Apprentissage incrémental et décrémental

Manuel Bouillon

INSA de Rennes

Master 2 Recherche en Informatique

Mémoire

Apprentissage incrémental et décrémental

Classification avec un système d'inférence floue évolutif appliquée à la reconnaissance de gestes manuscrits.

IRISA UMR 6074

Équipe IntuiDoc

Étudiant
Manuel Bouillon

Encadrants
Eric Anquetil
Abdullah Almaksour

5 juin 2012

Table des matières

Table des figures

Remerciements

Tout d'abord, je tiens à remercier Eric Anquetil de m'avoir accepté en stage dans l'équipe IntuiDoc, et de m'avoir proposé de poursuivre en thèse par la suite.

Ensuite, je remercie Abdullah Almaksour et encore Eric Anquetil pour leurs conseils et leur soutien tout au long de mon stage.

Enfin, je remercie mes collègues de bureau, Achraf et Ney, pour leur bonne humeur, et toute l'équipe IntuiDoc pour l'ambiance conviviale des pauses café.

Résumé

Ce mémoire de master présente la question de l'apprentissage incrémental et décrémental d'un système de classification évolutif. Il se place dans le cadre de la reconnaissance de raccourcis graphiques sur une interface homme-machine tactile ou avec stylet. Un panorama des systèmes incrémentaux évolutifs existants est présenté, et une attention particulière est portée aux systèmes d'inférence floue. L'apprentissage décrémental, qui constitue l'objet de ce stage, est ensuite présenté : l'introduction d'oubli dans l'optimisation des conclusions des règles d'inférence est explorée. Deux stratégies sont proposées – l'apprentissage décrémental sur une fenêtre et l'apprentissage décrémental progressif – qui sont ensuite évaluées suivant plusieurs protocoles expérimentaux. Il est en particulier montré que l'oubli est indispensable pour conserver un système dynamique et réactif à long terme ; ce qui rend l'apprentissage décrémental nécessaire pour l'utilisation à vie d'un système d'apprentissage incrémental évolutif.

Mots-clefs : classification, apprentissage incrémental, apprentissage décrémental, système d'inférence floue, moindres carrés récursifs, oubli.

Abstract

This master thesis tackles the problem of incremental and decremental learning of an evolving classification system. This study takes place in the context of graphical shortcuts recognition on touch sensitive screen. An overview of existing incremental evolving systems is presented, and special attention is paid to fuzzy inference systems. Decremental learning is then introduce, and the use of forgetting in optimizing inference rules conclusions is explored. Two strategies are proposed : windowed decremental learning and progressive decremental learning which are evaluated on several experimention protocols. In particular, it is shown that forgetting is essential to keep a system dynamic and reactive in the long term. Decremental learning is thus necessary for the life-time use of an evolving incremental learning system.

Keywords : classification, incremental learning, decremental learning, fuzzy inference system, recursive least square, forgetting.

Chapitre 1

Introduction

Depuis quelques années, on observe une forte augmentation des interfaces homme-machine (IHM) tactiles. L'utilisateur n'utilise alors plus de souris ni de clavier, mais interagit directement sur l'écran avec un stylet ou même ses doigts. Avec l'arrivée de ces nouvelles interfaces, c'est toute l'ergonomie des applications qui doit être repensée. Toutes les commandes précédemment effectuées à l'aide de la souris et du clavier doivent être revues pour devenir facilement utilisable avec un stylet. En particulier, les anciens raccourcis claviers n'ont plus raison d'être, et laissent place à des raccourcis graphiques, que l'utilisateur peut directement dessiner avec les doigts ou un stylet sur l'écran.

Ces raccourcis graphiques peuvent être des lettres, ou d'autres symboles, que l'utilisateur peut rapidement dessiner pour effectuer une action sans avoir besoin de parcourir de menus. La machine doit être capable de reconnaitre ces différents symboles pour interpréter les ordres de l'utilisateur. Comme les raccourcis clavier, les raccourcis graphiques peuvent être amenés à changer. Que ce soit parce que l'utilisateur a besoin de commandes supplémentaires ou qu'il n'arrive pas à retenir les raccourcis existants, il est indispensable que l'utilisateur puisse ajouter ou enlever des raccourcis. De même, la manière de dessiner ces symboles va probablement évoluer et il faut que le système de reconnaissance soit capable de s'y adapter.

Pour cela, il est nécessaire de disposer d'un système de reconnaissance de forme pour identifier les symboles dessinés. La reconnaissance de forme est un problème de classification, puisqu'il faut prédire la classe d'appartenance d'exemples inconnus du système. On y répond le plus souvent par de l'apprentissage supervisé, en entrainant le système sur un ensemble d'individus pour lesquels on connait la catégorie d'appartenance.

Cependant, cet ensemble d'apprentissage n'est pas toujours disponible avant le début de l'utilisation, il faut alors pouvoir entrainer en-ligne (à la volée) le système au fur et à mesure de son utilisation. Il est nécessaire que l'algorithme d'apprentissage soit incrémental, pour ne pas avoir besoin de reconstruire tout le système à l'arrivée de nouvelles données, et qu'il ne stocke pas toutes les données afin d'être utilisable à vie. En plus d'adapter ses paramètres, il faut également que le système soit capable de modifier sa structure pour permettre l'ajout et la suppression de classes.

La suite de ce mémoire se présente comme suit : dans le chapitre 2, nous commencerons par bien définir le problème d'apprentissage artificiel sur lequel nous travaillons. Puis, nous étudierons les différentes approches d'apprentissage incrémental existant dans la littérature, et en particulier

les systèmes d'inférence floue. On explorera ensuite l'apprentissage décrémental dans le chapitre 3 où nous étudierons deux approches : l'apprentissage décrémental sur une fenêtre et l'apprentissage décrémental progressif. Enfin, ces deux approches seront évaluées sur trois jeux de données différents et plusieurs scénarios expérimentaux dans le chapitre 4. Le chapitre 5 conclura ce mémoire et discutera des perspectives de ce travail.

Chapitre 2

État de l'Art

Dans ce chapitre, nous commencerons par définir le problème d'apprentissage artificiel auquel nous faisons face. Ensuite, nous présenterons un panorama des principaux systèmes d'apprentissage incrémental existants dans la littérature et en particulier les systèmes d'inférence floue.

2.1 Définitions

L'apprentissage artificiel est un vaste domaine sur lequel beaucoup de travaux ont été publiés. Cependant, le système recherché ici vis-à-vis de notre contexte applicatif doit posséder certaines caractéristiques bien précises.

Avec l'arrivée des interfaces homme-machine tactiles, l'ergonomie des applications est amenée à changer. Les raccourcis claviers cèdent place à des raccourcis graphiques pour effectuer les commandes les plus courantes. Il est souhaitable que l'utilisateur ait la possibilité de modifier ces raccourcis en fonction de ses attentes. Il faut donc disposer d'un système de reconnaissance de gestes manuscrits capable d'identifier ces raccourcis graphiques. Ce système doit alors être dynamique et flexible, il doit pouvoir s'adapter précisément à l'utilisateur final et permettre l'ajout ou la suppression de raccourcis.

Dans cette section, nous définirons précisément les différentes catégories de systèmes d'apprentissage qui existent, et en particulier les types de systèmes qui peuvent convenir à notre contexte applicatif. En effet, certaines définitions ne font pas l'unanimité dans la littérature, nous allons donc expliquer ici celles qui seront utilisées dans ce mémoire.

2.1.1 Apprentissage supervisé et non-supervisé

Le domaine de l'apprentissage artificiel peut être divisé en deux parties : l'apprentissage supervisé et l'apprentissage non-supervisé [Cornuéjols and Miclet, 2002].

Apprentissage supervisé En apprentissage supervisé, on cherche à prédire la catégorie ou la valeur d'individus inconnus à partir d'un ensemble de référence, pour lesquels on connaît la vraie classe ou valeur. Cette catégorie d'appartenance, ou valeur associée, étant renseignée par un expert, on parle alors d'apprentissage supervisé.

Apprentissage non-supervisé Au contraire, en apprentissage non-supervisé on ne connaît pas la classe ou la valeur des individus de notre ensemble d'entrainement. L'apprentissage non-supervisé consiste à chercher une structure ou des motifs dans les données. L'objectif est d'obtenir un partitionnement des données en ensembles d'individus ayant des caractéristiques assez similaires entre eux mais relativement différentes des autres individus.

Notre contexte applicatif correspond donc à de l'apprentissage supervisé puisque nous cherchons à prédire la classe d'appartenance de symboles inconnus du système, mais connu de l'utilisateur.

2.1.2 Classification et régression

Les problèmes d'apprentissage supervisé peuvent être divisés en deux catégories : les problèmes de classification et les problèmes de régression [Cornuéjols and Miclet, 2002].

Classification Un problème de classification est un problème où l'on cherche à prédire la catégorie, ou classe, des individus. En classification supervisée, on dispose d'un ensemble d'apprentissage composé d'individus pour lesquels on connaît d'une part les valeurs de leurs caractéristiques, et d'autre part, leur catégorie d'appartenance. On cherche ensuite à ranger les individus inconnus dans les différentes catégories à l'aide des valeurs de leurs caractéristiques.

Régression Un problème de régression est un problème où l'on cherche à prédire la valeur d'une variable inconnue en fonction d'autres variables descriptives. En régression, on cherche donc à modéliser la relation entre cette variable inconnue et les variables d'entrées.

Notre contexte applicatif correspond donc à un problème de classification puisque nous cherchons à prédire la classe d'appartenance de symboles inconnus.

2.1.3 Apprentissage hors-ligne et Apprentissage en-ligne

Un système de reconnaissance peut être utilisé soit de manière en-ligne soit de manière hors-ligne [Almaksour, 2011]. On parle bien ici de la manière dont est utilisé le système et non pas de la manière dont est réalisé l'apprentissage comme on le verra ensuite.

Apprentissage hors-ligne Le mode de fonctionnement hors-ligne sépare les phases d'apprentissage et d'utilisation. Dans un premier temps, le système est entrainé de manière statique ou incrémentale à partir d'un ensemble d'entrainement. Ensuite, dans un deuxième temps, le système préalablement appris est utilisé.

Apprentissage en-ligne Le mode de fonctionnement en-ligne ne sépare pas l'apprentissage et l'utilisation du système, les deux tâches sont effectuées de manière conjointe. Tout au long de son utilisation, le système continue d'apprendre dès qu'une nouvelle donnée est disponible afin d'améliorer ses performances.

L'avantage des systèmes en-ligne par rapport aux systèmes hors-ligne est qu'ils vont pouvoir s'ajuster très précisément à l'utilisateur final en continuant à apprendre tout au long de leur utilisation. L'ensemble d'apprentissage initial n'a plus besoin d'être aussi conséquent et diversifié puisque le système s'adaptera pendant son utilisation.

L'apprentissage d'un système fonctionnant en-ligne est le plus souvent réalisé incrémentalement, même s'il pourrait également être fait de manière statique en reconstruisant tout le système (en stockant toutes les données).

2.1.4 Apprentissage statique et apprentissage incrémental

L'entrainement d'un système d'apprentissage peut se faire de deux manières : de manière statique ou de manière incrémentale. Nous allons voir les principales différences entre ces deux approches [Almaksour, 2011].

Apprentissage statique En apprentissage statique (*batch learning*), le jeu de données d'entrainement est prédéfini et disponible lors de l'entrainement du système. L'algorithme d'apprentissage utilise l'ensemble du jeu de données pour minimiser le taux d'erreur.

Apprentissage incrémental En apprentissage incrémental, le jeu de données n'est pas forcément disponible dès le début de l'entrainement. Les données sont introduites au fur et à mesure et le système doit être capable d'apprendre à partir de chacun de ces exemples séparément. Le système doit donc être en mesure de se modifier et d'ajuster ses paramètres après l'observation de chaque exemple pour apprendre à partir de celui-ci ; mais néanmoins sans oublier la connaissance acquise à partir des exemples précédents.

Ce mode d'apprentissage est utilisé soit lorsque le jeu de données est trop grand pour être utilisé en une seule fois ; soit lorsque l'ensemble d'apprentissage n'est pas disponible dans son intégralité et que les données d'entrainement arrivent de manière incrémentale.

L'apprentissage incrémental peut facilement être réalisé en-ligne et permet d'utiliser le système pendant son apprentissage, ou formulé autrement, d'améliorer le système tout au long de son utilisation.

Nous souhaitons ici que notre système apprenne en-ligne et soit utilisable en temps réel, il faut donc que l'apprentissage soit incrémental.

