apprentissage en-ligne et sur flux de données. Il ne nécessite en entrée que des probabilités conditionnelles $P(X_i|C)$ et sa complexité en prédiction est très faible, ce qui le rend adapté aux flux. La difficulté majeure pour la mise en place de ce classifieur, dans le contexte des flux, est l'estimation en-ligne des $P(X_i|C)$. Cette difficulté est résolue à l'aide de nos résumés du chapitre précédent qui fournissent directement cette estimation. De plus nous proposons une nouvelle méthode de moyennage de classifieurs bayésien adaptée à une utilisation en-ligne. Cette nouvelle méthode utilise une descente de gradient pour optimiser les poids des variables du classifieur bayésien naïf moyenné.

4.2.1 Le classifieur bayésien naïf

4.2.1.1 Présentation

Le classifieur bayésien naïf est une méthode d'apprentissage supervisé qui repose sur une hypothèse simplificatrice forte : les variables X_i sont indépendantes conditionnellement aux valeurs de la classe à prédire. Cette hypothèse naïve ne permet pas de modéliser les interactions entre différentes variables. Cependant sur de nombreux problèmes réels cette limitation n'a que peu d'impact [HY01, LIT92].

L'idée de départ de ce classifieur vient de la formule de Bayes :

$$P(C|X) = \frac{P(C)P(X|C)}{P(X)}$$

La probabilité conditionnelle jointe $P(X|C)$ étant difficilement estimable on utilise la version naïve (appelée par la suite NB) de ce classifieur. La probabilité de la classe devient dans ce cas :

$$P_{NB}(C|X) = \frac{P(C)\prod_i^d P(X_i|C)}{P(X)}$$

Ce classifieur nécessite en entrée la probabilité de la classe $P(C)$ et la probabilité de la variable X_i sachant la classe C. La probabilité $P(C)$ se calcule facilement en-ligne en conservant le nombre d'apparitions de chaque classe. La probabilité $P(X_i|C)$ n'est pas facilement calculable en-ligne car cela nécessiterait de conserver toutes les instances. Nos résumés à deux niveaux sont capables de retourner une bonne estimation de cette probabilité et fonctionnent en-ligne. Le premier niveau de notre méthode fonctionne de manière non supervisée et est capable de retourner une première estimation. Le deuxième niveau permet, à l'aide de la méthode supervisée MODL, d'avoir une estimation plus robuste. À partir de nos résumés en-ligne, il est donc possible de directement construire un classifieur bayésien naïf.

L'intérêt du classifieur bayésien naïf, dans le cadre des flux de données, provient de sa faible complexité temporelle en prédiction qui dépend seulement du nombre de variables. Il en va de même pour sa complexité mémoire car il ne nécessite qu'un résumé par variable. Sa consommation mémoire est bien moindre que des modèles multivariés, comme les arbres, qui nécessitent un résumé par sous région de l'espace découpé (feuille).

4.2.1.2 Expérimentations

Les expérimentations pour ce classifieur sont lancées sur les bases du challenge Large Scale Learning (voir section 2.5.2). Nous utilisons les bases alpha, beta, delta et gamma, qui possèdent

Niveau 1

	Alpha			Beta			Delta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass10	54,49	54,55	54,54	51,02	51,27	51,28	80,08	81,58	82,24
GKClassAcc10	50,24	49,83	51,44	49,98	50,03	50,65	50,02	52,72	50,04
OnLineMODL10	54,45	54,54	54,66	50,67	51,23	51,27	71,91	74,61	76,16
CPiD10	54,48	54,49	54,53	51,07	51,35	51,28	81,29	82,35	82,84
Gaussien	54,62	54,67	54,67	51,21	51,50	51,31	84,58	85,10	85,08
Hors-ligne (MODL)	54,60	54,61	54,61	49,79	51,36	51,19	80,78	82,44	83,91

	Gamma			Epsilon			Zeta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass10	85,68	90,09	92,03	64,04	70,01	70,00	72,69	76,27	78,16
GKClassAcc10	52,11	50,61	50,19	49,88	50,48	49,88	51,25	49,76	50,34
OnLineMODL10	83,74	86,73	88,16	63,77	66,43	68,79	68,30	73,00	76,01
CPiD10	92,08	92,44	92,59	70,19	70,36	70,21	78,32	78,34	78,11
Gaussien	95,09	95,10	95,16	70,66	70,57	70,43	78,96	78,77	78,52
Hors-ligne (MODL)	92,95	93,99	94,63	70,35	71,04	70,58	78,39	78,34	78,26

Niveau 1 et 2

	Alpha			Beta			Delta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass10	54,00	54,11	54,35	49,79	49,79	51,05	72,42	74,97	75,10
GKClassAcc10	54,58	54,56	54,82	49,79	50,59	51,31	75,55	78,36	77,01
OnLineMODL10	50,20	50,20	50,20	49,79	49,79	50,89	50,59	59,31	65,65
CPiD10	54,50	54,61	54,62	49,79	49,79	51,29	78,95	80,04	81,14
Gaussien (niveau 1)	54,62	54,67	54,67	51,21	51,50	51,31	84,58	85,10	85,08
Hors-ligne (MODL)	54,60	54,61	54,61	49,79	51,36	51,19	80,78	82,44	83,91

	Gamma			Epsilon			Zeta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass10	81,23	83,82	84,16	70,14	69,97	70,02	77,96	78,13	78,10
GKClassAcc10	86,94	89,54	90,34	70,40	70,15	70,24	77,93	78,31	78,12
OnLineMODL10	76,97	77,77	76,22	69,15	69,04	69,90	77,50	73,75	74,48
CPiD10	90,34	90,99	91,28	70,38	70,41	70,23	78,49	78,48	78,36
Gaussien (niveau 1)	95,09	95,10	95,16	70,66	70,57	70,43	78,96	78,77	78,52
Hors-ligne (MODL)	92,95	93,99	94,63	70,35	71,04	70,58	78,39	78,34	78,26

TABLE 4.1 – Résultats synthétiques sur les bases du challenge Large Scale Learning avec des résumés de 10 tuples (indicateur : précision).

500 variables numériques et les bases epsilon et zeta, qui en contiennent 2000. Chaque base contient 500 000 exemples.

Les jeux de données sont séparés en ensemble de test/apprentissage. Les premiers exemples sont pris comme ensemble de test et leurs tailles sont les suivantes :

– 100 000 exemples pour les jeux qui ont 500 variables numériques

– 40 000 pour les autres (du à une contrainte mémoire imposée par une bibliothèque 32 bits).

Les expérimentations ont été lancées sur tous les types de résumés précédemment vus (GKClass, GKClassAcc, OnLineMODL, CPiD, Gaussien) avec deux tailles différentes : 10 et 100 tuples. Les résultats, en termes de précision, pour les résumés avec 10 tuples sont présentés dans le tableau 4.1, et pour 100 tuples dans le tableau 4.2. La dernière ligne du tableau donne les résultats de la méthode MODL utilisée hors-ligne avec toutes les données.

Discussion des résultats

Les meilleurs résultats sont obtenus par le résumé basé sur une gaussienne. Il semble que ces données aient été générées par un générateur basé sur une gaussienne. Cependant, nos résumés

Niveau 1

	Alpha			Beta			Delta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass100	54,34	54,43	54,59	50,35	50,68	51,18	60,65	67,94	79,82
GKClassAcc100	53,49	53,30	53,35	50,07	50,29	50,45	55,13	56,28	57,40
OnLineMODL100	54,19	54,54	54,52	50,17	50,68	51,03	59,78	68,38	78,78
CPiD100	54,54	54,64	54,60	50,93	51,22	51,24	75,93	80,10	83,46
Gaussien	54,62	54,67	54,67	51,21	51,50	51,31	84,58	85,10	85,08
Hors-ligne	54,60	54,61	54,61	49,79	51,36	51,19	80,78	82,44	83,91

	Gamma			Epsilon			Zeta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass100	69,22	80,36	90,87	56,68	60,92	65,75	58,13	63,14	72,32
GKClassAcc100	60,36	60,15	59,82	53,96	53,93	53,60	55,02	54,75	54,81
OnLineMODL100	67,02	77,26	88,62	56,85	58,94	64,72	57,41	61,10	69,04
CPiD100	90,97	93,31	94,46	68,58	69,63	70,09	75,06	76,99	77,93
Gaussien	95,09	95,10	95,16	70,66	70,57	70,43	78,96	78,77	78,52
Hors-ligne	92,95	93,99	94,63	70,35	71,04	70,58	78,39	78,34	78,26

Niveau 1 et 2

	Alpha			Beta			Delta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass100	54,40	54,49	54,56	49,79	51,05	51,23	80,56	82,22	83,47
GKClassAcc100	54,40	54,49	54,59	49,79	51,08	51,12	80,57	82,29	83,62
OnLineMODL100	54,35	54,39	54,56	49,79	51,01	51,14	79,72	81,51	83,01
CPiD100	54,36	54,52	54,57	49,79	51,09	51,12	80,66	82,39	83,77
Gaussien (niveau 1)	54,62	54,67	54,67	51,21	51,50	51,31	84,58	85,10	85,08
Hors-ligne	54,60	54,61	54,61	49,79	51,36	51,19	80,78	82,44	83,91

	Gamma			Epsilon			Zeta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
GKClass100	92,63	93,51	94,23	70,48	70,37	70,43	78,35	78,63	78,48
GKClassAcc100	92,80	93,71	94,42	70,47	70,38	70,44	78,36	78,49	78,24
OnLineMODL100	92,34	93,07	93,68	70,34	70,22	70,20	78,29	78,27	78,24
CPiD100	92,93	93,83	94,41	70,52	70,38	70,36	78,50	78,48	78,42
Gaussien (niveau 1)	95,09	95,10	95,16	70,66	70,57	70,43	78,96	78,77	78,52
Hors-ligne	92,95	93,99	94,63	70,35	71,04	70,58	78,39	78,34	78,26

TABLE 4.2 – Résultats synthétiques sur les bases du challenge Large Scale Learning avec des résumés de 100 tuples (indicateur : précision).

configurés avec 100 tuples et le second niveau ont des performances très proches et ont l'avantage de ne pas faire d'hypothèses sur la distribution des données.

Les résultats sur les jeux de données alpha et beta donnent tous les mêmes résultats quels que soient les résumés utilisés, leur taille ou l'application ou non du deuxième niveau MODL. Leurs résultats ne sont donc pas discutés par la suite.

Pour les résumés à 10 tuples (tableau 4.1) et sur la base gamma on observe, plus particulièrement pour les résumés GKClass et OnLineMODL, une baisse des performances après l'application de la méthode MODL du deuxième niveau. Par contre la méthode GKClassAcc, qui a de mauvaises performances pour le niveau 1, profite du niveau 2. Cette baisse de performance se retrouve aussi sur la base delta et tout particulièrement pour le résumé OnLineMODL dont les performances sont fortement dégradées après application du deuxième niveau. Le faible nombre de tuples du niveau 1 (10 tuples) ne permet pas à toutes les méthodes d'avoir des estimations assez fiables. Par conséquent le niveau 2 va fusionner une partie des tuples, ce qui va aboutir à plus de robustesse mais aussi, pour un si faible nombre de tuples, à moins de performance dans de nombreux cas.

Pour les résumés à 100 tuples (tableau 4.2), les résultats après le second niveau sont toujours meilleurs qu'avec seulement le premier niveau. L'intérêt du second niveau est évident dès qu'il y a suffisamment de tuples dans le premier niveau. Les performances, tout particulièrement avec peu de données (40 000 exemples), sont très bonnes et bien meilleures qu'en utilisant seulement le premier niveau.

Étude de l'impact du second niveau

Afin d'observer l'impact d'avoir deux niveaux, de nouvelles expérimentations ont été lancées avec et sans le second niveau. Différentes tailles de résumé sont choisies : 10, 30 et 100 tuples. La figure 4.1 montre le comportement du second niveau pour le jeu de données gamma.

On observe un impact important du second niveau sur les performances du classifieur. Tout d'abord si le nombre de tuples est trop faible (10 dans le cas de cette base), alors le second niveau détériore les performances. Cela s'explique par le critère MODL qui fusionne des tuples ayant des densités proches en pensant que ces densités ne sont pas significativement différentes. Avec plus de tuples dans le niveau 1 (100 tuples ici) l'effet inverse est observé.

Les performances du premier niveau avec 100 tuples et peu de données sont très mauvaises. L'ajout du deuxième niveau permet d'atteindre, avec une faible consommation mémoire, les performances qu'on obtiendrait en utilisant la méthode MODL hors-ligne sur la même quantité de données. Le niveau 2 améliore donc grandement les prédictions car les fusions qu'ils réalisent permettent d'obtenir de meilleures estimations de densité si le nombre de tuples du premier niveau est suffisant.

Il est difficile, *a priori*, de fixer idéalement le nombre de tuples pour le premier niveau et donc de s'assurer une estimation robuste des densités. Le deuxième niveau, ne nécessitant pas de réglages, apporte robustesse et performance au classifieur bayésien naïf pour un nombre suffisant de tuples dans le premier niveau. Les performances sont alors très proches de celles obtenues avec la méthode MODL hors-ligne.

4.2.1.3 Conclusion

Les expérimentations sur le classifieur bayésien naïf montrent que, pour un nombre suffisant de tuples et avec toutes les méthodes du premier niveau, l'ajout du second niveau permet d'obtenir de très bons résultats. Ce comportement rend le choix du premier niveau moins déterminant. L'intérêt d'une méthode non paramétrique pour le second niveau montre tout son intérêt. Il suffit de fixer la taille des résumés du niveau 1 en fonction de la mémoire disponible, le niveau 2 va par

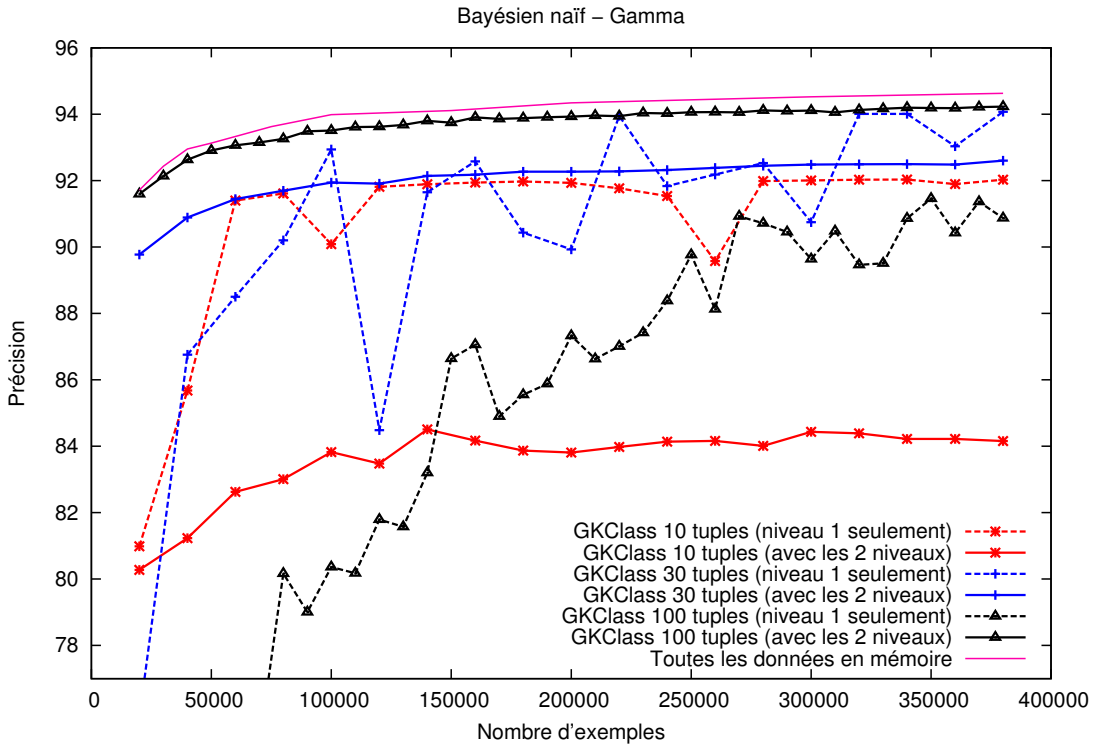


FIGURE 4.1 – Comparaison des performances du classifieur bayésien naïf sur la base Gamma avec et sans deuxième niveau pour le résumé GKClass.

la suite automatiquement regrouper les tuples du niveau 1 pour obtenir une estimation de densité robuste. Cette propriété est très intéressante car elle permet d'éviter une phase de réglage des paramètres. Cette phase est d'autant plus critique que, dans le cadre des flux de données, il n'est pas facile de mettre en place des méthodes de validations croisées pour régler les paramètres. De plus, le flux peut varier au cours du temps et les paramètres optimaux peuvent être différents selon la position dans le flux.

La seule limite de notre méthode correspond au cas d'une très faible mémoire disponible. Dans ce cas le niveau 1 est suffisamment robuste dans ces estimations de densité. Cette faible taille du résumé peut impliquer un mélange assez important des classes dans les tuples. Ce mélange est alors assimilé à du bruit par le niveau 2 (MODL) et celui-ci fusionne des tuples ce qui aboutit à une dégradation des estimations.

Bien que le niveau 2 fasse aboutir la plupart des méthodes aux mêmes résultats, nous recommanderions l'utilisation de la méthode GKClass comme premier niveau pour ses garanties en termes d'erreurs sur les comptes et son absence d'hypothèses sur la distribution des données en entrée.

4.2.2 Un classifieur bayésien naïf moyenné en-ligne

4.2.2.1 Présentation

La littérature montre que le classifieur bayésien naïf peut être amélioré de deux manières :
- en sélectionnant les variables [KS96, LS94]
- en pondérant les variables [HMR99]

La sélection de variables consiste à ne prendre qu'une partie des variables, on parle alors de bayésien naïf sélectif (SNB : « Selective Naïve Bayes »). Une manière effective de choisir les variables est de supprimer celles qui ne sont pas informatives. Cette information est directement disponible dans nos résumés à l'aide du deuxième niveau supervisé MODL. En effet celui-ci retourne un gain de compression pour chaque discrétisation/groupage. Ce gain correspond à la probabilité qu'un modèle de discrétisation/groupage M explique les données. On note M_* le meilleur modèle de discrétisation/groupage et $M_\emptyset$ le modèle à un seul intervalle/groupe. Soit $C(M)$ le coût du modèle M (cette fonction de coût a été explicitée pour la discrétisation et le groupage dans la section 3.3.3). Le gain de compression [Bou09] est défini selon la formule :

$$l = 1 - \frac{C(M_*)}{C(M_\emptyset)}$$

Si le meilleur modèle de discrétisation/groupage est le modèle à un intervalle/groupe, alors $l = 0$ et la variable est ignorée.

Les capacités de prédictions peuvent être encore améliorées en pondérant chacune des variables. Cette approche revient à un moyennage de modèles et en possède les qualités. Le moyennage de modèles vise à combiner la prédiction d'un ensemble de classifieurs de façon à améliorer les performances prédictives. Ce principe a été appliqué avec succès dans le cas du bagging [Bre96], qui exploite un ensemble de classifieurs appris sur une partie des exemples. Dans ces approches, le classifieur moyenné résultant procède par vote des classifieurs élémentaires pour effectuer sa prédiction. A l'opposé des approches de type bagging, où chaque classifieur élémentaire se voit attribuer le même poids, le moyennage de modèles bayésiens (BMA = Bayesian Model Averaging) [HMR99] pondère les classifieurs selon leur probabilité *a posteriori*. Le classifieur bayésien naïf moyenné résultant [HMR99] est similaire au classifieur bayésien naïf mais ajoute une pondération par variable tel que décrit dans l'équation 4.1 (où i est l'indice de la variable, k une classe d'intérêt et K le nombre de classes du problème). Cette pondération permet de limiter le biais introduit par l'hypothèse d'indépendance des variables.

$$P(C_k|X) = \frac{P(C_k)\prod_i P(X_i|C_k)^{w_i}}{\sum_{j=1}^{K}(P(C_j)\prod_i P(X_i|C_j)^{w_i})} \tag{4.1}$$

Le classifieur bayésien naïf sélectif et moyenné obtient de meilleurs résultats que le modèle non moyenné [LS94, Bou06b]. Cette supériorité est peu visible lorsque le nombre d'exemples utilisés pour entraîner le classifieur est faible mais s'accroît à mesure que le nombre d'exemples augmente. Dans le cadre des flux de données, le nombre d'exemples est important et on peut espérer une amélioration des performances grâce à la pondération.

Notre approche : optimisation directe des poids

Lorsque l'on se place dans le cadre de l'apprentissage hors-ligne, les poids du classifieur naïf bayes moyenné peuvent être estimés de différentes manières :

- par moyennage de modèles [HMR99] ;
- par moyennage de modèles avec optimisation des poids basée sur un critère MDL (Minimum Description Length) [Bou06b] ;
- par optimisation directe des poids par une descente de gradient [GB11].

Toutes ces méthodes fonctionnent en chargeant toutes les données en mémoire et nécessitent de les relire plusieurs fois. Par conséquent, elles ne peuvent pas être directement adaptées pour traiter des flux de données. L'approche présentée dans cette section consiste à directement optimiser les poids du classifieur. Cette optimisation directe a déjà fait l'objet d'étude et montré son intérêt

dans le cas d'un apprentissage batch [GB11]. On propose ici de la réaliser dans le cas d'un apprentissage incrémental sur flux.

La première étape consiste à créer un modèle graphique (voir Figure 4.2) dédié à l'optimisation de ces poids. Il permet de réécrire l'équation 4.1 sous la forme d'un modèle graphique où le classifieur bayésien naïf moyenné reçoit un poids par variable et par classe tel que présenté dans l'équation 4.4. Les poids que nous cherchons à optimiser sont donc plus nombreux dans ce modèle graphique. En effet le poids n'est plus seulement associé à la variable, mais à la variable conditionnellement à la classe, soit :

- w_{ik} le poids associé à la variable i et à la classe k
- b_k le biais lié à la classe k à apprendre

La première couche du modèle graphique est une couche linéaire réalisant pour chaque classe k à apprendre une somme pondérée. Chaque sortie de la première couche du modèle graphique est égale à :

$$H_k = \sum_{i=1}^{d} w_{ik} log(P(X_i|C_k)) + b_k \tag{4.2}$$

La couche suivante du modèle graphique est un $log(softmax)$. Chaque sortie est donc de la forme :

$$y_k = log\left(\frac{e^{H_k}}{\sum_{j=1}^{K} e^{H_j}}\right) \tag{4.3}$$

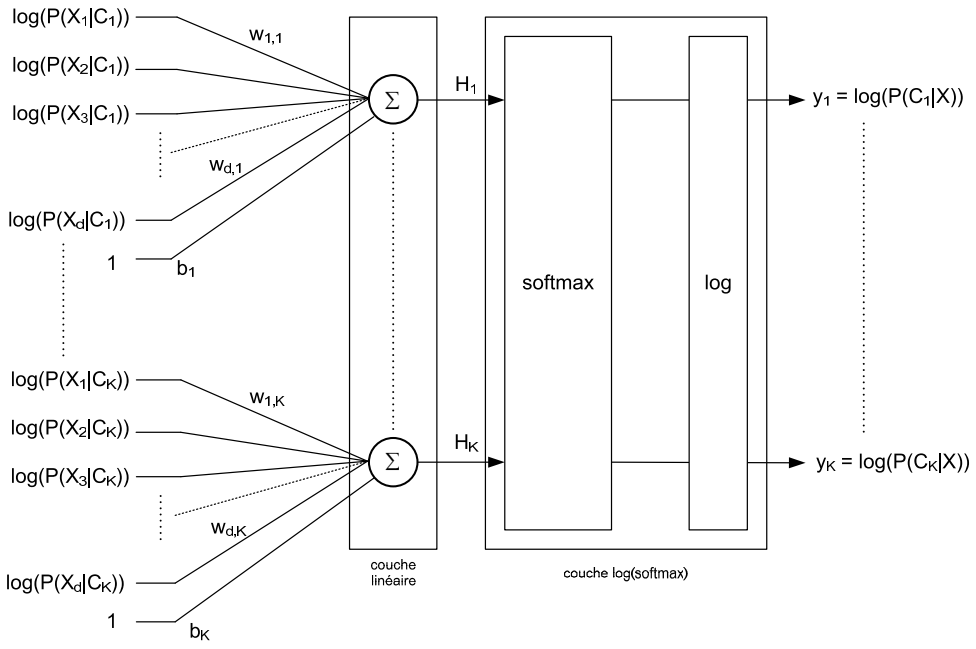


FIGURE 4.2 – Modèle graphique pour l'optimisation des poids du classifieur bayésien moyenné.

Ce modèle graphique permet d'avoir directement en sortie la valeur des $log(P(C_k|X))$:

$$y_k = log(P(C_k|X)) = log\left(\frac{e^{b_k + \sum_{i=1}^{d} w_{i,k} log(p(X_i|C_k))}}{\sum_{j=1}^{K} e^{b_j + \sum_{i=1}^{d} w_{i,j} log(p(X_i|C_j))}}\right) \tag{4.4}$$

Les variables positionnées en entrée de ce modèle sont issues des résumés univariés produits à partir du flux présentés dans le chapitre précédent. L'optimisation des poids est réalisée à l'aide d'une descente de gradient stochastique. Pour un exemple X donné, la règle de modification des poids est :

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial coût^X}{\partial w_{ij}} \tag{4.5}$$

où $coût^X$ est la fonction de coût appliquée à l'exemple X et $\frac{\partial coût^X}{\partial w_{ij}}$ la dérivée de la fonction de coût vis à vis des paramètres du modèle, ici les poids w_{ij}. On présente dans l'annexe A le calcul de cette dérivée dans le cas du modèle graphique proposé dans la figure 4.2 ainsi que les performances de ce classifieur dans le cas hors-ligne. Cette annexe montre le bon comportement de cette approche lorsque toutes les données peuvent être chargées en mémoire.

Configuration de la descente de gradient en-ligne

Les 3 principaux paramètres que l'on retrouve dans la littérature [LBOM98] de la descente de gradient stochastique pour les modèles graphiques de type « perceptron » (ou perceptron multi-couches) sont les suivants :

- la fonction de coût
- le nombre d'itérations
- le pas d'apprentissage

La fonction de coût peut être de plusieurs natures : erreur quadratique, vraisemblance, log vraisemblance... Elle doit être choisie selon le problème d'apprentissage à réaliser et les données disponibles. Dans ce mémoire on s'intéresse aux problèmes de classification supervisée où la variable cible possède seulement quelques modalités. L'utilisation d'une fonction minimisant l'erreur quadratique est équivalente à maximiser la vraisemblance sous l'hypothèse que les modalités de la variable cible aient été générées sous l'hypothèse d'un modèle gaussien. Etant donné que nous souhaitons pouvoir apprendre tous types de distribution, cette hypothèse n'est pas appropriée [Bis95] et nous avons préféré utiliser le log-vraisemblance qui optimise directement l'estimation de la vraisemblance $log(P(C_k|X))$. Cette fonction a aussi l'avantage de limiter les problèmes numériques dus aux calculs sur des faibles valeurs de probabilité.

Le nombre d'itérations est dans notre cas un paramètre absent du fait que le modèle est mis à jour après chaque exemple et seulement une fois. Etant donné que l'apprentissage est réalisé sur un flux de données, nous choisissons de n'effectuer qu'une itération par exemple du flux, de ne pas utiliser d'early stopping [Pre97] ni d'ensembles de validation [AMM+97].

Le choix du pas d'apprentissage est lui plus difficile. Une valeur trop faible aboutit à une convergence longue pour atteindre le minimum de la fonction de coût, alors qu'un pas trop grand ne permet pas d'atteindre ce minimum. Dans le cas d'un apprentissage batch il est possible de régler sa valeur par une méthode de validation croisée mais dans le cas de l'apprentissage sur flux ou en une passe ceci n'est pas envisageable. Pour les expérimentations de ce chapitre un pas fixe ($\eta = 10^{-4}$) a été choisi. Cependant dans le cadre d'un éventuel changement de concept (voir section 2.4), il serait intéressant que ce pas soit adaptatif afin de réagir plus vite lorsqu'un changement est détecté. Dans ce cas on peut s'intéresser aux méthodes qui adaptent le pas d'apprentissage automatiquement au cours de l'apprentissage comme par exemple R-Prop qui est l'une des plus connues [RB93] ou encore la méthode proposée par Kuncheva [KP08].

4.2.2.2 Expérimentations

Le protocole expérimental est identique à celui utilisé pour évaluer le classifieur bayésien naïf. Les 6 bases du challenge Large Scale Learning sont utilisées avec différentes tailles pour les jeux de données en apprentissage. Les résultats de notre classifieur bayésien naïf moyenné en-ligne sont comparés à deux classifieurs hors-ligne. Tout d'abord la comparaison est faite avec le classifieur bayésien naïf moyenné avec optimisation des poids à l'aide d'un critère MDL [Bou06b]. Ce classifieur était parmi les meilleurs du challenge Large Scale Learning. Afin d'observer l'amélioration due à la pondération des variables nous comparons notre méthode au classifieur bayésien naïf non moyenné de la section précédente (version hors-ligne et version avec un résumé à deux niveaux GKClass100). Ces résultats sont présentés dans le tableau 4.3.

	Alpha			Beta			Delta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
NB moyenné en-ligne	64,03	66,40	67,61	49,79	49,79	52,20	75,35	77,74	79,53
NB moyenné hors-ligne	66,13	67,30	68,77	49,79	51,39	53,24	80,80	82,46	83,91
NB en-ligne (GKClass100)	54,40	54,49	54,56	49,79	51,05	51,23	80,56	82,22	83,47
NB hors-ligne	54,60	54,61	54,61	49,79	51,36	51,19	80,78	82,44	83,91

	Gamma			Epsilon			Zeta		
Nb exemples appris →	40 000	100 000	380 000	40 000	100 000	380 000	40 000	100 000	380 000
NB moyenné en-ligne	90,25	91,05	91,76	69,25	74,76	79,95	73,82	79,91	84,52
NB moyenné hors-ligne	92,95	94,00	94,64	84,36	85,34	86,01	88,67	89,51	90,43
NB en-ligne (GKClass100)	92,63	93,51	94,23	70,48	70,37	70,43	78,35	78,63	78,48
NB hors-ligne	92,95	93,99	94,63	70,35	71,04	70,58	78,39	78,34	78,26

TABLE 4.3 – Résultats synthétiques du classifieur bayésien moyenné en-ligne avec des résumés à 2 niveaux (niveau 1 : GKClass avec 100 tuples) sur les bases du challenge Large Scale Learning (indicateur : précision).

Discussion des résultats

Sur la base alpha les résultats de notre classifieur bayésien moyenné en-ligne sont biens meilleurs que pour le NB hors-ligne et presque aussi bon que le NB moyenné hors-ligne. Pour la base beta l'amélioration n'est visible que pour 380 000 exemples appris, auparavant les résultats sont légèrement moins bons que les autres méthodes. delta et beta sont deux bases sur lesquelles le passage du NB au NB moyenné n'apporte rien pour les versions hors-ligne. Notre NB moyenné est pour ces deux bases moins performant que la version non moyennée (4% et 3% de précision perdue respectivement). Pour les bases epsilon et zeta les performances de notre classifieur moyenné sont à mi chemin entre celles du NB hors-ligne et celles du NB moyenné hors-ligne. Sur ces deux bases, les performances croissent rapidement avec le nombre d'exemples. On aurait pu espérer atteindre des performances identiques à la version hors-ligne si plus de données avaient été disponibles pour l'apprentissage.

4.2.2.3 Conclusion

Il apparaît que les résultats pour la version moyennée du classifieur bayésien naïf sont en demi-teintes. Il est capable sur certaines bases d'améliorer les résultats significativement par rapport à la version non moyennée mais il lui arrive aussi de dégrader les performances sur d'autres bases. La quantité de données de ce challenge n'est peut être pas suffisante pour arriver à converger vers les meilleurs poids.

Plusieurs pistes pourraient être envisagées pour améliorer ce classifieur qui, dans sa version hors-ligne, ne souffre pas des problèmes ci-dessous. On pourrait par exemple imaginer de conserver un réservoir de données de faible taille de manière à pouvoir réaliser plusieurs itérations dans la descente de gradient afin qu'elle se fasse plus rapidement. De la même manière le pas d'apprentissage pourrait être différent au cours de l'apprentissage : plus rapide en début d'apprentissage et plus lent par la suite.

4.3 Arbres en-ligne

Parmi les méthodes d'apprentissage incrémental, les modèles à base d'arbres de décision inspirés de l'algorithme « Very Fast Decision Tree » (VFDT) [DH00], sont très largement utilisés (voir section 2.3.2). La construction de l'arbre est incrémentale et les feuilles se transforment en nœuds 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 de pureté (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 [GRM03].

4.3.1 Présentation

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 4.3 illustre par un exemple simple l'approche des arbres en-ligne et leurs différents composants.

4.3.1.1 Résumés

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 télécommunications ou d'énergie, le flux est potentiellement de taille infinie. L'arbre généré peut posséder un grand nombre de nœuds de décision. Cependant la mémoire disponible est elle toujours de taille finie. Il en découle que ces résumés doivent être de faibles tailles et gérer le compromis précision/erreur. Ce cas d'utilisation correspond parfaitement aux résumés présentés au chapitre précédent et seront donc utilisés dans les feuilles. Idéalement il faudrait aussi que les résumés soient adaptés à la méthode permettant de trouver les points de coupure et au modèle local.

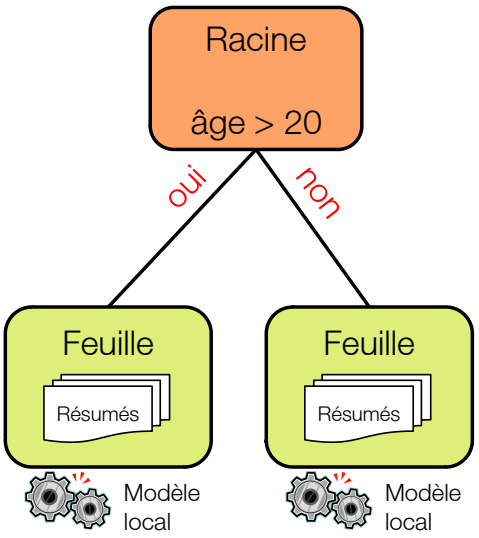


FIGURE 4.3 – Principaux composants d'un arbre en-ligne

4.3.1.2 Critère

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 [BFOS84] et le « gain ratio » basé sur l'entropie utilisé dans C4.5 [Qui93]. Ces deux critères permettent de trouver le meilleur point de coupure pour un attribut et fournissent une valeur de critère. Il suffit ensuite de réaliser la coupure sur l'attribut ayant la meilleure valeur de critère. Ce processus de transformation d'une feuille en nœud se répète afin d'induire l'arbre de décision final.

Dans le cas de la construction d'un arbre hors-ligne toutes les données sont disponibles. On peut donc s'assurer que le choix de l'attribut sur lequel couper est le meilleur choix au vu de toutes les données. Pour un arbre en-ligne, construit sur un flux de données, ce choix doit se faire seulement avec les données vues (à l'aide des résumés) au moment d'évaluer les attributs. Dans ce cas, 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 proposent dans VFDT d'utiliser la borne de Hoeffding [Hoe63]. Cette borne permet de s'assurer que la vraie moyenne d'une variable aléatoire comprise dans un intervalle R ne sera pas différente, à ϵ près, de sa moyenne estimée après n observations indépendantes, tout cela avec une probabilité de $1-\delta$: $\epsilon = \sqrt{\frac{R^2}{2n} ln(\frac{1}{\delta})}$. Elle apporte une garantie sur le choix du bon attribut. La borne de Hoeffding a par la suite été très souvent utilisée pour la réalisation d'arbre de décision en-ligne (voir section 2.3.2). Nous utilisons aussi cette borne dans le cadre de la construction de l'arbre en-ligne proposé.

Utilisation du critère MODL pour les coupures Afin d'être cohérent avec nos résumés, le critère MODL, basé sur les comptes, a été choisi pour réaliser les coupures et les groupements. L'approche MODL (présentée section 3.3.3), 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 ou 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 (voir section 4.2.2.1).

Les méthodes MODL de discrétisation [Bou06a] et de groupage de valeurs [Bou05] (présentés section 3.3.3) 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 bayésienne. Son but est de trouver les meilleurs paramètres de la discrétisation/groupage au sens bayésien : nombre d'intervalles, bornes des intervalles et répartition des classes dans les intervalles, qui maximisent la vraisemblance *a posteriori*.

Notre arbre est construit en-ligne à la manière de VFDT en utilisant la borne de 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 pourrait permettre de construire des arbres ayant des nœuds avec plus de deux fils.

4.3.1.3 Modèle local

Gama et al. [GRM03] observent empiriquement qu'il faut de 100 à 1000 exemples avant de transformer une feuille en nœud. Ces exemples, stockés dans les résumés des feuilles, ne sont pas utilisés pour améliorer le modèle global tant que la feuille n'est pas transformée en nœud. Ils proposent avec VFDTc d'utiliser ces exemples en ajoutant dans chaque feuille un modèle local. Cette technique d'utilisation de modèles locaux positionnés dans les feuilles d'un arbre existe depuis longtemps. Par exemple l'algorithme NBTree [Koh96] 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. Nous avons menés une étude [SL11c] (voir annexe B) sur la vitesse (en nombre d'exemples) d'apprentissage de différents classifieurs. Celle-ci montre que les forêts d'arbres 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 au 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 $P(X_i|C)$. De plus il est rapide à construire et possède une faible complexité algorithmique en prédiction. Ce classifieur est d'ailleurs celui choisi dans VFDTc (il améliore la prédiction de l'arbre sur des jeux de données comme Waveform et LED de l'UCI [GRM03]).

Nous avons donc choisi le classifieur bayésien naïf comme modèle local dans les feuilles. L'estimation de la densité conditionnelle aux classes est fournie par nos résumés à deux niveaux. L'intérêt de la méthode MODL utilisée dans le second niveau 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 où 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 au vu des données observées. Dans ce cas, c'est la classe majoritaire qui est prédite et donc revient à la prédiction d'un arbre sans modèle local. Kirkby dans [Kir08] étudie justement ce comportement en comparant des arbres de 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.

4.3.2 Expérimentations

4.3.2.1 Algorithmes comparés

Pour nos expérimentations nous avons utilisé la boîte à outils MOA : Massive Online Analysis [BK09] développée par l'université de Waikato qui reprend la librairie VFML fournie par Hulten et Domingos [HD]. 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 [3].

Tous les arbres évalués partent du même algorithme basé sur les arbres de Hoeffding. Cet algorithme contient trois paramètres :

- n : le nombre d'exemples à considérer avant d'essayer de trouver un point de coupure ;
- δ : la confiance que l'on souhaite avoir sur la moyenne dans la borne de Hoeffding ;
- τ : paramètre qui est utilisé pour imposer le choix entre deux attributs dont la différence de critère est trop proche pour que la borne de 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, notre résumé CMSClass et le groupage MODL car pour ces jeux de données les attributs catégoriels 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 GKClass.

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 modèle local soit avec un classifieur bayésien naïf dans chaque feuille. Le tableau 4.4 présente une vue synthétique des algorithmes présentés ci-dessous :

- **HT : Arbre de Hoeffding.** Cet algorithme correspond à l'arbre de Hoeffding de VFDT [DH00]. Les résumés numériques sont basés sur l'estimation de densité par une Gaussienne par classe (évalué comme étant le plus performant dans [PHK08]). 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 de 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.

3. `http://chercheurs.lille.inria.fr/salperwy/index.php?id=moa-extension`

- **HTmGKc : Arbre de Hoeffding avec critère de coupure MODL.** Cette version correspond à notre proposition. Le résumé pour les variables numériques est notre résumé de quantiles GKClass. Chaque variable est résumée par 10 tuples. Le critère de coupure utilisé est basé sur le critère MODL décrit section 4.3.1.2. Les 10 coupures binaires évaluées proviennent des quantiles du résumé GK. Cette version ne possède pas de modèle local dans les feuilles.
- **HTmGKcNBm : Arbre de 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 de notre résumé à 2 niveaux GKClass 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	GKClass 10 tuples	Level MODL	Aucune	Aucun
HTmGKcNBm	GKClass 10 tuples	Level MODL	MODL sur GKClass	NB

TABLE 4.4: Spécifications des différents algorithmes testés (NB : bayésien naïf).

4.3.2.2 Jeux de données utilisés

Inadéquation des bases du challenge Large Scale Learning avec les arbres

La comparaison des divers algorithmes se fait avec des jeux de données différents de la section précédente. En effet les données précédentes fonctionnent bien avec des classifieurs style SVM ou bayésien naïf moyenné [4] mais pas les arbres. On observe qu'il faut beaucoup de données pour que le classifieur bayésien naïf ait de bons résultats sur ces jeux de données. Les arbres de Hoeffding découpent l'espace de recherche ce qui implique que les classifieurs locaux ne s'entraînent que sur une partie des données. De plus les données arrivant au classifieur local correspondent seulement aux données vues après que la coupure soit effectuée. Par conséquent ces classifieurs locaux n'apprennent que sur une petite partie des données. La Figure 4.4 montre sur la base gamma l'évolution du nombre de feuilles de l'arbre et sa performance au fur et à mesure de l'apprentissage. On observe clairement une baisse des performances après l'ajout de nouvelles feuilles dans l'arbre. Selon le moment auquel on observe les résultats (longtemps après la création de nouvelles feuilles ou pas) les résultats sont très variables rendant une comparaison des algorithmes difficile. Les arbres en-ligne ne sont clairement pas adaptés à ce jeu de données dont la volumétrie n'est pas suffisante au vu de l'espace de recherche (500 à 2000 variables explicatives).

Générateurs de données pour tester les arbres

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. Ces générateurs sont utilisés dans la littérature pour tester les arbres de Hoeffding. On retrouve les publications correspondantes à ces générateurs dans l'état de l'art présenté dans la section 4.3.2.2. Pour nos expérimentations nous allons utiliser les générateurs suivants :

4. http://largescale.ml.tu-berlin.de/submission/

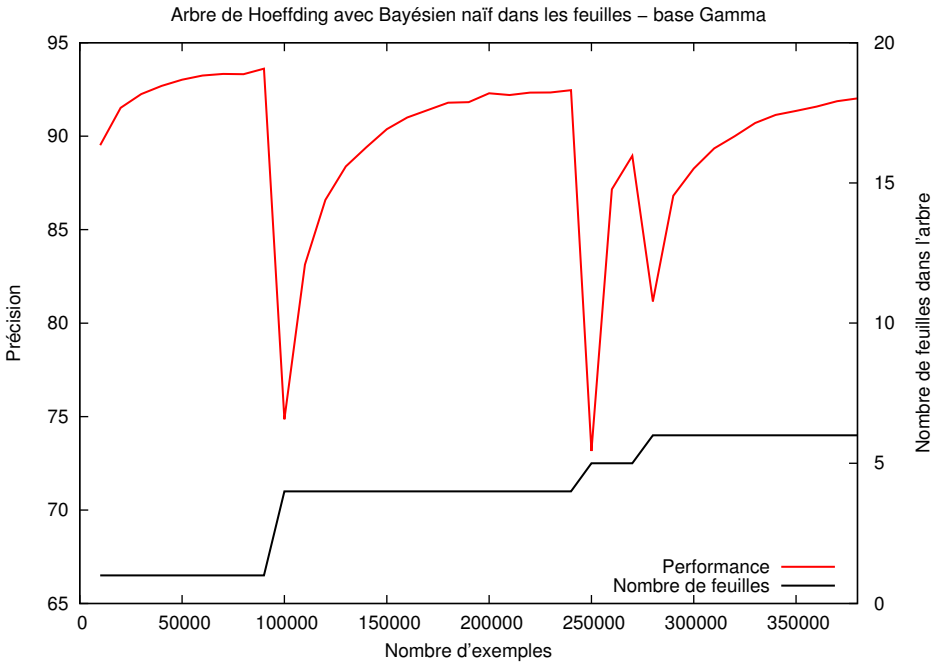


FIGURE 4.4 – Mise en évidence de l'inadéquation des arbres de Hoeffding avec les bases du challenge Large Scale Learning (base Gamma).

- Random RBF (Radial Basis Function) : plusieurs centroïdes définis par leur centre et leur diamètre sont tout d'abord générés à différentes positions dans l'espace. Puis des exemples appartenant à ces sphères sont générés en s'éloignant du centre d'une distance générée aléatoirement à partir d'une loi gaussienne. Une classe est affectée à chaque centroïde. Ce générateur ne contient que des attributs numériques. Pour les expériences 1000 centres et 50 attributs sont choisis.
- 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. Ce jeu de donnée à deux classes possède 10 attributs numériques et 10 attributs nominaux (5 valeurs différentes). L'arbre généré pour les expériences a 509 feuilles.
- 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. Les données comportent 21 attributs numériques.
- Fonction 6 (F6) d'Agrawal : 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 proposée par Agrawal avec 5% de bruit.

Chaque générateur crée un flux de 11 millions d'exemples. Parmi les méthodes d'évaluation présentées section 2.5, 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.

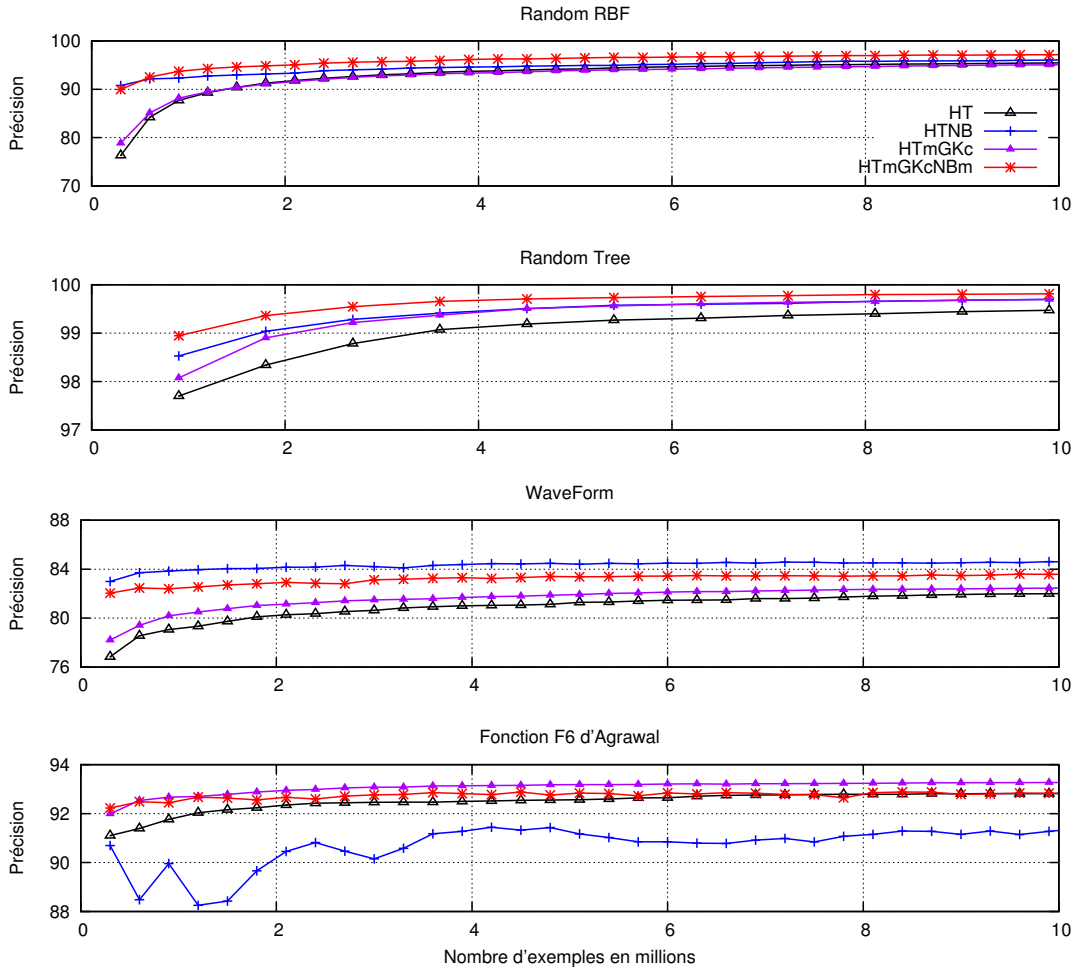


FIGURE 4.5 – Précision des différents arbres de Hoeffding expérimentés en fonction du nombre d'exemples appris.

4.3.2.3 Résultats

Les résultats de nos expérimentations sont présentés dans la figure 4.5 sous la forme de quatre courbes par jeu de données. 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 (Agrawal 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 due 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 le flux de données « Waveform ». 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 du générateur « Waveform » qui utilise des gaussiennes pour générer les données.

4.3.3 Perspective : gestion de la mémoire

Une piste d'amélioration pour nos arbres de décisions serait la gestion de la consommation mémoire. Celle-ci existe déjà dans VFDT [DH00] et Domingos et Hulten proposent de détecter les attributs les moins prometteurs et de ne plus stocker les comptes correspondants. Ils proposent aussi de conserver en priorité les parties de l'arbre qui ont le plus de chance de se développer en observant celles qui ont le plus grand nombre d'erreurs, un nombre d'erreurs plus important laissant supposer que l'arbre peut se développer pour s'améliorer. Nos modifications apportées aux arbres de Hoeffding permettent d'utiliser ces méthodes de la même manière que celles proposées dans VFDT. Cependant on peut proposer d'utiliser le critère MODL de discrétisation/groupage pour évaluer l'informativité d'une variable et donc savoir si l'on souhaite conserver son résumé.

4.3.4 Une autre application des arbres et résumés

Le challenge ICML 2011 Exploration & Exploitation[5], de part ses données réelles et ses contraintes, nous a semblé pertinent pour tester nos méthodes.

Les données de ce challenge proviennent de données réelles fournies par Adobe pour un site d'affichage de contenus. Celles-ci sont très déséquilibrées (la classe minoritaire représente 0,24% des données) et contiennent beaucoup de bruit. Ce jeu de données est constitué de trois millions d'exemples. Chaque exemple est décrit par 100 valeurs numériques et 19 valeurs catégorielles et est étiqueté par un booléen (clic/non clic). Le but de ce challenge est d'évaluer des algorithmes en-ligne pour la sélection de contenu. Les algorithmes doivent réaliser une séquence d'itérations. Pour chaque itération, une liste de 6 paires (visiteurs, contenu) est donnée à l'algorithme qui doit choisir quelle paire a la plus de chance d'être cliquée. De part sa nature en-ligne, ce challenge impose de fortes contraintes : (i) 100 ms par itération (pour traiter les 6 paires), (ii) 1,7G octets de mémoire maximale utilisable.

Les spécificités de ce challenge nous ont obligé à adapter notre méthode de résumés à deux niveaux. Le jeu de données est très déséquilibré et contient peu de clics. Les méthodes comme MODL sont très robustes mais cette robustesse entraîne une découverte tardive des points de coupure de l'arbre. Cela se traduit par une performance très faible sur les 200 000 premières itérations comme le montre la courbe verte (MODL) de la figure 4.6. Le critère d'évaluation du challenge est le nombre cumulé de clics bien prédits et impose au modèle d'être capable de prendre des décisions rapidement. Attendre pour trouver de meilleurs points de coupure peut permettre d'avoir un meilleur modèle en fin d'évaluation, mais résultera en un score plus faible sur le critère d'évaluation de ce challenge. Pour être plus réactif, nous avons choisi de construire, directement sur le premier niveau de résumé, des arbres renvoyant des probabilités (PETs : Probabilities Estimation Trees [PD03]). Ces arbres, construits sur une seule variable et de manière supervisée, peuvent être vus comme une discrétisation et donc comme un second niveau différent de la discrétisation MODL. La prédiction finale est réalisée en moyennant la prédiction de ces arbres pour chaque variable. La figure 4.6 montre les résultats de ce challenge : Inria (premier avec

5. `http://explo.cs.ucl.ac.uk/`

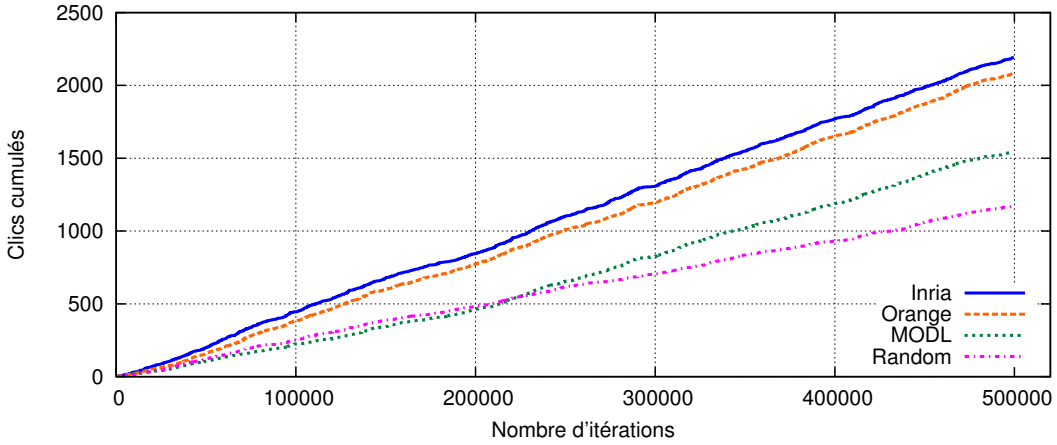


FIGURE 4.6 – Résulats du challenge Exploration & Exploitation ICML'11.

Olivier Nicol et Jérémie Mary [OJP12]), notre soumission (deuxième [SU12]) et un prédicteur choisissant de manière aléatoire la paire. Les détails de nos expérimentations ont été publiés dans [SU12]. Cet article contient des expérimentations similaires mais sur des données provenant des serveurs de publicités d'Orange. Cette publication est présente dans l'annexe C.

Les résultats de ce challenge confirment les performances en-ligne et la robustesse de nos méthodes de résumé ainsi que celles de nos classifieurs à base d'arbres.

4.4 Conclusion

Ce chapitre présente nos adaptations de classifieurs existants hors-ligne aux flux de données. Nos adaptations se basent sur nos résumés en-ligne présentés dans le chapitre précédent. Ceux-ci permettent d'obtenir en-ligne les probabilités conditionnelles $P(X_i|C)$ robustes. Deux classifieurs ont été adaptés : le classifieur bayésien naïf et les arbres de décisions.

Les probabilités en entrée du classifieur bayésien naïf ont été adaptées afin de rendre ce classifieur incrémental. Les probabilités conditionnelles $P(X_i|C)$, qui sont habituellement estimées par des méthodes hors-ligne telles que « Equal Frequency », MDLP, MODL..., sont remplacées par celles provenant de nos résumés. On observe que pour une taille de résumé suffisante les performances de la version adaptée pour les flux est similaire à la version hors-ligne. Si la mémoire disponible est suffisante les résumés sont capables de modéliser finement les probabilités conditionnelles aux classes. Les expérimentations sur les bases du challenge Large Scale Learning montrent aussi tout l'intérêt du second niveau. Celui-ci permet d'assurer la robustesse des estimations. Son apport est particulièrement remarquable pour les résumés de taille moyenne à grande dont le premier niveau ne permet pas d'obtenir une estimation robuste. L'utilisation seule du premier niveau dégrade fortement les performances du classifieur bayésen naïf. Le second niveau basé sur l'approche MODL permet de fusionner les tuples du premier niveau et d'obtenir de meilleures estimations. L'approche MODL ne nécessitant pas de paramètre, ces bonnes performances sont accessibles sans réglage préalable.

Le classifieur bayésien naïf moyenné diffère du classifieur bayésien naïf par l'ajout d'un poids sur chacune des variables. L'estimation de ces poids se fait habituellement hors-ligne. Une solution basée sur une descente de gradient stochastique est proposée afin de réaliser l'optimisation de ces poids en-ligne. Cette nouvelle méthode nécessite beaucoup de données pour converger vers

les poids optimaux. Sur certains jeux de données, cette pondération en-ligne améliore les résultats par rapport à la version non moyennée. Cependant sur d'autres jeux, cette méthode a des performances inférieures au classifieur bayésien naïf non moyenné. Par conséquent son utilisation dépend de la quantité de données disponible et de la rapidité à laquelle les poids peuvent être appris.

Pour les arbres de décision en-ligne, le choix s'est porté sur les arbres de Hoeffding largement utilisés dans le cadre de l'apprentissage en-ligne sur flux de données. Ceux-ci sont adaptés afin qu'ils puissent utiliser les probabilités conditionnelles $P(X_i|C)$ de nos résumés. Notre proposition porte sur les différents éléments qui les composent : (i) les résumés dans les feuilles, (ii) le critère de coupure, (iii) le modèle local. Sur les jeux de données issus des générateurs utilisés par la communauté nos modifications améliorent les résultats. Tout particulièrement le classifieur bayésien naïf, utilisé comme modèle local, qui tire avantage de la robustesse de notre résumé à deux niveaux. Notre résumé renvoie la classe majoritaire s'il n'y pas assez de données pour réaliser une meilleure prédiction. Ce comportement permet à l'arbre d'utiliser le modèle local seulement si celui-ci est plus performant que la prédiction de la classe majoritaire.

L'utilisation de nos résumés en-ligne sur ces deux classifieurs permet de les rendre incrémentaux et en-ligne de manière efficace. Cependant ces classifieurs sont prévus pour apprendre sur un flux stationnaire. Les flux de données sont rarement indéfiniment stationnaires mais peuvent varier au cours du temps. On parle alors de changement de concept (voir section 2.4). Différentes techniques existent pour traiter cette problématique : détecteur qui permet d'apprendre un nouveau classifieur, ensemble de classifieurs qui se renouvellent... Le chapitre suivant propose une nouvelle technique de détection de changement des données en observant les variations de distribution du flux de données.

Chapitre 5

Détection de changement dans un flux de données

Sommaire

5.1 Introduction

Les chapitres 3 et 4 ont respectivement présenté des méthodes capables de réaliser des résumés et des modèles de classification sur des flux de données. Dans ces deux chapitres, les flux utilisés sont stationnaires car ces méthodes fonctionnent de manière optimale sur des données sans changement de concept (voir section 2.4). Or comme l'indique [HSD01], l'apprentissage sur les flux nécessite d'être capable de traiter les flux avec changements de concept. Certains classifieurs sont naturellement capables de s'adapter aux changements. On peut, par exemple, citer les réseaux de neurones, les SVM linéaires utilisant la descente stochastique de gradient,... Les méthodes que nous avons proposées dans les chapitres 3 et 4 ne possèdent aucun mécanisme naturel pour s'adapter aux changements de concept dans le flux. Par conséquent, elles ont besoin d'un mécanisme de gestion du changement de concept.

L'état de l'art, présenté dans la section 2.4, propose différentes techniques pour la gestion du changement de concept dans un flux de données. Parmi celles-ci on trouve des méthodes basées sur des détecteurs, des ensembles de classifieurs ou des fenêtres/pondérations d'exemples. Dans ce chapitre nous nous intéressons uniquement à la problématique de la détection de changement. Cette détection peut être réalisée en observant les performances du modèle, la distribution des exemples ou l'évolution du modèle (voir section 2.4). La méthode que nous proposerons au cours de ce chapitre a pour but de détecter les changements de distribution dans les exemples. Mais nous nous comparerons aussi aux méthodes détectant les changements de performance du modèle.

Dans ce chapitre, nous proposons une nouvelle méthode de détection des changements basée sur l'observation des variables en entrée du classifieur. Notre méthode utilise deux fenêtres et permet d'identifier si les données de ces deux fenêtres proviennent ou non de la même distribution. Notre méthode a l'intérêt de n'avoir aucun *a priori* sur la distribution des données, ni sur le type de changement (graduel, soudain, incrémental - voir section 2.4). De plus, à part la taille des fenêtres, elle ne requiert aucun paramètre utilisateur.

Plan du chapitre

La deuxième section de ce chapitre positionne la méthode proposée vis-à-vis des méthodes de l'état de l'art. Elle formalise par la même occasion la détection du changement sur la distribution des données d'un point de vue bayésien. Dans la section 3, nous décrivons notre nouvelle méthode de détection basée sur deux fenêtres. Les changements de distribution entre les deux fenêtres sont détectés à l'aide de l'approche MODL. Nous proposons d'utiliser un critère bivarié afin d'être capables d'observer les changements dans la distribution conditionnellement à la classe. La section 4 montre, sur des exemples artificiels, que notre méthode est capable de détecter tous types de changement tout en étant robuste. Dans la section 5, nous proposons une méthode utilisant un classifieur pour comparer notre approche avec des méthodes basées sur la performance du classifieur. Les données de ces expérimentations proviennent de deux générateurs de la littérature dont la vitesse de changement est paramétrable. La dernière section conclut ce chapitre en récapitulant les apports de notre méthode et son intérêt dans le cadre du changement de concept sur les flux de données.

5.2 Positionnement de notre méthode de détection

Cette section présente tout d'abord le cadre et les changements que l'on souhaite détecter. Puis les méthodes de l'état de l'art, adaptées à ce cadre, sont décrites. Finalement, les caractéristiques des méthodes présentées sont synthétisées dans un tableau. Celui-ci décrit par la même occasion les caractéristiques de la méthode que l'on souhaite développer.

5.2.1 Changement de concept d'un point de vue bayésien

5.2.1.1 Cas général

Dans ce chapitre nous nous intéressons seulement aux méthodes de détection de changement qui sont basées sur l'observation des données X du flux. La détection se fait à partir de l'observation de deux distributions. La première distribution provient du concept de départ et la deuxième du concept actuel. Une détection a lieu si on observe une différence significative entre les distributions de ces deux concepts. Si les différences ne sont pas jugées significatives alors le flux est considéré comme stationnaire.

Nous choisissons d'observer ces changements d'un point de vue bayésien. D'après la formule de Bayes la probabilité d'une classe C conditionnellement aux données X est la suivante :

$$P(C|X) = \frac{P(C)P(X|C)}{P(X)}$$

Un changement dans chacun des trois termes peut faire varier $P(C|X)$ au cours du temps :

1. $P(C)$: correspond à la proportion des classes dans les données. Celle-ci peut évoluer au cours du temps.
2. $P(X)$: probabilité des données. Leur distribution peut évoluer dans le temps. On parle alors de covariate shift [QCSSL09].
3. $P(X|C)$: changement de la probabilité des X connaissant la classe C.

On retrouve une classification similaire (changements 1 et 3) dans [KHA99] ainsi que des expérimentations montrant l'impact de chacun des changements. Le but de certain classifieur est justement d'essayer d'apprendre $P(C|X)$. Par exemple, un classifieur bayésien naïf réalise son apprentissage en estimant au mieux $P(C)$ et $P(X_i|C)$. Des classifieurs comme les arbres vont segmenter l'espace pour réaliser ces estimations dans des sous-espaces (les feuilles de l'arbre).

5.2.1.2 Hypothèse naïve d'indépendance des variables

La probabilité conditionnelle jointe $P(X|C)$ étant difficilement estimable, nous utilisons la version naïve qui a pour hypothèse l'indépendance des variables X_i. Cette hypothèse fait qu'il ne sera pas possible de voir certaines interactions entre les variables pour la détection de changement. Cependant cette hypothèse a montré sur des problématiques réelles qu'elle n'empêchait pas d'avoir de bonnes performances [HY01]. Si nous reprenons la formule de Bayes et que nous faisons l'hypothèse de l'indépendance des variables nous obtenons :

$$P(C|X) = \frac{P(C)\prod_i^d P(X_i|C)}{P(X)}$$

On s'intéressera donc par la suite à la probabilité $P(X_i|C)$ au lieu de la probabilité $P(X|C)$. Pour les changements sur $P(X)$, on retrouve le problème du nombre d de dimensions que peut

avoir X. Si d est grand il est difficile de trouver des méthodes capables de détecter un changement sur la distribution de X avec une faible complexité. On choisit donc de reprendre l'hypothèse naïve et d'observer les changements variable par variable, c'est-à-dire sur les d probabilités $P(X_i)$. Les changements sur $P(C)$ ne posent pas de problèmes particuliers et peuvent être traités assez simplement avec, par exemple, un test du χ^2.

5.2.1.3 Changements sur $P(X_i|C)$

La détection de changement sur $P(X_i|C)$ implique d'avoir une méthode capable d'observer une distribution conditionnellement aux classes. On pourrait aussi imaginer observer la distribution pour chaque classe k prise indépendamment $P(X_i|C_k)$, mais cette solution pose d'autres problèmes :

1. Les distributions des classes les unes par rapport aux autres ne sont sans doute pas indépendantes. La détection sur une classe entrainera sans doute des détections sur les autres classes.
2. Si le nombre de classes est important, alors le nombre de tests de détection va augmenter. Par conséquent le risque de réaliser de fausses détections va aussi augmenter.
3. Si les classes sont déséquilibrées alors certains tests seront basés sur moins de données et seront donc moins robustes et/ou moins sensibles. De plus, si on souhaite avoir une mesure de la quantité de changement, il faudra alors trouver un moyen d'agréger de manière pertinente ces différentes mesures.

Il paraît donc peu intéressant d'observer les $P(X_i|C_k)$. Une méthode capable d'évaluer un changement sur $P(X_i|C)$ dans sa globalité ne pose pas ces problèmes et semble être plus adéquate.

5.2.2 État de l'art des approches de détection

L'état de l'art sur la détection de changement est abondant. On retrouve dans [Kun11, BK09] de nombreuses références et une présentation des méthodes dédiées à une utilisation sur les flux de données. De nombreux tests statistiques [Kan06] existent aussi pour comparer deux distributions. Nous présentons dans cette section uniquement des méthodes de l'état de l'art qui permettent de détecter des changements sur les probabilités $P(X_i)$. A notre connaissance, aucune méthode de détection sur les probabilités $P(X_i|C)$ adaptée aux flux de données n'existe dans la littérature. Les méthodes présentées dans cette section ne s'appliquent donc qu'à la détection de changement sur les probabilités $P(X_i)$.

5.2.2.1 CUSUM

L'algorithme CUSUM (CUmulative SUM) a été proposé dans [Pag54] puis réutilisé dans le cadre des flux par Gama dans [GRSR09]. Cet algorithme observe une variable ϵ_t et déclenche une alerte quand sa moyenne est significativement différente de zéro. La variable observée peut être, par exemple, l'écart à la moyenne comme dans [GRSR09]. Son fonctionnement est le suivant : soit $S_0 = 0$ et $S_{t+1} = max(0, S_t + \epsilon_t - v)$. Une détection correspond à avoir $S_t > h$ avec h le seuil de détection.

L'intérêt de ce test est qu'il ne nécessite pas de mémoire. Par contre il nécessite de fixer les paramètres v et h.

5.2.2.2 ADWIN

La méthode proposée dans [BG07] se base sur des fenêtres dont la taille s'adapte (ADaptative WINdowing). Cette méthode observe la moyenne des valeurs d'un flux. La taille de la fenêtre est variable et s'adapte de manière à avoir la meilleure estimation de la moyenne qui soit consistante avec l'hypothèse qu'il n'y a pas eu de changement dans la fenêtre. Si aucun changement ne se produit alors la taille de la fenêtre augmente et elle diminue lorsque des changements sont détectés. La méthode possède un paramètre δ qui définit la confiance désirée dans le résultat.

5.2.2.3 Test t de Welch

Le test t de Welch est une adaptation du test t de Student qui s'applique à deux échantillons X_1 et X_2 de tailles respectives N_1 et N_2. Ce test permet de tester statistiquement l'hypothèse d'égalité de deux moyennes ($\overline{X}_1$ et $\overline{X}_2$) avec deux échantillons de variances inégales (s_1^2 et s_2^2). La formule du t est la suivante :

$$pvalue = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

On utilise le test t de Welch pour tester l'hypothèse nulle suivante : « les moyennes de deux populations sont égales ». Ce test retourne une *pvalue* qui permet de rejeter ou non l'hypothèse nulle. Ce test a déjà été utilisé dans le cadre des flux de clics pour identifier les revisites de pages Web [Yam08].

5.2.2.4 Test de Kolmogorov-Smirnov

Le test d'hypothèse de Kolmogorov-Smirnov est utilisé pour déterminer si un échantillon suit bien une loi donnée ou bien si deux échantillons suivent la même loi. Ce test est basé sur les propriétés des fonctions de répartition empirique. Nous utiliserons ce test pour vérifier si deux échantillons suivent la même loi. Soient deux échantillons de tailles N_1 et N_2 possédant respectivement les fonctions de répartition empirique $F_1(x)$ et $F_2(x)$. La distance de Kolmogorov-Smirnov est définie de la manière suivante :

$$D = \max_x |F(x) - F_2(x)|$$

L'hypothèse nulle, stipulant que les deux échantillons proviennent de la même distribution, est rejetée avec une confiance α si :

$$\sqrt{\frac{N_1 N_2}{N_1 + N_2}} D > K_\alpha$$

K_α se retrouve à l'aide des tables de Kolmogorov-Smirnov [6].

5.2.2.5 Discrétisation MODL pour la détection des changements de distribution

Cette méthode a été proposée par Bondu et Boullé [BB11] pour la détection de changement dans le cadre de l'observation d'une variable numérique. Elle voit le problème de la détection de changement comme un problème d'apprentissage à deux classes. La discrétisation MODL [Bou06a] est utilisée comme méthode de détection de changement.