2.1.5 Système adaptatif et système évolutif

Les systèmes d'apprentissage incrémental sont donc dynamiques car ils apprennent au fur et à mesure que les données arrivent, contrairement aux systèmes d'apprentissage classique, dits statiques. Néanmoins, ces systèmes incrémentaux peuvent encore être séparés en deux catégories : les systèmes adaptatifs et les systèmes évolutifs [Lughofer and Angelov, 2011].

Système adaptatif On appelle systèmes adaptatifs les systèmes d'apprentissage incrémental qui vont apprendre de manière incrémentale leurs paramètres mais dont la structure est fixe. Cet apprentissage des paramètres peut être considéré comme un algorithme « d'adaptation ». La structure de ces systèmes est initialisée au départ et reste ensuite inchangée pendant l'apprentissage.

Système évolutif Les systèmes évolutifs sont une sous-catégorie des systèmes incrémentaux capables de changer leur structure. Ils vont non seulement apprendre leurs paramètres de manière incrémentale, mais également modifier leur structure, comme par exemple lors de l'apparition d'une nouvelle classe. Il est souhaitable qu'un système évolutif n'ait pas de paramètres dépendants du problème qui seraient difficilement ajustables de manière incrémentale.

Un système évolutif est donc bien plus flexible qu'un système adaptatif puisqu'il permet, en cours d'utilisation, d'ajouter des classes supplémentaires lorsque cela s'avère nécessaire.

2.1.6 Mémoire des données et mémoire des concepts

Un système d'apprentissage à besoin de mémoriser l'information apprise. Pour cela il peut soit mémoriser les données, soit mémoriser les concepts extraits des données ; ou encore combiner les deux [Maloof and Michalski, 2004].

Mémoire des données Mémoriser toutes les données est une méthode simple qui permet de mettre à jour l'ensemble d'apprentissage facilement. Cette méthode est par exemple utilisée par les classifieurs de type plus proches voisins. Elle peut également être utilisée pour reconstruire tout un système à l'arrivée de nouvelles données.

Cependant, cette méthode possède plusieurs inconvénients. Le nombre de données à mémoriser augmente tout au long de l'utilisation du système et cette méthode va nécessiter beaucoup d'espace mémoire. De plus, si l'on reconstruit tout le système, son apprentissage va être de plus en plus long et complexe au fur et à mesure que la taille du jeu de données d'entrainement augmente.

Une approche dérivée limitant ces inconvénients consiste à sélectionner et maintenir un sous-ensemble des données d'apprentissage sur lequel reconstruire le système. Cette méthode est appelée mémoire partielle des données, à opposer à la mémoire complète des données.

Mémoire des concepts Une autre méthode consiste non pas à mémoriser les données, mais plutôt à mémoriser les concepts, c'est-à-dire la connaissance extraite de ces données. Cette méthode est par exemple utilisée par les classifieurs de type réseau connexionnistes. La connaissance, le modèle, doit alors être mis à jour à l'arrivée de nouveaux exemples de manière à prendre en compte ces nouvelles informations mais sans pour autant oublier celles issues des exemples précédents.

2.2 Contexte applicatif

Dans cette section, nous allons présenter le contexte applicatif de la reconnaissance de gestes manuscrits, et les caractéristiques du système que nous recherchons pour y répondre.

2.2.1 Reconnaissance de gestes manuscrits

Ces travaux se placent dans le cadre de la reconnaissance de gestes manuscrits qui est un problème d'apprentissage supervisé. La manière la plus courante d'y faire face est d'entrainer hors-ligne un système de classification, puis ensuite de l'utiliser de manière statique. Ce système peut être omni-scripteur – indépendant du scripteur – ou bien mono-scripteur – spécialisé sur un scripteur.

Les systèmes mono-scripteur permettent d'atteindre de bonnes performances de reconnaissance mais nécessitent pour cela que le scripteur ait saisi une base de données d'entrainement suffisamment large au préalable. Les systèmes omni-scripteurs sont entrainés sur une base composée par le plus de scripteurs possible. Ils ont ainsi l'avantage de pouvoir être utilisé par un nouveau scripteur sans que celui-ci ne doit au préalable saisir une base d'entrainement personnalisée. Par contre, les systèmes omni-scripteur ont des performances plus limitées puisqu'ils ne sont pas adaptés précisément à l'utilisateur final.

Il existe également dans la littérature des approches combinant les deux types de systèmes [LaViola Jr. and Zeleznik, 2007].

Notre approche est d'utiliser un système mono-scripteur initialisé pas l'utilisateur avec très peu de données, entre un et trois exemples par classe. On apprend ensuite notre système pendant son utilisation pour le spécialiser et obtenir ensuite de bonnes performances. Pour cela, on offre à l'utilisateur la possibilité de corriger notre système lorsqu'il commet une erreur, ou on valide implicitement la classe reconnue si l'utilisateur ne corrige pas le système, pour pouvoir continuer à apprendre pendant l'utilisation.

D'autres approches sont également envisageables : il serait par exemple possible d'utiliser de l'apprentissage actif ou de l'apprentissage semi-supervisé pour limiter le besoin d'étiquetage, de validation de la classe reconnue, par l'utilisateur.

2.2.2 Objectif recherché

Notre contexte applicatif correspond à une utilisation **en-ligne** d'un système de **classification supervisée**. Si l'on veut que notre système soit utilisable sur un ordinateur standard, voir une tablette ou d'autres appareils mobiles, il est indispensable que ce système ne soit pas entièrement reconstruit à l'arrivée de nouvelles données. Ce système doit donc être **incrémental**, c'est à dire satisfaire les critères suivants :

- pouvoir apprendre de nouvelles connaissances à partir de nouvelles données ;
- utiliser une mémoire des concepts (ne pas nécessiter d'accès aux anciennes données) ;
- préserver la connaissance préalablement acquise à partir des anciennes données lors de l'apprentissage de nouvelles connaissances à partir de nouvelles données.

Ensuite, nous souhaitons que ce système soit **évolutif** et en particulier qu'il puisse :

- apprendre en partant de zéro (sans connaissance a priori) ;
- permettre l'ajout de classe ;
- éviter les paramètres dépendants du problème.

2.3 Apprentissage incrémental : panorama des principaux systèmes

Dans cette partie, nous allons présenter les principaux systèmes incrémentaux évolutifs possédant les caractéristiques que nous recherchons.

Les premiers systèmes d'apprentissage incrémental étaient des systèmes d'inférence floue (SIF) évolutifs appliqués à la régulation et au contrôle de systèmes physiques complexes. On trouve donc de nombreuses approches basées sur les SIF dans la littérature, même si elles sont plus souvent appliquées à des problèmes de régression que de classification. Depuis, plusieurs systèmes à ensembles évolutifs ont été proposés, ainsi que des algorithmes d'apprentissage incrémental de machines à vecteurs supports.

2.3.1 Machines à vecteurs supports (SVM) évolutives

Les machines à vecteurs supports (SVM) sont des systèmes de classification très puissants qui vont déterminer la séparatrice maximisant la marge entre les classes. Cette marge est définie comme la distance entre la séparatrice et les individus les plus proches, appelés vecteurs supports.

2.3.1.1 SVM incrémentale

Les premiers algorithmes d'apprentissage incrémental de machine à vecteurs supports ont été proposés dans [Syed et al., 1999] et [Ruping, 2001]. Cette approche vise à gérer l'ensemble de vecteurs supports de manière incrémentale. Lorsqu'une nouvelle donnée d'apprentissage est mal classée ou à l'intérieur de la marge, la séparatrice est recalculée à partir des vecteurs supports et de ce nouveau point.

Le principal inconvénient des SVM incrémentales est leur difficulté à gérer l'ajout de nouvelles classes. Une classe récemment ajoutée ne possèdera que peu d'individus, et en maximisant la marge, une SVM aura tendance à isoler ces quelques points sans vraiment généraliser la représentation de cette nouvelle classe.

2.3.2 Systèmes à ensemble évolutifs

Les systèmes à ensemble sont composés d'un groupe de classifieurs faibles dont les résultats sont ensuite combinés, le plus souvent par un système de vote majoritaire. Ces classifieurs faibles peuvent être construits avec un sous-ensemble de données d'apprentissage, de caractéristiques, de classes ou une combinaison des trois.

L'ajout de nouveaux classifieurs à l'arrivée de nouvelles données permet l'apprentissage incrémental et l'ajout de classes. Néanmoins, une attention particulière doit être portée à la diversité de cet ensemble de classifieurs afin que la combinaison soit représentative de toutes les données.

2.3.2.1 Learn++

L'algorithme Learn++, proposé dans [Polikar et al., 2001], permet de construire un système à ensemble incrémental où de nouveaux classifieurs faibles sont créés à l'arrivée de nouvelles données. L'implémentation proposée utilise des Perceptrons multi-couches (MLP) comme classifieurs faibles, mais comme pour l'algorithme AdaBoost, d'autres choix sont également possibles.

Learn++ nécessite une mémoire partielle des données afin de maintenir un ensemble d'apprentissage. Cet ensemble d'apprentissage est ensuite échantillonné, suivant une distribution de probabilité correspondant à la difficulté des individus, afin d'obtenir des sous-ensembles sur lesquels seront appris les classifieurs faibles. Ensuite, les résultats de l'ensemble de classifieurs faibles générés sont combinés par une méthode de vote pondéré spécifique.

2.3.2.2 *Growing Negative Correlation Learning* (GNCL)

GNCL [Minku et al., 2009] est une méthode d'apprentissage d'ensemble de classifieurs également basée sur des réseaux de neurones. GNCL est une méthode de *bagging*, et d'autres types de classifieurs faibles sont également utilisables.

Cette méthode vise à produire un ensemble de classifieurs faibles le plus varié possible en insérant une pénalité dans la fonction d'erreur de chaque classifieur faible. Cette pénalité négative de corrélation augmente lorsqu'un classifieur se « rapproche » d'un autre, et assure ainsi la diversification de l'ensemble en pénalisant les classifieurs corrélés.

2.3.3 Systèmes d'inférence floue évolutifs

Dans un système d'inférence floue, un individu active plus ou moins des règles d'inférence. Les sorties de ces règles vont ensuite être combinées, en fonction de leur activation, pour prédire la classe de cet individu. Les systèmes d'inférence floue sont, sous certaines conditions, équivalent à des réseaux de neurones à fonction radiale [Hunt et al., 1996].

L'apprentissage incrémental d'un système d'inférence floue se fait par modification de ces règles d'inférence. En particulier, de nouvelles classes peuvent facilement être ajoutées en créant de nouvelles règles.

2.3.3.1 *FLEXible Fuzzy Inference System* (FLEXFIS)

FLEXFIS est un système d'inférence floue de Takagi-Sugeno d'ordre un [Lughofer, 2008b], c'est-à-dire avec une structure de conclusion linéaire. L'idée principale de cet algorithme est d'utiliser les principes de quantification vectorielle de manière incrémentale pour mettre à jour les prémisses des règles d'inférence.

Le principal inconvénient de cette méthode de mise à jour des prémisses vient du fait qu'elle nécessite un seuil de distance. Ce seuil contrôle l'équilibre entre la mise à jour des *clusters* existants et la création de nouveaux. Ce paramètre doit être ajusté pour chaque problème de classification, ce qui est difficilement réalisable de manière incrémentale.

2.3.3.2 *Evolving Fuzzy Neural Networks* (EFuNN)

EFuNN est un système de classification flou composé de règles associant une hypersphère de l'espace d'entrée à une hypersphère de l'espace de sortie [Kasabov, 2001]. L'auteur propose une application de cette méthode à la reconnaissance de phonèmes.

Dans ce système, une nouvelle règle est créée lorsqu'une nouvelle donnée d'apprentissage n'active pas de règle au-dessus d'un certain seuil. Une nouvelle règle est également créée lorsqu'une donnée d'apprentissage active une règle mais que l'erreur dépasse un autre seuil. Ces principes d'évolution dépendent fortement de seuils, qui sont des paramètres qu'il faut ajuster pour chaque problème.

2.3.3.3 *Dynamic Evolving Neural-Fuzzy Inference System* (DENFIS)

DENFIS est un système d'inférence floue de Takagi-Sugeno d'ordre un où la fonction d'appartenance floue est la fonction triangulaire [Kasabov and Song, 2002]. L'auteur utilise ce système pour faire de la prédiction dynamique de séries temporelles.

L'évolution des prémisses des règles est faite à l'aide d'une méthode de *clustering* évolutif (ECM) qui nécessite un paramètre spécifiant la taille des *clusters*. DENFIS partage donc l'inconvénient des systèmes précédents, c'est à dire l'ajustement d'un paramètre dépendant du problème.

Les conclusions des règles sont apprises par la méthode des moindres carrés récursifs.