6. http://www.eridlc.com/onlinetextbook/appendix/table7.htm

Fonctionnement Cette méthode utilise deux fenêtres pour détecter un changement La première, appelée fenêtre de référence W_{ref}, contient les données du concept de départ. La deuxième, appelée fenêtre courante W_{cur}, est une fenêtre glissante/sautante sur le flux. Les exemples de la fenêtre de référence ont comme étiquette la classe « ref », et ceux de la fenêtre courante la classe « cur ». Ces deux étiquettes constituent la variable cible $W \in \{W_{ref}, W_{cur}\}$ du problème d'apprentissage. Ces deux fenêtres sont fusionnées et la discrétisation MODL appliquée sur ces données. Si la discrétisation MODL arrive à discrétiser la variable X_i en au moins 2 intervalles cela signifie qu'il y a au moins 2 zones où la distribution des exemples est significativement différente entre les deux fenêtres. Dans ce cas un changement a été observé dans les $P(W|X_i)$ et on peut donc conclure qu'un changement s'est produit.

Cette méthode s'applique sur une variable numérique ou catégorielle. Si le flux contient plusieurs variables alors la méthode est appliquée à chacune des variables. Cette méthode s'intéresse aux changements de régime d'une variable numérique et ne peut fonctionner pour détecter un changement de concept dans un flux de données étiquetés.

5.2.3 Discussion

Le tableau 5.1 présente une synthèse des méthodes décrites ci-dessus. On note que (i) les méthodes Kolmogorov-Smirnov et MODL, ne font pas d'hypothèses sur la distribution des données, (ii) seule la méthode basée sur l'approche MODL n'a pas de paramètres utilisateur, (iii) aucune des méthodes présentées n'est capable de détecter des changements sur $P(X_i|C)$.

Aucune de ces méthodes ne possède toutes les propriétés désirées. Nous proposons donc, dans la section suivante, une nouvelle méthode ayant toutes ces propriétés : (i) pas d'*a priori* sur la distribution des données, (ii) sans paramètre utilisateur et (iii) capable de détecter les changements de probabilités $P(X_i|C)$.

	CUSUM	ADWIN	Welch	KS	MODL	Méthode désirée
Hypothèse sur la distribution des données	dépend de la variable observée	oui	oui	non	non	non
Paramétres utilisateur	oui	oui	oui	oui	non	non
Détection des changements sur $P(X_i\|C)$	non	non	non	non	non	oui

TABLE 5.1 – Synthèse des méthodes de détection.

5.3 Une nouvelle méthode de détection de changement

La méthode proposée dans cette section utilise uniquement les données étiquetées provenant du flux et ne nécessite pas de classifieurs. Elle cherche à distinguer si la distribution des données dans deux fenêtres d'observation a changé.

5.3.1 Une nouvelle méthode à deux fenêtres

5.3.1.1 Présentation de l'approche

Notre idée pour détecter les changements de concept s'inspire de la méthode proposée par Bondu et Boullé [BB11] présentée dans la section 5.2.2.5. De la même manière que pour la méthode de Bondu et Boullé, on observe les variables sur deux fenêtres. La première, appelée fenêtre de référence W_{ref}, contient les données du concept de départ. La deuxième, appelée fenêtre courante W_{cur}, est une fenêtre glissante/sautante sur le flux qui permet de capturer les données d'un éventuel nouveau concept. Les données sont étiquetées par fenêtre : $W \in \{W_{ref}, W_{cur}\}$ de manière similaire à la méthode de Bondu et Boullé.

Les variables que nous observons sont donc :

1. $P(W|C)$: la probabilité de la fenêtre connaissant la classe ;
2. $P(W|X_i)$: la probabilité de la fenêtre connaissant X_i ;
3. $P(W|C, X_i)$: la probabilité de la fenêtre connaissant à la fois la classe C et la variable X_i.

5.3.1.2 Un nouveau critère bivarié pour la détection en classification

Pour les variables $P(W|C)$ et $P(W|X_i)$ on peut respectivement utiliser la méthode de groupage MODL et la méthode de discrétisation MODL (voir section 3.3.3) comme proposé par Bondu et Boullé. Par contre pour la variable $P(W|C, X_i)$, qui dépend à la fois de la variable X_i et de la classe, leur méthode ne peut s'appliquer. Dans ce cas il faut utiliser une méthode de discrétisation/groupage bivarié capable de prendre en compte la variable X_i et la classe. Cette extension au cas bivarié de l'approche MODL existe déjà et a prouvé son efficacité dans [Bou09, Bou07b].

Le critère bivarié MODL utilise un formalisme similaire au critère univarié et devient pour une variable numérique (X_i) et la variable catégorielle représentant la classe (C) avec comme classe à prédire W :

$$C_{discBivarié} = \tag{5.1}$$

$$logN + \tag{5.2}$$

$$log\binom{N+I-1}{I-1} + \tag{5.3}$$

$$logV + \tag{5.4}$$

$$logB(V,G) + \tag{5.5}$$

$$\sum_{g=1}^{G}\sum_{i=1}^{I} log(c_{gi}+1) + \tag{5.6}$$

$$\sum_{g=1}^{G}\sum_{i=1}^{I} log\frac{c_{gi}!}{c_{gi1}!c_{gi2}!\ldots c_{giK}!} \tag{5.7}$$

avec

- N : taille des données (nombre d'individus)

- I : nombre d'intervalles de la variable numérique (inconnu)
- G : nombre de groupes de la classe (inconnu)
- V : nombre de valeurs de la classe
- c_{gi} : nombre d'individus dans la cellule gi
- c_{gik} : nombre d'individus dans la cellule gi pour la fenêtre k
- $B(V,G)$ correspond à une somme de nombre de Stirling de deuxième espèce (nombre de partitions de V valeurs en groupes non vides).

Les termes (5.2) à (5.5) correspondent au choix du modèle de discrétisation/groupage. C'est-à-dire le choix des intervalles I pour la variable numérique X et le choix des groupes G pour la variable catégorielle C. Le terme (5.6) représente le choix des distributions dans chaque cellule. Le dernier terme (5.7) exprime la probabilité d'observer ces distributions dans les données (vraisemblance) connaissant le modèle.

Si la variable observée n'est pas numérique mais catégorielle alors le critère MODL bivarié pour une variable catégorielle est utilisé. Celui-ci est très similaire au critère présenté ci-dessus.

La complexité temporelle de l'algorithme naïf pour trouver le meilleur modèle est en N^5, mais à l'aide d'heuristiques d'optimisation [Bou09], cette complexité est ramenée à $N\sqrt{N}log(N)$ avec N la somme de la taille de nos deux fenêtres (référence et courante).

Interprétation du gain de compression MODL

Le critère l d'évaluation MODL a été présenté dans la section 4.2.2.1. Il s'applique aussi bien pour la discrétisation ($P(W|X_i)$), le groupage ($P(W|C)$) ou un modèle bivarié ($P(W|C,X_i)$). Quand la variable à expliquer (dans notre cas W) ne peut être distinguée à partir des différentes variables que nous observons alors $l = 0$. Si $l > 0$ alors cela signifie que la variable observée permet de distinguer la variable à expliquer (le fenêtre W dans notre cas) et donc qu'un changement a été détecté. La valeur l représente la quantité de changement.

Changement non détecté : le cas du XOR qui s'inverse brutalement La méthode MODL ne fait pas d'hypothèse sur la distribution des données et est capable de différencier tous types de changements (moyenne, variance...) intervenus dans une variable. La grande majorité des changements sur plusieurs variables induisent un changement sur au moins une des variables et seront donc détectés.

Cependant, l'hypothèse d'indépendance des variables de notre méthode fait qu'elle ne sera pas capable de détecter un changement brutal impliquant simultanément deux variables X_1 et X_2. Il n'y aura pas de détection dans le cas où il n'y a pas de changement sur les $P(C)$, $P(X_1)$, $P(X_2)$, $P(X_1|C)$ et $P(X_2|C)$ mais seulement un changement sur $P(X_1,X_2|C)$. La figure 5.1 illustre cette inversion soudaine des classes sans aucun autre changement. Elle montre que si le changement est brutal et complet il n'y a pas de détection. Au contraire, s'il y a une phase de transition, ne serait-ce que minime, alors notre méthode détecte un changement. Nous considérons qu'un changement soudain de cette nature a peu de chance de se produire dans des données réelles, et donc que notre méthode est capable de traiter un très grand nombre de changements.

5.3.1.3 Deux fenêtres contenant les données

Notre méthode implique l'utilisation de deux fenêtres. La première W_{ref} contient les données du concept de départ. La deuxième W_{cur} conserve les données récentes. Le réglage de ces fenêtres a plusieurs implications.

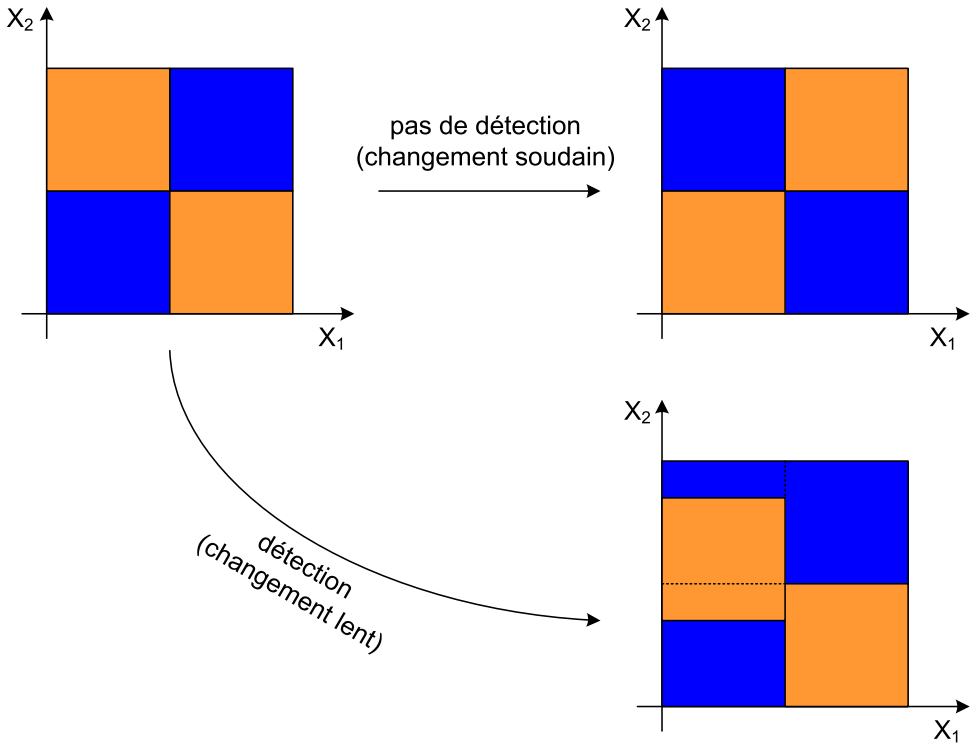


FIGURE 5.1 – Cas du XOR où notre méthode n'arrive pas à détecter un changement soudain mais seulement lent.

Taille des fenêtres Elle impacte la réactivité aux changements ainsi que le nombre de mauvaises détections. Une grande taille de fenêtre permet de détecter avec plus de confiance ainsi que des motifs plus complexes. Une plus petite taille permet d'être plus réactif. Fixer cette taille dépend du flux observé et des changements que l'on veut détecter. De manière générale on peut traiter ce problème en utilisant plusieurs tailles de fenêtre en parallèle. Cette problématique de la taille des fenêtres a fait l'objet de nombreuses publications (voir section 2.4.2) et ne sera pas traitée de manière approfondie dans ce chapitre.

Positionnement des fenêtres Il est possible d'avoir différentes implémantations pour le positionnement des fenêtres sur le flux.

1. Une première implémentation serait d'avoir deux fenêtres qui se suivent. Cette solution peut cependant ne pas détecter des changements progressifs lents. En effet la différence de distribution entre les deux fenêtres sera faible et les méthodes de détection, censées être robustes, interprèteront probablement la différence comme du bruit. Ce cas se produira d'autant plus que la taille des fenêtres sera petite.
2. Une seconde implémentation consiste à avoir une fenêtre de référence positionnée au début du flux et repositionnée à la position de la détection après une détection. La fenêtre courante est glissante (ou sautante) avec l'arrivée des exemples du flux. Si un changement progressif lent se produit alors les méthodes de détection observeront de plus en plus de différence et finiront pas détecter un changement entre les deux fenêtres.
3. On pourrait aussi imaginer, pour la fenêtre de référence, de réutiliser le premier niveau de nos résumés (voir section 3.3.2) pour avoir des tailles de fenêtre encore plus grande sans

avoir besoin de stocker toutes les données.

Nous choisissons d'utiliser la seconde proposition, c'est-à-dire une fenêtre de référence dont la position est fixe et une fenêtre courante glissante (ou sautante) avec le flux.

5.3.2 Résumé et discussion

Dans cette section, nous avons étendu à la classification supervisée la méthode de détection de Bondu et Boullé basée sur l'approche MODL. Cette méthode utilise deux fenêtres et fait l'hypothèse d'indépendance des variables. Elle est capable d'observer les changements apparus à la fois sur les probabilités $P(C)$ et $P(X_i)$ mais pas sur $P(X_i|C)$. Nous proposons d'utiliser le critère bivarié MODL pour prendre en compte les changemens sur les probabilités $P(X_i|C)$. L'intérêt de la méthode MODL réside dans sa robustesse (pas d'hypothèse sur la distribution des données et faible sensibilité au bruit) et son absence de paramètres utilisateur. Cependant, la taille de la fenêtre reste à fixer. Cette taille dépend de la complexité du changement que l'on veut détecter et de la réactivité désirée.

La suite de ce chapitre est composée de deux sections expérimentales évaluant notre méthode de détection. La première s'intéresse à la détection de changement dans un flux sans utiliser de classifieur. La deuxième propose un nouvel algorithme de remplacement du classifieur suite à une détection.

5.4 Expérimentations sans classifieur

Le but de cette section est de montrer, à l'aide d'exemples artificiels, les capacités de notre nouvelle méthode à détecter tous types de changements sans réaliser de fausses détections.

5.4.1 Méthodes comparées

Dans ces expérimentations nous nous intéressons aux changements sur les probabilités $P(X_i|C)$ et $P(X_i)$. Parmi les méthodes de l'état de l'art, nous choisissons différents types de méthodes capables de détecter des changements sur $P(X_i)$: une méthode détectant les changements dans les moyennes : le test t de Welch ; une méthode n'ayant pas d'*a priori* sur les distributions comparées : le test de Kolmogorov-Smirnov. Enfin, la méthode supervisée de Bondu et Boullé basée sur MODL $P(W|X_i)$. Pour les changements sur $P(X_i|C)$, seule notre méthode est utilisée car nous n'avons pas connaissance de méthodes capables de traiter ce genre de problème pour les flux.

Pour résumer, les méthodes suivantes sont comparées :

1. Test t de Welch avec signification statistique de 1%, 5% et 10%
2. Test de Kolmogorov-Smirnov (KS) avec signification statistique de 1%, 5% et 10%
3. Méthode supervisée MODL $P(W|X_i)$
4. Méthode supervisée bivariée MODL $P(W|X_i, C)$

5.4.2 Fausses détections et capacité à détecter

5.4.2.1 Fausses détections : comportement dans le cas stationnaire

Le but de cette première expérimentation est d'étudier le comportement des méthodes dans le cadre d'un flux stationnaire où l'on ne doit détecteur aucun changement.

méthode→ ↓ taille	Welch (1%)	Welch (5%)	Welch (10%)	KS (1%)	KS (5%)	KS (10%)	MODL $P(W\|X_a)$	MODL $P(W\|X_a, C)$
10	0	9	20	0	0	8	3	0
20	0	7	13	0	5	22	4	0
30	1	6	19	0	3	12	3	0
50	0	5	21	0	2	17	6	0
100	0	3	19	0	4	20	1	0
200	0	5	28	0	5	13	0	0
300	0	6	16	1	5	21	0	0
500	1	6	22	2	7	18	1	0
1000	1	8	25	0	5	24	0	0
2000	0	4	13	0	7	20	0	0
5000	0	6	26	0	7	19	0	0

TABLE 5.2 – Nombre de mauvaises détections selon la méthode détection et la taille de la fenêtre, pour 1000 expérimentations.

Protocole expérimental Les fenêtres de référence et courante ont la même taille. Une seule variable X_a est utilisée. La distribution des classes se fait selon une gaussienne de paramètres $(\mu_1 = -1, \sigma_1 = 1)$ pour la classe 1 et une gaussienne de paramètres $(\mu_2 = 1, \sigma_2 = 1)$ pour la classe 2. Différentes tailles de fenêtre ont été choisies entre 10 et 5 000 exemples. Les expérimentations sont réalisées 1000 fois.

Résultats Les résultats sont présentés dans le tableau 5.2. Ceux-ci confirment la robustesse de la méthode MODL car aucune détection n'est observée sur $P(W|X_a, C)$. Par contre quelques fausses alarmes ont eu lieu sur $P(W|X_a)$ mais dans moins de 1% des cas et pour de petites tailles de fenêtre. Les autres tests se comportent bien pour des petites valeurs de signification statistique (1%). Pour de plus grandes valeurs (10%), de 1% à 3% de fausses détections sont réalisées par le test t de Welch et le test de Kolmogorov-Smirnov.

5.4.2.2 Capacité à détecter un changement brutal

Le but de cette expérimentation est d'observer le temps nécessaire (en nombre d'exemples) pour détecter un changement en fonction de la taille de la fenêtre et des méthodes.

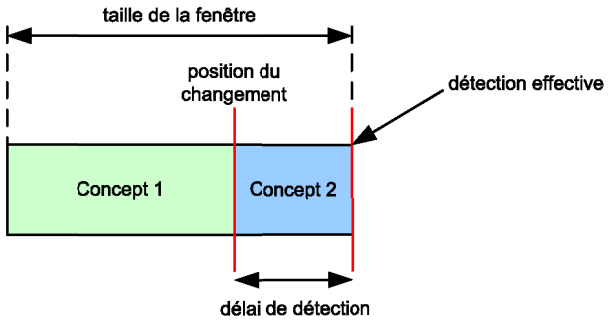
Protocole expérimental Les fenêtres de référence et courante ont la même taille. Une seule variable X_a est utilisée. Le concept 1 est défini ainsi :

- la classe 0 suit une distribution $\mathcal{N}_0(\mu = -1, \sigma = 1)$;
- la classe 1 suit une distribution $\mathcal{N}_1(\mu = 1, \sigma = 1)$.

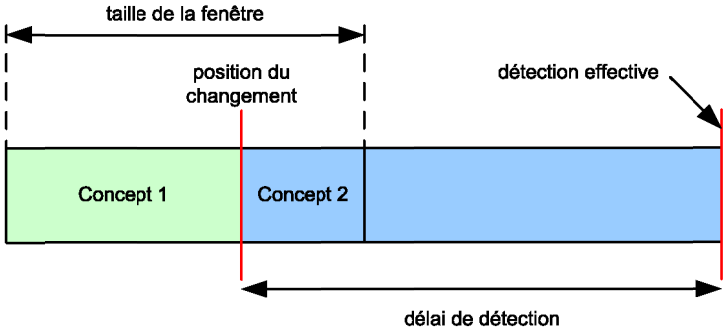
On simule ce changement en changeant la moyenne de la classe 0, qui passe de -1 à 2 et sa variance qui passe de 1 à 0,5. Pour la classe 1, seule la moyenne change en passant de 1 à 0. Ce changement est présenté dans la figure 5.6. On obtient donc le concept 2 suivant :

- la classe 0 suit une distribution $\mathcal{N}_0(\mu = 2, \sigma = 0,5)$;
- la classe 1 suit une distribution $\mathcal{N}_1(\mu = 0, \sigma = 1)$.

Différentes tailles de fenêtres ont été testées entre 10 et 5 000 exemples et les expérimentations sont réalisées 1000 fois. La position du changement dans la fenêtre est tirée de manière aléatoire car dans un cas réel la position du changement n'est pas connue. Les résultats obtenus correspondent aux délais moyens de détection du changement en fonction de la taille de la fenêtre et des méthodes étudiées. La figure 5.2 explique la méthode de calcul du délai de détection pour cette expérimentation.



(a) Détection dans la fenêtre où a lieu le changement.



(b) Détection dans la fenêtre suivant le changement.

FIGURE 5.2 – Calcul du délai de détection selon que la détection a lieu ou non dans la fenêtre du changement.

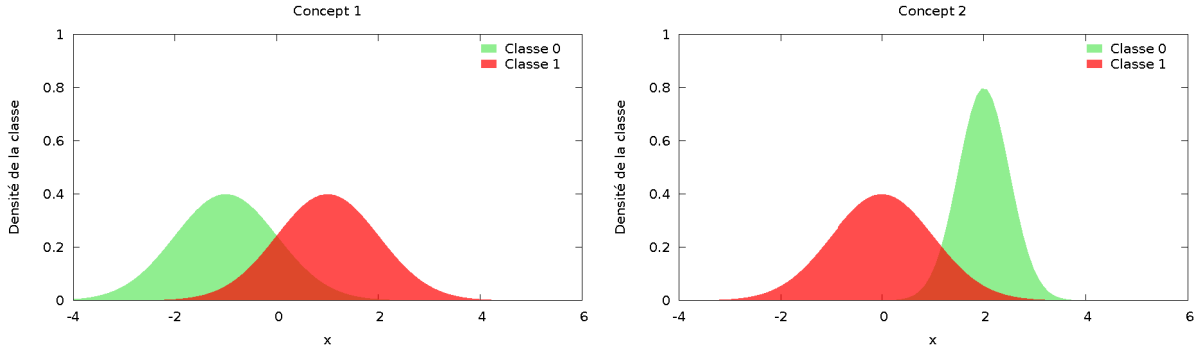


FIGURE 5.3 – Changement brutal : passage du concept 1 au concept 2.

méthode→ ↓ taille	Welch (1%)	Welch (5%)	Welch (10%)	KS (1%)	KS (5%)	KS (10%)	MODL $P(W\|X_a)$	MODL $P(W\|X_a, C)$
10	15	14	14	15	15	15	15	15
20	29	26	24	29	28	26	28	29
30	41	36	33	43	40	36	41	40
50	62	53	49	67	57	54	65	58
100	103	90	86	110	96	90	109	96
200	178	160	150	185	166	160	195	163
300	241	218	211	252	227	214	269	218
500	367	337	321	375	344	330	404	325
1000	665	620	598	678	636	613	719	600
2000	1224	1188	1154	1260	1188	1174	1334	1120
5000	2886	2766	2741	2911	2781	2756	3081	2671

TABLE 5.3 – Délai moyen de détection d'un changement brutal selon la méthode de détection et la taille de la fenêtre.

Résultats Les résultats sont présentés dans le tableau 5.3. On observe que jusqu'à une taille de fenêtre de 100 exemples la détection est difficile dans la fenêtre où a lieu le changement. A partir d'une taille de 200 exemples, le délai moyen est inférieur à la taille de la fenêtre et ce pour toutes les méthodes. L'augmentation du seuil de signification à 5% et 10% entraine, pour les tests de Welch et de Kolmogorov-Smirnov, une baisse du délai de détection. La méthode basée sur MODL $P(W|X_a)$ est légèrement plus longue à détecter que les méthodes paramétriques à 1%. Cependant, si on prend notre méthode MODL $P(W|X_a, C)$ alors celle-ci est meilleure que les deux autres méthodes paramétrées à 1%.

5.4.2.3 Conclusion

D'une part, notre méthode basée sur MODL $P(W|X_a, C)$ est robuste car elle ne réalise aucune fausse détection sur un flux stationnaire. Les méthodes basées sur le test de Welch et de Kolmogorov-Smirnov ont peu de fausses détections pour un seuil de significativité de 1%. Cependant, ces fausses détections augmentent rapidement avec l'augmentation du seuil. D'autre part, pour un changement brutal, notre méthode détecte plus rapidement le changement que les deux autres méthodes paramétrées à 1%. Pour arriver aux mêmes résultats, leur seuil de significativité doit être augmenté. Cette augmentation se fait au détriment de la robustesse dans le cas stationnaire. En conclusion, notre méthode est à la fois robuste et rapide à détecter un changement.

5.4.3 Changement progressif

Le but de cette section est d'étudier le passage progressif d'un concept 1 à un concept 2 au travers de la variation des critères de chaque méthode selon la quantité de changement.

Protocole expérimental Les indicateurs observés proviennent des différentes méthodes testées auparavant c'est-à-dire :

- pour le test t de Welch : p-value du test ;
- pour le test de Kolmogorov-Smirnov : distance D ;
- pour la méthode MODL $P(W|X)$: gain en compression l ;
- pour la méthode bivariée MODL $P(W|X, C)$: gain en compression l.

Pour cette expérimentation nous avons généré deux concepts :

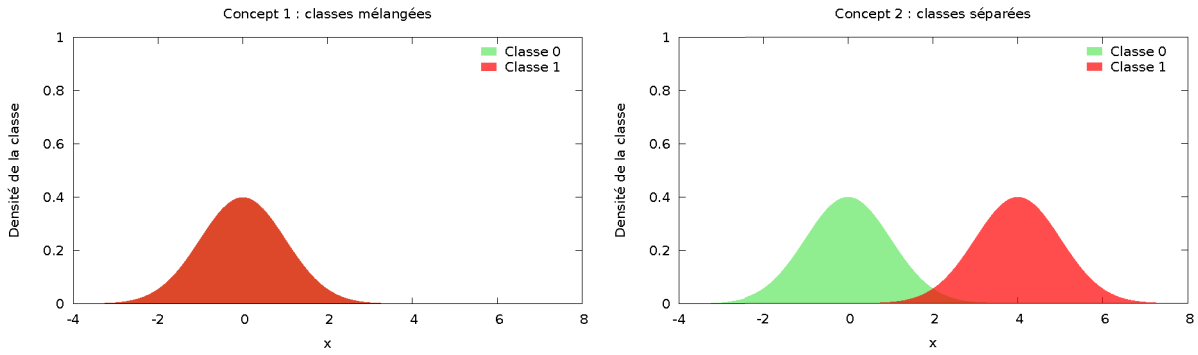


FIGURE 5.4 – Changement de moyenne entre le concept 1 et le concept 2.

- Concept 1 avec classe 0 : distribution $\mathcal{N}_0(\mu = 0, \sigma = 1)$ et classe 1 : distribution $\mathcal{N}_1(\mu = 0, \sigma = 1)$;
- Concept 2 avec classe 0 : distribution $\mathcal{N}_0(\mu = 0, \sigma = 1)$ et classe 1 : distribution $\mathcal{N}_1(\mu = 4, \sigma = 1)$.

La fenêtre de référence contient le concept 1 et la fenêtre courante le concept 1 et le concept 2 à raison du pourcentage de changement désiré (allant de 0% à 100% avec un pas de 5%).

Résultats Les résultats de cette expérimentation sont présentés dans la figure 5.5. Celle-ci permet de voir que les valeurs des différents critères sont bien proportionnelles à la quantité de changement dans les données. Les variances sont relativement faibles pour toutes les méthodes avec un léger avantage aux méthodes basées sur MODL qui la maintienne faible peu importe la quantité de changement.

Conclusion La valeur du critère de notre méthode croit bien avec la quantité de changement.

5.4.4 Différents types de changements détectés

Les expériences de cette section ont pour but d'observer quels types de changement (moyenne, variance, inversion des classes) les différentes méthodes sont capables de détecter. Pour tous les types de changement testés, le concept de départ est le Concept 1 défini ainsi :

- la classe 0 suit une distribution $\mathcal{N}_0(\mu = 0, \sigma = 0,5)$;
- la classe 1 suit une distribution $\mathcal{N}_1(\mu = 2, \sigma = 1)$.

Différents changements sont appliqués au concept 1 pour expérimenter le comportement des différentes méthodes.

Changement de moyenne On simule ce changement en changeant la moyenne de la classe 0, qui passe de 0 à 1 comme présenté dans la figure 5.6. On obtient le concept 2 suivant :

- la classe 0 suit une distribution $\mathcal{N}_0(\mu = 1, \sigma = 0,5)$;
- la classe 1 suit une distribution $\mathcal{N}_1(\mu = 2, \sigma = 1)$.

Changement de variance On simule ce changement en changeant la variance de la classe 0, qui passe de 1 à 0,5 comme présenté dans la figure 5.7. On obtient le concept 2 suivant :

- la classe 0 suit une distribution $\mathcal{N}_0(\mu = 0, \sigma = 1)$;
- la classe 1 suit une distribution $\mathcal{N}_1(\mu = 2, \sigma = 1)$.

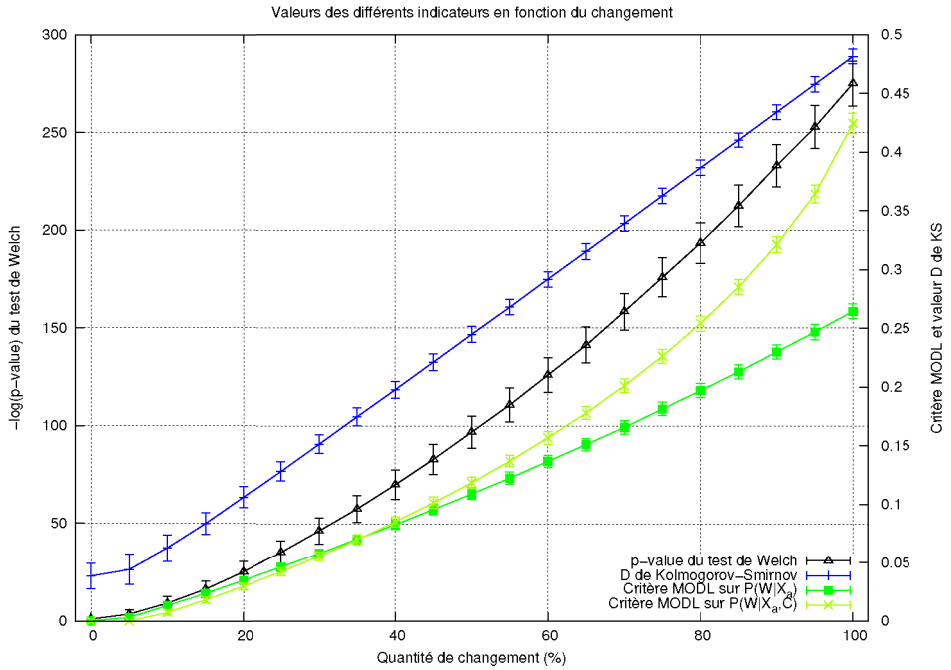


FIGURE 5.5 – Évolution des différents critères en fonction du changement de concept.

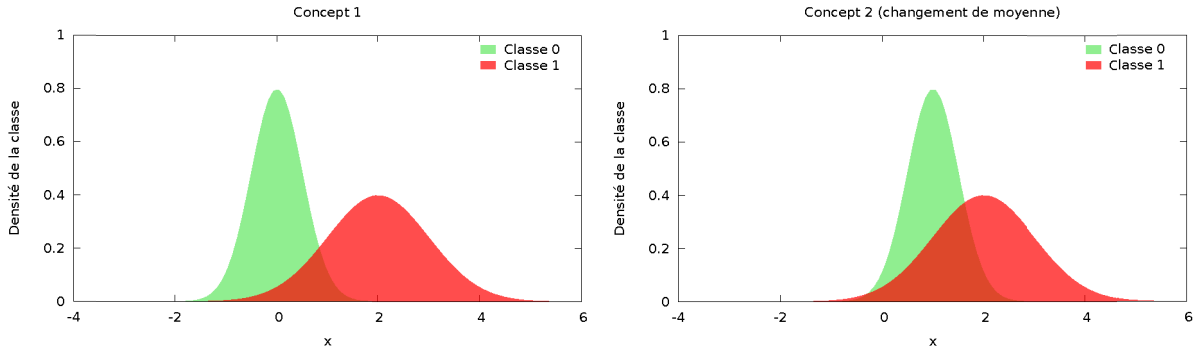


FIGURE 5.6 – Changement dans la moyenne entre le concept 1 et le concept 2.

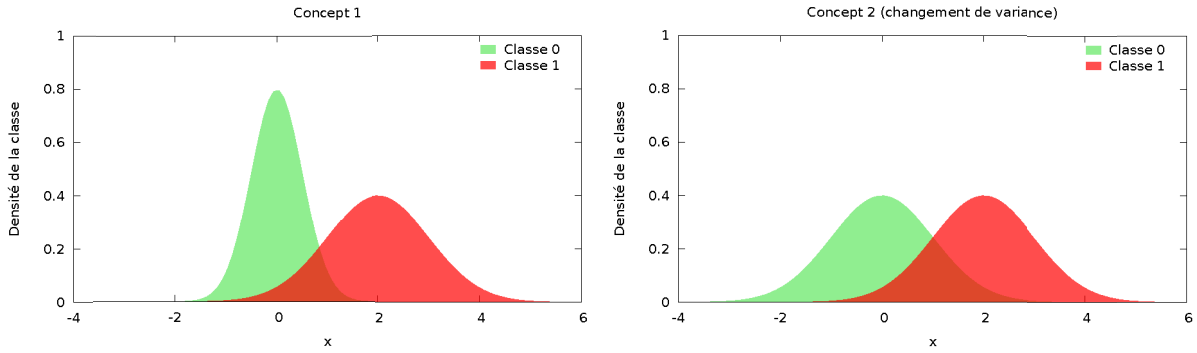


FIGURE 5.7 – Changement dans la variance entre le concept 1 et le concept 2.

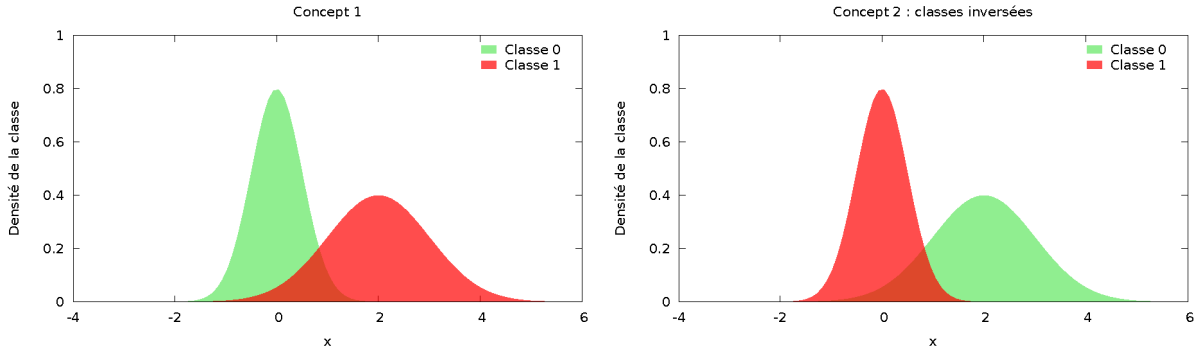


FIGURE 5.8 – Changement par inversion des classes entre le concept 1 et le concept 2.

Méthode	Moyenne	Variance	Inversion des classes
Welch (1%)	1000	0	0
Welch (2%)	1000	0	0
Welch (10%)	1000	19	10
KS (1%)	1000	998	0
KS (2%)	1000	1000	0
KS (10%)	1000	1000	15
MODL $P(W\|X_a)$	1000	1000	1
MODL $P(W\|X_a, C)$	1000	1000	1000

TABLE 5.4 – Nombre de détections par changement pour les différentes méthodes de détection.

Inversion des classes On simule ce changement en inversant les étiquettes des classes du concept 1 comme présenté dans la figure 5.8.

Résultats Pour ces expérimentations et pour les différents types de changement, les fenêtres sont de taille 1000. La fenêtre de référence contient le concept 1 et la fenêtre courante le concept 2. Le tableau 5.4 présente les résultats en termes de nombre de détections pour 1000 expérimentations. Les meilleurs méthodes sont celles qui sont capables de réaliser le plus de détections.

Toutes les méthodes sont capables de détecter de manière fiable un changement dans la moyenne. Le test de Welch n'est pas capable de détecter les changements de variance et ces expériences le confirment. Quelques détections (19) pour le test à 10% se produisent, mais celles-ci correspondent à des fausses détections et non pas à une détection de changement de variance. Le test de Kolmogorov-Smirnov et la détection MODL $P(W|X_a)$ se comportent très bien pour les détections dans les changements de moyenne et variance. Tous les tests précédents détectent seulement des changements dans la répartition des données mais sans s'intéresser à la classe. Par conséquent aucun d'entre eux n'est capable de détecter un changement qui n'intervient que par rapport aux classes. Contrairement à ces tests statistiques, la méthode de détection bivariée MODL $P(W|X_a, C)$ est capable de détecter ce genre de changements. Les expérimentations confirment cette capacité.

Conclusion Seule notre méthode MODL $P(W|X_a, C)$ est capable de détecter tous les types de changements étudiés dans cette section.

Critère → Méthode ↓	Fausses détections	Délai de détection	Changement moyenne	Changement variance	Inversion des classes
Welch (1%)	+	+	+	-	-
Welch (2%)	-	-	+	-	-
Welch (10%)	- -	+ +	+	-	-
KS (1%)	+	+	+	+	-
KS (2%)	-	-	+	+	-
KS (10%)	- -	+ +	+	+	-
MODL $P(W\|X_a)$	+	-	+	+	-
MODL $P(W\|X_a, C)$	+ +	+ +	+	+	+

TABLE 5.5 – Comparaison synthétique des différentes méthodes de détection.

5.4.5 Discussion de notre méthode sans classifieur

Les méthodes de l'état de l'art, testées dans cette section, fonctionnent bien pour un grand nombre de changements. Cependant elles ne sont pas capables de détecter tous les types de changement. Le test de Welch ne s'intéresse qu'à la moyenne. Le test de Kolmogorov-Smirnov n'a pas d'hypothèse sur la distribution des données mais a une tendance à réaliser de fausses détections sur un flux stationnaire. De plus, les performances de ces deux méthodes dépendent du paramétrage de leur seuil de détection qui arbitre leur sensibilité par rapport à leur robustesse.

La nouvelle méthode que nous proposons utilise le critère bivarié MODL $P(W|X_a, C)$ et ne souffre d'aucun de ces inconvénients. Elle est à la fois : (i) robuste dans le cas d'un flux stationnaire (section 5.4.2.1), (ii) rapide à détecter tous types de changements dans la distribution des données (section 5.4.2.2), (iii) capable d'utiliser l'information de classe (section 5.4.4). La valeur du critère donne aussi une très bonne indication de la quantité de changement entre les deux concepts observés. Le tableau 5.5 présente une vue synthétique des différentes méthodes en fonction des différents critères sur lesquels nous les avons observées.

5.5 Expérimentations avec classifieur

La méthode décrite dans la section précédente ne s'intéresse qu'aux changements dans un flux de données mais pas à la mise à jour du classifieur après la détection du changement. Nous proposons une nouvelle méthode de gestion de changement, avec un classifieur, se basant sur notre méthode de détection. Celle-ci est comparée aux méthodes de détection de l'état de l'art. Nous ne nous comparerons pas aux autres types de méthodes de l'état de l'art comme les méthodes à base d'ensembles de classifieurs ou les méthodes pondérant les exemples (voir section 2.4).

5.5.1 Une nouvelle méthode de gestion de changements avec un classifieur

5.5.1.1 Critère de détection

Le critère de détection utilisé dans les expérimentations suivantes est le critère bivarié MODL $P(W|X_i, C)$ appliqué à chacune des variables X_i. Il suffit qu'une détection ait lieu sur l'une des

variables pour déclencher le mécanisme de remplacement du classifieur présenté dans la section suivante.

5.5.1.2 Remplacement de l'ancien classifieur

La méthode de remplacement de l'ancien classifieur par un nouveau après une détection est présentée dans l'algorithme 6. Cet algorithme (appelé par la suite MDD : MODL Drift Detection) attend que le nouveau classifieur ait un plus faible taux d'erreurs que l'ancien pour utiliser le nouveau. La méthode `tauxErreur` utilisée pour nos expérimentations correspond à une moyenne mobile exponentielle de paramètre $\alpha = 1/tailleMaxW_{cur}$. Le paramètre n_{min} est le nombre minimum d'exemples utilisé pour comparer les performances des deux classifieurs, sa valeur est fixée à 30 ($n_{min} = 30$) pour toutes les expérimentations. Cette même valeur est utilisée par les méthodes DDM et EDDM (auxquelles nous allons nous comparer, voir section 5.5.2.2) avant qu'elles ne commencent à chercher un changement.

5.5.2 Protocol expérimental

5.5.2.1 Jeux de données

Les expériences menées dans cette section sont réalisées sur des flux de données avec un changement de concept. Nous utilisons deux générateurs de la littérature qui permettent de simuler un changement progressif.

Hyperplan en mouvement Les « hyperplans en mouvement » ont été proposés par Hulten et al. pour tester leur version des arbres de Hoeffding (CVFDT [HSD01]) dans le cadre du changement de concept. L'idée de base du générateur est de générer un hyperplan et de positionner les étiquettes des exemples selon leur position par rapport à l'hyperplan. Un hyperplan dans d-dimensions correspond à un ensemble de points tel que :

$$\sum_{i=0}^{d} w_i x_i = w_0$$

Le générateur à base d'hyperplans est intéressant dans le cadre du changement de concept car il est facile de faire évoluer le concept en changeant la position de l'hyperplan. Pour cela les poids w_i changent au cours du temps. La vitesse à laquelle les poids sont changés permet de contrôler la vitesse du changement de concept. Ce générateur possède deux paramètres :

- s : la vitesse de changement
- σ : la probabilité de changer de direction

Le poids associé à chaque attribut est mis à jour de la manière suivante : $w_i = w_i + s\sigma$. Les exemples pour lesquels $\sum_{i=0}^{d} w_i x_i > w_0$ sont étiquetés avec la classe 0 et les autres avec la classe 1.

Pour les expérimentations ce générateur a été configuré avec 10 attributs, $s = 10^{-3}$, $\sigma = 10\%$ et 10% de bruit de classe. Tous les attributs subissent un changement de concept.

Random RBF en mouvement Ce générateur reprend le celui utilisé dans la section 4.3.2.2 du chapitre sur les classifieurs. Des centroïdes définies par leur centre et leur diamètre sont générées à différentes positions dans l'espace. Des exemples appartenant à ces sphères sont générés en s'éloignant du centre d'une distance générée aléatoirement à partir d'une loi gaussienne. Afin

Notations :
x : un exemple du flux
W_{ref} : fenêtre de référence ($tailleW_{ref}$: sa taille)
W_{cur} : fenêtre sautante ($tailleW_{cur}$: sa taille)
$classif$: classifieur en cours d'utilisation ($perf$: sa performance)
$nouvClassif$: classifieur qui apprend depuis la détection ($nouvPerf$: sa performance)
n : nombre d'exemples utilisés pour l'apprentissage du nouveau classifieur
n_{min} : nombre minimum d'exemples avant de comparer la performance des deux classifieurs

tant que $x \leftarrow$ `flux` *()* **faire**
  `// s'il y a un nouveau classifieur, est-il meilleur que l'ancien ?`
  si `estDémarré` *(nouveauClassif)* **alors**
    $n \leftarrow n + 1$
    $tauxErrClassif \leftarrow$ `tauxErreur` ($tauxErreurClassif$, classif, x)
    $tauxErrNouvClassif \leftarrow$ `tauxErreur` ($tauxErreurNouvClassif$, nouvClassif, x)
    `apprendre` (nouvClassifieur, x)
    si $n > n_{min}$ *et* $tauxErreurNouveauClassif < tauxErreurClassif$ **alors**
      $W_{ref} \leftarrow \emptyset$
      $W_{cur} \leftarrow \emptyset$
      classif $\leftarrow$ nouvClassif
      $tauxErrClassif \leftarrow 0$
      $tauxErrNouvClassif \leftarrow 0$
      $n \leftarrow 0$
  `// remplissage des fenêtres`
  si $|W_{ref}| < tailleMaxW_{ref}$ **alors**
    $W_{ref} \leftarrow \mathrm{W}_{ref} \cup X$
  sinon si $|W_{cur}| < tailleMaxW_{cur}$ **alors**
    $W_{cur} \leftarrow \mathrm{W}_{cur} \cup X$
  sinon
    pour $i \leftarrow 1$ *à* d **faire**
      `// calcul du critère pour toutes les variables`
      $l \leftarrow l +$ `CalculerMODLBivarié` (W_{ref}, W_{cur}, i) (voir équation 5.1)
    `// détection ?`
    si $l > 0$ **alors**
      `// changement détecté : apprentissage d'un nouveau classifieur`
      `créer` (nouveauClassif)
    sinon
      `// flux stationnaire : plus besoin d'avoir un nouveau classifieur`
      `détruire` (nouveauClassif)
  `apprendre` (classif, x)

Algorithme 6 – Notre algorithme MDD (MODL Drift Detection) de remplacement du nouveau classifieur après changement de concept.

d'introduire un changement de concept dans ce générateur, Bifet et al. [BHP+09] proposent d'ajouter un paramètre qui définit la vitesse de déplacement de chaque centroïde.

Ce générateur ne contient que des attributs numériques. Pour les expériences, les paramètres suivants ont été choisis : 50 centres, 10 attributs et une vitesse de déplacement de 10^{-3}.

5.5.2.2 Méthodes comparées

Nous comparons notre méthode à deux méthodes de la littérature basées sur les performances du classifieur. Ces méthodes observent les performances du classifieur et détectent un changement quand celles-ci varient de manière importante. Cependant, pour que ces méthodes fonctionnent bien, il faut supposer que :

- les observations de la performance soient indépendamment et identiquement distribuées (iid) ;
- le processus (un classifieur dans notre cas) générant ces observations soit toujours le même.

Or, le classifieur est constamment en train d'apprendre, ce qui implique que le processus change après chaque nouvel exemple appris. Par conséquent les observations seront difficilement identiquement distribuées. Bien que toutes les hypothèses nécessaires au bon fonctionnement de ces méthodes ne soient pas validées, celles-ci ont prouvé leur intérêt sur diverses expérimentations [GMCR04, BGDF+06, BHP+09].

Notre méthode n'utilise pas de classifieur pour la détection mais observe directement les données du flux. Elle ne rencontrera donc pas les problèmes des méthodes qui utilisent un classifieur qui se construit avec le flux.

DDM La méthode DDM a été proposée par Gama et al. [GMCR04] et détecte les changements en observant l'évolution du nombre d'erreurs du classifieur. L'algorithme prend en entrée une distribution binomiale provenant de la variable binaire qui indique si l'exemple est bien classé (0) ou mal classé (1) par le classifieur. Cette loi binomiale est approximée par une loi normale après avoir vu 30 exemples. Ils estiment pour chaque exemple la probabilité qu'il soit mal classé p_i (p_i correspond aussi au taux d'erreur) et son écart type $s_i = \sqrt{p_i(1-p_i)/i}$. Ils font l'hypothèse que le classifieur ne fait que s'améliorer avec l'arrivée de nouveaux exemples si le flux est stationnaire, et donc que p_i va diminuer. Ils estiment qu'une augmentation significative du taux d'erreur signifie un changement de concept et donc qu'il faut réapprendre. Dans leur algorithme ils conservent les valeurs p_i et s_i et les minima atteints p_{min} et s_{min}. Leur méthode fonctionne en deux temps, tout d'abord une alerte est levée (« warning ») puis une détection. Ces deux niveaux sont définis de la manière suivante :

- alerte : $p_i + s_i \geqslant p_{min} + 2 \cdot s_{min}$
- détection : $p_i + s_i \geqslant p_{min} + 3 \cdot s_{min}$

Entre la phase d'alerte et de détection un nouveau classifieur est construit. De cette manière, après à la détection, il est possible de ne pas repartir d'un classifieur sans apprentissage mais d'un classifieur ayant appris sur les données du flux comprises entre l'alerte et la détection.

EDDM La méthode EDDM [BGDF+06] reprend la mode de fonctionnement de DDM mais propose un autre critère pour évaluer les seuils d'alerte et de détection. Cette méthode utilise la distance entre les erreurs de classification plutôt que le taux d'erreur. Cette distance correspond aux nombres de bonnes prédictions entres deux mauvaises prédictions. EDDM calcule la distance moyenne entre les erreurs p'_i et l'écart type s'_i et conserve les maxima de la moyenne p'_{max} et l'écart s'_{max}. De la même manière que pour DDM un seuil d'alerte et de détection sont définis :

- alerte : $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \alpha$

– détection : $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \beta$

Pour leurs expérimentations ils utilisent comme paramètres $\alpha = 90\%$ et $\beta = 95\%$. Ils constatent sur des jeux de données synthétiques et réels que EDDM détecte plus rapidement que DDM les changements graduels.

5.5.2.3 Évaluation des performances

Pour les mesures de performance, nous utilisions dans les chapitres précédents un jeu de données servant d'ensemble de test car les flux n'évoluaient pas. Cette méthodologie n'est pas adaptée à l'évaluation dans le cadre d'un changement de concept. Plusieurs méthodes existent pour évaluer en-ligne les classifieurs. Nous choisissons l'une des méthodes présentées dans l'état de l'art de la section 2.5.1. Cette méthode mesure la performance en utilisant les exemples du flux comme données de test avant qu'ils ne soient appris. Nous utilisons cette méthode afin d'obtenir deux mesures de performance :

– performance « instantanée » : précision calculée sur une fenêtre sautante de taille 10 000 ;
– performance globale : précision moyenne à un instant t. La valeur de cet indice à la fin de l'apprentissage est un indicateur synthétique de la performance de la méthode. Elle est d'ailleurs utilisée dans [BHP+09] pour comparer les différents algorithmes.

5.5.3 Expérimentations

Notre méthode a été implémentée dans l'environnement d'expérimentations pour les flux MOA (voir section 2.5.3). Les méthodes DDM et EDDM, présentées dans la section 5.5.2.2, sont disponibles dans MOA. Notre algorithme est configuré avec une même taille de fenêtre de 1000 pour la fenêtre de référence W_{ref} et la fenêtre sautante W_{cur}.

5.5.3.1 Bayésien naïf

Le classifieur utilisé est le bayésien naïf utilisant nos résumés à deux niveaux. Le premier niveau est le résumé de quantiles GKClass configuré avec 100 tuples et le second niveau la discrétisation MODL.

Les figures 5.9 et 5.10 présentent les résultats pour le jeu de données des « Hyperplan en mouvement », respectivement avec la précision sur une fenêtre de taille 10 000 et la précision moyenne. Les figures 5.11 et 5.12 reprennent les mêmes indicateurs mais pour le jeu de données « Random RBF en mouvement ».

Pour les « Hyperplan en mouvement » (Figures 5.9 et 5.10) toutes les méthodes permettent au classifieur d'avoir les mêmes performances jusqu'à 100 000 exemples appris. Entre 100 000 et 200 000 exemples DDM devient meilleur, notre méthode MDD reste relativement stable et EDDM bien moins bon. Après 300 000 exemples alors que MDD reste stable, DDM devient bien moins bon et EDDM meilleur, mais moins bon que notre méthode. On observe que notre méthode de détection est meilleure et beaucoup plus stable que les deux autres méthodes sur le jeu de données des « Hyperplan en mouvement ».

Pour les « Random RBF en mouvement » (Figures 5.11 et 5.12), DDM n'arrive pas à gérer le changement de concept. EDDM y parvient seulement pour les 500 000 premiers exemples. Notre méthode basée sur MODL est stable au cours du temps et donne les meilleures performances.

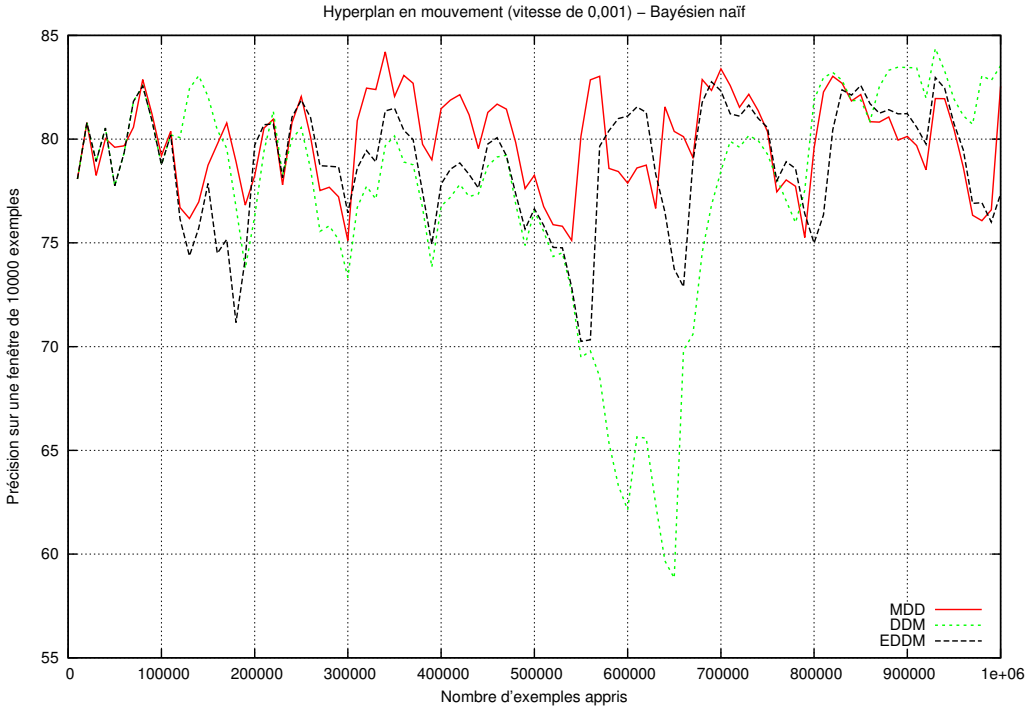


FIGURE 5.9 – Précision sur une fenêtre de taille 10000 sur les différentes méthodes de détection (classifieur bayésien naïf avec le jeu de données Hyperplan en mouvement).

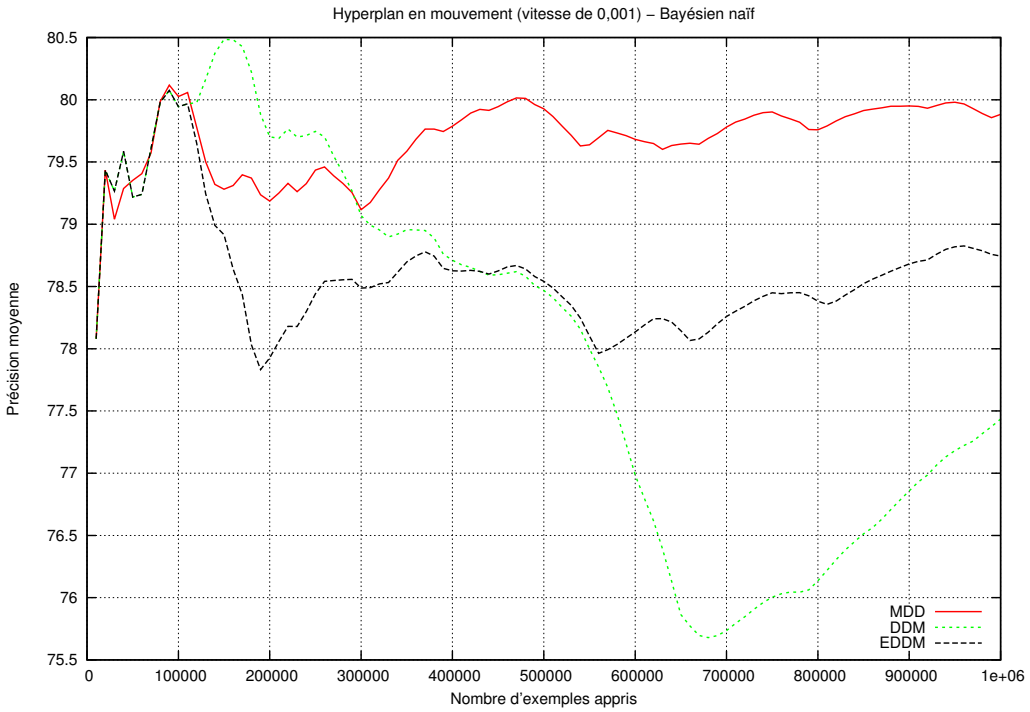


FIGURE 5.10 – Précision moyenne sur les différentes méthodes de détection (classifieur bayésien naïf avec le jeu de données Hyperplan en mouvement).

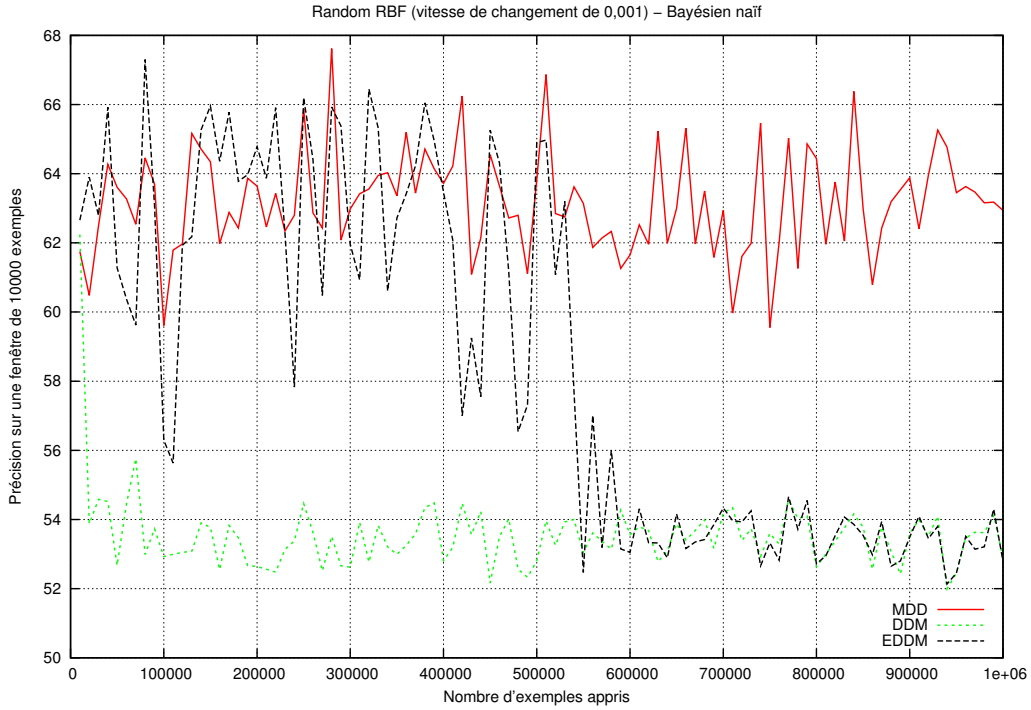


FIGURE 5.11 – Précision sur une fenêtre de taille 10000 sur les différentes méthodes de détection (classifieur bayésien naïf avec le jeu de données Random RBF en mouvement).

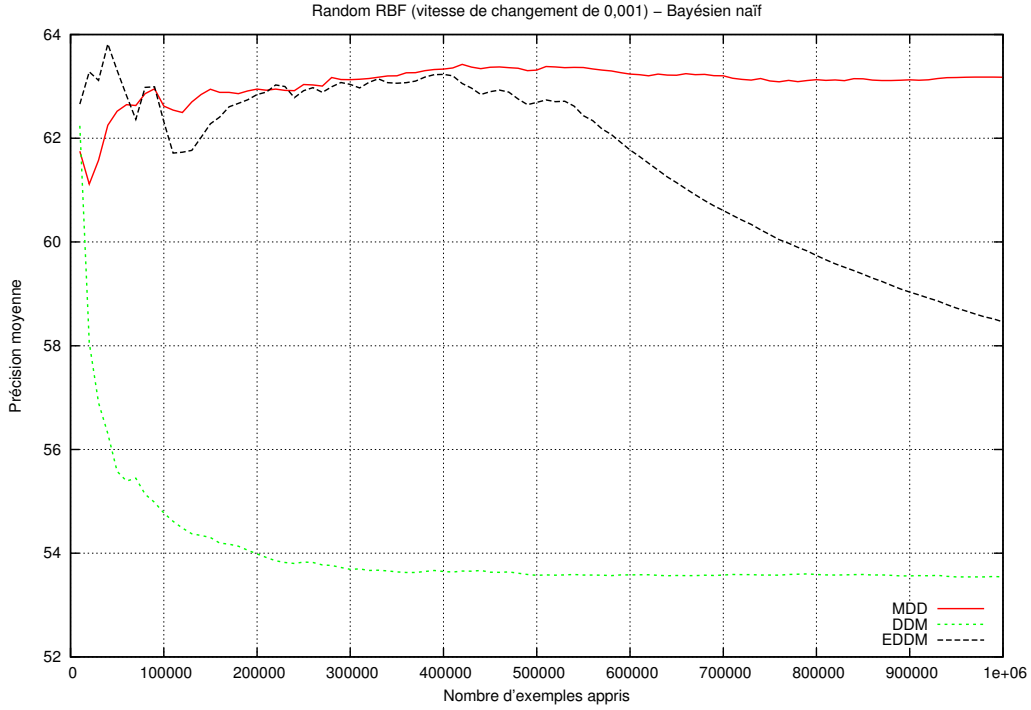


FIGURE 5.12 – Précision moyenne sur les différentes méthodes de détection (classifieur bayésien naïf avec le jeu de données Random RBF en mouvement).

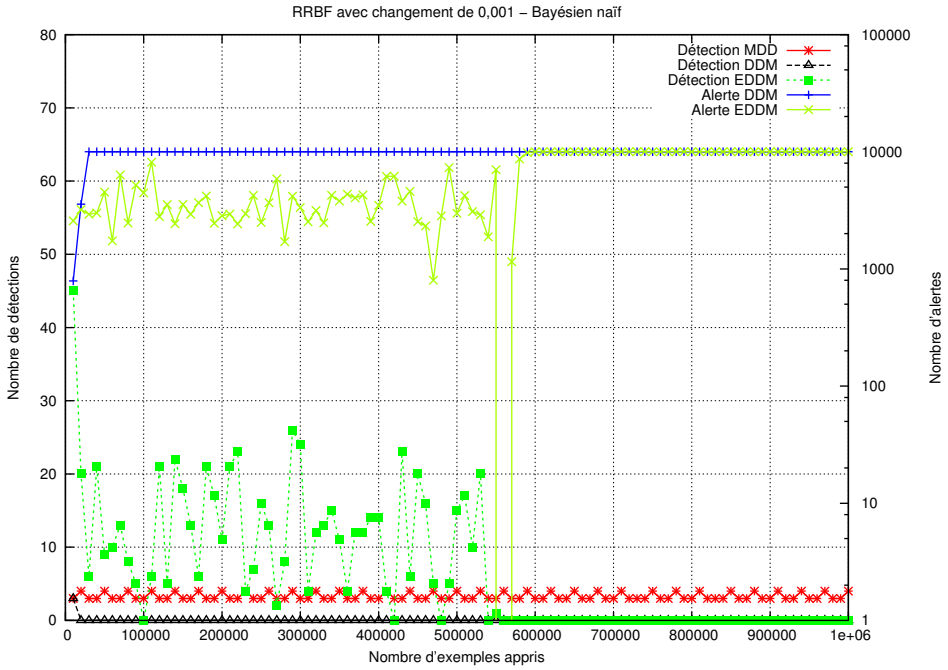


FIGURE 5.13 – Nombre de détections et alertes en fonctions du nombre d'exemples du flux pour différentes méthodes de détection, pour le jeu de données Random RBF en mouvement.

Observations des alertes et détections au cours du temps

Afin de mieux comprendre le comportement et les résultats observés, nous avons tracé les alertes et les détections des différentes méthodes en fonction du nombre d'exemples appris. Notre méthode ne fait que des détections et n'a pas de seuil d'alerte, par conséquent seulement ses détections seront tracées. La figure 5.13 montre ces résultats pour le jeu de données des « Random RBF en mouvement ». On observe que DDM est en permanence en alerte mais n'arrive pas à détecter. EDDM réalise beaucoup de détections et d'alertes jusqu'à 500 000 exemples mais n'arrive plus à détecter par la suite. Notre méthode MDD est quant à elle très stable et détecte de manière régulière des changements de concept. Ces observations expliquent clairement les variations de performance au cours du temps observés dans les figures 5.11 et 5.12. Le changement se réalisant toujours à la même vitesse, il paraît souhaitable que la détection se fasse aussi de manière régulière. Ces expériences montrent que notre méthode est bien plus stable et permet de garantir de bonnes performances au cours du temps.

Différentes tailles de fenêtre

La taille de la fenêtre est le paramètre le plus important de notre méthode. Pour toutes les expériences précédentes cette taille a été fixée arbitrairement à 1000 pour montrer l'intérêt de l'approche. Une grande taille de fenêtre permet de détecter avec une plus grande confiance ainsi que des motifs plus compliqués. Une plus petite taille permet d'être plus réactif. Un bon paramétrage de cette taille dépend du flux observé et des changements que l'on veut détecter. Afin d'observer la variation de performances qu'induisent les différentes tailles de fenêtre, nous avons lancé une expérimentation sur des tailles allant de 100 à 2 000. Le tableau 5.6 présente les résultats pour le jeu de données « Random RBF en mouvement » avec le classifieur bayésien naïf. Les meilleurs résultats sont obtenus pour des tailles de fenêtre allant de 250 à 750. Pour des

Taille W_{ref}	Taille W_{cur}	Précision moyenne
100	100	63,50
250	250	65,52
500	500	65,54
750	750	64,59
1000	1000	63,18
2000	2000	60,30
750	250	65,45
1000	100	63,59

TABLE 5.6 – Précision moyenne en fonction de la taille des fenêtres
(classifieur bayésien naïf sur le jeu de données « Random RBF en mouvement (0,001) » avec 1 million d'exemples appris).

tailles plus petites et plus grande les performances en précision se dégradent : 2% de moins pour les tailles de 100 et de 1000 et 5% de moins pour une taille de 2000. Pour des tailles de fenêtre asymétriques, 1000/100 ou 750/250, les résultats sont identiques à ceux des tailles symétriques, 100/100 ou 250/250. Les sections 5.3.1.3 et 2.4.2 abordent la problématique du réglage de la taille des fenêtres. Pour les expérimentations suivantes avec les arbres d'Hoeffding nous conservons une fenêtre de taille 1000. Le but de cette section, présentant notre approche avec un classifieur, est de montrer que l'association de notre méthode de détection avec un algorithme simple (algorithme 6) donne de bonnes performances et possède de bonnes propriétés (robustesse, nombre de détections dépendant de la vitesse de changements...). L'optimisation de la taille de la fenêtre ne sera donc pas plus approfondie dans ce chapitre.

5.5.3.2 Arbre de décision

Le classifieur utilisé est l'arbre de Hoeffding HTmGKcNBm présenté dans la section 4.3.2.1. Cet arbre utilise le critère MODL pour les coupures, nos résumés à deux niveaux (premier niveau GKClass avec 10 tuples) et le classifieur bayésien naïf dans les feuilles.

Les figures 5.14 et 5.15 présentent les résultats pour le jeu de données des « Hyperplan en mouvement », respectivement avec la précision sur une fenêtre de taille 10 000 et la précision moyenne. Les figures 5.16 et 5.17 reprennent les mêmes indicateurs mais pour le jeu de données « Random RBF en mouvement ».

Pour les « Hyperplan en mouvement » (Figures 5.14 et 5.15) toutes les méthodes permettent au classifieur d'avoir les mêmes performances jusqu'à 200 000 exemples appris. Entre 200 000 et 600 000 exemples DDM devient meilleur, la méthode de détection MDD reste relativement stable et EDDM bien moins bon. Après 600 000 exemples alors que MDD continue à s'améliorer, DDM devient bien moins bon et EDDM un peu meilleur. Comme pour la classifieur bayésien naïf, on observe que notre méthode de détection est bien meilleure et beaucoup plus stable que les deux autres méthodes sur le jeu de données des « Hyperplan en mouvement ».

Pour les « Random RBF en mouvement » (Figures 5.16 et 5.17), DDM n'arrive pas à gérer le changement de concept. EDDM n'y parvient pas pour les 300 000 premiers exemples mais réussit à identifier quelques changements par la suite. Notre méthode MDD est largement plus performante et stable au cours du temps.