2.3.3.4 *Evolving Takagi-Sugeno* (eTS) et eClass

eTS est un système d'inférence floue de Takagi-Sugeno d'ordre un qui ajuste les règles d'inférences pour suivre l'évolution du flux de données [Angelov and Filev, 2004]. eClass est une extension de eTS appliquée au domaine de la classification [Angelov and Zhou, 2008].

L'ajustement du système se fait à plusieurs niveaux. Tout d'abord, les prémisses des règles existantes sont mises à jour à l'arrivée de nouvelles données. Ensuite, de nouvelles règles sont créées si les nouvelles données ne correspondent pas aux prémisses des règles existantes. Enfin, les paramètres des conclusions des règles d'inférence sont également mis à jour.

Il est important de préciser que la mise à jour des prémisses et la création de règles sont faites par une méthode de *clustering* basée sur la densité. L'avantage de cette solution est qu'elle ne nécessite pas l'ajustement de paramètres dépendant du problème.

2.3.3.5 Evolve++

Evolve++ est un système d'inférence floue de Takagi-Sugeno d'ordre un, basé sur eTS, avec une structure de prémisse améliorée [Almaksour and Anquetil, 2010] pour prendre en compte la corrélation entre les axes.

La spécificité de l'algorithme Evolve++ est que l'apprentissage est dirigé par la confusion [Almaksour and Anquetil, 2009]. Lors de l'apprentissage de nouvelles données, une plus grande importance est accordée aux individus qui sont mal classés par le système, et une plus faible importance aux individus bien classés. Cela permet d'accélérer le discernement des individus mal reconnus et de limiter les modifications inutiles.

2.3.4 Discussion

Si les SVM incrémentales sont très performantes, leur utilisation de manière évolutive est beaucoup moins aisée. L'algorithme de calcul des SVM cherche à séparer deux classes avec la plus grande marge ce qui réduit la généralisation des classes récentes sous représentées.

Le principal inconvénient des méthodes à ensembles est le problème d'*outvoting* lorsque l'on introduit de nouvelles classes. Ce problème est dû au fait que les anciens classifieurs votent pour des individus appartenant à des classes sur lesquelles ils n'ont pas été entrainés. Cela va donc fortement défavoriser les dernières classes introduites.

Les systèmes d'inférence floue permettent généralement d'obtenir de bonnes performances, et se comportent bien à l'ajout de nouvelles classes grâce à leur représentation floue des ensembles. De plus, ces systèmes ne nécessitent pas de mémoire des instances, mais seulement une mémoire des concepts. Les systèmes d'inférence floue semblent donc bien correspondre à nos attentes.

Nous allons donc étudier plus en détails le fonctionnement des systèmes d'inférence flou ainsi que leur utilisation de manière incrémentale et évolutive dans la prochaine partie.

2.4 Système d'inférence floue (SIF) incrémental évolutif

Dans cette partie, nous allons commencer par aborder les principes de la logique floue. Ensuite, nous étudierons les différentes structures de SIF. Enfin, nous verrons les différents niveaux de fonctionnement de l'apprentissage incrémental d'un SIF.

2.4.1 Théorie de la logique floue

La théorie de la logique floue a été formalisée dans [Zadeh, 1965].

2.4.1.1 Théorie des ensembles flous

En logique floue, l'appartenance d'un individu à un ensemble n'est pas binaire, mais floue. Cette appartenance n'est plus limitée aux deux valeurs $\{0; 1\}$, mais peut varier dans tout l'intervalle $[0; 1]$. Un individu peut ainsi appartenir, à différents degrés, à plusieurs ensembles flous.

Différentes fonctions peuvent être utilisées comme fonction d'appartenance, les plus utilisées sont la fonction triangulaire [Kasabov and Song, 2002], la fonction trapézoïdale et la fonction gaussienne [Lughofer, 2008b][Angelov and Filev, 2004].

2.4.1.2 Inférence floue

Un système d'inférence floue est composé de règles structurées selon la forme suivante :

$$IF\ (prémisse)\ THEN\ (conclusion) \tag{2.1}$$

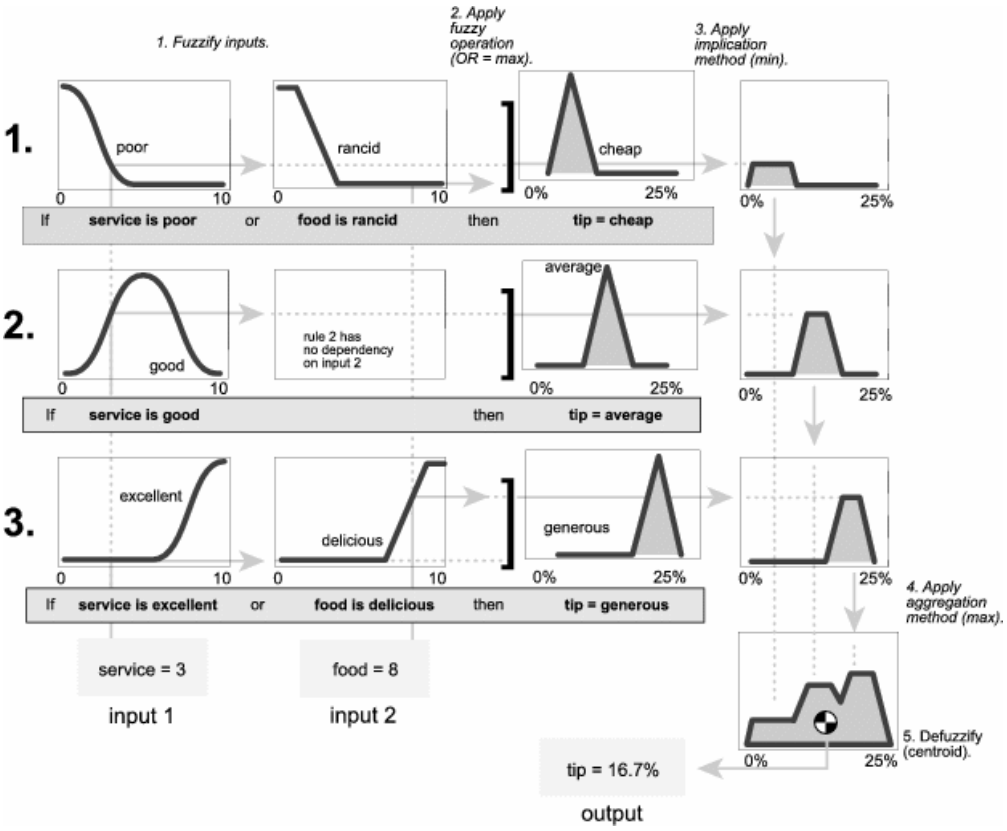

FIGURE 2.1 – Exemple d'inférence floue sur un système à trois règles, avec deux entrées (*service*, *food*) et une sortie (*tip*) – source : mathworks.com

Le processus d'inférence floue peut être détaillé en cinq étapes (voir figure 2.1) :

Fuzzyfication : les données « classiques » sont rendues floues ;

Unification : les données floues en entrée sont unifiées avec les prémisses des règles ;

Implication : en fonction de leur activation, les règles produisent les sorties floues ;

Agrégation : les sorties floues des règles sont agrégées pour produire la sortie floue du système ;
Défuzzyfication : la sortie floue est rendue « classique ».

La structure des prémisses et la structure des conclusions sont détaillées dans la section suivante.

2.4.2 Architecture d'un SIF

2.4.2.1 Structure des prémisses

La prémisse d'une règle d'inférence floue est une agrégation d'ensembles flous :

$$IF\ (x_1\ is\ A_1),\ \ldots,(x_k\ is\ A_k)\ THEN\ (conclusion) \tag{2.2}$$

où $x_1, \ldots, x_n$ sont les variables d'entrée et $A_1 \ldots A_k$ sont les différents ensembles flous d'entrée.

La zone d'influence d'une règle floue est appelée le prototype de cette règle. Les prototypes des règles d'inférence peuvent être calculés à l'aide d'algorithmes de *clustering*. Un algorithme de regroupement basé sur un seuil de distance a l'inconvénient d'introduire un paramètre dépendant du problème [Lughofer, 2008b], ce qui est évité avec une méthode basée sur la densité [Angelov and Filev, 2004].

Dans le cas de la fonction d'appartenance gaussienne, les prototypes sont de formes hyperelliptiques. Le prototype d'une règle est caractérisé par son centre $\mu = (\mu_1, \ldots, \mu_n)$ et son rayon $\sigma = (\sigma_1, \ldots, \sigma_n)$.

Le degré d'activation β d'une règle par un individu $x = (x_1, \ldots, x_n)$ est l'union – opérateur produit – des degrés d'activation des différentes composantes de cet individu selon les différents axes. Ils sont obtenus en appliquant la fonction d'appartenance, paramétrée par le rayon, à la distance entre les composantes de cet individu et celles du centre du prototype de la règle.

Pour la fonction d'appartenance gaussienne, le degré d'activation β est donné par la formule :

$$\beta(x) = \prod_{i=1}^{n} exp(-\frac{||x_i - \mu_i||}{2\sigma_i}) \tag{2.3}$$

2.4.2.2 Structure des conclusions

La sortie floue d'une règle est un vecteur regroupant les degrés d'appartenance aux k classes :

$$S^{(i)} = (s_1^{(i)}, \ldots, s_k^{(i)}) \tag{2.4}$$

La sortie floue $\hat{y}$ du système est la somme des sorties floues des r règles pondérées par leur activation β :

$$\hat{y} = (s_1, \ldots, s_k) = \sum_{i=1}^{r} \beta_i (s_1^{(i)}, \ldots, s_k^{(i)}) \tag{2.5}$$

La classe prédite $\tilde{y}$ est celle avec le plus fort degré d'appartenance :

$$\tilde{y} = \arg \max_{1 \leq i \leq k} \hat{y} = \arg \max_{1 \leq i \leq k} (s_1, \ldots, s_k) \tag{2.6}$$

Les conclusions des règles, qui produisent leur sortie floue, peuvent être de deux type : d'ordre zéro ou d'ordre un.

Conclusion d'ordre zéro Les SIF avec une structure de conclusion d'ordre zéro ont été proposés dans [Mamdani, 1977]. EFuNN [Kasabov, 2001] est un exemple de SIF de Mamdani.

Dans un tel système, les règles sont de la forme :

$$IF\ (prémisse)\ THEN\ \hat{y}^{(i)}\ is\ (s_1, \ldots, s_k) \tag{2.7}$$

Un SIF de Mamdani va donc créer un partitionnement flou de l'espace d'entrée et le relier à un partitionnement de l'espace de sortie. Le principal avantage des SIF d'ordre zéro est leur transparence : on peut facilement interpréter les règles produites en langage naturel.

La figure 2.2 représente un SIF d'ordre zéro sous la forme d'un réseau de neurones.

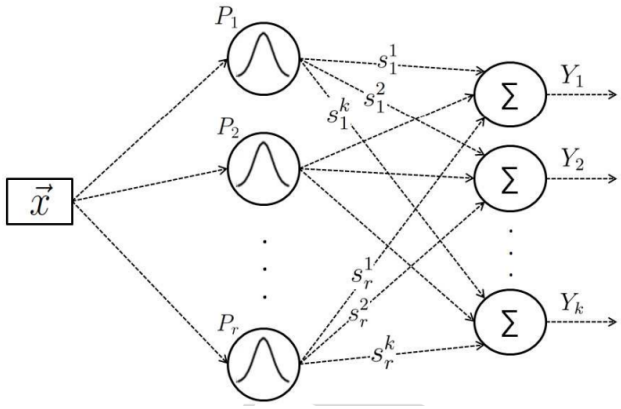

FIGURE 2.2 – Système d'inférence floue d'ordre zéro

Pour chacune des r règles, la conclusion est un vecteur regroupant les degrés d'appartenance à chacune des k classes :

$$S^{(i)} = (s_1^{(i)}, \ldots, s_k^{(i)}) \quad pour\ 1 \leq i \leq r \tag{2.8}$$

Conclusion d'ordre un Les systèmes d'inférence floue avec une structure de conclusion d'ordre un ont été proposés dans [Takagi and Sugeno, 1985]. eTS [Angelov and Filev, 2004] est un exemple de SIF de Takagi-Sugeno.

Dans un tel système, les règles sont de la forme :

$$IF\ (prémisse)\ THEN\ \hat{y}^{(i)} = (\Pi_1^{(i)}(x_1, \ldots, x_n),\ \ldots, \Pi_k^{(i)}(x_1, \ldots, x_n)) \tag{2.9}$$

où $\Pi_1^{(i)} \ldots \Pi_k^{(i)}$ sont des fonctions linéaires des variables d'entrées.

Les SIF d'ordre un permettent généralement d'obtenir une meilleure précision que les SIF d'ordre zéro, mais sont moins facilement interprétables.