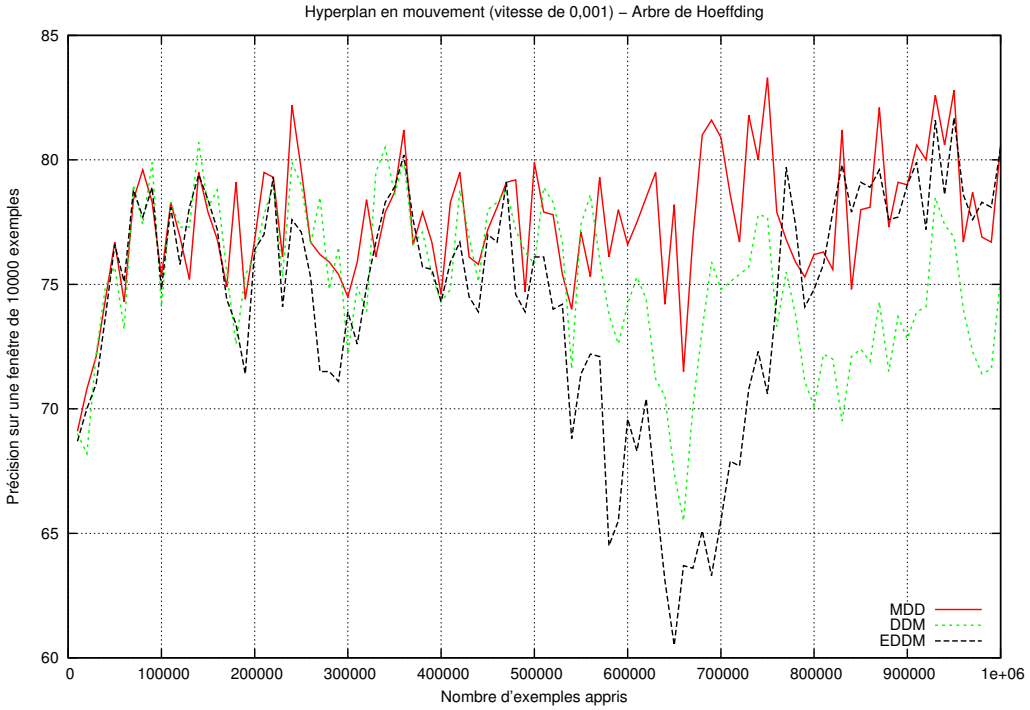


FIGURE 5.14 – Précision sur une fenêtre de taille 10000 sur les différentes méthodes de détection (Arbre de Hoeffding avec le jeu de données Hyperplan en mouvement).

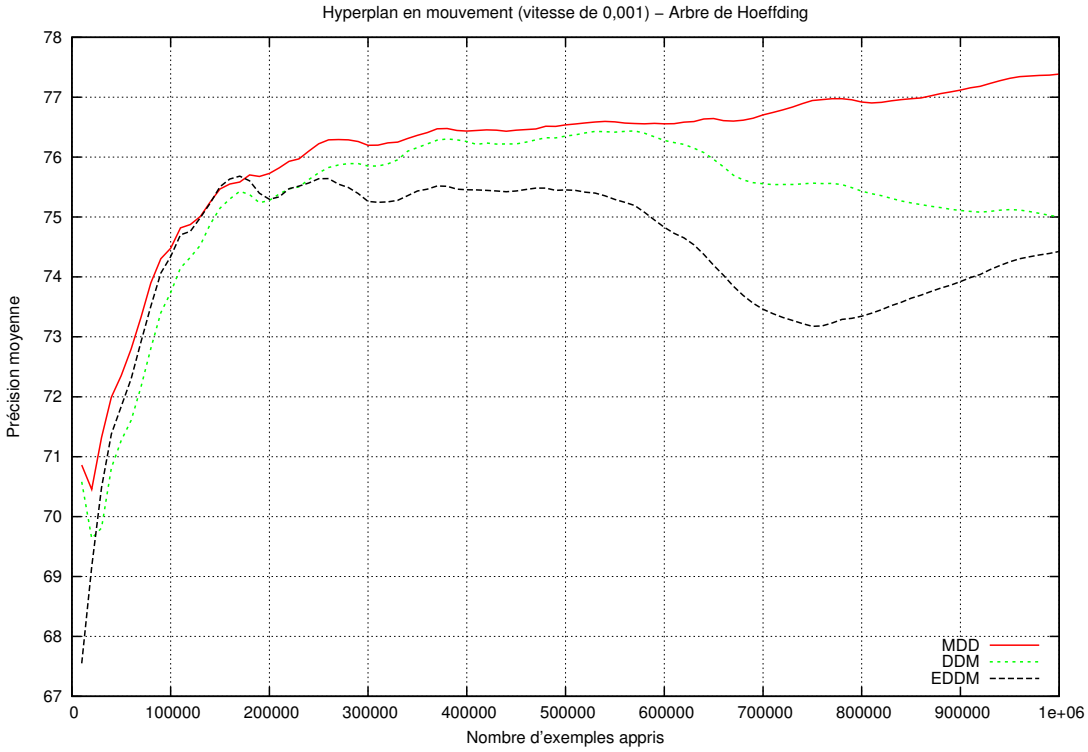


FIGURE 5.15 – Précision moyenne sur les différentes méthodes de détection (Arbre de Hoeffding avec le jeu de données Hyperplan en mouvement).

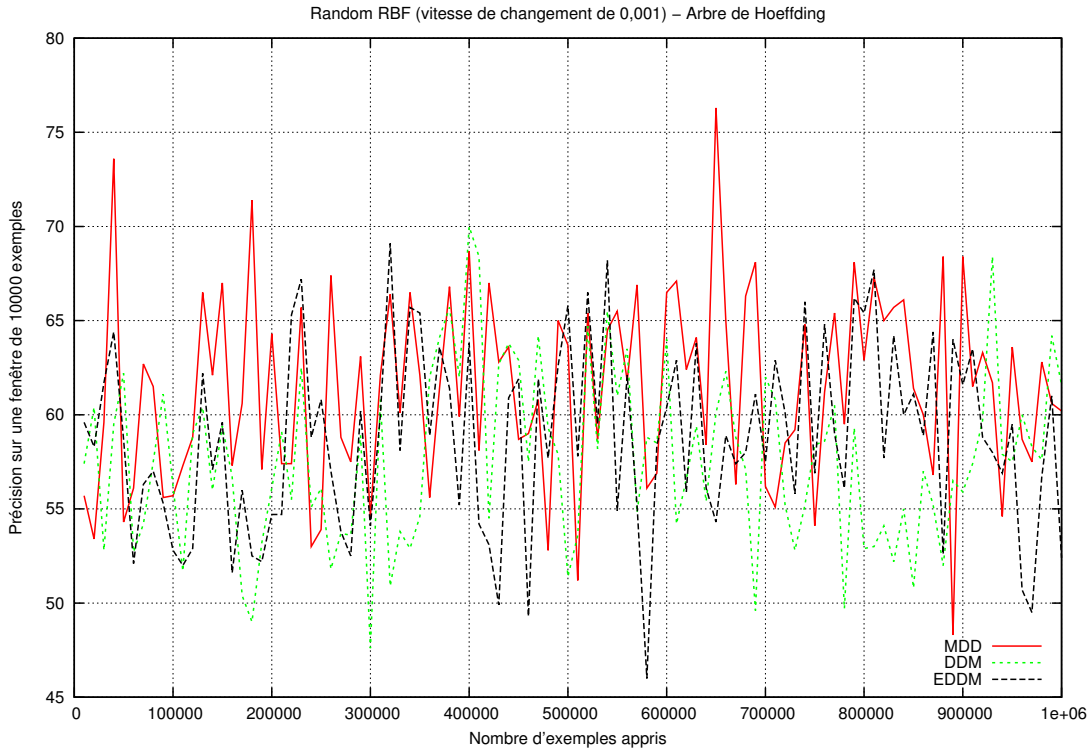


FIGURE 5.16 – Précision sur une fenêtre de taille 10000 sur les différentes méthodes de détection (Arbre de Hoeffding avec le jeu de données Random RBF en mouvement).

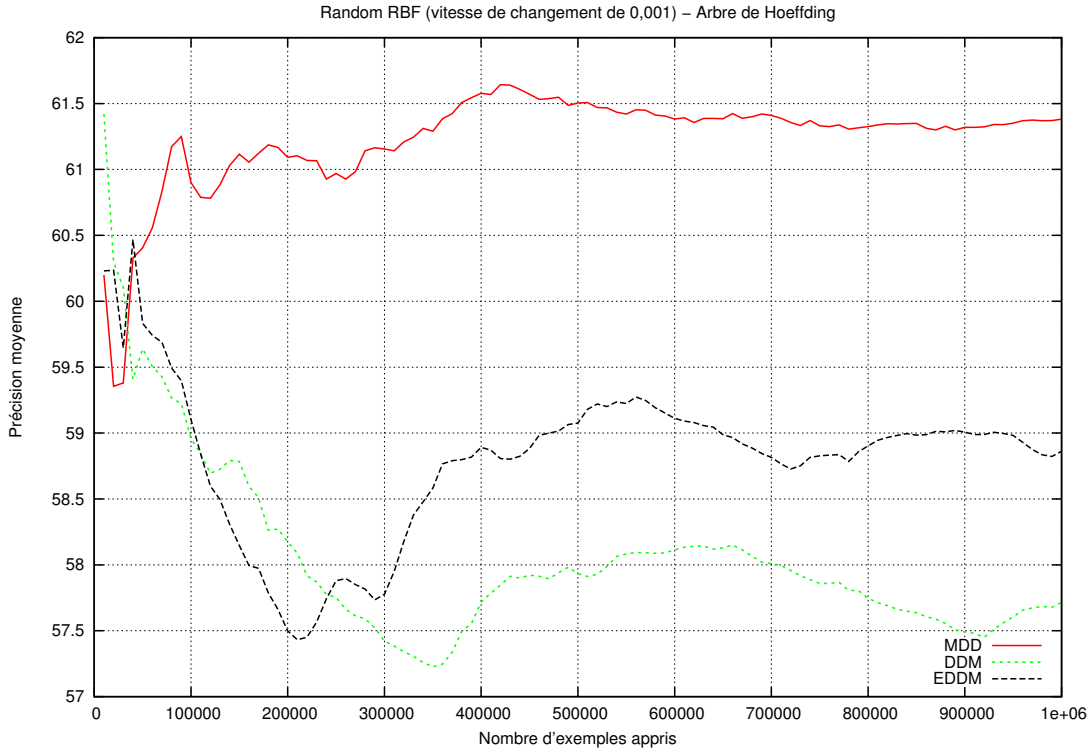


FIGURE 5.17 – Précision moyenne sur les différentes méthodes de détection (Arbre de Hoeffding avec le jeu de données Random RBF en mouvement).

Les expérimentations sur le nombre d'alertes/détections de DDM et EDDM, ainsi que celles sur la taille des fenêtres, ne sont pas présentées ici car elles donnent des résultats et conclusions similaires à celles réalisées sur le classifieur bayésien naïf.

5.5.4 Discussion de notre méthode avec classifieur

On observe, sur les deux générateurs de données de l'état de l'art et pour les deux méthodes concurrentes testées, que leur comportement n'est pas stable. En effet, le changement est contrôlé et régulier mais ces méthodes ont tendance à détecter beaucoup de changements sur certaines périodes et aucun sur d'autres. Leurs hypothèses de départ (processus stationnaire et données iid), ne sont sans doute pas valides, ce qui aboutit à un manque de robustesse de ces méthodes. Notre méthode de détection n'utilise pas le classifieur pour la détection mais seulement les données du flux. On observe que son nombre de détections par intervalle de temps est régulier pour un changement ayant une vitesse constante. Les performances de notre méthode, en termes de précision, sont bien meilleures et plus constantes que les méthodes de l'état de l'art testées.

5.6 Conclusion

Ce chapitre présente notre nouvelle méthode de détection des changements dans les flux de données. Celle-ci est basée sur l'observation du changement des distributions des exemples dans deux fenêtres. La première est placée au début du flux et ne change pas de position. La deuxième est glissante avec l'arrivée de nouveaux exemples. Notre méthode n'a pas d'*a priori* sur la distribution des données ni sur le type de changement à détecter. Elle est capable de détecter des changements rapides ou lents, que cela soit sur la moyenne, l'écart type ou tout autre changement dans la distribution. Le formalisme bayésien est utilisé pour caractériser les changements détectés. Notre méthode fait l'hypothèse naïve d'indépendance des variables et est capable de détecter des changements sur les $P(C)$, $P(X_i)$ et $P(X_i|C)$ en utilisant l'approche MODL. Chaque fenêtre est étiquetée et un changement est détecté s'il existe un modèle de discrétisation/groupage permettant de distinguer les deux fenêtres.

Notre approche a été évaluée et comparée sur des exemples artificiels pour en étudier les avantages et limites. Parmi les méthodes de l'état de l'art, nous nous sommes comparés aux tests de Welch et de Kolmogorov-Smirnov. Le premier est assez robuste mais ne s'intéresse qu'à la moyenne. Le second n'a pas d'hypothèse sur la distribution des données, mais tend à réaliser de fausses détections sur un flux stationnaire. Les performances de ces deux méthodes dépendent du paramétrage de leur seuil de détection. La nouvelle méthode que nous proposons utilise le critère bivarié MODL $P(W|X_i, C)$ et ne souffre d'aucun de ces inconvénients. Elle est à la fois : (i) robuste dans le cas d'un flux stationnaire, (ii) rapide à détecter tous types de changements dans la distribution des données, (iii) capable d'utiliser l'information de classe. La valeur du critère permet aussi de mesurer la quantité de changement entre les deux concepts observés.

Nous comparons aussi notre méthode à deux méthodes de l'état de l'art détectant les variations de performance d'un classifieur. Nous proposons un nouvel algorithme, appelé MDD, basé sur notre méthode de détection permettant au classifieur de se mettre à jour. Celle-ci n'utilise pas le classifieur pour la détection mais seulement les données du flux. Ses détections sont régulières pour un changement ayant une vitesse constante. Ses performances, en termes de précision, sont bien meilleures et plus constantes que les méthodes de l'état de l'art testées.

Chapitre 6

Conclusion et perspectives

Sommaire

6.1 Conclusion

Les apports de cette thèse ont été présentés dans les chapitres 3, 4 et 5. Nos contributions à la classification en-ligne sur les flux portent sur trois points : des résumés de données supervisés incrémentaux, des classifieurs incrémentaux et une méthode de détection de changement. Ceux-ci sont rappelés ci-dessous.

Résumés supervisés incrémentaux en-ligne pour flux de données

Les débits des flux de données rendent impossible l'utilisation de la plupart des méthodes d'apprentissage hors-ligne. Nous avons proposé une nouvelle méthode incrémentale de résumés des données univariés adaptés à un usage en-ligne pour flux de données. Notre méthode est basée sur deux niveaux et est capable de traiter aussi bien des variables numériques que catégorielles.

Pour le premier niveau traitant les attributs numériques, nous avons choisi des méthodes contrôlant le compromis mémoire / vitesse / précision comme le résumé de quantiles de Greenwald et Khanna (GK). Ce résumé n'étant pas adapté à la classification supervisée, nous l'avons modifié en ajoutant les comptes par classe (GKClass) dans les tuples afin d'améliorer ses performances en classification. Nous avons aussi proposé une autre modification de l'algorithme GK d'origine permettant d'augmenter jusqu'à un ordre de grandeur la vitesse d'insertion dans le résumé (GKClassAcc). Nous avons adapté le critère MODL afin de pouvoir l'utiliser dans un environnement en-ligne à mémoire limitée (OnLineMODL). Ce résumé supervisé est capable d'allouer plus de mémoire aux régions de l'espace les plus informatives et moins aux autres. Notre dernière proposition de résumé (cPiD) repart de la première couche de PiD mais en l'adaptant afin qu'elle utilise une quantité de mémoire fixe. La méthode OnLineMODL se comporte bien mieux que les autres sur des données à base de créneaux et plus particulièrement quand l'espace contient des zones « vides » (plateaux). Pour le premier niveau traitant les attributs catégoriels, notre choix s'est porté sur le Count-min Sketch qui possède comme le résumé GK des garanties théoriques sur le nombre d'erreurs en fonction de la mémoire utilisée. Nous l'avons adapté (CMSClass) à la classification en ajoutant les comptes par classe dans son tableau de stockage.

Le deuxième niveau utilise l'approche MODL pour la discrétisation et le groupage hors-ligne. Cette approche fonctionne aussi bien pour les variables numériques que catégorielles en prenant en entrée les comptes fournis par les résumés de notre premier niveau. Le résumé du premier niveau est de faible taille ce qui permet une exécution rapide des méthodes MODL bien qu'elles soient prévues pour fonctionner hors-ligne. Les qualités de notre approche à deux niveaux ont été validées sur différentes expérimentations.

Classifieurs incrémentaux en-ligne pour flux de données

La plupart des classifieurs, prévus pour fonctionner hors-ligne, ne sont pas capables de traiter des flux de données. Nous avons adapté des classifieurs existants aux flux de données à l'aide de nos résumés en-ligne : le classifieur bayésien naïf et les arbres de décisions.

Les probabilités conditionnelles aux classes du classifieur bayésien naïf, habituellement estimées par des méthodes hors-ligne telles que « Equal Frequency », MDLP, MODL..., sont remplacées par celles provenant de nos résumés. On observe que pour une taille de résumé suffisante les performances de la version adaptée pour les flux est similaire à la version hors-ligne. Si la mémoire disponible est suffisante les résumés sont capables d'obtenir des estimations fiables des probabilités conditionnelles aux classes. L'apport du second niveau est particulièrement remarquable pour les résumés de taille moyenne à grande pour lesquels le premier niveau ne permet pas d'obtenir une estimation fiable de la densité conditionnelle aux classes. L'approche MODL ne nécessitant pas de paramètre, ces bonnes performances sont accessibles sans réglage préalable.

Le classifieur bayésien naïf moyenné diffère du classifieur bayésien naïf par l'ajout d'un poids sur chacune des variables. L'estimation de ces poids se fait habituellement hors-ligne. Nous avons proposé une solution basée sur une descente de gradient stochastique afin de réaliser l'optimisation de ces poids en-ligne. Cette nouvelle méthode nécessite beaucoup de données pour converger vers les poids optimaux. Par conséquent son utilisation dépend de la quantité de données disponible et de la rapidité à laquelle les poids peuvent être appris.

Pour les arbres de décision en-ligne, nous avons adapté les arbres de Hoeffding afin qu'ils puissent utiliser les probabilités conditionnelles aux classes de nos résumés. Notre proposition porte sur les différents éléments qui les composent : (i) les résumés dans les feuilles, (ii) le critère de coupure, (iii) le modèle local. Sur les jeux de données issus des générateurs nos modifications améliorent les résultats. Le classifieur bayésien naïf, utilisé comme modèle local, tire tout particulièrement avantage de la robustesse de notre résumé à deux niveaux.

Détection de changement dans un flux de données

Les flux de données peuvent ne pas être stationnaires et comporter des changements de concept. Nous proposons une nouvelle méthode de détection des changements qui est basée sur l'observation du changement des distributions univariées des exemples entre deux fenêtres.

La première fenêtre est placée au début du flux et ne change pas de position. La deuxième est glissante avec l'arrivée de nouveaux exemples. Notre méthode n'a pas d'*a priori* sur la distribution des données ni sur le type de changement à détecter. Elle est capable de détecter des changements rapides ou lents, que cela soit sur la moyenne, l'écart type ou tout autre changement dans la distribution. Notre nouvelle méthode utilise le critère bivarié MODL qui permet d'observer des changements de distribution sur les données conditionnellement aux classes. Elle est à la fois : (i) robuste dans le cas d'un flux stationnaire, (ii) rapide à détecter tout type de changements dans la distribution des données, (iii) capable d'utiliser l'information de classe. La valeur du critère permet aussi de mesurer la quantité de changement entre les deux concepts observés.

Nous proposons aussi un nouvel algorithme, appelé MDD, basé sur notre méthode de détection permettant de remplacer le classifieur. Celle-ci n'utilise pas le classifieur pour la détection mais seulement les données du flux. Ses détections sont régulières pour un changement ayant une vitesse constante. Ses performances, en termes de précision, sont bien meilleures et plus constantes que les méthodes de l'état de l'art comme DDM et EDDM.

6.2 Utilisation de ces nouvelles méthodes

Dans de nombreuses grandes entreprises possédant des systèmes d'informations complexes, le processus de mise en production de nouvelles méthodes requiert de nombreuses étapes et nécessite un certain temps. À ce jour, les méthodes décrites dans ce manuscrit n'ont pas encore été industrialisées chez Orange. Cependant les cas d'utilisation présentés ci-dessous pourraient être envisagés par rapport aux apports de cette thèse.

6.2.1 Sélection de variables

Les informations sur les clients d'Orange et les logs d'usage liés aux appels de ceux-ci sont utilisés pour modéliser les profils clients pour des besoins marketing comme le changement d'opérateur (churn), l'appétence pour un produit ou la montée en gamme (upselling). La jointure complète entre ces deux tables produirait une quantité de données trop importantes. L'approche utilisée chez Orange consiste à construire de nombreux agrégats sur les logs d'usage. Chaque

agrégat correspond à la création d'une nouvelle variable dans la table client. Des exemples de ces agrégats sont le nombre de communications par jour/semaine/mois, la durée moyenne des communications... L'un des premiers traitements consiste à sélectionner les variables informatives parmi toutes ces variables. Cette étape est très coûteuse en ressources pour des bases de données de millions de clients et ne peut être réalisée, pour l'instant, que sur un échantillon de la base. L'idée serait d'utiliser nos résumés qui consomment peu de ressources mémoire et processeur afin de pouvoir utiliser toutes les données. Le second niveau de nos résumés utiliserait la discrétisation MODL dont le critère indique l'informativité d'une variable. On pourrait donc adapter la première passe de sélection de variables avec nos résumés afin de réduire les besoins en ressources de ce pré-traitement.

6.2.2 Augmentation de la volumétrie traitée

Ce cas d'utilisation est la suite logique de la sélection de variables. Une fois les variables sélectionnées on pourrait utiliser un de nos classifieurs : bayésien naïf, bayésien naïf moyenné ou arbres de décision, pour faire de la classification sur la base clients d'Orange. Pour l'instant les algorithmes de classification n'utilisent qu'une partie de la base clients. L'intérêt est ici d'augmenter le volume de données utilisées pour être capable d'avoir de meilleurs modèles.

6.2.3 Mise à jour des modèles

Les modèles de classification utilisés chez Orange se font sur des données qui peuvent évoluer. On peut citer, par exemple, les profils des clients qui changent au cours des mois, les profils de fraude qui évoluent à l'apparition de nouvelles offres commerciales... Pour l'instant ce problème du changement est traité par un ré-apprentissage régulier (tous les semaines, mois...) ou la détection d'une baisse de la performance du modèle. On pourrait utiliser notre méthode de détection de changement pour évaluer si les données sont toujours les mêmes et donc que le modèle actuel est toujours pertinent ou si au contraire les données ont changé et que le modèle doit être ré-appris. Ainsi on serait plus réactif à un changement en ré-apprenant le modèle dès qu'un changement est apparu au lieu d'attendre et on éviterait l'apprentissage du nouveau modèle quand cela n'est pas nécessaire.

6.3 Perspectives

Parmi les différents travaux futurs qui pourraient être envisagés suite à cette thèse, nous proposons deux pistes. La première est relative au résumé OnLineMODL pour en diminuer sa variance et la deuxième concerne un arbre de décisions en ligne régularisé.

6.3.1 Ensemble de résumés OnLineMODL

Dans le chapitre 3, nous avons vu que le résumé OnLineMODL fonctionnait très bien mais avait une forte variance. Cette forte variance est due à la non remise en question des points de coupure au cours du temps : les intervalles ne peuvent être que fusionnés mais jamais divisés. Afin d'améliorer ce résumé on peut imaginer d'avoir un ensemble de résumés OnLineMODL construit sur différentes parties du flux. De cette manière ces résumés ont des points de coupures à différentes positions. Si un résumé contient un ou des points de coupure à des positions non pertinentes alors le coût de son modèle sera plus élevé que les autres. Cet ensemble peut se combiner par vote selon le coût du modèle pour donner les comptes finaux par intervalle. Cette

méthode nécessite le réglage de la taille de l'ensemble, la taille des fenêtres des différents résumés et la politique de combinaison des votes. L'utilisation d'un ensemble de résumés OnLineMODL devrait diminuer la variance de ce résumé.

6.3.2 Arbre en-ligne avec un critère global MODL

La construction d'un arbre de décision en-ligne, dans un environnement à mémoire limitée, pose la question de savoir quelles parties de l'arbre doivent continuer à se développer ou au contraire quelles parties doivent être élaguées. Le critère MODL est utilisé pour réaliser les coupures et il donne, pour une feuille, l'informativité de la variable choisie au moment de la coupure. Le choix de l'expansion de l'arbre est fait localement mais il serait intéressant d'avoir un critère global pour tout l'arbre afin de le régulariser. Ce critère global permettrait d'évaluer le coût de l'ajout de nouvelles feuilles et le coût d'un élagage d'une partie de l'arbre. Un arbre utilisant un tel critère (basé sur l'approche MODL) a déjà été proposé mais uniquement dans le cadre de l'apprentissage hors-ligne [VBH09]. L'idée serait d'adapter le critère pour une utilisation en-ligne afin de trouver le meilleur modèle, du point de vue du critère, pour une quantité de mémoire donnée.

Annexe A

Classifieurs bayésien moyenné hors-ligne

Comparaison de classifieurs bayésiens moyenné hors-ligne

Le but de cette expérience est de comparer les performances prédictives de deux versions hors-ligne du classifieur bayésien naïf moyenné afin de montrer que ces deux classifieurs ont des performances très proches. On pourra ainsi conclure que le modèle graphique proposé dans le chapitre 3 permet un bon apprentissage des poids pour sa version hors-ligne.

Le classifieur bayésien naïf moyenné est utilisé chez Orange [Bou06b] sur de nombreuses problématiques. Parmi ces problématiques, on peut citer les problèmes marketing du churn, de l'appétence pour un produit ou de la montée en gamme (upselling). Des données d'Orange, liées à ces problèmes marketing, ont été rendues publiques pour organiser le challenge KDD 2009 [7] [GLB+09]. Ce jeu de données contient 50 000 exemples en apprentissage et en test. Chaque exemple est composé de 190 variables numériques et 40 variables catégorielles.

Nous proposons d'étudier les performances prédictives de deux classifieurs bayésien naïf moyenné sur ce jeu de données. Les pondérations des variables sont obtenues de manière différente selon le classifieur. Le classifieur proposé dans [Bou06b] utilise un moyennage de modèles avec optimisation des poids basée sur un critère MDL (Minimum Description Length) et est disponible à l'adresse suivante : `www.khiops.com`. Ce classifieur a de très bonnes performances sur de nombreux jeux de données [Bou06b]. Le second classifieur provient d'une application non publique développée chez Orange qui optimise les poids directement à l'aide d'une descente de gradient. Cette application est prévue pour fonctionner hors-ligne et peut donc utiliser plusieurs itérations sur le même exemple et des ensembles de validation croisée. Le modèle graphique de cette application est le même que celui présenté dans le chapitre 3.

Le tableau A.1 présente les résultats en test de ces deux algorithmes sur les données du challenge KDD 2009. On observe bien que les résultats sont très proches sauf pour la précison sur la base « Upselling ».

Précision

	Moyennage de modèles	Optimisation directe
Churn	92,58%	92,66%
Upselling	94,90%	92,64%
Appetency	98,22%	98,22%

AUC

	Moyennage de modèles	Optimisation directe
Churn	72,70%	73,09%
Upselling	86,32%	86,38%
Appetency	82,22%	82,02%

TABLE A.1 – Résultats des deux classifieurs bayésien moyenné sur les données du challenge KDD 2009.

Calcul de la dérivée de la fonction de coût

Cette section explicite le calcul de la dérivée de la fonction de coût pour le classifieur bayésien moyenné avec optimisation des poids par descente de gradient. Le modèle graphique utilisé est le même que pour notre classifieur bayésien naïf moyenné du chapitre 3 et donc ce calcul est

7. `http://www.kddcup-orange.com/`

identique pour notre méthode. On rappelle par la figure A.1 le modèle graphique utilisé dans le chapitre 3. Les notations utilisées sont les suivantes :

- k l'indice des classes du problème de classification supervisé : $k \in \{1, ..., K\}$
- X un vecteur d'entrée représentant un exemple d'apprentissage
- d le nombre de variables explicatives
- b_k le biais lié à la classe k à apprendre
- w_{ik} le poids assoicié à la variable j et la classe k

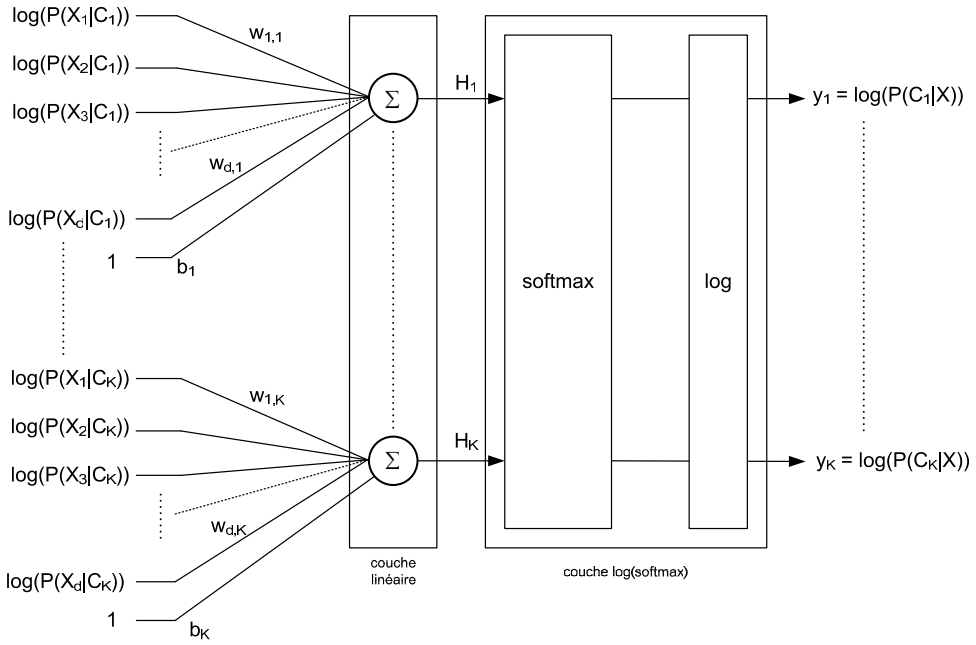


FIGURE A.1 – Modèle graphique pour l'optimisation des poids du classifieur bayésien moyenné.

La première couche du modèle graphique est une couche linéaire réalisant pour chaque classe k à apprendre une somme pondérée. Chaque sortie de la première couche du modèle graphique est égale à :

$$H_k = \sum_{i=1}^{d} w_{ik} log(P(X_i|C_k)) + b_k \tag{A.1}$$

La couche suivante du modèle graphique est un $log(softmax)$. Chaque sortie est donc de la forme :

$$y_k = log\left(\frac{e^{H_k}}{\sum_{j=1}^{K} e^{H_j}}\right) \tag{A.2}$$

$$y_k = log\frac{exp(\sum_{i=1}^{d} w_{ik} log(P(X_i|C_k)) + b_k)}{\sum_{j=1}^{K} exp(\sum_{i=1}^{d} w_{ij} log(P(X_i|C_j)) + b_j)} \tag{A.3}$$

Le modèle graphique permet d'avoir directement en sortie la valeur des $log(P(C_k|X))$. Le but étant de maximiser la vraisemblance il suffit alors de minimiser le log vraisemblance.

On décompose tout d'abord la partie softmax en considérant que chaque sortie de la fonction $softmax$, avant la phase de normalisation, peut être vue comme étant la succession de deux

étapes : une phase d'activation suivi d'une fonction recevant la valeur de l'activation. Ici la fonction d'activation peut être vu comme étant :

$$O_k = f(H_k) = exp(H_k) \tag{A.4}$$

Et la sortie formalisée de la partie softmax de notre modèle graphique est :

$$P_k = \frac{O_k}{\sum_{j=1}^{K} O_j} \tag{A.5}$$

La dérivée de la fonction d'activation est :

$$\frac{\partial O_k}{\partial H_k} = f'(H_k) = exp(H_k) = O_k \tag{A.6}$$

Avec notre modèle graphique qui ajoute une couche « log » et le fait que l'on cherche à minimiser comme fonction de coût le $-log\, vraisemblance$, il faut à présent considérer deux cas :

- soit on désire apprendre pour la classe k la valeur 1
- soit on désire apprendre pour la classe k la valeur 0

Si on désire obtenir la valeur 1 pour la classe k alors :

$$\frac{\partial C}{\partial P_k} = \frac{-1}{P_k} \tag{A.7}$$

Si on désire obtenir la valeur 0 pour la classe k alors :

$$\frac{\partial C}{\partial P_k} = 0 \tag{A.8}$$

On pose pour la suite :

$$\frac{\partial C}{\partial H_k} = \frac{\partial C}{\partial P_k}\frac{\partial P_k}{\partial O_k}\frac{\partial O_k}{\partial H_k} \tag{A.9}$$

Dans le cas où l'on désire obtenir la valeur 1 pour la classe k alors en remplaçant A.6 et A.7 dans A.9 on a :

$$\frac{\partial C}{\partial H_k} = \frac{\partial C}{\partial P_k}\frac{\partial P_k}{\partial O_k}\frac{\partial O_k}{\partial H_k} \tag{A.10}$$

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}\frac{\partial P_k}{\partial O_k}O_k \tag{A.11}$$

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}\left[\sum_{l=1,l\neq k}^{K}\left(\frac{O_l}{(\sum_{j=1}^{K} O_j)^2}\right)\right]O_k \tag{A.12}$$

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}\left[\frac{(\sum_{j=1}^{K} O_j) - O_k}{(\sum_{j=1}^{K} O_j)^2}\right]O_k \tag{A.13}$$

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}\left[\frac{(\sum_{j=1}^{K} O_j) - O_k}{(\sum_{j=1}^{K} O_j)}\right]\frac{O_k}{(\sum_{j=1}^{K} O_j)} \tag{A.14}$$

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}\left[1 - \frac{O_k}{(\sum_{j=1}^{K} O_j)}\right]\frac{O_k}{(\sum_{j=1}^{K} O_j)} \tag{A.15}$$

D'où on obtient finalement :

$$\frac{\partial C}{\partial H_k} = \frac{-1}{P_k}[1 - P_k]P_k = P_k - 1 \tag{A.16}$$

Dans le cas où l'on désire obtenir la valeur 0 pour la classe k l'effet de l'erreur est uniquement transmis par la normalisation issue de la fonction $softmax$ (la dérivée de la fonction d'erreur vis-à-vis d'une unité de sortie pour laquelle la sortie désirée est 0 est nulle).

On obtient à l'aide de calculs similaires :

$$\frac{\partial C}{\partial H_k} = P_k \tag{A.17}$$

On conclut alors que

$$\frac{\partial C}{\partial H_k} = P_k - T_k, \forall k \tag{A.18}$$

où T_k désigne les valeurs de probabilité désirées (target).

Il ne reste ensuite plus qu'à inclure la partie couche linéaire de notre modèle graphique pour avoir les dérivées partielles $\frac{\partial C}{\partial w_{ik}}$.