La figure 2.3 représente un SIF d'ordre un sous la forme d'un réseau de neurones.

Pour chacune des r règles, la conclusion est une matrice regroupant les coefficients des fonctions linéaires $\Pi_1^{(i)} \ldots\ \Pi_k^{(i)}$:

$$\Pi^{(i)} = \begin{pmatrix} \pi_{1,1}^{(i)} & \ldots & \pi_{1,k}^{(i)} \\ \vdots & & \vdots \\ \pi_{n,1}^{(i)} & \ldots & \pi_{n,k}^{(i)} \end{pmatrix} \quad pour\ 1 \leq i \leq r \tag{2.10}$$

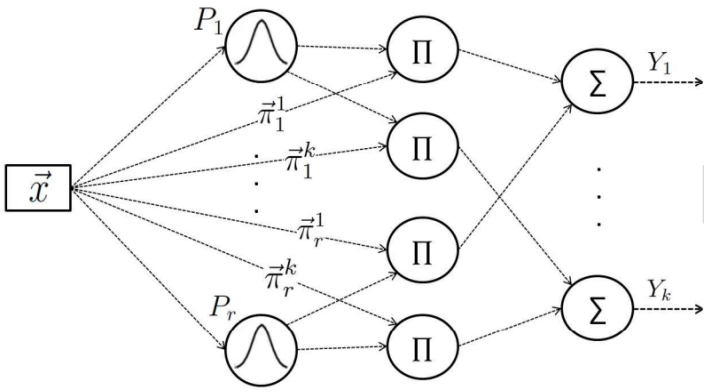

FIGURE 2.3 – Système d'inférence floue d'ordre un

La sortie floue d'une règle est alors obtenue en multipliant x^T par la matrice de conclusion $\Pi^{(i)}$:

$$S^{(i)} = (s_1^{(i)}, \ldots, s_k^{(i)}) = x^T \Pi^{(i)} \tag{2.11}$$

2.4.3 Apprentissage incrémental d'un SIF

2.4.3.1 Évolution des prémisses

Lorsqu'une nouvelle donnée d'apprentissage arrive, il est nécessaire de faire évoluer les prémisses de certaines règles.

Ajustement des prototypes Lorsqu'une nouvelle donnée arrive, il faut alors mettre à jour les prototypes des différentes règles en fonction de leur activation. Leur centre μ et leur rayon σ sont alors recalculés récursivement à l'aide des formules statistiques suivantes :

$$\mu_t = \frac{\alpha_{t-1}}{\alpha_t}\mu_{t-1} + \frac{\beta_t}{\alpha_t}x_t \tag{2.12}$$

$$\sigma_t = \frac{\alpha_{t-1}}{\alpha_t}\sigma_{t-1} + \frac{\beta_t}{\alpha_t}(\mu_t - x_t) \tag{2.13}$$

où β_t est l'activation du prototype et $\alpha_t = \sum_{i=1}^{t} \beta_i$ est le cumul des activations de ce prototype à l'instant t.

Création de nouvelles règles Si la nouvelle donnée ne correspond à aucun prototype de règle déjà existante, il est alors nécessaire de créer une nouvelle règle autour de cette donnée. Cette décision peut être prise sur différents critères :

- si l'activation de toutes les règles est inférieure à un certain seuil ;
- à l'aide d'un algorithme de *clustering* basé sur un seuil de distance [Kasabov and Song, 2002] ;
- à l'aide d'un algorithme de *clustering* basé sur la densité [Angelov and Filev, 2004].

Les deux premières solutions ont l'inconvénient d'introduire un paramètre qui est dépendant du problème et qui est difficilement estimable de manière incrémentale, nous préférerons donc la troisième solution.

On peut notamment retenir l'algorithme RepStream [Lühr and Lazarescu, 2009], basé sur une représentation des données sous forme de graphe, ainsi que l'algorithme eClustering qui procède par estimation récursive de la densité [Ramezani et al., 2008].

2.4.3.2 Évolution des conclusions

Lorsqu'une nouvelle donnée d'apprentissage arrive, il est également nécessaire de faire évoluer les conclusions de certaines règles. Cette optimisation est le plus souvent réalisée de manière supervisée à l'aide de l'algorithme des moindres carrés récursifs [Lughofer, 2008b], mais peut aussi être réalisée de manière non supervisée par *clustering* [Kasabov, 2001].

Cette optimisation des conclusions par moindres carrés récursifs peut être réalisée de manière globale ou de manière locale. Il est montré dans [Almaksour, 2011] que les deux méthodes donnent des résultats similaires, mais que l'optimisation locale est de complexité plus faible. L'optimisation locale est préférable dans le cas d'une utilisation en-ligne où l'apprentissage est réalisé en arrière-plan. Enfin, toujours de part sa plus faible complexité, l'optimisation locale des conclusions permet de travailler dans un espace d'entrée de plus grande dimension (nombre de caractéristiques plus important).

2.4.3.3 Stabilité de l'apprentissage

Lors de l'apprentissage incrémental d'un SIF, l'évolution des prémisses et des conclusions est réalisée de manière simultanée. Cette évolution des prémisses pendant l'optimisation des conclusions crée des instabilités [Lughofer, 2008a]. En effet, lorsque les prototypes changent, les anciennes modifications apportées aux conclusions ne correspondent plus tout à fait aux nouveaux prototypes. Une méthode pour limiter ces instabilités est de maintenir une mémoire partielle des données et de réaliser la mise à jour des prémisses de manière ponctuelle ; une seconde est de concentrer les mises à jour sur les individus mal ou difficilement classés [Almaksour and Anquetil, 2009]. Une autre piste pourrait être d'introduire de l'oubli dans l'optimisation récursive des conclusions pour limiter le poids des anciennes modifications qui ne sont plus forcement pertinentes.

2.4.3.4 Ajout de classes

L'ajout de classes est relativement facile dans un SIF. L'arrivée d'un individu d'une nouvelle classe va mettre à jour les prémisses de manière classique, ou créer une nouvelle règle si besoin, puis la nouvelle classe sera ajoutée aux règles activées.

2.5 Apprentissage décrémental

Nous allons maintenant étudier ce que l'on peut appeler l'apprentissage décrémental. L'apprentissage incrémental est le fait d'apprendre petit à petit, au fur et à mesure que les données arrivent. Par opposition, on peut appeler apprentissage décrémental le fait de désapprendre, d'oublier petit à petit la connaissance issue des données d'apprentissage les plus anciennes.

Dans cette partie, nous commencerons par les concepts inhérents à l'apprentissage décrémental, puis nous étudierons les systèmes existants.

2.5.1 Concepts

2.5.1.1 Suppression de classes

La suppression de classes est un sujet qui n'est pas abordé dans la littérature, du moins pas à notre connaissance. Cela est probablement dû au fait que l'on peut très bien laisser un système avec des classes inutilisées. D'autre part, il est plutôt rare de se retrouver avec des classes inutilisées. C'est ici le cas avec notre contexte applicatif de reconnaissance de raccourcis graphiques où l'utilisateur peut être amené à modifier les classes.

Supprimer des classes lorsqu'elles sont inutilisées pourrait permettre de diminuer les risques de confusion et ainsi améliorer la reconnaissance des classes restantes. Cela permettrait également de pouvoir simplifier le système.

Dans un SIF, il serait possible de supprimer les règles n'intervenant que dans ces classes, mais aussi de simplifier les conclusions d'autres règles. Cependant, cette suppression doit être effectuée avec précaution pour ne pas générer d'instabilité ou de perte de précision dans la reconnaissance des autres classes. En effet, chaque règle participe à la reconnaissance, à différents degrés, de toutes les classes.

2.5.1.2 Dérive temporelle

Une dérive temporelle est un changement, plus ou moins progressif, mais durable, des caractéristiques du flot de données. Cette dérive correspond à un changement du concept que le système essaye de modéliser, et le système doit donc s'y adapter.

Par exemple, la manière de dessiner de l'utilisateur va probablement évoluer au fur et à mesure qu'il va s'habituer à utiliser l'IHM tactile, et donc le système de reconnaissance. Ainsi l'utilisateur va probablement accélérer sa manière de dessiner et ses dessins vont alors avoir tendance à être plus déformés : passage d'un mode novice à un mode expert. Il peut alors être intéressant de concentrer l'apprentissage sur la manière courante d'utilisation, et ce au détriment des données issues des premières utilisations lorsque l'utilisateur prenait le système en main.

De même, si l'utilisateur change, il peut être nécessaire de recentrer le système sur la manière de dessiner de cette nouvelle personne et d'oublier petit à petit les particularités de l'ancien utilisateur.

Néanmoins, cette dérive temporelle du système ne doit pas être trop rapide pour ne pas se spécialiser sur de petites déformations passagères. Une dérive temporelle trop rapide pourrait limiter la capacité de généralisation du système. Une attention particulière doit également être prêtée aux classes peu fréquentes afin qu'elles ne soient pas oubliées ou mal reconnues.

2.5.2 Systèmes existants

2.5.2.1 SVM décrémentale

[Klinkenberg and Joachims, 2000] soulève le problème de dérive temporelle pour l'apprentissage incrémental de SVM et propose de maintenir une fenêtre glissante sur les données d'apprentissage. Les auteurs proposent une méthode de détection des dérives de concept qui ajuste automatiquement la fenêtre d'apprentissage. Cependant, cette approche nécessite de maintenir une mémoire des données, au moins partielle, mais ne permet pas d'oublier une donnée d'apprentissage sans reconstruire tout le système.

[Boukharouba et al., 2009] propose un algorithme permettant de mettre à jour une SVM de manière incrémentale, mais aussi de manière décrémentale. Cette méthode permet d'apprendre à

partir de nouvelles données, mais également de desapprendre ce qui a été appris à partir de certaines anciennes données lorsqu'elles deviennent obsolètes sans reconstruire tout le système.

2.5.2.2 SIF décrémental

Il existe dans la littérature consacrée au contrôle plusieurs approches pour permettre à un système de régression de s'adapter aux modifications de son environnement.

L'approche la plus courante est l'utilisation d'un facteur d'oubli exponentiel [Moustakides, 1997] dans l'algorithme des moindres carrés récursifs utilisé pour l'optimisation des conclusions. Cela permet de créer une dérive temporelle dans le système qui suit tout changement de concept dans les données.

Une autre approche [Lughofer and Angelov, 2011] est de détecter les changements de concept pour ensuite réagir en déplaçant les prototypes ou en créant de nouvelles règles.

2.5.3 Discussion

Plusieurs approches d'apprentissage décrémental existent pour des systèmes de régression, mais peu de système de classification décrémental existent. On peut retenir deux idées, l'utilisation d'une fenêtre glissante de données et l'utilisation d'un facteur d'oubli exponentiel dans l'algorithme des moindres carrés récursifs utilisé pour le calcul des conclusions.

Nous allons donc explorer ces deux pistes pour l'apprentissage décrémental d'un SIF dédié à la classification de gestes manuscrits.

Chapitre 3

Apprentissage décrémental d'un système d'inférence floue (SIF)

Nous allons maintenant étudier l'apprentissage décrémental d'un SIF. L'apprentissage incrémental d'un SIF est chose courante pour apprendre petit à petit, au fur et à mesure que les données arrivent. Par contre, l'apprentissage décrémental, le fait de désapprendre, d'oublier petit à petit, est beaucoup moins étudié en classification à l'aide de SIF.

Dans ce chapitre, nous commencerons par expliciter l'intérêt de l'apprentissage décrémental, notamment vis à vis du contexte de la classification de gestes manuscrits, puis nous proposerons deux algorithmes d'apprentissage décrémental des conclusions d'un SIF avec des conclusions d'ordre un.

3.1 Intérêts de l'apprentissage décrémental

L'apprentissage décrémental n'est que très peu exploré dans la littérature, pourtant l'oubli est intéressant, voir indispensable, dans plusieurs cas.

Nous commencerons par expliquer comment l'apprentissage décrémental permet de conserver un système dynamique et réactif tout au long de son utilisation. Ensuite, nous verrons comment l'oubli permet d'améliorer la réaction du système aux perturbations. Enfin, nous citerons les limites de l'apprentissage décrémental.

3.1.1 Conserver un système dynamique et réactif

Habituellement, les systèmes de classification incrémentale accordent une importance équivalente à toutes les données d'apprentissage. Cette importance est donc inversement proportionnelle au nombre de données utilisées pour l'apprentissage du système. Ainsi une donnée qui arrive au début de l'apprentissage du système aura un poids relatif fort, puisque peu de données auront déjà été utilisées pour l'apprentissage. Au contraire, une donnée qui arrive après une longue période d'utilisation, et donc d'apprentissage, aura un poids relatif faible puisque beaucoup de données auront déjà servi à l'apprentissage.

Ce comportement est problématique puisqu'il tend à augmenter l'inertie du système jusqu'à le figer. Si l'âge du système est élevé, les nouvelles données ont un poids relatif très faible et le système

ne s'adaptera plus à d'éventuels changements. Il est nécessaire de limiter le poids des anciennes données : soit l'annuler à partir d'une limite d'âge, soit le diminuer progressivement, pour qu'une dérive des nouvelles données puisse avoir une influence sur le système.

L'apprentissage décrémental est donc indispensable pour conserver un système dynamique et réactif même après une longue période d'utilisation.

3.1.2 Améliorer la réaction du système aux perturbations

Si l'on perturbe le système, en modifiant des classes par exemple, les anciennes données deviennent obsolètes et il est intéressant de diminuer leur poids dans le processus d'apprentissage. De même, si l'on ajoute ou l'on supprime des classes.

Introduire de l'oubli permet d'augmenter la vitesse d'évolution du système en concentrant l'apprentissage sur les dernière données, et d'oublier les anciennes devenues obsolètes, pour adapter le système à son utilisation courante.

3.1.3 Discussion

L'apprentissage décrémental à donc deux grands intérêts, permettre au système de suivre une dérive des données, et améliorer la réaction du systèmes aux perturbations.

Si l'apprentissage décrémental et l'oubli des anciennes données est indispensable pour conserver un système dynamique, il ne faut pas « trop » oublier. Il ne faut pas oublier des données qui ne sont pas encore obsolète et qui contiennent encore des informations utiles vis-à-vis de l'utilisation courante du système, sous peine de détériorer ses performances. La réactivité obtenue par l'oubli ne doit pas se faire au détriment des performances d'apprentissage, que ce soit en taux de reconnaissance à convergence ou en vitesse de convergence.

3.2 Apprentissage décrémental d'un SIF

L'intégration d'oubli dans l'apprentissage incrémental d'un SIF peut se faire à plusieurs niveaux : dans l'apprentissage des prémisses, dans l'apprentissage des conclusions, ou encore dans la gestion de l'ensemble de règles d'inférence.

Nous allons ici nous aborder l'apprentissage décrémental des prémisses puis nous nous concentrerons sur l'apprentissage décrémental des conclusions. En effet, ce sont les conclusions qui sont à l'origine des frontières de décisions et intégrer de l'oubli dans leur calcul permettra d'obtenir une dérive temporelle du système.

3.2.1 Apprentissage décrémental des prémisses

Plutôt que de donner le même poids relatif à tous les données, il est aussi possible d'accorder un poids fixe aux nouvelles données afin de créer une dérive temporelle. La mise à jour des centres μ et des rayons σ des prototypes des règles se ferait alors par les formules suivantes :

$$\mu_t = \begin{cases} \frac{\alpha_{t-1}}{\alpha_t}\mu_{t-1} + \frac{\beta_t}{\alpha_t}x_t & \text{si } \alpha_t < \zeta \\ \frac{\zeta-\beta_t}{\zeta}\mu_{t-1} + \frac{\beta_t}{\zeta}x_t & \text{sinon} \end{cases} \tag{3.1}$$

$$\sigma_t = \begin{cases} \frac{\alpha_{t-1}}{\alpha_t}\sigma_{t-1} + \frac{\beta_t}{\alpha_t}(\mu_t - x_t) & \text{si } \alpha_t < \zeta \\ \frac{\zeta-\beta_t}{\zeta}\sigma_{t-1} + \frac{\beta_t}{\zeta}(\mu_t - x_t) & \text{sinon} \end{cases} \tag{3.2}$$

Il faut alors fixer la valeur de ζ pour que la dérive soit suffisamment rapide pour suivre toute dérive des données mais sans compromettre la stabilité des prototypes.

3.2.2 Apprentissage décrémental des conclusions

Comme nous l'avons vu au chapitre précédent, les conclusions d'ordre un des règles d'inférence sont le plus souvent mises à jour à l'aide de l'algorithme des moindres carrés récursifs. Nous allons voir dans cette section comment cette optimisation se fait de manière locale, puis comment intégrer de l'oubli dans cet algorithme.

3.2.2.1 Optimisation locale par moindre carrés récursifs

La fonction de coût de notre SIF est la somme pondérée par les activations des fonctions de coût de chacune des r règles :

$$J = \sum_{i=1}^{r} \beta^{(i)} J^{(i)} \tag{3.3}$$

On cherche donc à minimiser localement chacune des fonctions de coût pondérées :

$$\beta^{(i)} J^{(i)} = \sum_{i=0}^{t} \beta^{(i)} (y_i - x_i^T \pi_t^{(i)})^2 = (Y - X^T \Pi^{(i)})^T B^{(i)} (Y - X^T \Pi^{(i)}) \quad pour\ 1 \leq i \leq r \tag{3.4}$$

Avec $B^{(i)}$ la matrice de pondération définie comme suit :

$$B^{(i)} = \beta^{(i)} Id \quad pour\ 1 \leq i \leq r \tag{3.5}$$

La solution de ce problème de moindre carrés récursifs pondérés est :

$$\Pi^{(i)} = (X^T B^{(i)} X)^{-1} X^T B^{(i)} Y \quad pour\ 1 \leq i \leq r \tag{3.6}$$

En posant $\Phi_t^{(i)} = X^T B^{(i)} X$, on peut en dériver les équations récursives suivantes pour mettre à jour les conclusions [Angelov and Filev, 2004] :

$$\Pi_t^{(i)} = \Pi_{t-1}^{(i)} + (\Phi_t^{(i)})^{-1} x_t \beta^{(i)} (y_t - x_t^T \Pi_{t-1}^{(i)}) \quad pour\ 1 \leq i \leq r \tag{3.7}$$

$$(\Phi_t^{(i)})^{-1} = (\Phi_{t-1}^{(i)})^{-1} - \frac{(\Phi_{t-1}^{(i)})^{-1} x_0 x_0^T (\Phi_{t-1}^{(i)})^{-1}}{\frac{1}{\beta^{(i)}} + x_0^T (\Phi_{t-1}^{(i)})^{-1} x_0} \quad pour\ 1 \leq i \leq r \tag{3.8}$$

Avec $\Pi_0^{(i)} = (0)$ et $(\Phi_0^{(i)})^{-1} = \Omega Id$.
La valeur de Ω dépend du rapport signal sur bruit [Moustakides, 1997], on prend classiquement $\Omega = 1000$ pour l'apprentissage des conclusions d'un SIF car c'est un contexte relativement bruité.

3.2.2.2 Moindres carrés récursifs et oubli

Une possibilité est d'optimiser les conclusions sur une fenêtre glissante, sans toutefois reconstruire le système à chaque fois car cela n'est pas envisageable en temps réel.

Une autre possibilité pour introduire de l'oubli est d'utiliser un facteur d'oubli exponentiel dans l'optimisation des conclusions. Pour cela, on s'inspire de l'algorithme de moindres carrés récursifs exponentiellement pondérés qui est utilisé en régression [Vahidi et al., 2005].

Cet algorithme à l'inconvénient de parfois être numériquement instable et de provoquer une divergence de la matrice de corrélation utilisée [Liavas and Regalia, 1999]. Plusieurs approches *ad hoc* existent en régression pour tenter de pallier à ce problème. On peut ainsi borner la matrice de corrélation, faire varier le facteur d'oubli à l'aide d'heuristique [Fortescue et al., 1981] ou l'enlever [Rao Sripada and Grant Fisher, 1987], ou encore réinitialiser la matrice de corrélation lorsqu'elle augmente anormalement [Salgado et al., 1988].

Nous allons donc explorer ces pistes dans la section suivante et proposer deux nouvelles approches.

3.3 Approche décrémentale sur une fenêtre

Une première approche d'apprentissage décrémental est de considérer une fenêtre glissante de données d'apprentissage. Nous allons commencer par expliquer le principe de fonctionnement de l'approche que nous proposons, puis sa formulation mathématique, et enfin nous discuterons de ses limites et des améliorations possibles.

3.3.1 Principe

Le principe proposé ici est de maintenir une fenêtre glissante sur les données. On n'utilise comme données d'apprentissage que les n dernières données, et on considère que les données plus anciennes sont devenues obsolètes. Cette approche s'inspire de la machine à vecteurs supports (SVM) décrémentale proposée dans [Boukharouba et al., 2009].

Cependant, reconstruire tout le système sur cette fenêtre à l'arrivée de chaque nouvelle donnée n'est pas envisageable d'un point de vue complexité et temps de calcul. Pour que le système soit utilisable en temps réel, il faut le mettre à jour récursivement. Ainsi, nous proposons la stratégie suivante : à l'arrivée d'une nouvelle donnée, le système est deux fois mis à jour, une première fois pour prendre en compte la nouvelle donnée, et une deuxième fois pour désapprendre l'ancienne donnée considérée comme obsolète.

Les équations de mise à jour récursive de la solution des moindres carrés, pour ajouter une données, ont été données dans la section précédente. Les équations de mise à jour récursive de la solution des moindres carrés, pour supprimer l'effet d'une données, sont données dans la section suivante.

3.3.2 Formulation mathématique

Nous disposons déjà de l'algorithme de calcul incrémental de la solution de moindres carrés, qui permet de mettre à jour la solution à l'arrivée d'une nouvelle donnée. Nous cherchons maintenant l'algorithme décrémental pour mettre à jour cette solution lorsqu'une donnée devient obsolète, et que l'on souhaite la supprimer.

Soit $\Pi_{0-t}^{(i)}$ $(1 \leq i \leq r)$ les solutions des moindres carrés calculées sur les données x_0 à x_t.

$$\Pi_{0-t}^{(i)} = (\Phi_{0-t}^{(i)})^{-1} X_{0-t}^T B^{(i)} Y_{0-t} \quad \textit{pour } 1 \leq i \leq r \tag{3.9}$$

Nous allons mettre à jour ces matrices récursivement pour enlever l'effet de la donnée x_0 et obtenir $\Pi_{1-t}^{(i)}$ $(1 \leq i \leq r)$:

$$\Pi_{1-t}^{(i)} = \Pi_{0-t}^{(i)} + (\Phi_{1-t}^{(i)})^{-1} x_0 \beta^{(i)} (x_0^T \Pi_{0-t}^{(i)} - y_0) \quad \textit{pour } 1 \leq i \leq r \tag{3.10}$$

Preuve (*pour* $1 \leq i \leq r$) :

$$\begin{aligned}
\Phi_{1-t}^{(i)}\Pi_{1-t}^{(i)} &= X_{1-t}^T B^{(i)} Y_{1-t} \\
&= X_{0-t}^T B^{(i)} Y_{0-t} - x_0 \beta^{(i)} y_0 \\
&= \Phi_{0-t}^{(i)}\Pi_{0-t}^{(i)} - x_0 \beta^{(i)} y_0 \\
&= (\Phi_{1-t}^{(i)} + x_0 \beta^{(i)} x_0^T)\Pi_{0-t}^{(i)} - x_0 \beta^{(i)} y_0 \\
&= \Phi_{1-t}^{(i)}\Pi_{0-t}^{(i)} + x_0 \beta^{(i)} x_0^T \Pi_{0-t}^{(i)} - x_0 \beta^{(i)} y_0 \\
\Phi_{1-t^{(i)}}\Pi_{1-t}^{(i)} &= \Phi_{1-t}^{(i)}\Pi_{0-t}^{(i)} + x_0 \beta^{(i)} (x_0^T \Pi_{0-t}^{(i)} - y_0)
\end{aligned}$$

Soit $\Phi_{0-t}^{(i)}$ les matrices de corrélation des moindres carrés récursifs calculés sur les données x_0 à x_n. Nous allons mettre à jour ces matrices pour enlever l'effet de la donnée x_0 et obtenir $\Phi_{1-t}^{(i)}$:

$$\Phi_{1-t}^{(i)} = \Phi_{0-t}^{(i)} - x_0 x_0^T \quad \textit{pour } 1 \leq i \leq r \tag{3.11}$$

Pour éviter de réaliser des inversions matricielles, nous allons directement mettre à jour les matrices de corrélation inverse $(\Phi_{1-t}^{(i)})^{-1}$. Pour cela, nous utilisons le lemme d'inversion matriciel :

$$(B^{-1} + CD^{-1}C^T)^{-1} = B - BC(D + C^T BC)^{-1} C^T B \tag{3.12}$$

avec $B^{-1} = \Phi_{0-t}^{(i)}$, $C = x_0$ et $D^{-1} = -\beta^{(i)}$. On obtient alors :

$$(\Phi_{1-t}^{(i)})^{-1} = (\Phi_{0-t}^{(i)})^{-1} - \frac{(\Phi_{0-t}^{(i)})^{-1} x_0 x_0^T (\Phi_{0-t}^{(i)})^{-1}}{\frac{-1}{\beta^{(i)}} + x_0^T (\Phi_{0-t}^{(i)})^{-1} x_0} \quad \textit{pour } 1 \leq i \leq r \tag{3.13}$$

3.3.3 Discussion

Le point sensible de cette approche est la taille de la fenêtre. En effet, une fenêtre trop petite réduira le taux de reconnaissance du système mais une fenêtre trop grande réduira la réactivité du système.