Annexe B

Apprendre avec peu d'exemples : une étude empirique

Learning with few examples: an empirical study on leading classifiers

Christophe Salperwyck and Vincent Lemaire

Abstract— **Learning algorithms proved their ability to deal with large amount of data. Most of the statistical approaches use defined size learning sets and produce static models. However in specific situations: active or incremental learning, the learning task starts with only very few data. In that case, looking for algorithms able to produce models with only few examples becomes necessary. The literature's classifiers are generally evaluated with criterion such as: accuracy, ability to order data (ranking)... But this classifiers' taxonomy can dramatically change if the focus is on the ability to learn with just few examples. To our knowledge, just few studies were performed on this problem. The study presented in this paper aims to study a larger panel of both algorithms (9 different kinds) and data sets (17 UCI bases).**

I. Introduction

Learning machines have shown their ability to deal with huge volumetry on real problems [1], [2]. Nevertheless most of the works were realized for data analysis on homogeneous and stationary data. Usually learning machines use data set with fixed sizes and produce static models. However in certain situations, the learning task starts with only few data. In such cases finding algorithms able to produce accurate models with few data and low variance is an advantage. Active and incremental learning are the two main learning problems where a learning machine able to learn with few data is necessary. This study only focuses on supervised learning.

Active learning [3] is used when lots of data are available but labeling them is expensive (indeed labels are bought). In that case the goal is to select the smallest amount of data which will provide the best model. These data are expected to be very expressive and an algorithm able to deliver an accurate model with just few data is needed in order to avoid buying more data.

Incremental learning [4] start to learn with few examples as it theoretically has to learn from the first provided examples. The model is then improved as new examples are arriving. The model quality at the beginning depends on the algorithm capacity to learn fast with few examples. Incremental learning research started a long time ago but it recently reappears with data stream mining. Indeed numerous software are generating data streams: sensor networks, web pages access logs... These data arrive fast and are only visible once. Therefore it is mandatory to learn them as soon as they are arriving (on-line learning). Incremental learning appears to be a natural solution to solve streams problems. An example is Hoeffding trees which are widely used in incremental learning on data streams. The tree construction is incremental and nodes are transformed into leaves as examples are arriving. Having a classifier in the tree leaves [5] before they will be transformed, appears to improve the tree accuracy. A classifier that can learn with few data will provide a pertinent local model in the leaves.

The most used classifiers such as decision tree, neural network, support vector machine... are often evaluated with criterion such as accuracy, ability to rank data... But this classifiers taxonomy can be completely different if the focus is on their ability to learn with just few examples.

To our knowledge, the state of art presents only few studies on the learning performance versus the size of the learning data set: in [6] the performance on small and not balanced text datasets is studied using 3 different classifiers (Support Vector Machine (SVM), naive Bayes and logistic regression). In [7], the authors focus on the learning time contrary to this study which focus on the performance versus the size of the training set. In [8] and in [9] the focus is respectively on Parzen Windows and k nearest neighbor. In [10] the construction of linear classifiers are considered for very small sample sizes using a stability measure. In [11] the influence of the training set size is evaluated on 4 computer-aided diagnosis problems using 3 different classifiers (SVM, C4.5 decision tree, k-nearest neighbor). In [12] the authors look at how increasing data set size affects bias and variance error decompositions for classification algorithms. The conclusions of these papers will be compared to the results of this empirical study at the end of this paper.

The present work aims to study a larger panel both of learning algorithms (9 different kinds) and data sets (17 from UCI). In a first part (section II) we will present the classifiers that will be used in this study and their parameters. The experimental protocol will be presented section III: data sets, split between training and test sets, evaluation criterion. Section IV will present the results and analyze them depending on the typology of the classifiers. In the last part we will conclude and propose future works related to this study.

II. Learning speed and classifiers typology

A. Learning speed

Firstly this study does not focus on the bounds or convergence time for a classifier given a training set of n examples. This would correspond to determine the CPU time needed for this classifier to learn n examples.

Authors are in the group 'Profiling and Datamining', Orange Labs, 2 avenue Pierre Marzin, 22300 Lannion, France (phone: +33 296 053 107; email: firstname.name@orange-ftgroup.com).

In this study the "learning speed" means the ability for a given classifier to obtain an "interesting" solution to a problem with a minimum of training examples. Figure 1 shows an example of two classifiers on a same problem with two different learning speeds. X-axis shows the number of examples (n) used to train the classifier and Y-axis the AUC obtained (AUC=$f(n)$).

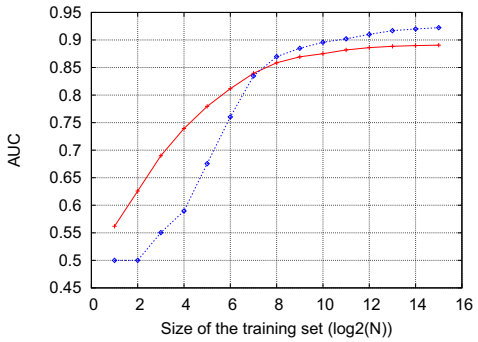


Fig. 1. Illustration of the learning speed of two classifiers (AUC in testing)

This figure illustrates two different behaviors: the "red" (+) algorithm reach a better performance in a first stage but then when n become larger the "blue" (∘) algorithm is finally better. In order to always have the best model available, we obviously have to use both models: at the beginning the red (+) model and then the blue (∘) one.

This perfectly illustrates the purpose of this study: are there better types of learning algorithms to learn on small training data sets?

B. Typology of benchmarked algorithms

Different families of classification models based on learning parameters exist. They could be: (i) linear classifier for which a hyperplane is learned (this kind of classifiers is based on a linear combination of features such as $y = \sum_j w_j X_j$ where X_j represents a feature and w_j its weight); (ii) non linear classifiers such as multi layer perceptrons, k nearest neighbors or decision trees.

Knowing that: (i) these families can partially overlap each other; (ii) their placement is not always easy (a decision tree can be seen as many hyperplanes - one for each leaf); (iii) a linear classifier can deal with non linear classification problem if a projection is done previously (the "kernel trick" in SVM); (iv) sub families also exist. For example linear classifiers can be dispatched into two big sub families to estimate vectors parameters w:

- generative model: it uses an estimate of the conditional probability $P(X|C)$ and the class probability $P(C)$. It is used in the linear discriminant analysis (LDA - [13]) and the naive Bayes classifier [14].
- discriminative model: the goal is to maximize the classification quality on a test set with estimating $P(C|X)$ or a similar score on the train set labels. Linear regression, logistic regression, perceptron and support vector machines are discriminative models.

Other classifiers can also be found: (i) probabilistic classifiers which estimate conditional probabilities of classes given a input vector ($P(C|X)$) constituted by combinations of explanatory variables (this is not the case with many classifiers as scores returned by support vector machines (SVM)); (ii) parametric classifiers which assume that explanatory variables follow a probability law defined beforehand (a priori). One example is the naive Bayes classifier when presupposing that data follow Gaussian distributions [15].

This study can not cover all the classifier families but tried to explore the space at best. Table I summarizes the tested classifiers in relation to their families. These classifiers and their parameters are presented in the following sub section.

	Linear classifier	Non linear classifier
Generative model	Naive Bayes, Selective naive Bayes	Bayesian network, nearest neighbor
Discriminant model	Logistic regression, SVM	Decision tree, Forest of decision trees

TABLE I
TYPOLOGY VERSUS BENCHMARKED CLASSIFIERS.

C. Benchmarked classifiers

In order to evaluate the classifiers presented in the previous section (see Table I), two public software[1] were used: WEKA [16] (version 3.7.1) and Khiops [17] (version 7.0). For both software default values were mostly used (if not, parameters which were tuned are described below) and results are presented in section IV.

- Weka - Bayes
  - Unsupervised [18]: standard naive Bayes. Numeric estimator precision values are chosen based on analysis of the training data.
  - Supervised: same as before but a supervised discretization (MDL) is used to convert numeric attributes to nominal ones.
  - BayesNet [19]: Bayes network learning using various search algorithms and quality measures. Used with default parameters: SimpleEstimator and K2 search algorithm.
- Weka - Nearest-neighbor [20]: uses normalized Euclidean distance to find the training instance closest to the given test instance, and predicts the same class as this training instance.
- Weka - Regression:
  - Logistic regression: multinomial logistic regression model with a ridge estimator which is based on [21].
  - SVM: Support Vector Machine implementation by [22]. The Weka wrapper is made by [23]. Parameters used were: C-SVC for SVM type, 100MB for

[1] All experiments in this benchmark are reproducible.

cache size and "normalize" set to true. Two kernel types were tried: linear and radial basis function (RBF). A cross-validation was used to optimize RBF parameters for all training set sizes.

- Weka - Trees:
  - ADTree [24]: decision tree with many subtrees and combined with boosting.
  - J48 [25]: C4.5 decision tree proposed by Quinlan.
  - SimpleCart [26]: binary tree based on the Gini coefficient.
  - Random Forest [27]: forest of random trees. Default number of trees in the forest is 10. 40 was also tried.
- Weka - Vote: VFI [28]: classification by voting feature intervals. Intervals are constructed around each class for each attribute (basically discretization). Class counts are recorded for each interval on each attribute and the classification is by voting. The default parameters are using the "weight feature intervals by confidence". It was also tried without it.
- Khiops[2]: a tool developed by Orange Labs. It implements, for supervised classification, a naive Bayes and a selected naive Bayes.
  Naive Bayes or Selective Naive Bayes [29] were tested with different pretreatments on numerical and nominal attributes. On nominal attributes two pretreatments were tested:
  - Basic Grouping: a group per observed value
  - MODL discretization [30]

  On numerical attributes three pretreatments were tested:
  - Equal Frequency
  - Equal Width
  - MODL grouping method [31]

  The two unsupervised Equal Frequency and Equal Width methods are used with a fixed number of bins set to 10. If the number of observed values is below 10 the number of bins is reduced to the number of observed data.

III. Experimental protocol

The experimental protocol aims to show the impact of the training size on the learner performances. First data sets are presented: they have different characteristics and are recognized by the data mining community. Then the training/testing split of the data sets is explained. Finally a criterion which point up algorithms that learn well with few data is presented.

A. Data set

Different data sets from the data mining community were used in this study. We chose a panel of data sets from the UCI repository [32]. Most of these data sets were already used for benchmarking classifiers in the literature. Data sets with only nominal features, or only numerical features, or a mix of both were used. Sizes are from one hundred to many thousands examples. Table II presents the data sets characteristics in terms of number of examples, number of numerical and nominal features, accuracy of the majority vote classifier.

This study only focuses on binary classification problems. For problems with more than two classes the experimental protocol should be different.

	Data set	#Var	#Num. feat.	#Nom. feat.	#Exa. (n)	Maj. acc.
1	Adult	15	7	8	48842	0.7607
2	Australian	14	6	8	690	0.5550
3	Breast	10	10	0	699	0.6552
4	Bupa	6	6	0	345	0.5797
5	Crx	15	6	9	690	0.5550
6	German	24	24	0	1000	0.7
7	Heart	13	10	3	270	0.5555
8	Hepatitis	19	6	13	155	0.7935
9	Horsecolic	27	7	20	368	0.6304
10	Hypothyroid	25	7	18	3163	0.9522
11	Ionosphere	34	34	0	351	0.6410
12	Mushroom	22	0	22	8416	0.5332
13	Pima	8	8	0	768	0.6510
14	SickEuthyroid	25	7	18	3163	0.9073
15	Sonar	60	60	0	208	0.5336
16	Spam	57	57	0	4307	0.6473
17	Tictactoe	9	0	9	958	0.6534

TABLE II
DATASET CHARACTERISTICS

B. Split of the data sets into learning and testing sets

In order to generate small data sets and keep the widely used 10 cross validations, first a 90%/10% split was used. Then small training data sets were drawn from the 90% training set. The training data sets sizes are taken from the following sizes: $S = \{S_1, S_2, ..., S_{max}\} = \left\{2, 2^2, 2^3, , \ldots, 2^{(\lfloor log_2(0.9n-1) \rfloor)}\right\}$; where n is the original size of the data set[3]. This draw was performed 10 times for all data sets in order to obtain a mean and variance on the result. The algorithm performance is evaluated on the remaining 10% of the cross validation. Figure 2 illustrates this protocol.

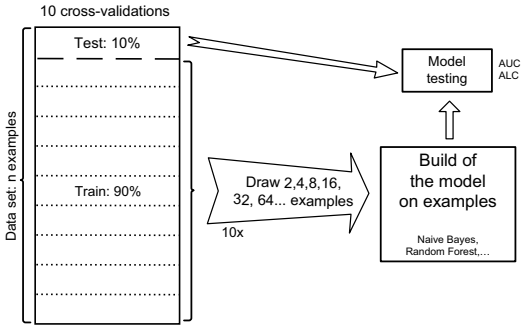


Fig. 2. Data sets construction

[2] www.khiops.com

[3] 0.9n come from the 10 cross validation.

C. Evaluating criterion: ALC

Those experiments give for each dataset a AUC curve [33] on the test dataset versus the number of examples used to train the learning machine. Each curve is constituted of $|S|$ points for all the learning machines as shown in figure 3.

Performances in prediction were evaluated with ALC (Area under the Learning Curve - [34]). AUC (Area Under the ROC Curve) is calculated for every point $(2, 4, 8...)$ defined in the data set. The obtained score corresponds to the normalized ALC calculated as follow:

$$score = \frac{(ALC - Arand)}{(Amax - Arand)}$$

where $Amax$ is the area under the best achievable learning curve (*i.e.* 1) and $Arand$ is the area under the learning curve obtained by random predictions (*i.e.* 0.5).

This criterion has good properties considering the goal of this benchmark. Indeed this criterion emphasizes the AUC on smaller subsets, what is especially the focus of this study. The x-axis is logarithmic which means that the AUC for 2 examples will contribute to the ALC as much as the AUC for 4096 examples.

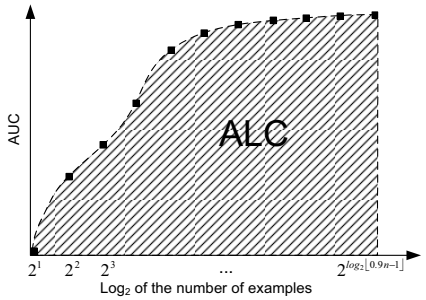


Fig. 3. ALC computation: area under AUC curve

IV. Results

This section presents the study results. The notations used are first described, then tables and curves are presented, finally they are analyzed and an interpretation is proposed.

A. Notations

In order to show compact results, we had to use abbreviation for algorithm names. The first part of the name represents the software used: "W" for algorithms in Weka and no prefix for those which came from Khiops. The name of the tested algorithm constitutes the second part of the abbreviation. The following list gives the meaning of all abbreviations:

- Weka
  - W-ADT: ADTree
  - W-BN: BayesNet - Bayesian network
  - W-IB1: Nearest-neighbor
  - W-RegLog: logistic regression
  - W-NB-NS: non supervised naive Bayes
  - W-NB-S: supervised naive Bayes
  - W-RF10/40: Random Forest with 10/40 trees
  - W-VFI / W-VFIn: VFI with/without option "weight feature intervals by confidence"
  - W-SCart: SimpleCart
  - W-SVM-Lin/W-SVM-RBF: SVM with a linear/radial basis function kernel type
- Khiops. The format is Algorithm – Cont_Var – Nom_Var
  - Algorithm - NB: Naive Bayes or SNB: Selective Naive Bayes
  - Cont_Var for continuous variables - EF: EqualFrequency / EW: EqualWidth / M: MODL
  - Nom_Var for nominal variables - BG: Basic Grouping / M: MODL

B. Tables and curves

1) Tables: The ALC performances obtained by the different algorithms on all data sets are presented in table III. The last line gives the averaged ALC on all data sets for a given algorithm. In order to have a more synthetic view, table V shows the averaged rank, averaged ALC and averaged final AUC (AUC on the largest training set: $2^{(\lfloor log_2(0.9n-1) \rfloor)}$).

As this study focuses on learning from small data sets, the beginning of the AUC curve is particularly interesting. To highlight it, ALC is calculated just between 2 and 2^6 examples so that the ending of the AUC curve is ignored. These results are presented in table IV which is similar to the previous one (table III). The computation was done in the same way for the ranks. The results are presented in table VI.

The results variances are shown using a box plot per algorithm on figure 5.

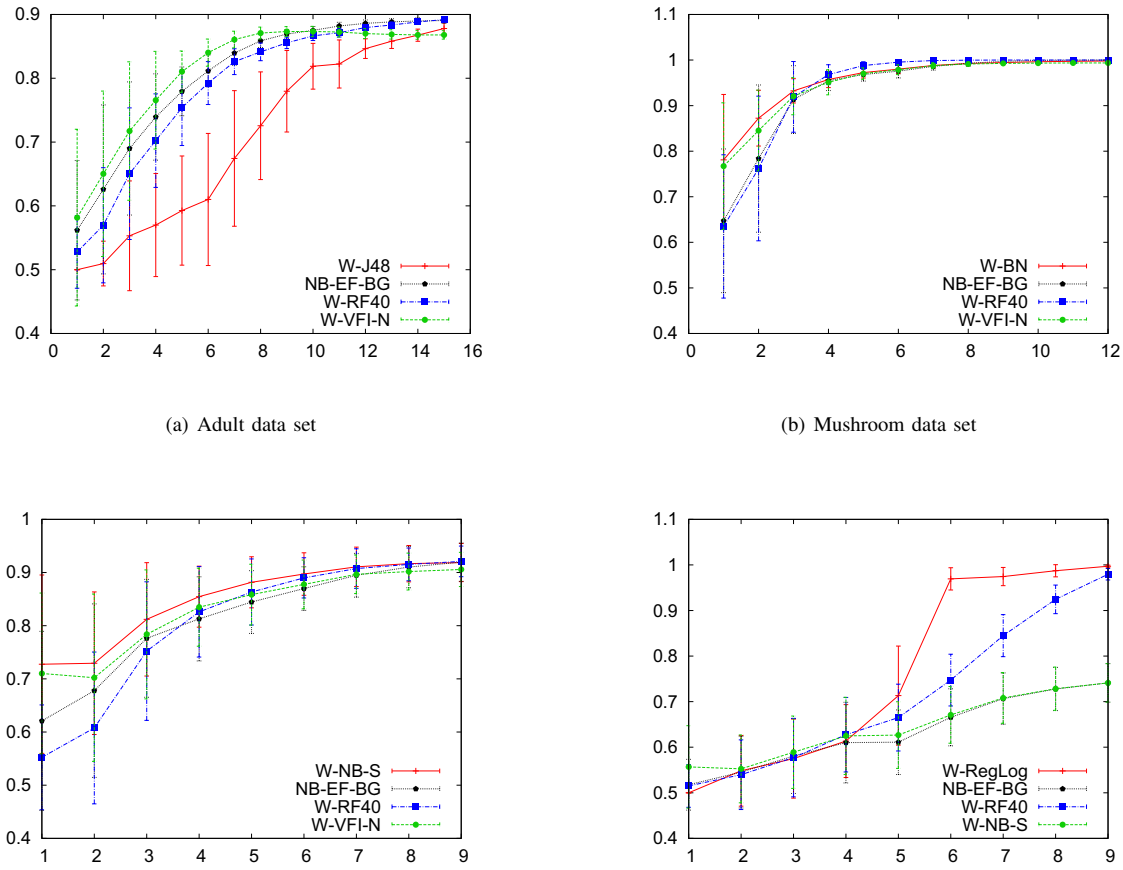
2) Curves: Four curves were chosen to show the performances of different algorithms. The idea is to compare the behavior of a particular algorithm with the best algorithms in this study. The comparison is done on data sets adapted to this particular algorithm in order to observe how the best algorithms perform in that case. The best example is the figure 4(d) which shows a data set working very well for the logistic regression but not that well for the naive Bayes.

C. Analysis

1) Global analysis: This study was done without a fine tuning of the algorithms' parameters but mainly using their default values. C4.5 (J48), CART (SimpleCart) and SVM are references in data mining but this study shows that they don't perform well on small datasets. Indeed they have the worst averaged ranks and worst averaged ALCs.

On the contrary, naive Bayes algorithms are known to perform well on small datasets [35]. Figures 4(a) and 4(b) confirm it. Very rapidly the generated classifiers perform well. After seeing around $2^5(32)$ to $2^8(256)$ examples they are almost at their maximum accuracy. Even if naive Bayes classifiers perform better than C4.5, they are not the best on the smaller datasets.

Tree classifiers do not perform well out of the box (C4.5, Cart), but surprisingly in combination with bagging/boosting



(a) Adult data set

(b) Mushroom data set

(c) Crx data set

(d) TicTacToe data set

Fig. 4. AUC versus the size of the training set (log_2)

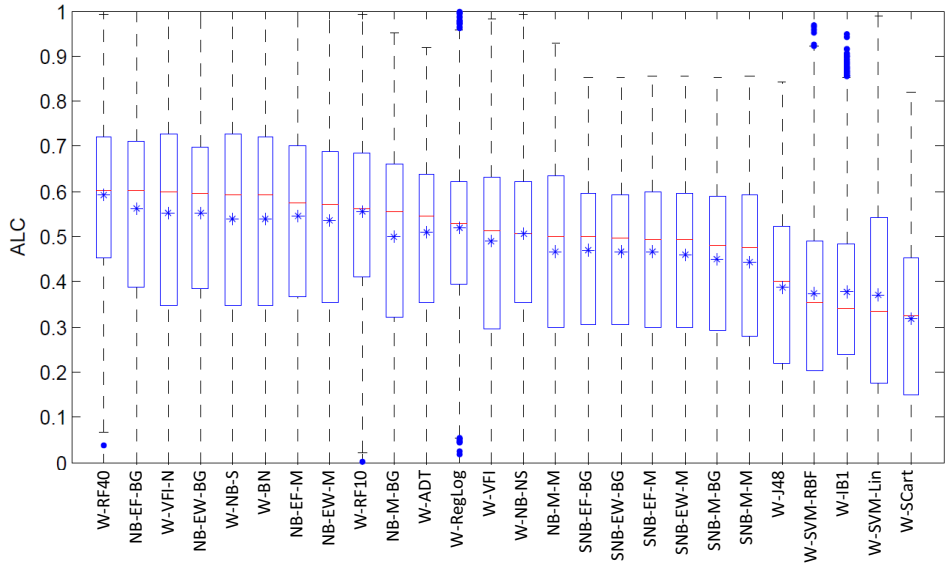


Fig. 5. Results variance using box plots

	W-ADT	W-BN	W-IB1	NB-EF-BG	NB-EF-M	NB-EW-BG	W-NB-NS	W-NB-S	W-RegLog	W-RF10	W-RF40	W-SVM-Lin	W-VFI	W-VFI-N
Adult	58.05	**64.28**	32.60	62.31	58.98	62.64	59.88	63.70	54.15	54.73	58.44	18.04	58.02	63.80
Australian	59.40	71.55	48.99	65.75	64.07	64.11	56.43	**72.15**	55.16	59.88	63.23	53.56	59.03	69.06
Breast	74.97	89.12	80.85	**92.31**	**92.31**	92.18	88.26	89.04	87.28	88.25	89.89	83.30	88.40	92.30
Bupa	20.17	2.02	8.79	14.81	14.81	13.20	12.52	1.98	**24.06**	21.74	23.99	3.02	9.05	9.32
Crx	60.16	70.12	47.73	63.91	62.79	62.32	56.30	**70.63**	54.59	59.02	62.47	51.44	57.33	66.57
German	25.98	13.10	11.23	27.61	27.61	26.13	30.53	13.12	26.69	27.29	**31.69**	18.58	24.84	26.89
Heart	47.19	62.19	42.42	61.31	58.26	60.21	54.56	62.84	51.25	53.64	57.15	47.33	49.75	**64.22**
Hepatitis	39.92	60.38	26.77	49.64	46.22	47.78	37.92	**63.16**	37.27	46.93	50.36	31.86	46.71	58.88
Horsecolic	52.73	59.81	41.35	51.87	44.01	52.06	40.61	61.58	48.31	57.34	**62.27**	36.28	54.20	58.31
Hypothyroid	59.21	50.78	25.42	65.87	**66.75**	64.06	52.81	48.77	59.27	63.05	66.13	24.89	52.94	60.41
Ionosphere	57.42	59.18	39.91	65.91	65.91	64.68	56.58	59.06	43.17	65.86	**70.29**	42.43	52.06	61.54
Mushroom	82.20	**92.23**	83.38	88.99	86.26	88.99	92.07	92.07	87.20	88.76	89.97	84.20	89.25	91.01
Pima	37.70	30.76	21.79	41.50	41.50	40.16	38.35	30.84	41.99	40.68	**44.43**	24.63	26.77	30.06
SickEuthyroid	60.46	52.57	22.90	59.44	59.29	59.50	50.28	49.25	57.49	61.87	**66.85**	3.70	45.41	47.93
Sonar	32.64	36.54	32.29	38.10	38.10	37.32	31.72	36.51	36.05	41.80	**48.32**	31.54	23.12	34.37
Spam	66.53	72.86	45.95	79.93	79.93	77.29	71.07	72.76	68.76	75.92	**80.08**	58.29	67.72	78.29
Tictactoe	32.40	28.40	30.55	26.91	20.52	26.91	28.73	28.73	**53.24**	37.77	41.83	16.32	26.00	27.61
Averaged ALC	51.01	53.88	37.82	56.25	54.55	55.27	50.51	53.89	52.11	55.56	**59.26**	37.02	48.86	55.33

	W-J48	NB-EW-M	NB-M-BG	NB-M-M	SNB-EF-BG	SNB-EF-M	SNB-EW-BG	SNB-EW-M	SNB-M-BG	SNB-M-M	W-SVM-RBF	W-SCart
Adult	41.68	59.25	64.06	56.63	55.25	55.13	55.73	55.52	57.92	56.02	15.41	40.97
Australian	51.60	62.32	65.60	63.22	53.20	54.13	52.04	53.04	54.85	57.18	42.16	42.70
Breast	66.84	92.18	76.24	76.24	79.29	79.29	79.40	79.40	76.30	76.30	81.05	57.85
Bupa	10.54	13.20	1.91	1.91	13.58	13.58	12.00	12.00	1.91	1.91	16.61	8.53
Crx	51.68	61.01	63.73	62.90	52.92	54.38	51.99	53.15	54.50	57.31	40.21	44.03
German	14.39	26.13	14.14	14.14	22.27	22.27	22.11	22.11	14.15	14.15	16.17	10.06
Heart	29.21	56.90	57.34	45.95	40.91	40.75	38.92	38.76	42.81	41.28	43.17	22.88
Hepatitis	21.05	44.66	46.83	40.49	30.99	32.43	31.32	32.50	30.52	31.34	25.28	10.62
Horsecolic	40.18	44.06	55.08	49.01	45.12	40.67	45.36	40.90	47.42	43.50	10.45	2.04
Hypothyroid	43.79	65.06	54.46	52.09	56.56	56.62	60.77	61.03	53.04	52.56	42.55	39.54
Ionosphere	42.89	64.68	53.04	53.04	55.15	55.15	55.48	55.48	53.42	53.42	45.23	36.27
Mushroom	77.71	86.26	88.99	86.26	82.56	81.77	82.56	81.77	82.56	81.77	79.76	74.62
Pima	25.34	40.16	29.71	29.71	37.65	37.65	37.56	37.56	29.89	29.89	26.63	20.08
SickEuthyroid	47.80	59.64	52.14	45.32	56.16	55.95	55.57	55.68	46.20	45.49	39.23	45.21
Sonar	19.40	37.32	31.97	31.97	25.26	25.26	25.21	25.21	27.12	27.12	29.37	16.20
Spam	52.67	77.29	66.63	66.63	65.93	65.93	60.53	60.53	66.33	66.33	55.96	46.11
Tictactoe	21.79	20.52	26.91	20.52	25.50	20.29	25.50	20.29	25.50	20.29	24.02	25.47
Average	38.74	53.57	49.93	46.83	46.96	46.54	46.59	46.17	44.97	44.46	37.25	31.95

TABLE III

ALC PER ALGORITHM AND DATA SET

	W-ADT	W-BN	W-IB1	NB-EF-BG	NB-EF-M	NB-EW-BG	W-NB-NS	W-NB-S	W-RegLog	W-RF10	W-RF40	W-SVM-Lin	W-VFI	W-VFI-N
Adult	28.64	39.37	23.07	40.85	30.41	40.73	29.73	38.23	30.01	30.58	33.46	9.02	37.73	**46.18**
Australian	45.86	65.32	42.35	57.02	53.94	55.99	44.12	**66.17**	45.52	48.43	52.13	42.89	51.11	62.37
Breast	61.52	83.70	75.44	88.89	88.89	**88.92**	82.76	83.63	82.09	82.88	85.20	77.85	84.21	88.86
Bupa	14.19	2.02	5.74	10.48	10.48	9.76	8.26	1.96	**17.75**	14.90	16.28	3.01	8.30	8.66
Crx	46.73	62.86	40.99	54.27	52.00	53.40	43.71	**63.59**	44.37	46.86	50.82	39.15	48.45	58.91
German	17.49	4.54	7.54	17.20	17.20	16.92	18.51	4.59	15.51	18.64	**21.74**	12.00	15.29	18.44
Heart	42.48	59.44	41.15	58.07	54.24	57.68	49.47	60.14	47.95	50.22	53.59	44.20	46.64	**61.78**
Hepatitis	36.16	57.85	24.18	46.30	41.94	45.26	32.16	**60.91**	35.04	43.53	46.90	28.87	43.34	55.85
Horsecolic	44.41	54.46	35.25	44.70	33.88	45.11	29.08	**57.08**	41.05	49.18	54.92	27.15	49.17	52.52
Hypothyroid	26.39	9.91	9.17	38.32	**38.81**	37.85	16.01	7.44	33.30	35.21	37.81	18.25	35.21	34.46
Ionosphere	45.63	47.00	30.88	58.20	58.20	56.82	44.75	46.89	35.35	55.36	**60.61**	32.75	37.83	50.35
Mushroom	61.75	**84.61**	65.36	77.27	71.14	77.27	84.41	84.41	73.47	75.56	78.09	67.80	77.66	82.42
Pima	28.26	19.00	17.53	31.50	31.50	31.17	26.76	19.12	30.58	32.82	**35.95**	18.55	26.51	31.45
SickEuthyroid	31.74	17.08	10.56	37.90	36.64	39.65	20.62	12.17	32.90	35.28	**42.33**	4.40	33.59	34.56
Sonar	26.88	31.71	26.03	33.12	33.12	33.15	26.72	31.65	33.80	36.10	**42.42**	27.83	22.51	33.67
Spam	44.03	54.31	29.49	68.25	68.25	67.19	55.39	54.33	57.14	60.64	66.77	46.13	55.83	**68.26**
Tictactoe	17.09	19.82	16.59	17.54	10.61	17.54	20.27	20.27	**27.38**	19.21	21.63	11.03	16.04	18.71
Average	36.43	41.94	29.49	45.88	43.01	45.55	37.22	41.92	40.19	43.26	47.10	30.05	40.55	**47.50**

	W-J48	NB-EW-M	NB-M-BG	NB-M-M	SNB-EF-BG	SNB-EF-M	SNB-EW-BG	SNB-EW-M	SNB-M-BG	SNB-M-M	W-SVM-RBF	W-SCart
Adult	11.21	30.34	40.53	19.72	23.43	20.65	23.48	21.34	24.63	17.83	2.04	4.35
Australian	38.85	52.99	57.47	52.79	36.58	37.57	36.15	37.43	39.30	42.05	27.54	24.83
Breast	54.86	88.92	63.19	63.19	67.84	67.84	68.03	68.03	63.14	63.14	74.68	40.97
Bupa	6.78	9.76	1.31	1.31	8.87	8.87	8.14	8.14	1.30	1.30	9.22	4.23
Crx	39.50	51.15	54.52	52.17	36.12	37.66	36.06	37.56	38.72	42.01	24.55	26.97
German	8.43	16.92	3.82	3.82	9.83	9.83	10.95	10.95	3.81	3.81	9.13	2.88
Heart	25.93	53.66	53.81	39.91	34.19	33.94	32.90	32.73	36.44	34.35	40.36	17.85
Hepatitis	18.19	41.07	43.88	35.83	26.66	27.93	26.98	28.10	25.99	26.20	22.47	8.54
Horsecolic	31.30	34.56	48.65	39.77	34.61	27.70	35.41	28.48	37.81	30.81	4.59	1.67
Hypothyroid	11.47	38.19	18.51	14.28	19.32	19.50	29.47	29.63	14.87	14.49	20.44	7.11
Ionosphere	31.15	56.82	39.30	39.30	41.67	41.67	42.30	42.30	38.53	38.53	33.14	21.60
Mushroom	52.99	71.14	77.27	71.14	62.30	60.64	62.30	60.64	62.30	60.64	59.59	46.68
Pima	18.38	31.17	15.81	15.81	24.77	24.77	26.04	26.04	15.86	15.86	19.37	9.92
SickEuthyroid	16.31	38.65	22.80	9.66	25.84	25.53	27.55	27.60	9.53	8.90	16.14	10.63
Sonar	15.39	33.15	26.04	26.04	18.60	18.60	19.24	19.24	21.24	21.24	25.02	11.52
Spam	32.00	67.19	41.92	41.92	38.96	38.96	35.31	35.31	40.13	40.13	38.09	21.05
Tictactoe	8.92	10.61	17.54	10.61	13.53	9.83	13.53	9.83	13.53	9.83	9.99	5.50
Average	24.80	42.72	36.85	31.60	30.77	30.09	31.40	30.79	28.65	27.71	25.67	15.66

TABLE IV

ALC PER ALGORITHM AND DATA SET WITH AREA BETWEEN 2 AND $2^6 = 64$ EXAMPLES DURING TRAINING

Algorithm	Avg rank	Avg ALC	Avg final AUC
W-RF40	**3.65**	**59.26**	**91.30**
NB-EF-BG	4.65	56.25	88.04
NB-EW-BG	6.59	55.27	86.82
W-VFI-N	6.94	55.33	83.13
NB-EF-M	7.24	54.55	88.04
W-RF10	7.29	55.56	89.70
W-BN	8.53	53.88	87.36
W-NB-S	8.82	53.89	87.35
NB-EW-M	9.18	53.57	86.80
W-RegLog	10.94	52.11	88.55
W-ADT	11.24	51.01	88.66
W-NB-NS	11.29	50.51	87.09
NB-M-BG	12.53	49.93	86.81
W-VFI	13.94	48.86	82.04
SNB-EF-BG	15.76	46.96	88.15
SNB-EW-BG	16.35	46.59	87.06
NB-M-M	16.65	46.83	86.84
SNB-EF-M	16.94	46.54	88.20
SNB-EW-M	17.47	46.17	86.99
SNB-M-BG	17.53	44.97	87.07
SNB-M-M	18.94	44.46	87.19
W-IB1	20.59	37.82	78.07
W-SVM-Lin	20.65	37.02	76.44
W-SVM-RBF	20.71	37.25	81.51
W-J48	22.18	38.74	82.84
W-SCart	24.41	31.95	81.70

TABLE V

AVERAGED RANK, AVERAGED ALC AND AVERAGED FINAL AUC

Algorithm	Avg rank	Avg ALC	Avg final AUC
W-RF40	**4.35**	47.10	**86.21**
W-VFI-N	4.65	**47.50**	82.43
NB-EF-BG	4.65	45.88	83.45
NB-EW-BG	5.41	45.55	82.34
W-RF10	6.88	43.26	84.09
NB-EF-M	7.65	43.01	83.29
NB-EW-M	8.47	42.72	82.12
W-BN	9.41	41.94	81.58
W-NB-S	9.47	41.92	81.28
W-RegLog	10.00	40.19	79.98
W-VFI	10.53	40.55	77.91
W-NB-NS	11.88	37.22	82.60
W-ADT	12.24	36.43	83.02
NB-M-BG	12.76	36.85	80.92
SNB-EW-BG	16.88	31.40	81.93
SNB-EF-BG	17.18	30.77	82.44
NB-M-M	17.59	31.60	80.29
W-SVM-Lin	17.59	30.05	72.97
SNB-EW-M	18.24	30.79	81.76
SNB-EF-M	18.59	30.09	82.20
W-IB1	18.71	29.49	72.46
SNB-M-BG	19.59	28.65	80.18
W-SVM-RBF	19.82	25.67	72.98
SNB-M-M	21.35	27.71	80.12
W-J48	21.94	24.80	74.14
W-SCart	25.18	15.66	70.52

TABLE VI

AVERAGED RANK, AVERAGED ALC AND AVERAGED FINAL AUC WITH AREA BETWEEN 2 AND $2^6 = 64$ EXAMPLES DURING TRAINING

techniques they are the best of this study: Random Forest. Figure 4 shows Random Forest (with a forest size of 40: W-RF40) results on four datasets. W-RF40 is the overall best classifier in this study. The data sets Crx and TicTacToe on sub-figures 4(c) and 4(d) are favorable to naive Bayes and logistic regression, but even on them Random Forest is still performing well. In a general manner its performances are better than the other classifiers on most of the data sets.