Une taille de fenêtre de dix fois le nombre de classes semble raisonnable. Une tel fenêtre ne limite pas les performances expérimentales du système sur les principaux *benchmark* de classification de gestes manuscrits, tout en permettant de conserver un système relativement dynamique.

Le principal inconvénient de cette approche est qu'elle nécessite une mémoire partielle des données. Il faut en effet mémoriser les données de la fenêtre d'apprentissage pour pouvoir ensuite les oublier. Suivant la taille de la fenêtre, et la dimension de l'espace d'entrée, le coût en termes de mémoire peut être relativement élevé.

3.4 Approche décrémentale progressive

Une deuxième approche d'apprentissage décrémental est de considérer un oubli progressif des anciennes données d'apprentissage. C'est l'approche couramment utilisée en régression que nous allons transposer à notre système de classification. Nous allons commencer par expliquer le principe de fonctionnement de cette approche, puis sa formulation mathématique, enfin nous discuterons de ses limites et des améliorations possibles.

3.4.1 Principe

Le principe proposé ici est de diminuer progressivement le poids des anciennes données d'apprentissage. Pour cela, on utilise l'algorithme de moindres carrés récursifs exponentiellement pondérés qui est utilisé en régression [Vahidi et al., 2005].

Les équations de mise à jour récursive des solutions des moindres carrés, avec un facteur d'oubli exponentiel, sont données dans la section suivante.

3.4.2 Formulation mathématique

Nous disposons déjà de l'algorithme de calcul incrémental de la solution de moindres carrés, qui permet de mettre à jour la solution à l'arrivée d'une nouvelle donnée. Nous allons maintenant rajouter une pondération exponentielle lors de la mise à jour de cette solution pour réduire le poids des anciennes données.

Les fonctions de coût pondérées $\beta^{(i)} J_t^{(i)}$ deviennent :

$$\beta^{(i)} J_t^{(i)} = \sum_{i=0}^{t} \lambda^{t-i} \beta^{(i)} (y_i - x_i^T \pi_t^{(i)})^2 \qquad pour\ 1 \leq i \leq r \tag{3.14}$$

$$\beta^{(i)} J_t^{(i)} = (Y_t - X_t^T \Pi_t^{(i)})^T B^{(i)} \Lambda_t (Y_t - X_t^T \Pi_t^{(i)}) \qquad pour\ 1 \leq i \leq r \tag{3.15}$$

Avec les matrices de pondération $B^{(i)}$ définies par l'équation (3.5) et Λ_t définie comme suit :

$$\Lambda_t = \mathrm{diag}(\lambda^{t-1}, \lambda^{t-2}, \ldots, \lambda, 1) = \begin{pmatrix} \lambda[\Lambda_{t-1}] & (0) \\ (0) & 1 \end{pmatrix} \tag{3.16}$$

Où $\lambda \in [0;1]$ (en pratique on observe $\lambda \in [0.95;1]$), on prend ici $\lambda = 0.99$ pour avoir un oubli modéré et ne pas dégrader les performances générales du système.

Les solutions des moindres carrés $\Pi^{(i)}$ $(1 \leq i \leq r)$ des r règles deviennent :

$$\Pi^{(i)} = (\Phi^{(i)})^{-1} X^T B^{(i)} \Lambda Y \qquad pour\ 1 \leq i \leq r \tag{3.17}$$

On obtient les équations récursives de $\Pi^{(i)}$ $(1 \leq i \leq r)$:

$$\Pi_t^{(i)} = \Pi_{t-1}^{(i)} + (\Phi_t^{(i)})^{-1} x_t \beta^{(i)} (y_t - x_t^T \Pi_{t-1}^{(i)}) \qquad pour\ 1 \leq i \leq r \tag{3.18}$$

Preuve (*pour* $1 \leq i \leq r$) :

$$\begin{aligned}
\Phi_t^{(i)} \Pi_t^{(i)} &= \lambda X_t^T B^{(i)} Y_t \\
&= \lambda X_{t-1}^T B^{(i)} \Lambda_{t-1} Y_{t-1} + x_t \beta^{(i)} y_t \\
&= \lambda \Phi_{t-1}^{(i)} \Pi_{t-1}^{(i)} + x_t \beta^{(i)} y_t \\
&= \lambda (\lambda^{-1} \Phi_t^{(i)} - \lambda^{-1} x_t \beta^{(i)} x_t^T) \Pi_{t-1}^{(i)} + x_t \beta^{(i)} y_t \\
&= \Phi_t^{(i)} \Pi_{t-1}^{(i)} - x_t \beta^{(i)} x_t^T \Pi_{t-1}^{(i)} + x_t \beta^{(i)} y_t \\
\Phi_t^{(i)} \Pi_t^{(i)} &= \Phi_t^{(i)} \Pi_{t-1}^{(i)} + x_t \beta^{(i)} (y_t - x_t^T \Pi_{t-1}^{(i)})
\end{aligned}$$

Les matrices de corrélation des moindres carrés récursifs $\Phi_t^{(i)}$ deviennent :

$$\Phi_t^{(i)} = X_t^T B^{(i)} \Lambda_t X_t = \lambda \Phi_{t-1}^{(i)} + x_t \beta^{(i)} x_t^T \quad pour\ 1 \leq i \leq r \tag{3.19}$$

La formulation récursive de $(\Phi_t^{(i)})^{-1}$ devient alors :

$$(\Phi_t^{(i)})^{-1} = \lambda^{-1}(\Phi_{t-1}^{(i)})^{-1} - \lambda^{-1} \frac{(\Phi_{t-1}^{(i)})^{-1} x_0 x_0^T (\Phi_{t-1}^{(i)})^{-1}}{\frac{\lambda}{\beta^{(i)}} + x_0^T (\Phi_{t-1}^{(i)})^{-1} x_0} \quad pour\ 1 \leq i \leq r \tag{3.20}$$

Preuve : On utilise le lemme d'inversion matriciel :

$$(B^{-1} + CD^{-1}C^T)^{-1} = B - BC(D + C^T BC)^{-1} C^T B \tag{3.21}$$

avec $B^{-1} = \lambda \Phi_{t-1}^{(i)}$, $C = x_t$ et $D^{-1} = \beta^{(i)}$.

3.4.3 Discussion

L'inconvénient de cet algorithme est que la matrice de covariance inverse augmente anormalement lorsque l'excitation du système est faible (*covariance wind-up problem* [Vahidi et al., 2005]).

Pour pallier à ce phénomène, on réinitialise la matrice de covariance lorsqu'elle augmente déraisonnablement. Ainsi lorsqu'un des éléments de $\Phi_t^{(i)}$ dépasse 1000, on réinitialise $\Phi_t^{(i)}$ à Id. Ces réinitialisations provoquent des perturbations sur le système qui peuvent en dégrader momentanément les performances mais elles sont indispensables pour ne pas qu'il s'écroule.

Pour limiter les réinitialisations de la matrice de corrélation, on va utiliser une heuristique pour introduire de l'oubli seulement lorsque celui-ci peut être utile. Les techniques utilisées en régression sont principalement *ad hoc* et n'ont pas donné de résultats sur notre système. Nous proposons de mettre en place ici la stratégie suivante : utiliser comme heuristique l'évolution du taux d'erreur du système et mettre un facteur d'oubli lorsque le taux d'erreur remonte.

Pour cela, on calcule une estimation du taux d'erreur sur une petite et une grande fenêtre exponentielle (environ 20 et 100 exemples) et l'on met un facteur d'oubli ($\lambda = 0.99$) lorsque le taux d'erreur sur la petite fenêtre dépasse celui sur la grande.

On obtient les taux d'erreur $\tau_t^{(nb)}$ par la formule récursive suivante :

$$\tau_t^{(nb)} = (1 - \frac{1}{nb})\tau_{t-1} + \frac{e_t}{nb} \tag{3.22}$$

où nb est une approximation du nombre d'exemples utilisés et e_t vaut 1 lorsqu'une erreur de reconnaissance est faite sur l'exemple t, et vaut 0 sinon.

On obtient donc

$$\lambda = \begin{cases} 0.99 & \text{si } \tau_t^{(20)} > \tau_t^{(100)} \\ 1 & \text{sinon} \end{cases} \tag{3.23}$$

Une amélioration envisageable serait de mettre au point un algorithme faisant varier le facteur d'oubli dans l'intervalle $[0.95; 1]$ en fonction des besoins du système.

Chapitre 4

Validation expérimentale

Dans le chapitre précédent, nous avons étudié les différents intérêts de l'apprentissage décrémental, ainsi que ses limites. Nous allons maintenant présenter plusieurs protocoles de test permettant de mettre en valeur ces intérêt et ces limites. Ces protocoles nous permettrons ensuite d'évaluer les approches d'apprentissage décrémental proposées.

4.1 Protocoles expérimentaux

Pour ces expériences, nous avons utilisé comme base le système d'inférence floue incrémental et évolutif Evolve++ [Almaksour, 2011] auquel on ajoute nos deux approches d'apprentissage décrémental. Nous comparerons l'approche sur une fenêtre – *Evolve Windowed (W)* – et l'approche progressive – *Evolve with Variable Forgetting and Reseting (VFR)* – à *Evolve++* .

4.1.1 Évaluation incrémentale

Pour évaluer les performance des différents systèmes d'une manière représentative des performances ressenties par l'utilisateur, nous utilisons un taux d'erreur incrémental.

Plutôt que d'utiliser une base de test, nous évaluons les performances du système sur les futures données d'apprentissage juste avant qu'elles ne servent à entrainer le système. Pour chaque donnée d'apprentissage, on commence par essayer de la reconnaître avec le système courant, ensuite on l'utilise pour entrainer le système ; puis on passe à la donnée suivante et ainsi de suite. Chaque exemple sert d'abord comme donnée de test, et seulement ensuite comme donnée d'apprentissage pour ne pas biaiser les résultats.

Á chaque point de test, on compte le nombre d'erreurs faites depuis le point de test précédent et on divise par le nombre d'exemples présentés pour avoir un taux d'erreur incrémental. On obtient ainsi un taux d'erreur représentatif des performances d'un système incrémental.

4.1.2 Jeux de données

Les expérimentations ont été réalisées sur trois bases de données de reconnaissance de gestes manuscrits publiquement disponibles : ILGDB [1], Laviola-DB [2] et Ironoff-digit [3].

Les données utilisées sont en-ligne, elles contiennent les informations dynamiques comme le sens et la vitesse du tracé en plus des information hors-ligne (le tracé en lui même). Un jeu de cinquante caractéristiques (HBF50 [4]) développé par l'équipe IntuiDoc , a été extrait de ces données pour ces expériences.

4.1.2.1 ILGDB

La base de données ILGDB – IntuiDoc Loustic Gesture Data Base [5] – est une base de données recueillie par l'équipe IntuiDoc en partenariat avec le laboratoire d'observation des usages des technologies de l'information et de la communication de Rennes (Loustic). Elle est composée de 6 629 gestes, répartis dans 588 classes, qui ont été dessinés par 38 scripteurs.

Cette base de données est très intéressante car elle a été recueillie dans un environnement immersif où l'utilisateur n'avait pas conscience que ses gestes étaient enregistrés. Les gestes ainsi obtenus sont ordonnés chronologiquement contrairement à la majorité des autres bases de gestes.

Les gestes dessinés sont différents suivant le groupe de l'utilisateur :

- Groupe 1 : les gestes sont librement choisis pas les utilisateurs (voir figure 4.1) ;
- Groupe 2 : les gestes sont structurés avec des racines libres ;
- Groupe 3 : les gestes sont structurés avec des racines imposées ;

Figure 4.1 – Exemples de gestes du groupe 1 de ILGDB (gestes libres)

Les données sont réparties en 5 phases dont certaines ne contiennent pas le même nombre d'exemples par classe. Les 173 données de chaque scripteurs sont réparties en 5 phases :

- La phase 0 est la phase d'initialisation, elle contient 3 exemples de chacune des 21 classes ;
- La phase 1 est une phase d'utilisation, elle contient un nombre d'exemples par classe variable ;
- La phase 2 est une phase de test, elle contient un exemple par classe ;
- La phase 3 est une phase d'utilisation, elle contient un nombre d'exemples par classe variable ;

1. http://www.irisa.fr/intuidoc/ILGDB.html
2. http://graphics.cs.brown.edu/research/pcc/symbolRecognitionDataset.zip
3. http://www.irccyn.ec-nantes.fr/~viardgau/IRONOFF/IRONOFF.html
4. Référence à paraître
5. Référence à paraître

– La phase 4 est une phase de test, elle contient un exemple par classe.