It is surprising to discover a pretty unknown classifier close to the top of this study: VFI - a vote by majority based on a discretization technique. This algorithm is working pretty well without using the option "weight feature intervals by confidence": W-VFI-N. If the learning phase is stopped after the first 64 examples (Table VI), W-VFI-N is first on ALC and is ranked second.

2) Discriminative models analysis: Discriminative algorithms are represented in this study by the Random Forests (W-RF and W-RF40), the logistic regression (W-RegLog) and SVM (W-SVM-Lin, W-SVM-Log). The Random Forest algorithm set up with a size of 40 trees is the best performer in this study on all aspects: rank, averaged ALC and averaged final AUC. This algorithm ability to try numerous trees with just few features gives good results both at the beginning and at the end of the learning phase.

Even if the logistic regression is positioned after Random Forest, naive Bayes and VFI, its ALC score is not that far. In certain cases it is even much better than all the others classifiers as shown on the figure 4(d) for the TicTacToe date set.

Support Vector Machines (SVM) are not performing well on small data sets. Linear SVMs are a bit better than RBF ones but still their performances are one of the lowest in this study. These results are surprising and further experiments should be conducted to double check them.

3) Generative models analysis: Among the generative models tested in this study, naive Bayes are the most evaluated. The methods used to discretize continuous variables and grouping nominal variables have a great influence on the results. The best choice is constituted by the pairs ("Equal-Frequency", "Basic Grouping") followed by ("EqualWidth", "Basic Grouping"). The regularized discretization [29] and the MODL grouping method [31] are too robust to be able to build a model with just few examples. Regularized methods (MODL approach and selective naive Bayes) are known to have low bias-variance [36]. Therefore, in order to maintain this low bias-variance, their variance but also AUC will be lower on the smaller datasets. This statement is confirmed as the AUC increases later: the classifier is more conservative and prefers not to make highly variable predictions. This robustness makes these classifiers to have a lower score but still they are better than C4.5 or Cart. In this study "simpler methods" have the advantage to more rapidly find patterns and give better results.

Bayesian network represents non linear generative model (see table I) in this study. It behaves relatively well but they are just below Random Forest, VFI and naive Bayes. On few

data sets: Adult and Mushrooms (figure 4(b)), it is the best classifier in terms of ALC.

4) Comparisons with existing analysis: The results in the literature are confirmed in this study. A ranking could be proposed:

- Generative classifiers are better than discriminative classifiers when the number of examples is low [37] and when only one classifier is used.
- Ensemble of classifiers performs very well [38]: bagging [39] of discriminative classifiers performs very well and allows to have a reduced variance (compare to an only one discriminative classifier). In this study, the bagging of discriminative classifiers performs better than a single generative classifier.
- Ensemble of generative classifiers [40] has to be tested and compared to ensemble of discriminative classifiers.
- Regularized methods are robust [36].

Few studies on the learning performance versus the size of the learning data set have been identified in the introduction of this paper. This paragraph seeks if our empirical study conclusions are confirmed.

In [6] the training size and class distribution vary but they are using a different measure: the learning surface, different from the traditional learning curve used in our study. In [7], the authors focus on the learning time contrary to this study which focuses on the performance versus the size of the training set. Therefore both of these studies can not be compared with ours.

In [9] the focus is on the k nearest neighbor. Experimental results show that the nearest neighbor classifier designed on the bootstrap samples outperforms the conventional k-NN classifiers, particularly in high dimensions. Since the bootstrapping method used is closed to the bagging procedure their conclusion does not enter in conflict with our recommendations. In [10] linear classifiers are considered for very small sample sizes using a stability measure. One conclusion of their paper is: if classifiers become instable for small sample sizes, a general stabilizing technique like bagging could improved classifiers. This conclusion is in accordance with the empirical study presented here. In [11] the influence of the training set size is evaluated on 4 computer-aided diagnosis problems using 3 different classifiers (C4.5, Linear and RBF SVM). They observe that classifier results on small data sets should not be generalized to larger. Their empirical results suggest that, given sufficient training data, SVMs tend to be the best classifiers. In [12] the authors study how the data set size affects bias and variance error decompositions for classification algorithms. The paper conclusion is in two parts. The first one indicates that there is no clear effect of training set size on bias. This is confirmed in our study since the mean rank does not really change between Table V and Table VI. The second part of the conclusion is that variance can be expected to decrease as the training set size increases. This second part confirms our results. Moreover the bagging procedure is able to reduce this variance [39].

5) Recommendation: The study presented in this paper recommends that few algorithms could be used to build a model on small training data sets: random forest and the naive Bayes classifier. Both these algorithms need to be parameterized to reach their best performances. The association ("EqualFrequency", "Basic Grouping") and ("EqualWidth", "Basic Grouping") are the best ones for the naive Bayes and a size of 40 trees for "Random Forest". If we only focus on the early learning stage, a method using vote on intervals (VFI) is ranked as the best methods in this study.

V. Conclusion and Future Works

A. Towards a new criterion?

We previously presented our results in two tables: one containing the results of the area under the AUC curve (ALC) finishing at 2^6 examples (table VI) and another one containing the final AUC (table V). The first table shows the algorithms behaviors at the early learning stage and the second one its performance at the end. On figure 6 the two axis use these two evaluating criterion. On average only "W-RFxx" and "NB-EF-BG" manage to be well positioned on these two criterion. Logistic regression and trees boosting (ADTree) have good final performances but there ALC_{64} is relatively low. On the contrary "VFI" is good on this criterion but its final AUC is not that good.

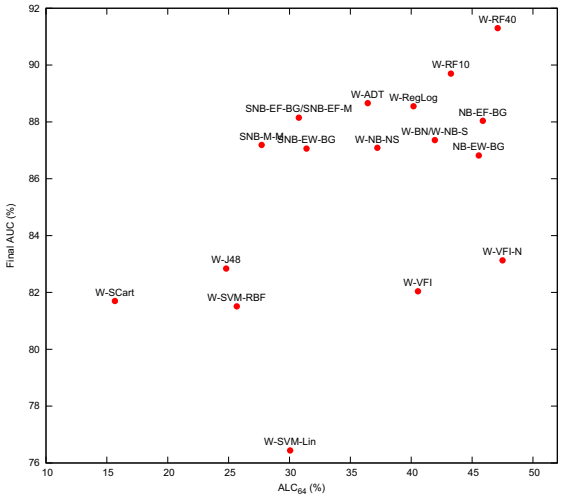


Fig. 6. Positioning of the different algorithms in terms of ALC_{2^6} and AUC_{final} (to avoid text overlapping the classifiers NB-EF-M, NB-EW-M, NB-M-BG, NB-M-M, SNB-EW-M and SNB-M-BG are omitted)

This analysis let us think that the ALC criterion could be replaced by a new criterion which would take into account both of these aspects. Indeed, it would be interesting to have a synthetic criterion allowing us to evaluate algorithm performances on these two axis. This criterion could be written as follow:

$$Criterion = \frac{ALC_{2^6} + AUC_{final}}{2}$$

B. Algorithms combination

In this study very few cases were found where a particular algorithm is very good on the early stage and another one is very good at the end of the learning phase. However if we would be in such situation as it is shown on figure 7, it would be interesting to use two different algorithms: W-VFI-N for the 2^8 first examples and then a selective naive Bayes at the end.

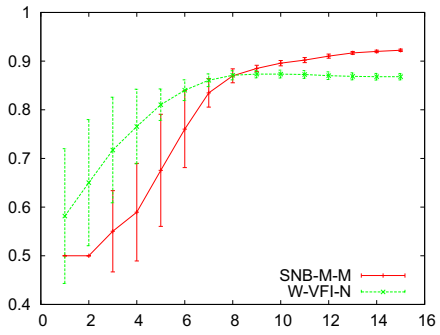


Fig. 7. A data set (Adult) where using 2 algorithms could be beneficial

REFERENCES

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

[2] R. Féraud, M. Boullé, F. Clérot, F. Fessant, and V. Lemaire, "The orange customer analysis platform," in *Industrial Conference on Data Mining (ICDM)*, 2010, pp. 584–594.

[3] B. Settles, "Active learning literature survey," 2010, http://pages.cs.wisc.edu/~bsettles/pub/settles.activelearning.pdf.

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

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

[6] G. Forman and I. Cohen, "Learning from little: Comparison of classifiers given little training," *Knowledge Discovery in Databases: PKDD 2004*, pp. 161–172, 2004.

[7] T. Lim, W. Loh, and Y. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Machine learning*, vol. 40, no. 3, pp. 203–228, 2000.

[8] Y. Muto and Y. Hamamoto, "Improvement of the parzen classifier in small training sample size situations," *Intelligent Data Analysis*, vol. 5, no. 6, pp. 477–490, 2001.

[9] Y. Hamamoto, S. Uchimura, and S. Tomita, "A bootstrap technique for nearest neighbor classifier design," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 19, NO. 1, JANUARY 1997*, vol. 19, no. 1, pp. 73–79, 1997.

[10] M. Skurichina and R. P. W. Duin, "Stabilizing classifiers for very small sample sizes," in *Proceedings of the 13th International Conference on Pattern Recognition - Volume 2*, 1996, pp. 891–.

[11] A. Basavanhally, S. Doyle, and A. Madabhushi, "Predicting classifier performance with a small training set: Applications to computer-aided diagnosis and prognosis," in *Biomedical Imaging: From Nano to Macro. IEEE International Symposium*, 2009.

[12] D. Brain and G. I. Webb, "On the effect of data set size on bias and variance in classification learning," in *Proceedings of the Fourth Australian Knowledge Acquisition Workshop (AKAW '99)*. The University of New South Wales, 1999, pp. 117–128.

[13] R. Fisher, "The use of multiple measurements in taxonomic problems." *Annals of eugenics*, vol. 7, pp. 179–188, 1936.

[14] P. Langley, W. Iba, and K. Thompson, "An analysis of Bayesian classifiers," in *Proceedings of the National Conference on Artificial Intelligence*, no. 415, 1992, pp. 223–223.

[15] J. Gama, P. Medas, and P. Rodrigues, "Learning Decision Trees from Dynamic Data Streams," in *Proceedings of the ACM Symposium on Applied ComputingLearning Decision Trees from Dynamic Data Streams*, 2005.

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

[17] M. Boullé, "Khiops: A Statistical Discretization Method of Continuous Attributes," *Machine Learning*, vol. 55, no. 1, pp. 53–69, Apr. 2004.

[18] G. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 338–345.

[19] R. Bouckaert, "Bayesian Network Classifiers in Weka," 2004, http://weka.sourceforge.net/manuals/weka.bn.pdf.

[20] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[21] S. L. Cessie and J. V. Houwelingen, "Ridge estimators in logistic regression," *Applied Statistics*, 1992.

[22] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[23] Y. EL-Manzalawy and V. Honavar, *WLSVM: Integrating LibSVM into Weka Environment*, 2005, software available at http://www.cs.iastate.edu/~yasser/wlsvm.

[24] Y. Freund and L. Mason, "The alternating decision tree learning algorithm," in *Machine learning*, 1999, pp. 124–133.

[25] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[26] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees*. Chapman and Hall/CRC, 1984.

[27] L. Breiman, "Random forests," *Machine learning*, vol. 25, no. 2, pp. 5–32, 2001.

[28] G. Demiröz and H. Güvenir, "Classification by voting feature intervals," *Machine Learning: ECML-97*, pp. 85–92, 1997.

[29] M. Boullé, "Regularization and Averaging of the Selective Naive Bayes classifier," *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pp. 1680–1688, 2006.

[30] ——, "MODL: A Bayes optimal discretization method for continuous attributes," *Machine Learning*, vol. 65, no. 1, pp. 131–165, May 2006.

[31] ——, "A grouping method for categorical attributes having very large number of values," *Machine Learning and Data Mining in Pattern Recognition*, pp. 228–242, 2005.

[32] C. L. Blake and C. J. Merz, "UCI Repository of machine learning databases," 1998, http://archive.ics.uci.edu/ml/ visit pour la dernire fois : 15/09/2010. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[33] T. Fawcett, "ROC graphs: Notes and practical considerations for researchers," *Machine Learning*, vol. 31, pp. 1–38, 2004.

[34] I. Guyon, G. Cawley, G. Dror, and V. Lemaire, "Results of the Active Learning Challenge," in *JMLR W&CP, Workshop on Active Learning and Experimental Design, collocated with AISTATS, Sardinia, Italy*, vol. 10, 2010, pp. 1–26.

[35] P. Domingos and M. Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," *Machine learning*, vol. 130, pp. 103–130, 1997.

[36] F. Cucker and S. Smale, "Best Choices for Regularization Parameters in Learning Theory: On the Bias-Variance Problem," *Foundations of Computational Mathematics*, vol. 2, no. 4, pp. 413–428, 2008.

[37] G. Bouchard and B. Triggs, "The tradeoff between generative and discriminative classifiers," in *IASC International Symposium on Computational Statistics (COMPSTAT)*, 2004, pp. 721–728.

[38] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1, pp. 105–139, 1999.

[39] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[40] A. Prinzie and D. Van den Poel, "Random multiclass classification: Generalizing random forests to random mnl and random nb," in *Database and Expert Systems Applications*. Springer - Lecture Notes in Computer Science, 2007, pp. 349–358.

Annexe C

Challenge Exploration & Exploitation (Workshop ICML 2011)

JMLR: Workshop and Conference Proceedings vol 26 (2012) 86–97 On-line Trading of Exploration and Exploitation 2

Stumping along a Summary for Exploration & Exploitation Challenge 2011

Christophe Salperwyck CHRISTOPHE.SALPERWYCK@ORANGE.COM
Orange Labs - Lannion, France
LIFL - Université de Lille 3,
Villeneuve d'Ascq, France

Tanguy Urvoy TANGUY.URVOY@ORANGE.COM
Orange Labs - Lannion, France

Editor: Dorota Głowacka, Louis Dorard and John Shawe-Taylor

Abstract

The *Pascal Exploration & Exploitation challenge 2011* seeks to evaluate algorithms for the online website content selection problem. This article presents the solution we used to achieve second place in this challenge and some side-experiments we performed. The methods we evaluated are all structured in three layers. The first layer provides an online summary of the data stream for continuous and nominal data. Continuous data are handled using an online quantile summary. Nominal data are summarized with a hash-based counting structure. With these techniques, we managed to build an accurate stream summary with a small memory footprint. The second layer uses the summary to build predictors. We exploited several kinds of trees from simple decision stumps to deep multivariate ones. For the last layer, we explored several combination strategies: online bagging, exponential weighting, linear ranker, and simple averaging.

Keywords: online learning, click through rate, content selection problem, online summary, probability decision tree, models averaging

1. Problem statement

The general *content selection problem* consists of selecting the best items to display for a given user profile in a given context to maximize the users' satisfaction (roughly represented by the number of clicks): this is a key problem of information retrieval. The considered items may be of any nature (news, ads, books, friends, etc.) and most of the time each couple (user profile, item) is described by a vector of features. Some features like *navigation history* or *cookies* are specific to the user, some features are item-specific, and some features like *textual* or *geographic distances* may involve both the user and the item.

The purpose of the *Pascal Exploration & Exploitation challenge 2011* was to evaluate online content selection algorithms by simulation from real data provided by *Adobe Test&Target* platform. The evaluation being done online, entrants had to submit a piece of Java code encoding their click prediction algorithm to the organizers. The experimental protocol is precisely described on the challenge website: http://explo.cs.ucl.ac.uk/description/. Each algorithm had to perform a sequence of iterations. For each iteration, a batch of six visitor-item pairs was extracted sequentially from the logs and given to the

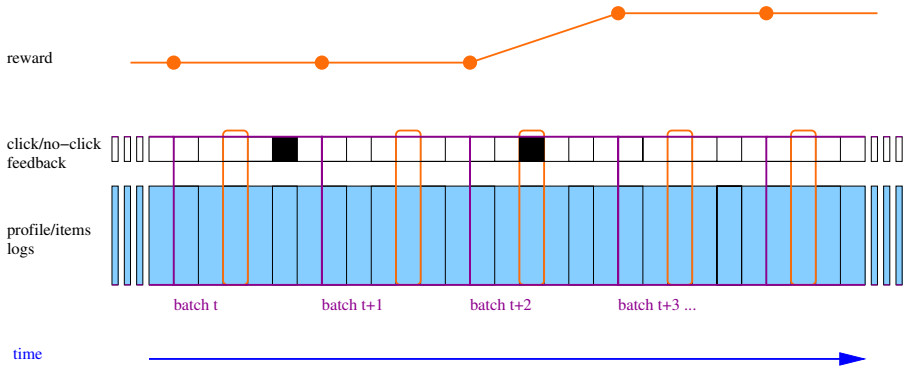


Figure 1: Challenge evaluation process

algorithm. The average click through rate being around 0.24% in the logs, most of the batches did not contain any click at all. The goal was to select the instance which was most likely to provoke a click, knowing that *click* or *no-click* feedback was only given for the selected instance (See Figure 1). The scarcity of clicks made the estimation of probabilities a difficult task. Fortunately, the goal was not to predict accurately the click probabilities of each instance but to respect their ordering according to these probabilities: the content selection problem is a ranking problem (Cortes and Mohri, 2004).

Other strong technical constraints of the challenge were the realistic time limit (100ms per iteration) and the limited space (1.7GB of memory) which motivated the use of stream summaries or incremental models.

In section 2, we detail our three-layers approach. The experiment results for the different layers configurations are given in section 3. In section 4, we discuss about the challenge offline evaluation protocol, and we conclude this article in section 5.

2. Our approach

The originality of our approach is to rely on a stream summary to build click predictions. Our methods are structured in three layers. The first layer, described in section 2.1, provides an online summary of the data stream for continuous and nominal data. The second layer, described in section 2.2, uses the summary to build incremental prediction trees, and the last layer, described in section 2.3, combines predictions trees in order to improve the final result. We experimented several methods for each layer.

2.1. Online summary

With only 100ms per round for predicting and learning and a bounded memory, the constraints of the challenge were rather tight (especially with Java garbage collector side effects). We first experimented a sliding reservoir of instances with standard batch prediction algorithms, but it consumed too much time and memory to fit the constraints of the challenge. Moreover, a basic sliding window is not appropriate for unbalanced data: a stratified (click/no-click) reservoir or an adapted stream summary seemed to be more promising options. We chose to start from MOA: Massive Online Analysis (Bifet et al., 2010). MOA

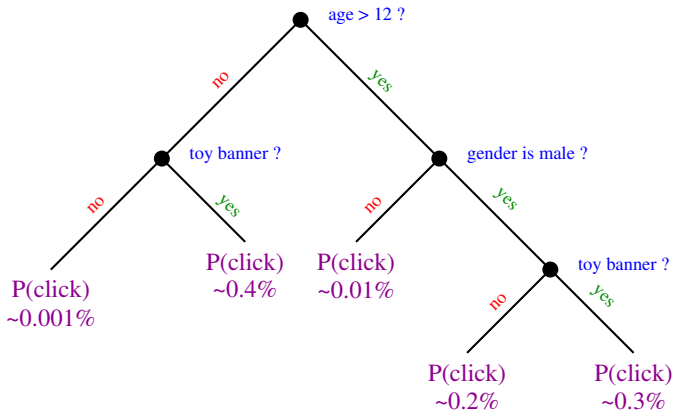


Figure 2: A toy illustration of multivariate probability estimation trees. Nodes are unfolded when there is enough data to estimate CTR accurately.

is developed by the Waikato University and is designed to work online and process data streams. We adapted their framework and classifiers to fit the challenge needs. Depending on the nature of the features: numerical or nominal, we built two kinds of summary.

Numerical features summary

In order to build probability estimation trees, we needed to keep the click/no-click counts by intervals for each numerical feature. This kind of summary is called a *quantile summary*. Greenwald and Khanna (2001) proposed an online merge-prune algorithm to maintain such a summary. Their algorithm is based on an array of tuples $< v_i, g_i, \Delta_i >$ where: v_i is a value seen in the stream, g_i is the number of values seen between v_{i-1} and v_i, and Δ_i is the maximum possible error on g_i. This algorithm has many interesting properties: it maintains online an approximate equal-frequency histogram with strong error guarantees (ϵ-approximate quantile summary for a given memory size), it uses a bounded memory space, and it is insensitive to data arrival order and distribution. We adapted the algorithm in order to store the counts per classes: the g_i counter was replaced by a counter per interval and per class: $g_{i,c}$.

Nominal features summary

A perfect nominal summary would keep click/no-click counts for all feature values. Unfortunately the number of nominal values is not supposed to be bounded. For instance some features can represent *client ids, cookies, city names, etc.* As we want a memory bounded summary we can only afford to focus on frequent values. The solution we used is based on hashing: nominal values are hashed and put into a fixed number of buckets, the click/no-click counts being stored for each bucket. We used a single hashing function, but in order to reduce errors several hashing functions may be combined as proposed in the count-min sketch algorithm (Cormode and Muthukrishnan, 2005).

2.2. Base predictors

The predictors we used were probability estimation trees, also known as PETs (Provost and Domingos, 2003). Their output is a click through rate estimate (CTR). We illustrate these

kinds of predictors in Figure 2. To estimate CTR in the leaves, we used an optimistically biased shrinkage-estimator:

$$\hat{P}(click) = \frac{n_{click} + m \cdot prior}{n_{total} + m}$$

where the prior was set to 0.0025 (a bit higher than global CTR) and the shrinkage parameter m was set to low values (less than 1000) in order to force a fast decreasing of the prior impact. The impact of these two parameters on performance seems to be slight. Depending on the feature types, numerical or nominal, we evaluated different strategies to estimate CTR.

Numerical features

We first started with decision stumps i.e. single level probability estimation trees. A decision stump is defined by two parameters: the features id i and the decision threshold θ. Figure 3 shows how the decision stump is built on a quantile summary.

$$\hat{P}(click \mid x_i) = \begin{cases} \hat{P}(click \mid x_i \leq \theta) & \text{if } x_i \leq \theta \\ \hat{P}(click \mid x_i > \theta) & \text{else} \end{cases}$$

The split criterion we used was the Gini impurity I_G (Entropy gives similar results). Given a split value θ on a feature i the Gini impurity is computed on two intervals (left: $x_i \leq \theta$ and right: $x_i > \theta$) using the previous estimates:

$$I_G(i) = \left(\hat{P}(click \mid x \leq \theta) - \hat{P}^2(click \mid x \leq \theta)\right) + \left(\hat{P}(click \mid x > \theta) - \hat{P}^2(click \mid x > \theta)\right)$$

The best split value is the value θ giving the lowest $I_G(i)$.

Then we explored deeper univariate trees by using a gradual unfolding. The tree is first a simple stump but new nodes are added when there is *'enough'* samples in the leaves for a correct CTR estimate (we fixed this amount to 1000 instances). To enhance the ranking quality, we used a bottom-up smoothing similar to the one proposed by Provost and Domingos (2003). This smoothing is defined recursively by:

$$\hat{P}(click \mid \text{node}) = \frac{n_{click} + m \cdot \hat{P}(click \mid \text{parent node})}{n_{total} + m}$$

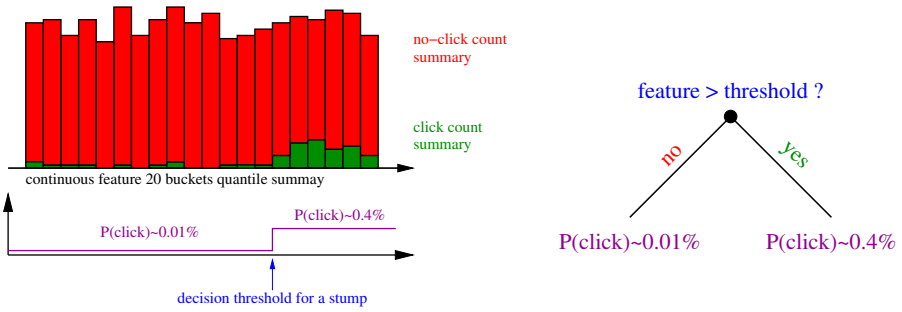


Figure 3: An illustration of how decision stumps are stacked on quantile summaries: the summary maintain a fine grained count of click/no-click for each bucket and the decision stump provides a rougher, but more reliable, CTR estimate.

Nominal features

We tried two different approaches. The first one transforms nominal values into numerical ones with a hash function and reuses the numerical-feature approach *(HStump)*. A combination of several hashing functions for the same feature avoids "unlucky orderings". The second option is to build a stump which contains one leaf for the most discriminant value and one leaf for all the other values *(1 vs. All)*.

Multivariate trees

Because it relies on a single feature, an univariate tree only requires a single summary: this simplifies the implementation of the system and reduces drastically the memory footprint. On the other hand, only multivariate models are able to catch interactions between features.

A multivariate tree needs to embed one summary per feature into each leaf. These summaries allow the algorithm to compute statistics to transform a leaf into a new decision node. The memory consumption (mainly taken by summaries) is proportional to the number of leaves. When a decision tree is built offline all the instances are available and the first node splits on the feature that maximizes the criterion. The next nodes are created in a similar way but only with the data in the subpartition induced by the parent node. In this challenge, instances arrive online and are collected by summaries. As a split (a node) is never reconsidered, we would like to guarantee that splits are made with a strong confidence. Domingos and Hulten proposed to use the Hoeffding bound to build trees online (Domingos and Hulten, 2000). These trees are called Hoeffding trees and are set up with a split confidence parameter δ.

2.3. Online combination of predictors

Without prior knowledge about features, we could not assume that all base predictors were accurate, especially the univariate ones based on poorly informative features. For this reason we looked for a combination system to favor the best predictors. This section describes the systems we experimented.

At each time step t each base predictor outputs a score $x_{i,t} \in [0,1]$. Let $\mathbf{x}_t = (x_{1,t}, \ldots, x_{N,t})$ be the predictors output vector at time t. Predictor's outputs are linearly combined in order to produce a score $\mathbf{w}_t \cdot \mathbf{x}_t \in [0,1]$ where $\mathbf{w_t} = (w_{1,t}, \ldots, w_{N,t})$ is a voting weight vector which verifies $|\mathbf{w_t}| = \sum_i |w_{i,t}| = 1$.

Baseline Our baseline was a simple averaging giving the same weight to all predictors.

Exponential weighting The main principle of this combination strategy is to reweight exponentially the predictors according to their past performance:

$$w_{i,t+1} \leftarrow w_{i,t} e^{\eta Reward_{t+1}(x_{i,t+1})} / Z_{t+1}$$

where Z_{t+1} is such that $|\mathbf{w_{t+1}}| = 1$. With a proper tuning of the parameter η, this combination strategy provides guarantees to converge to the best predictor performance (Cesa-Bianchi and Lugosi, 2006).

Online linear ranker Let $\mathcal{X}_0$, $\mathcal{X}_1$, be respectively the set of negative (no-click) and positive (click) instances. Set $\mathbf{w_t}$ to optimize *Area Under the roc Curve* (AUC) is equivalent

to minimize pairwise ranking loss:

$$RLoss(\mathbf{w}) = \sum_{(\mathbf{x_0},\mathbf{x_1})\in\mathcal{X}_0\times\mathcal{X}_1} [\mathbf{w}\cdot\mathbf{x_0} \geq \mathbf{w}\cdot\mathbf{x_1}]$$

Because the $\mathbf{x}_0 - \mathbf{x}_1$ are bounded, we can approximate the ranking loss with a linear function. This allows us to separate pairs:

$$RLoss(\mathbf{w}) \leq \sum_{(\mathbf{x_0},\mathbf{x_1})\in\mathcal{X}_0\times\mathcal{X}_1} 1+\mathbf{w}\cdot(\mathbf{x_0}-\mathbf{x_1}) = |\mathcal{X}_0|\cdot|\mathcal{X}_1|+\mathbf{w}\cdot\left(|\mathcal{X}_1|\cdot\sum_{\mathbf{x_0}\in\mathcal{X}_0}\mathbf{x_0} - |\mathcal{X}_0|\cdot\sum_{\mathbf{x_1}\in\mathcal{X}_1}\mathbf{x_1}\right)$$

The minimal linear loss is reached at $\lambda\mathbf{w}^*$ for any real λ with:

$$\mathbf{w}^* = \frac{1}{|\mathcal{X}_1|}\cdot\sum_{\mathbf{x_1}\in\mathcal{X}_1}\mathbf{x_1} - \frac{1}{|\mathcal{X}_0|}\cdot\sum_{\mathbf{x_0}\in\mathcal{X}_0}\mathbf{x_0}$$

Hence, for each t we simply maintain the centroids as in the Rocchio relevance feedback system (Rocchio, 1971):

- $\mathbf{w_{0,t}} = 1/|\mathcal{X}_0|\cdot\sum_{\mathbf{x}\in\mathcal{X}_0}\mathbf{x}$ (no click centroid)
- $\mathbf{w_{1,t}} = 1/|\mathcal{X}_1|\cdot\sum_{\mathbf{x}\in\mathcal{X}_1}\mathbf{x}$ (click centroid)
- $\mathbf{w_t} = (\mathbf{w_{1,t}} - \mathbf{w_{0,t}})/Z_t$ where Z_t such that $|\mathbf{w_t}| = 1$ ($w_{i,t}$ may be negative)

Online bagging Another kind of averaging that we experimented was online-bagging as proposed by (Oza, 2005). This bagging duplicates base classifiers several times and weights instances according to a Poisson law.

3. Experiments

Most of our experiments were done on the Adobe dataset but we did few experiments on our own advertising dataset (Orange web-mail). The characteristics of these two datasets are summarized in Table 1.

	Adobe (Challenge part 1)	**Orange web-mail**
Size	3M instances (i.e. 500K batches of 6)	6M instances (i.e. 1M batches of 6)
Features	99 continuous + 21 nominals *last nominal feature is "option id"*	33 nominals
Average CTR	~ 0.0024	~ 0.0005

Table 1: Datasets characteristics

submission	summary	base-predictors	combination	score
Random				1127 ± 41
LinRankP	GK-4/CMS-1	1R (proba. output)	LRank-120	~ 1600
LinRankB	GK-4/CMS-1	1R (0/1 output)	LRank-120	~ 1800
RHT	GK-100/CMS-1	200 trees (10 features)	OZA-200	~ 1900
Stumps1vAll	GK-100/CMS-1	Stump/1 vs. all	AVG-120	~ 2000
UTree	GK-100/CMS-1	UTree/HStump	AVG-120	~ 2030
Cnrs				$\sim$ **2040**
SimpleStumps	GK-100/CMS-1	Stump/HStump	AVG-120	~ 2050
Orange Best	GK-100/CMS-1	800 stumps + 200 RHT	OZA-1000	$\sim$ **2080**
Inria				$\sim$ **2200**
Perfect				6678

Table 2: Phase 1 results for the Adobe dataset
GK-100: Greenwald & Khanna (100 tuples), CMS-1: count-min sketch (1 hash),
1R: single rules $x_i \in \ ]a, b]$? 1 : 0*, HStump: hash then split like stump,*
RHT: random Hoeffding trees, UTree: univariate tree,
LRank-120: online linear ranker (120 predictors), AVG-120: simple averaging (120 pred.),
OZA-200: Oza bagging (instance weighting for 200 predictors)

3.1. Challenge results

Our experiments on Adobe dataset are detailed in Table 2. In this table, INRIA (1^{st} in this challenge), CNRS (3^{rd}), random and perfect predictor results are also presented.

We ran several random predictors with different seeds to estimate the standard reward deviation (± 41). The linear ranker stacked on probabilistic predictors (LinRankP) scored around 1600 while the one stacked on simple binary rules (LinRankB) scored around 1800. Similar results were obtained with exponential weighting. The combination strategies being difficult to tune, we backed up to a simple averaging and – surprisingly – obtained a significant improvement: around 2000 for decision stumps (Stumps1vAll), and 2030 for univariate trees (UTree). There seems to be a trade-off between the expressiveness of the base predictors and the power of the combination strategy.

Our best score using only univariate base classifiers was obtained with stumps for numerical features and HStump for the nominal ones ($\sim$ 2050 - SimpleStumps). Finally we have got our best result with a combination of 80% stumps and 20% Hoeffding trees with Oza bagging ($\sim$ 2080 - Orange Best). However, due to the multiplication of classifiers, the computing time of this model was much heavier than other submissions, especially the one with basic stumps. The 200 Hoeffding trees alone scored around 1900.

The evolution of the learning process is illustrated in Figure 4. This figure presents two sub-figures: the top one gives the evolution of the cumulated clicks and the bottom one the instantaneous CTR estimate with an exponential smoothing. Both curves are function of the batch number. The two random predictors' curves give an indication of the variability for a given strategy. The best entrant's CTR were around 2 times better than random.