Cette base de données est donc très proche de conditions d'utilisation réelles d'une interface tactile. La chronologie des données récoltées est réaliste (apprentissage de l'utilisateur et déformation progressive de ses gestes) ce qui est fondamental pour tester un système d'apprentissage incrémental et encore plus décrémental.

4.1.2.2 Bases de *benchmark* connues

Nous avons également utilisé deux bases de *benchmark* de reconnaissance de forme connue : Laviola-DB et Ironoff-digit.

Laviola-DB La base de données Laviola-DB [LaViola Jr. and Zeleznik, 2007] est une base de gestes manuscrits regroupant des chiffres, des lettres et des symboles mathématiques. Elle est composée de 16 896 gestes, répartis dans 48 classes, qui ont été dessinés par 11 scripteurs.

Ironoff-digit La base de données Ironoff [Viard-Gaudin et al., 1999] est une base de mots manuscrits réalisée par près de 700 scripteurs. Nous ne travaillons ici qu'avec les 4 086 chiffres.

4.1.3 Scénarios « macroscopiques »

Nous allons d'abord présenter un premier groupe de protocoles – dits « macroscopiques »– évaluant l'intérêt de l'oubli à long terme.

4.1.3.1 Utilisation réaliste

Ce premier protocole vise à évaluer le comportement des différents systèmes sur des données réalistes et dans des condition proches d'une utilisation réelle.

Pour cela, on utilise les données de la base ILGDB qui sont sont ordonnées chronologiquement. On présente toutes les données dans l'ordre et on mesure le taux d'erreur sur chaque phase.

Ces résultats, moyennés sur les 38 scripteurs des trois groupes, sont présentés section 4.2.1 page 30.

4.1.3.2 Inertie et réactivité au changement

Ce protocole vise à mesurer l'inertie et la réactivité du système au changement à long terme.

On commence par présenter au système les classes 0 à 6 de la base Ironoff-Digit de manière régulière pendant plus ou moins longtemps. On perturbe ensuite le système en arrêtant de présenter les classes 5 et 6, et en introduisant les classes 7 à 9. On observe alors le temps nécessaire à la reconvergence du système quand la perturbation arrive après une phase de 10, 100 et 300 exemples par classe.

Ces résultats, moyennés sur 100 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.2.2 page 30.

4.1.3.3 Changement de concept progressif

Ce protocole vise à mesurer la capacité du système à suivre une dérive progressive des classes à reconnaitre (*slow concept drift*).

Pour cela, on présente à la suite les 11 scripteurs du test 1 (gestes imposés) de ILGDB. Les symboles restent donc les mêmes mais la manière de les dessiner change avec les scripteurs.

Pour chaque scripteur, on mesure le taux d'erreur à convergence (sur les phases 3 et 4, soit 46 exemples sur les 173 de chaque scripteur). Ces résultats, moyennés sur 20 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.2.4 page 33.

4.1.3.4 Changement de concept brusque

Ce protocole vise à mesurer la capacité du système à suivre un changement brusque des classes à reconnaitre (*abrupt concept drift*).

Pour cela, on présente à la suite les 21 scripteurs du test 3 (gestes libres) de ILGDB. Les scripteurs ayant choisis eux-mêmes leurs gestes, les symboles d'une même classe change donc radicalement avec les scripteurs

Pour chaque scripteur, on mesure le taux d'erreur à convergence (sur les phases 3 et 4, soit 46 exemples). Ces résultats, moyennés sur 20 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.2.3 page 32.

4.1.3.5 Performances à convergence

Ce dernier protocole vise à mesurer les performances à convergence du système pendant une longue période d'utilisation.

On présente au système toutes les classes de la base de données Ironoff-Digit de manière régulière. On observe alors le taux d'erreur à convergence.

Ces résultats, moyennés sur 100 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.2.5 page 33.

4.1.4 Scénarios « microscopiques »

Nous allons maintenant présenter un second groupe de protocoles – dits « microscopiques » – évaluant l'intérêt de l'oubli à plus court terme.

4.1.4.1 Ajout de classes

Ce premier protocole vise à évaluer la réaction du système suite à un ajout de classes.

On commence par présenter au système 5 exemples des classes 0 à 23 de la base de données Laviola-DB. On présente ensuite 5 exemples des classes 0 à 47 et on observe le comportement du système après cet ajout de classes.

Ces résultats, moyennés sur 10 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.3.1 page 34.

4.1.4.2 Suppression de classes

Ce deuxième protocole vise à évaluer la réaction du système suite à une suppression de classes.

On commence par présenter au système 5 exemples des classes 0 à 47 de la base de données Laviola-DB. On présente ensuite 5 exemples des classes 24 à 47 et on observe le comportement du système après cette suppression de classes.

Ces résultats, moyennés sur 10 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.3.2 page 35.

4.1.4.3 Alternance de classes

Ce troisième protocole vise à évaluer la réaction du système lorsque l'on présente alternativement deux groupes de classes.

On commence par présenter au système 5 exemples des classes 0 à 23 de la base de données Laviola-DB. On présente ensuite 5 exemples des classes 24 à 47 puis on représente 5 exemples des classes 0 à 23 et 5 exemples des classes 24 à 47.

Ces résultats, moyennés sur 10 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.3.3 page 35.

4.1.4.4 Alternance de classes – test *batch* par groupe

L'objectif de ce quatrième protocole est de mettre en valeur les effets de l'oubli sur les classes que l'on arrête de présenter.

Pour cela, on reprend le scénario de test « alternance de classes » ci-dessus mais on teste cette fois sur une base de test. On présente les résultats de chaque système en deux courbes, une pour le premier groupe de classes (0 à 23) et une autre pour le second groupe (24 à 48).

Ces résultats, moyennés sur 10 expériences où les individus sont présentés dans des ordres différents, sont détaillés section 4.3.4 page 37.

4.2 Résultats « macroscopiques »

Nous allons discuter ici des résultats obtenus avec les différents systèmes sur les quatre protocoles de test « macroscopiques », c'est à dire sur de longues durées d'utilisation.

4.2.1 Utilisation réaliste

Les performances obtenues par les différents systèmes sur le scénario « utilisation réaliste » (décrit section 4.1.3.1 page 28) sont présentées figure 4.2.

Nos deux approches d'apprentissage décrémental ne permettent pas d'améliorer les performances ici. Cela est dû au fait que l'on a trop peu de données par scripteur. Il est alors plus intéressant pour le système de conserver les anciennes données, même si elles ne sont plus complètement d'actualité, plutôt que de les oublier et se concentrer sur les données courantes qui ne sont pas assez nombreuses.

On a en effet entre 5 et 20 exemples par classes ce qui est trop faible pour simuler une utilisation à long terme. Les premières données ne sont pas encore vraiment obsolètes à la fin du test, et le système ne dispose pas de suffisamment de données pour se permettre d'en oublier.

4.2.2 Inertie et réactivité au changement

Les performances obtenues par les différents systèmes sur le scénario « inertie et réactivité au changement » (décrit section 4.1.3.2 page 28) sont présentées figure 4.3.

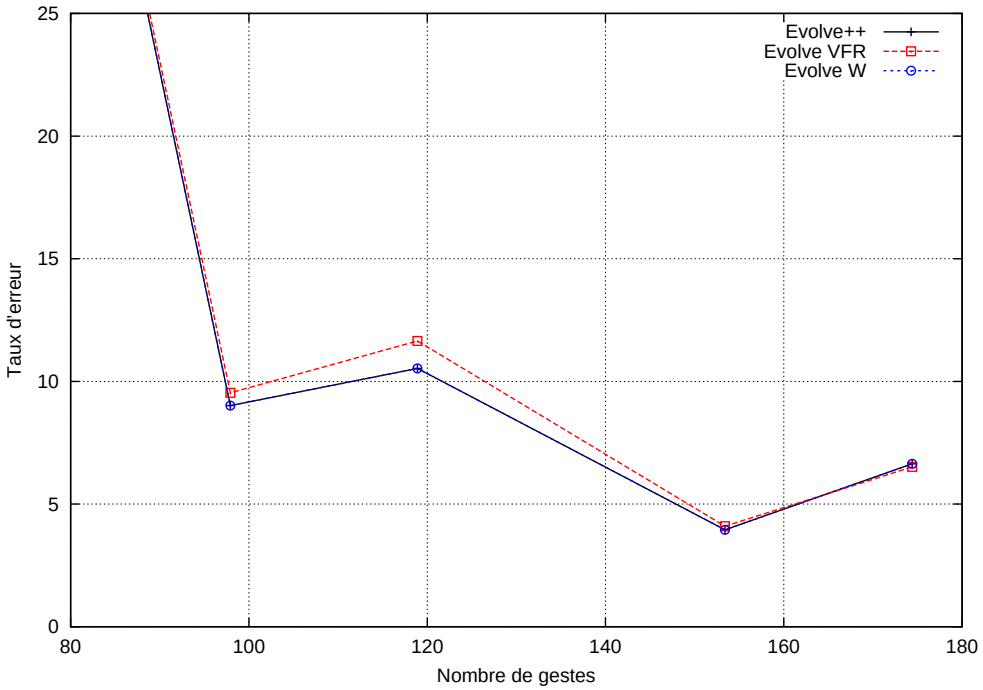

FIGURE 4.2 – Utilisation réaliste

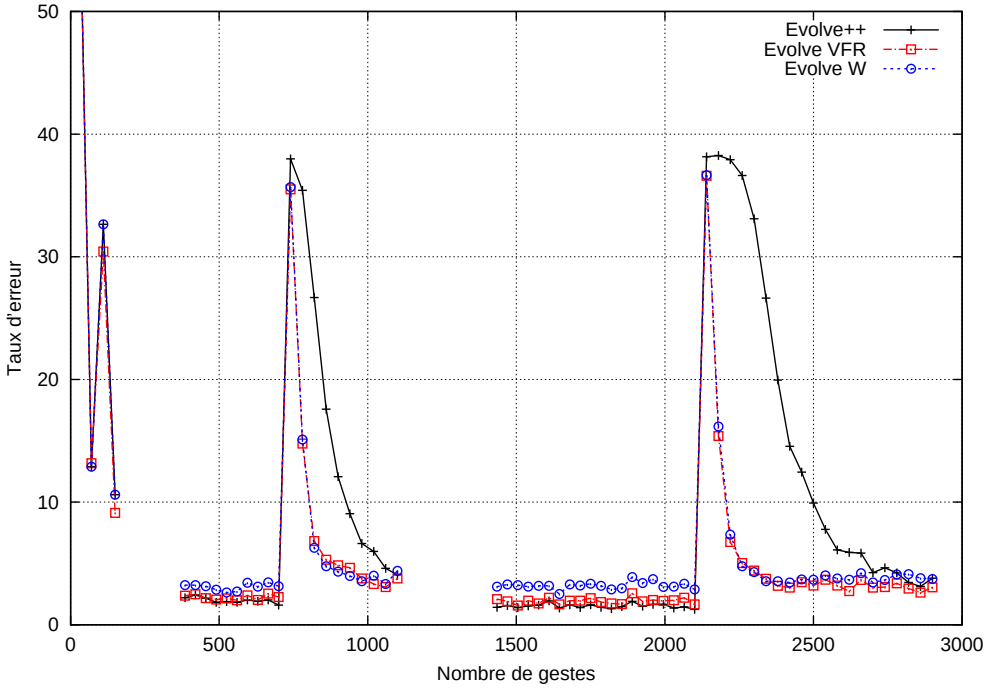

FIGURE 4.3 – Inertie et réactivité au changement

Sans oubli, tout changement est pondéré par l'âge du système. Plus la perturbation est introduite tardivement, plus *Evolve++* met du temps à reconverger (jusqu'à plus de 100 exemples par classes).

Les deux approches d'apprentissage décrémental *Evolve W* et *Evolve VFR* ont un comportement similaire : elles ne sont pas influencées par l'âge du système et ont toujours le même temps de reconvergence d'environ 5 exemples par classe.

L'oubli est donc bien indispensable pour conserver un système dynamique et réactif après une longue période d'utilisation.

4.2.3 Changement de concept progressif

Les performances obtenues par les différents systèmes sur le scénario « changement de concept progressif » (décrit section 4.1.3.4 page 29) sont présentées figure 4.4.