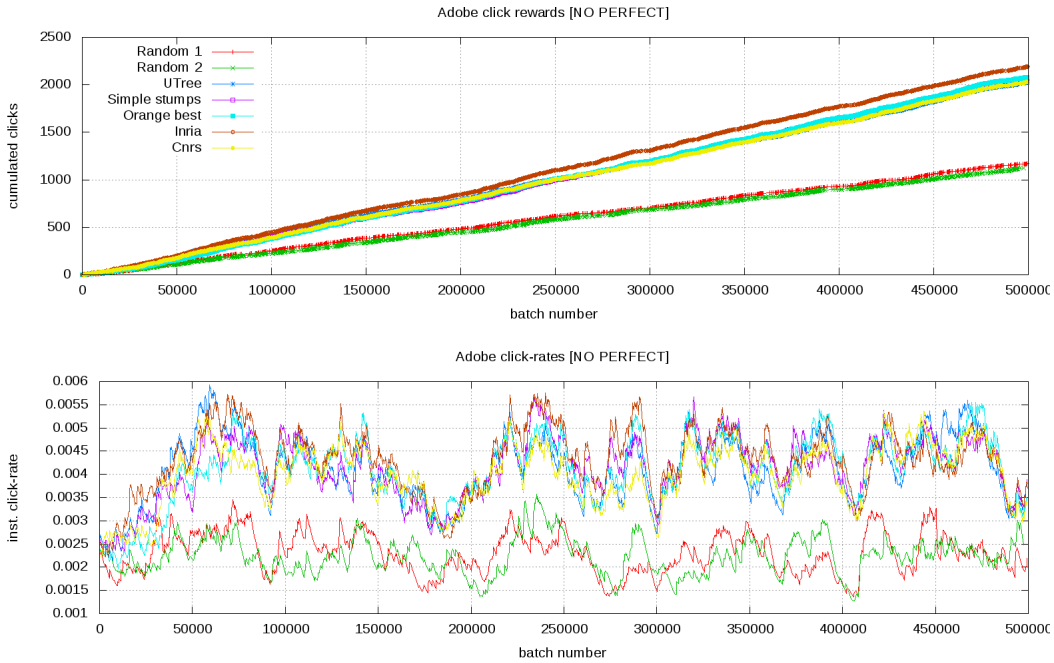


Figure 4: Phase 1 reward curves on Adobe dataset

3.2. Side Experiments

In order to deepen our understanding of the problem we did few side experiments on both Adobe and our own Orange web-mail dataset.

Offline analysis After the first phase of the challenge, a part of Adobe dataset was given to the entrants. Therefore we were able to refine our study of the predictors' behavior with an offline analysis on the whole dataset. This offline dataset analysis exhibited a maximum of 37 values for nominal features. For numerical features, we used the Khiops software (www.khiops.com) to perform an optimal discretization according to the MODL method (Boullé, 2006). This discretization resulted in one to three intervals depending on the feature. With 100 intervals for each of the 120 features our summary was small in memory (∼200kB) but fine enough to catch the data patterns.

To evaluate the quality of the online density estimation, we compared the discretization and click through rate estimates at different steps of the learning process to the MODL estimate on the whole dataset. Figure 5 illustrates this experiment. As expected, the online discretization converges to the MODL discretization and the average click through rate is higher.

Concept drift detection Another experiment was to check for *concept drifts* (i.e. abrupt changes of data distribution). Therefore we performed a comparison between a model which stops learning after 250,000 batches and the same model which keeps learning to the end. Figure 6 presents this experiment. There is no significant "drift" of the clicks distribution.

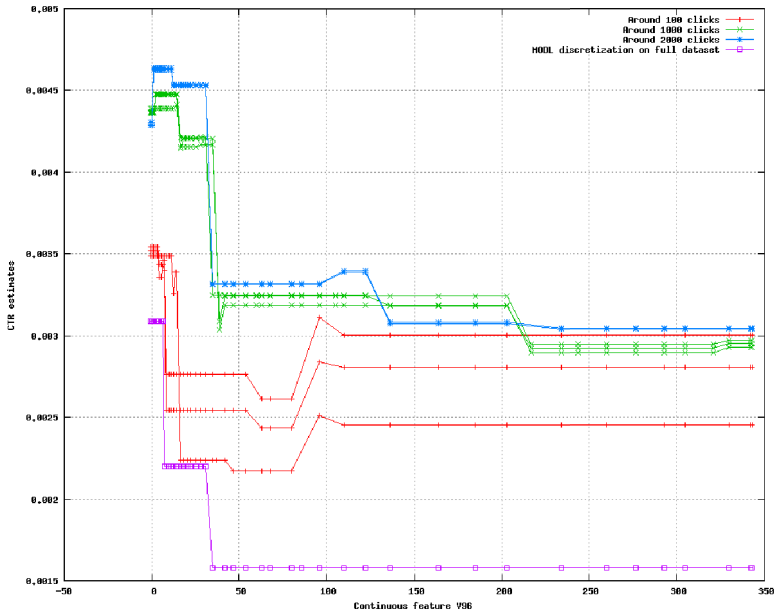


Figure 5: Discretization and CTR estimates at different steps for continuous feature 96 for different seeds.

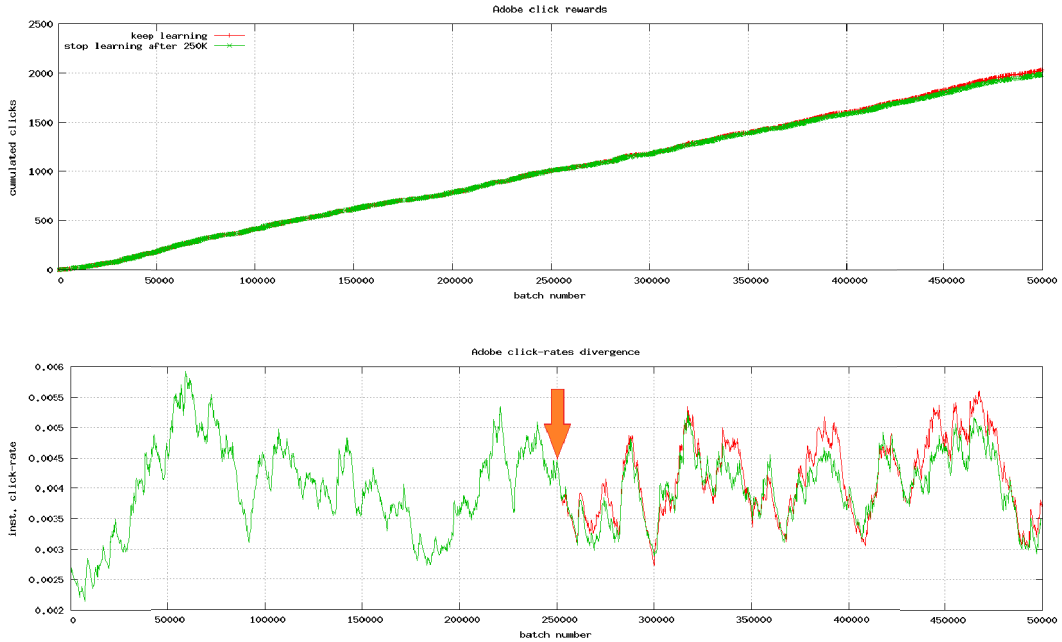


Figure 6: Deviation from a static model

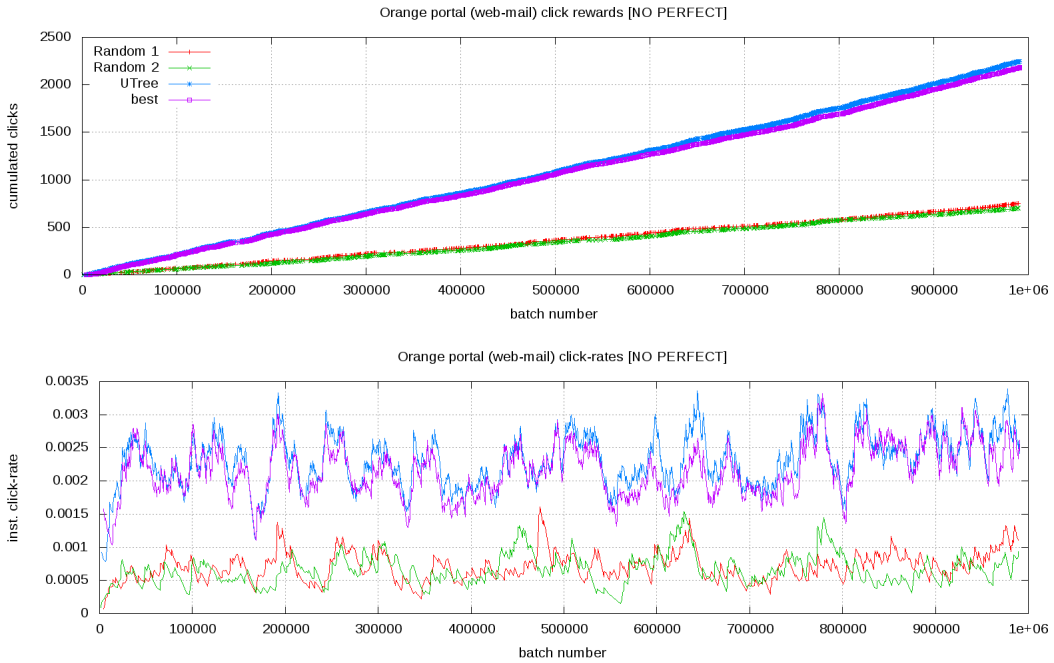


Figure 7: Challenge protocol applied on Orange dataset

Orange web-mail dataset The challenge protocol was also applied on our own – Orange web-mail – dataset. Figure 7 presents the results of two of our best predictors on this dataset. We obtain a click through rate around 3 times better than random policies.

3.3. Recommended algorithm

If we had to recommend a system to address a similar problem we would choose the following three layers because they performed well on both Adobe and Orange datasets with a small memory and time consumption:

1. Quantile summary for numerical features and Count-min Sketch for nominal ones;
2. Univariate PETs (one for each feature);
3. Averaging of the univariate predictors.

4. About Offline Evaluation Protocol

This challenge was one of the first to propose an evaluation of online content selection algorithms but the interaction between content selection and click feedback makes this evaluation difficult. The best solution is certainly to evaluate the different algorithms on similar subsets of real traffic but the cost of this method is prohibitive. A simulation from purely artificial model is not satisfying either: the most convenient solution is to evaluate offline from logged data.

The evaluation scheme proposed by this challenge was simple and made an efficient use of the logs: at each round a batch of 6 (profile, item, reward) instances was read from logs and submitted to the entrants' algorithms. The unrealistic aspect of this method was the possibility given for the algorithms to choose between different profiles: this is not the case for the real content selection problem where the profile is fixed. From our own experiments with Orange portal logs, some profile features are very discriminant. For instance the click through rate on Orange home page is five time higher than on other pages: the trained model only learns to select home page versus other pages. To reduce this side-effect, we only extracted logs from a single page (the web-mail).

For future challenges, if the log size is important and the number of items is limited, one may also use rejection sampling: at each round we take one log line and emulate all missing options features. If the selected option is the one of the log: we evaluate this instance based on known click/no-click results. If selected option is unknown we can reject (ignore) this instance. This method, well studied by (Langford et al., 2008; Li et al., 2010, 2011), rejects too many instances when the number of options is high. On the other hand, its great advantage is that it offers an unbiased estimate of the real algorithm performance when applied on uniform logs.

5. Conclusion

This challenge was difficult and interesting for both technical and scientific reasons. The need to embed the code into the challenge framework with limited resources required some technical investment from the entrants. On the other hand, the scarcity of clicks, the high level of noise, and the partial feedback which characterize the content selection problem required to explore innovative approaches.

The key ingredient of all our submissions was the elaboration of stream summaries: the Greenwald and Khanna stream summary for numerical features and the count-min sketch for nominal ones. From these summaries, we derived different kinds of click-predictors. Our best model was an online flavor of the *random forest* (although the term *"random tundra"* may appear more appropriate if we take into account the average tree size). A simpler model based only on decision stumps also proved to be remarkably efficient with lighter resource consumption. Some experiments on our own Orange dataset confirmed this phenomenon: the need for exploration, the high level of noise, and the click scarcity sharpen the need for very stable – hence simple, or strongly regularized – ranking models. We also observed a clear trade-off between the expressiveness of the base predictors and the power of the combination strategies.

Acknowledgments

We would like to thank the organizers for these very interesting challenge and workshop. We are already waiting for the next "Exploration/Exploitation" opus. We would also like to thank Sylvie Tricot for providing us the Orange advertising dataset and, Carine Hue and Vincent Lemaire for their comments. This work was partially supported by the FUI/FEDER Project DIFAC.

References

A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. J. Mach. Learn. Res., 99:1601–1604, August 2010. ISSN 1532-4435.

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

N. Cesa-Bianchi and G. Lugosi. Prediction, Learning, and Games. Cambridge University Press, 2006.

G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1):58–75, April 2005. ISSN 01966774.

C. Cortes and M. Mohri. AUC optimization vs. error rate minimization. In Advances in Neural Information Processing Systems. MIT Press, 2004.

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

M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. ACM SIGMOD Record, 30(2):58–66, June 2001. ISSN 01635808.

J. Langford, A. Strehl, and J. Wortman. Exploration scavenging. In Proceedings of the 25th international conference on Machine learning, ICML '08, pages 528–535, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

L. Li, W. Chu, J. Langford, and R.E. Schapire. A contextual-bandit approach to personalized news article recommendation. In Proceedings of the 19th international conference on World wide web, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8.

L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In Proceedings of the fourth ACM international conference on Web search and data mining, WSDM '11, pages 297–306, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0493-1.

N.C. Oza. Online Bagging and Boosting. 2005 IEEE International Conference on Systems, Man and Cybernetics, pages 2340–2345, 2005.

F. Provost and P. Domingos. Tree induction for probability-based ranking. Machine Learning, 52(3):199–215, 2003.

J. Rocchio. Relevance Feedback in Information Retrieval. In G Salton, editor, SMART Retrieval System Experiments in Automatic Document Processing, chapter 14, pages 313–323. Prentice Hall, 1971.

Bibliographie

[Aha97] David W Aha, editor. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[AMM+97] S Amari, N Murata, K R Muller, M Finke, and H H Yang. Asymptotic statistical theory of overtraining and cross-validation. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 8(5) :985–96, January 1997.

[BB05] A Bordes and Léon Bottou. The Huller : a simple and efficient online SVM. *Proceedings of the 16th European Conference on Machine Learning (ECML2005)*, 2005.

[BB11] A Bondu and Marc Boullé. A supervised approach for change detection in data streams. *International Joint Conference on Neural Networks (IJCNN)*, 2011.

[BEWB05] A Bordes, S Ertekin, J Weston, and Léon Bottou. Fast kernel classiffiers with online and active learning. *Journal of Machine Learning Research*, 6 :1579–1619, 2005.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees*. Chapman and Hall/CRC, 1984.

[BG07] Albert Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, pages 443–448, 2007.

[BGDF+06] Manuel Baena-García, José Del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early Drift Detection Method. *Fourth International Workshop on Knowledge Discovery from Data Streams*, 6 :77–86, 2006.

[BGV92] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM New York, USA, 1992.

[BH07] J Beringer and E Hüllermeier. Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6) :627–650, 2007.

[BHP+09] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, pages 139–147, 2009.

[BHTT10] Y Ben-Haim and E Tom-Tov. A streaming parallel decision tree algorithm. *Journal of Machine Learning*, 11 :849–872, 2010.

[Bis95] Christopher M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, USA, 1995.

[BK09] Albert Bifet and Richard Kirkby. DATA STREAM MINING A Practical Approach. *Journal of empirical finance*, 8(3) :325–342, 2009.

[BM98] C L Blake and C J Merz. UCI Repository of machine learning databases, 1998.

[BM02] Henry Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2) :153–172, 2002.

[BMS10] João Bártolo Gomes, Ernestina Menasalvas, and Pedro Sousa. Tracking recurrent concepts using context. In *Proceedings of the 7th international conference on Rough sets and current trends in computing*, pages 168–177. Springer, Springer-Verlag, 2010.

[Bou05] Marc Boullé. A grouping method for categorical attributes having very large number of values. *Machine Learning and Data Mining in Pattern Recognition*, pages 228–242, 2005.

[Bou06a] Marc Boullé. MODL : A Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1) :131–165, May 2006.

[Bou06b] Marc Boullé. Regularization and Averaging of the Selective Naive Bayes classifier. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1680–1688, 2006.

[Bou07a] Marc Boullé. *Recherche d'une représentation des données efficace pour la fouille des grandes bases de données*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, 2007.

[Bou07b] Marc Boullé. Une méthode optimale d'évaluation bivariée pour la classification supervisée. In *Extraction et gestion des connaissances (EGC'2007)*, pages 461–472, 2007.

[Bou09] Marc Boullé. Optimum simultaneous discretization with data grid models in supervised classification : a Bayesian model selection approach. *Advances in Data Analysis and Classification*, 2009.

[Bre96] L. Breiman. Bagging predictors. *Machine learning*, 24(2) :123–140, 1996.

[BS84] B G Buchanan and E H Shortliffe. *Rule-Based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984.

[CH08] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2) :1530–1541, January 2008.

[Cha04] M Charikar. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1) :3–15, January 2004.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary : the count-min sketch and its applications. *Journal of Algorithms*, 55(1) :58–75, April 2005.

[CM10] Antoine Cornuéjols and Laurent Miclet. *Apprentissage artificiel*. Eyrolles, 2010.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3) :273–297, September 1995.

[DB88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the seventh national conference on artificial intelligence*, pages 49–54, 1988.

[dCARJGMB06] J. del Campo-Avila, G Ramos-Jiménez, J. Gama, and R. Morales-Bueno. Improving Prediction Accuracy of an Incremental Algorithm Driven by Error Margins. *Knowledge Discovery from Data Streams*, pages 305–318, 2006.

[DG01] C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. In *ICDM*, pages 589–592, 2001.

[DH00] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM New York, USA, 2000.

[DH01] P. Domingos and G. Hulten. Catching up with the data : Research issues in mining data streams. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[DKS95] J. Dougherty, Ron Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann Publishers Inc., 1995.

[DKS05] Jian-xiong Dong, Adam Krzyzak, and Ching Y Suen. Fast SVM training algorithm with decomposition on very large data sets. *IEEE transactions on pattern analysis and machine intelligence*, 27(4) :603–618, April 2005.

[DNP09] T.N. Do, V.H. Nguyen, and F. Poulet. GPU-based parallel SVM algorithm. *Jisuanji Kexue yu Tansuo*, 3(4) :368–377, 2009.

[DP97] P. Domingos and Michael Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, 130 :103–130, 1997.

[DR09] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2 :311–327, December 2009.

[Faw04] Tom Fawcett. ROC graphs : Notes and practical considerations for researchers. *Machine Learning*, 31 :1–38, 2004.

[FI93] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the International Joint Conference on Uncertainty in AI*, pages 1022–1027, 1993.

[FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[FTARR05a] F. Ferrer-Troyano, J. S Aguilar-Ruiz, and J. C Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 ACM symposium on Applied computing*, page 572. ACM, 2005.

[FTARR05b] Francisco Ferrer-Troyano, Jesus S. Aguilar-Ruiz, and Jose C. Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the 2005 ACM symposium on Applied computing - SAC '05*, pages 568–572, New York, USA, 2005. ACM Press.

[FTARR06] F. Ferrer-Troyano, J.S. Aguilar-Ruiz, and J.C. Riquelme. Data streams classification by incremental rule learning with parameterized generalization. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 657–661. ACM, 2006.

[Gam10] J. Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC Press, 2010.

[GB11] Romain Guigourès and Marc Boullé. Optimisation directe des poids de modèles dans un prédicteur Bayésien naif moyenné. In *Extraction et gestion des connaissances EGC'2011*, pages 77–82, 2011.

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2) :58–66, June 2001.

[GK09] J. Gama and Petr Kosina. Tracking Recurring Concepts with Meta-learners. *Progress in Artificial Intelligence*, pages 423–434, 2009.

[GLB+09] I Guyon, V Lemaire, Marc Boullé, G Dror, and D Vogel. Analysis of the KDD Cup 2009 : Fast Scoring on a Large Orange Customer Database. *JMLR : Workshop and Conference Proceedings*, 7 :1–22, 2009.

[GMCR04] J. Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. *Advances in Artificial Intelligence - SBIA 2004*, pages 286–295, 2004.

[GMP02] PB Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database*, 27(3) :261–298, September 2002.

[GMR05] J. Gama, P. Medas, and P. Rodrigues. Learning decision trees from dynamic data streams. *Journal of Universal Computer Science*, 11(8) :1353–1366, 2005.

[GP06] J. Gama and Carlos Pinto. Discretization from data streams : applications to histograms and data mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667, 2006.

[GR05] A Globersonn and S Roweis. Metric Learning by Collapsing Classes. In *Neural Information Processing Systems(NIPS)*, 2005.

[GRG00] Johannes Gehrke, R. Ramakrishnan, and V. Ganti. RainForest - a framework for fast decision tree construction of large datasets. *Data Mining and Knowledge Discovery*, 4(2) :127–162, 2000.

[GRM03] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528. ACM New York, NY, USA, 2003.

[GRSR09] J. Gama, P. P Rodrigues, R. Sebastiao, and P.P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM New York, NY, USA, 2009.

[HD] Geoff Hulten and Pedro Domingos. VFML - A toolkit for mining high-speed time-changing data streams. In *http ://www.cs.washington.edu/dm/vfml/main.html*.

[HMR99] JA Hoeting, David Madigan, and AE Raftery. Bayesian model averaging : a tutorial. *Statistical science*, 14(4) :382–417, 1999.

[Hoe63] W Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 1963.

[HSD01] G. Hulten, Laurie Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM New York, USA, 2001.

[HSST06] John Hershberger, Nisheeth Shrivastava, Subhash Suri, and Csaba D. Toth. Adaptive Spatial Partitioning for Multidimensional Data Streams. *Algorithmica*, 46(1) :97–117, July 2006.

[HY01] David J Hand and Keming Yu. Idiot's Bayes ?Not So Stupid After All ? *International Statistical Review*, 69(3) :385–398, December 2001.

[JL95] GH John and Pat Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.

[Kan06] Gopal K Kanji. *100 Statistical Tests*, volume 129. Sage, 2006.

[Kas80] G V Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, 29(2) :119–127, 1980.

[Ker92] R. Kerber. Chimerge : Discretization of numeric attributes. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 123–128. AAAI Press, 1992.

[KHA99] Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '99*, volume 32, pages 367–371, New York, New York, USA, 1999. ACM Press.

[Kir08] Richard Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, 2008.

[KK97] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *ICML '97 : Proceedings of the Fourteenth International Conference on Machine Learning*, number 1996, pages 161–169. Morgan Kaufmann Publishers Inc., 1997.

[KM03] J.Z. Kolter and M.A. Maloof. Dynamic weighted majority : A new ensemble method for tracking concept drift. In *Proceedings of the Third International IEEE Conference on Data Mining*, pages 123–130, 2003.

[Koh96] Ron Kohavi. 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, 1996.

[KP08] Ludmila I Kuncheva and Catrin O Plumpton. Adaptive Learning Rate for Online Linear Discriminant Classifiers. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 510–519. Springer-Verlag, 2008.

[KR98] Ralf Klinkenberg and Ingrid Renz. Adaptive Information Filtering : Learning Drifting Concepts. *Artificial Intelligence*, 1998.

[KR03] I Kononenko and M Robnik. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning Journal*, 53 :23–69, 2003.

[KS96] Daphne Koller and Mehran Sahami. Toward Optimal Feature Selection. *International Conference on Machine Learning*, 1996(May) :284–292, 1996.

[Kun11] Ludmila I Kuncheva. Change Detection in Streaming Multivariate Data Using Likelihood Detectors. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–7, 2011.

[LBOM98] Y Lecun, L Bottou, G B Orr, and K R Müller. Efficient BackProp. In G Orr and K Müller, editors, *Lecture Notes in Computer Science*, volume 1524 of *Lecture Notes in Computer Science*, pages 5–50. Springer Verlag, 1998.

[LCB06] Gaélle Loosli, Stéphane Canu, and Léon Bottou. SVM et apprentissage des très grandes bases de données. *CAp Conférence d'apprentissage*, 2006.

[LIT92] Pat Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Proceedings of the National Conference on Artificial Intelligence*, number 415, pages 223–228, 1992.

[LS94] Pat Langley and S Sage. Induction of Selective Bayesian Classifiers. In R Lopez De Mantras Poole and D, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 399–406. Morgan Kaufmann, 1994.

[LTP07] Stephane Lallich, Olivier Teytaud, and Elie Prudhomme. Association Rule Interestingness : Measure and Statistical Validation. In Fabrice Guillet and Howard Hamilton, editors, *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 251–275. Springer Berlin / Heidelberg, 2007.

[LVB04] M. M Lazarescu, S. Venkatesh, and H. H Bui. Using multiple windows to track concept drift. *Intelligent Data Analysis*, 8(1) :29–59, 2004.

[LYW06] Jingli Lu, Ying Yang, and G Webb. Incremental Discretization for Naive-Bayes classifier. *Advanced Data Mining and Applications*, pages 223 – 238, 2006.

[LZ05] Y.N. Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. *Lecture notes in computer science*, 3721 :108, 2005.

[MAR96] M. Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ : A fast scalable classifier for data mining. *Lecture Notes in Computer Science*, 1057 :18–34, 1996.

[Mar03] S Marsland. Novelty Detection in Learning Systems. *Neural computing surveys*, 3 :157–195, 2003.

[MM00] M.A. Maloof and R.S. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41(1) :27–52, 2000.

[MMHL86] R. S. Michalski, I Mozetic, J Hong, and N Lavrac. The Multi-Purpose incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, 1986.

[MP78] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 253–258. IEEE, October 1978.

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. *Approximate medians and other quantiles in one pass and with limited memory.* ACM Press, New York, USA, 1998.

[MSMO02] F Moreno-Seco, L Mico, and J Oncina. Extending LAESA Fast Nearest Neighbour Algorithm to Find the k Nearest Neighbours. In *SSPR & SPR*, pages 718–724, 2002.

[NK07] A. Narasimhamurthy and Ludmila I Kuncheva. A framework for generating data to simulate changing environments. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference : artificial intelligence and applications*, volume 549, pages 384–389. ACTA Press, 2007.

[OJ97] T. Oates and D. Jensen. The Effects of Training Set Size on Decision Tree Complexity. In *ICML '97 : Proceedings of the Fourteenth International Conference on Machine Learning*, pages 254–262, 1997.

[OJP12] Nicol Olivier, Mary Jérémie, and Preux Philippe. ICML Exploration & Exploitation Challenge : Keep it simple ! *Journal of Machine Learning Research - Proceedings Track*, 26 :62–85, 2012.

[Pag54] ES Page. Continuous inspection schemes. *Biometrika*, 41(1-2) :100, 1954.

[PD03] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3) :199–215, 2003.

[PHK08] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. Handling numeric attributes in Hoeffding trees. *Advances in Knowledge Discovery and Data Mining*, pages 296–307, 2008.

[PK99] Foster Provost and Venkateswarlu Kolluri. A Survey of Methods for Scaling Up Inductive Algorithms. *Data Min. Knowl. Discov.*, 3(2) :131—-169, June 1999.

[Pre97] Lutz Prechelt. Early Stopping - but when ? In *Neural Networks : Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pages 55–69. Springer-Verlag, 1997.

[QCSSL09] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[Qui93] J. R. Quinlan. *C4.5 : programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[RB93] Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning : The RPROP Algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[RJdCAMB06] G. Ramos-Jimenez, J. del Campo-Avila, and R. Morales-Bueno. Incremental algorithm driven by error margins. *Lecture Notes in Computer Science*, 4265 :358–362, 2006.

[SAK+09] T. Seidl, I. Assent, P. Kranen, R. Krieger, and J. Herrmann. Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology*, pages 311–322. ACM, 2009.

[Sal97] Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1) :133–155, 1997.

[SAM96] J. Shafer, R. Agrawal, and M. Mehta. SPRINT : A scalable parallel classifier for data mining. In *Proceedings of the International Conference on Very Large Data Bases*, pages 544–555, 1996.

[SCZ05] Michael Stonebraker, Ugur Cetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4) :42–47, December 2005.

[SF86] J.C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501, 1986.

[SG86] J.C. Schlimmer and R.H. Granger. Incremental learning from noisy data. *Machine learning*, 1(3) :317–354, 1986.

[SK01] W.N. Street and Y.S. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM New York, NY, USA, 2001.

[SK07] M. Scholz and R. Klinkenberg. Boosting classifiers for drifting concepts. *Intelligent Data Analysis*, 11(1) :3–28, 2007.

[SL11a] Christophe Salperwyck and Vincent Lemaire. Classification incrémentale supervisée : un panel introductif. *Revue des Nouvelles Technologies de l'Information, Numéro spécial sur l'apprentissage et la fouille de données*, A5 :121–148, 2011.

[SL11b] Christophe Salperwyck and Vincent Lemaire. Incremental discretization for supervised learning. *CLADAG : CLAssification and Data Analysis Group - 8th International Meeting of the Italian Statistical Society*, 2011.

[SL11c] Christophe Salperwyck and Vincent Lemaire. Learning with few examples : An empirical study on leading classifiers. In *The 2011 International Joint Conference on Neural Networks*, pages 1010–1019. IEEE, July 2011.

[SL12a] Christophe Salperwyck and Vincent Lemaire. Arbres en ligne basés sur des statistiques d'ordre. In *Atelier CIDN de la conférence EGC 2012*, Bordeaux, 2012.

[SL12b] Christophe Salperwyck and Vincent Lemaire. Incremental Decision Tree based on order statistics. In *Workshop AIL : Active and Incremental Learning (ECAI 2012 : 20th European Conference on Artificial Intelligence)*, Montpellier, 2012.

[SL13] Christophe Salperwyck and Vincent Lemaire. A two layers incremental discretization based on order statistics. *Advances in Data Analysis and Classification*, 2013.

[SLS99] N.A. Syed, H. Liu, and K.K. Sung. 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*, pages 317–321. ACM New York, NY, USA, 1999.

[SSV07] J Sankaranarayanan, H Samet, and A Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 2007.

[SU12] Christophe Salperwyck and Tanguy Urvoy. Stumping along a Summary for Exploration & Exploitation Challenge 2011. *Journal of Machine Learning Research - Proceedings Track*, 26 :86–97, 2012.

[TKC06] I.W. Tsang, J.T. Kwok, and P.M. Cheung. Core vector machines : Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(1) :363, 2006.

[UBC97] P.E. Utgoff, N.C. Berkman, and J.A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1) :5–44, 1997.

[UPB10] Nicolas Usunier, Université Paris, and Léon Bottou. Guarantees for Approximate Incremental SVMs. *Processing*, 9 :884–891, 2010.

[Utg89] P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2) :161–186, 1989.

[VBH09] Nicolas Voisine, Marc Boullé, and Carine Hue. Un critère d'évaluation Bayésienne pour la construction d'arbres de décision. *Extraction et gestion des connaissances (EGC'2009)*, pages 259–264, 2009.

[Vit85] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1) :37–57, March 1985.

[VLC00] Hooman Vassef, Chung-Sheng Li, and Vittorio Castelli. Combining fast search and learning for fast similarity search. In *Proceedings of SPIE - The International Society for Optical Engineering*, volume 3972, pages 32–42, 2000.

[Ž10] I. Žliobaite. Learning under Concept Drift : an Overview. Technical report, https ://sites.google.com/site/zliobaite/Zliobaite_CDoverview.pdf, October 2010.

[WF05] I. H. Witten and E. Frank. *Data mining : practical machine learning tools and techniques*. Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, second edition, 2005.

[WFYH03] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, pages 226–235, New York, New York, USA, 2003. ACM Press.

[WK92] G. Widmer and M. Kubat. Learning flexible concepts from streams of examples : FLORA2. In *Proceedings of the 10th European conference on Artificial intelligence*, number section 5, pages 463–467. John Wiley & Sons, Inc., 1992.

[WK96] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1) :69–101, 1996.

[WS09] K Weinberger and L Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *The Journal of Machine Learning Research (JMLR)*, 10 :207–244, 2009.

[Yam08] Toshihiko Yamakami. A Stream-mining Oriented User Identification Algorithm Based on a Day Scale Click Regularity Assumption in Mobile Clickstreams. *Journal of Information Processing*, 16(July) :93–99, 2008.

[YW08] Ying Yang and Geoffrey Webb. Discretization for naive-Bayes learning : managing discretization bias and variance. *Machine Learning*, 74(1) :39–74, September 2008.

[ZR96] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1) :181–213, 1996.

[ZR00] Djamel A. Zighed and R. Rakotomalala. *Graphes d'induction*. HERMES Science Publications, 2000.

[ZRR98] Djamel A. Zighed, S. Rabaséda, and R. Rakotomalala. FUSINTER : a method for discretization of continuous attributes. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(3) :307–326, 1998.

[ZW07] Qi Zhang and Wei Wang. A fast algorithm for approximate quantiles in high speed data streams. *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, pages 29–29, July 2007.

Apprentissage Incrémental en-Ligne sur Flux de Données

Résumé

L'apprentissage statistique propose un vaste ensemble de techniques capables de construire des modèles prédictifs à partir d'observations passées. Ces techniques ont montré leurs capacités à traiter des volumétries importantes de données sur des problèmes réels. Cependant, de nouvelles applications génèrent de plus en plus de données qui sont seulement visibles sous la forme d'un flux et doivent être traitées séquentiellement. Parmi ces applications 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 l'apprentissage avec les nouvelles contraintes imposées par les flux de données. Nous proposons d'aborder ce problème en proposant de nouvelles techniques de résumé de flux de données dans le cadre de l'apprentissage supervisé. Notre méthode est constituée de deux niveaux. Le premier niveau utilise des techniques incrémentales de résumé en-ligne pour les flux qui prennent en compte les ressources mémoire et processeur et possèdent des garanties en termes d'erreur. Le second niveau utilise les résumés de faible taille, issus du premier niveau, pour construire le résumé final à l'aide d'une méthode supervisée performante hors-ligne. Ces résumés constituent un prétraitement qui nous permet de proposer de nouvelles versions du classifieur bayésien naïf et des arbres de décision fonctionnant en-ligne sur flux de données. Les flux de données peuvent ne pas être stationnaires mais comporter des changements de concept. Nous proposons aussi une nouvelle technique pour détecter ces changements et mettre à jour nos classifieurs.

Mots clés : Apprentissage incrémental, Flux de données, Résumés, Changements de concept

Incremental Online Learning on Data Streams

Summary

Statistical learning provides numerous algorithms to build predictive models on past observations. These techniques proved their ability to deal with large scale realistic problems. However, new domains generate more and more data which are only visible once and need to be processed sequentially. These volatile data, known as data streams, come from telecommunication network management, social network, web mining. The challenge is to build new algorithms able to learn under these constraints. We proposed to build new summaries for supervised classification. Our summaries are based on two levels. The first level is an online incremental summary which uses low processing and address the precision/memory tradeoff. The second level uses the first layer summary to build the final summary with an efficient offline method. Building these summaries is a pre-processing stage to develop new classifiers for data streams. We propose new versions for the naive-Bayes and decision trees classifiers using our summaries. As data streams might contain concept drifts, we also propose a new technique to detect these drifts and update classifiers accordingly.

Key words : Incremental learning, Data streams, Summaries, Concept drift