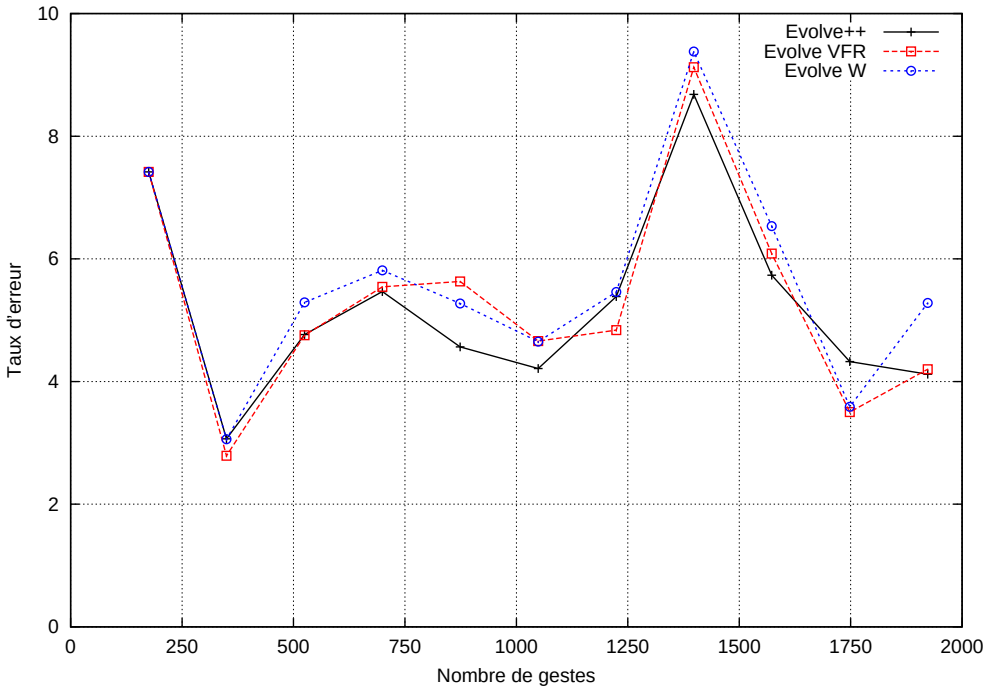

FIGURE 4.4 – Changement de concept progressif

Contrairement à ce que l'on aurait pu penser, nos approches d'apprentissage décrémental ne permettent pas d'améliorer les performances ici. Le système *Evolve++* obtient même des performances légèrement meilleures et plus régulières que *Evolve W* et *Evolve VFR* .

Cela est dû au fait que l'on a trop peu de données par scripteur. Il est alors plus intéressant pour le système de conserver les données des autres scripteurs – et de devenir un système omni-scripteur – plutôt que d'oublier les données des anciens scripteurs pour ce concentrer sur celles du scripteur courant – et devenir un système mono-scripteur. L'oubli deviendrait sans doute intéressant avec plus de données par scripteur car les systèmes mono-scripteur atteignent généralement de meilleurs performances que les systèmes omni-scripteur [LaViola Jr. and Zeleznik, 2007].

4.2.4 Changement de concept brusque

Les performances obtenues par les différents systèmes sur le scénario « changement de concept brusque » (décrit section 4.1.3.3 page 29) sont présentées figure 4.5.

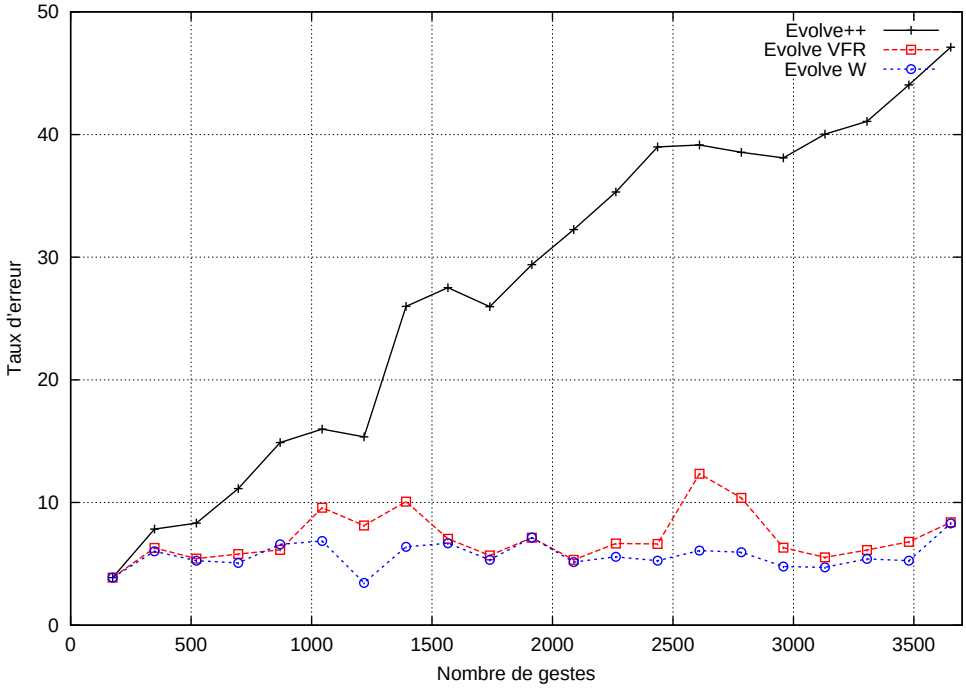

FIGURE 4.5 – Changement de concept brusque

Ce scénario est relativement difficile puisque, pour chaque scripteur, les symboles des différentes classes que l'on demande au système de reconnaitre changent complètement. Sans oubli, le système *Evolve++* n'est pas capable de s'adapter aux brusques changements de concept de ce scénario et finit avec un taux d'erreur proche de 50% à convergence. Les deux approches d'apprentissage décrémental sont capables de s'adapter complètement aux changements des données et de reconverger rapidement à des taux d'erreur inférieur à 10%.

On peut noter que le système *Evolve VFR* est moins régulier que le système *Evolve W* , ce qui est sans doute dû aux perturbations engendrées par les réinitialisations des matrices de corrélation.

4.2.5 Performances à convergence

Les performances obtenues par les différents systèmes sur le scénario « Performances à convergence » (décrit section 4.1.3.5 page 29) sont présentées figure 4.6.

Ce scénario permet de voir la dégradation des performances engendrée par l'apprentissage décrémental lorsque l'oubli n'est pas nécessaire.

L'oubli entraine ici un augmentation de 2 à 3% du taux d'erreur pour *Evolve W* et *VFR* ce qui est relativement faible. Les vitesses initiales de convergence des trois systèmes sont presque identiques, ce qui est normal car la fenêtre d'*Evolve W* n'est pas encore remplie et *Evolve VFR* n'utilise le facteur d'oubli exponentiel que lorsque le taux d'erreur augmente.

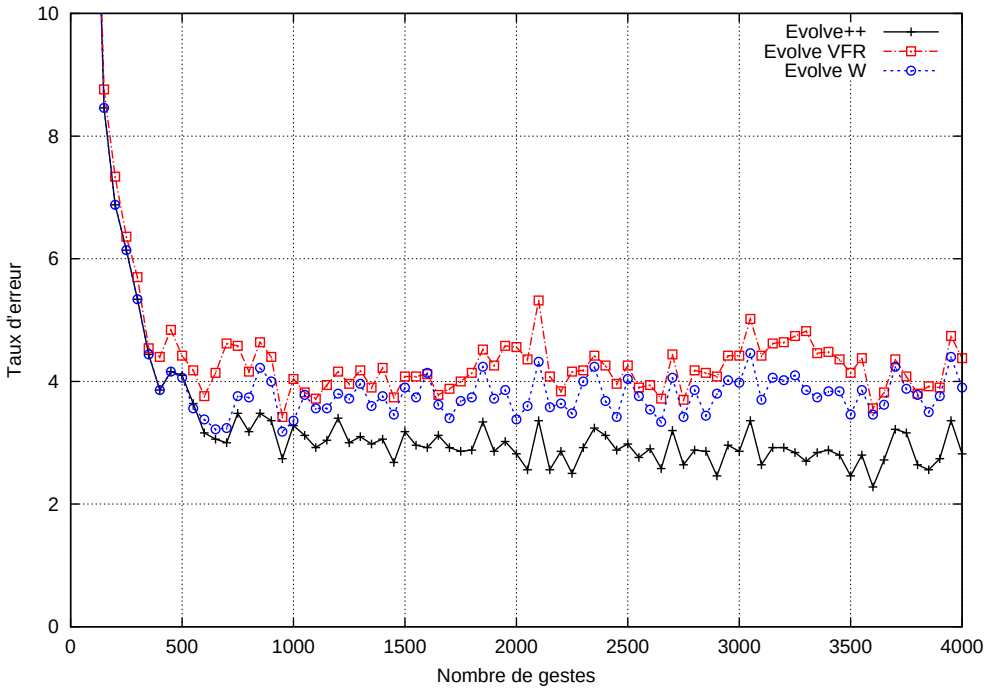

FIGURE 4.6 – Performances à convergence

L'aspect décrémental ne dégrade que très légèrement les performances générales de l'apprentissage incrémental. Il faut cependant faire attention à ne pas oublier les données peu fréquentes qui pourraient handicaper le système si elles ne sont jamais reconnues.

4.3 Résultats « microscopiques »

Nous allons discuter ici des réponses des différents systèmes sur les quatre protocoles de test « microscopiques », c'est à dire sur de courtes durées d'utilisation.

4.3.1 Ajout de classes

Les performances obtenues par les différents systèmes sur le scénario « ajout de classes » (décrit section 4.1.4.1 page 29) sont présentées figure 4.7.

Suite à un ajout de classes, les systèmes *Evolve VFR* et *Evolve VFR* reconvergent plus vite. Après 2 exemples par classes, leur taux d'erreur sont plus faibles que celui d'*Evolve++* de 5% ; il faut un troisième exemple par classes pour que *Evolve++* atteigne le même score. La différence de taux d'erreur est relativement faible, mais la différence de nombre d'exemples est assez importante : il faut un tiers d'exemples en moins aux systèmes décrémentaux pour reconverger à un taux d'erreur de 15%.

Nos deux approches décrémentales permettent d'accélérer la reconvergence du système après la perturbation et réduisent le nombre d'exemples nécessaire avant son utilisation dans de bonnes conditions.

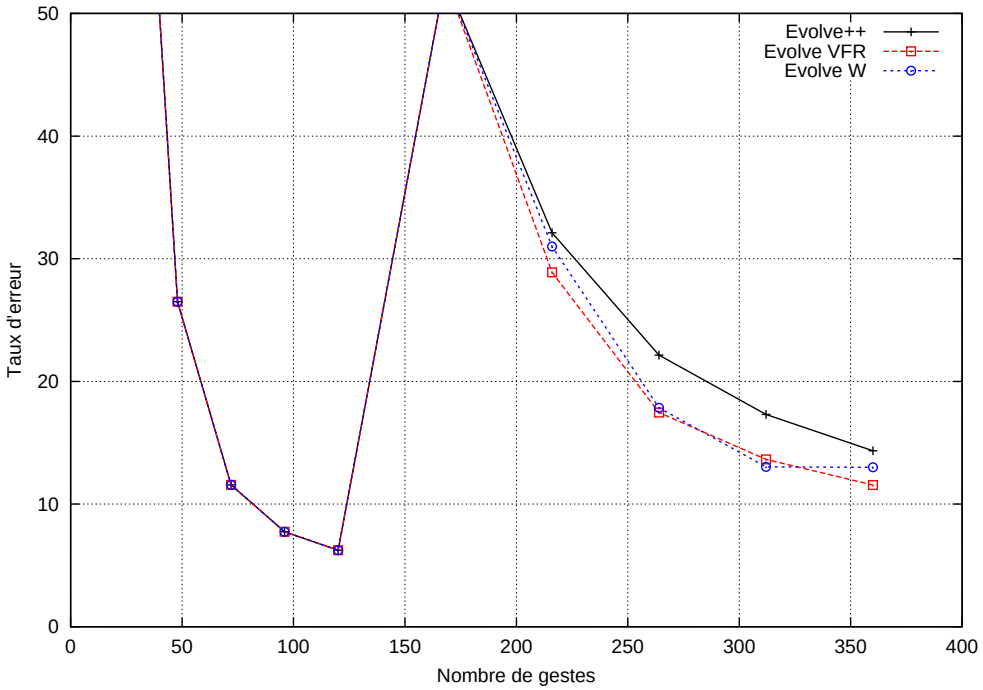

FIGURE 4.7 – Ajout de classes

4.3.2 Suppression de classes

Les performances obtenues par les différents systèmes sur le scénario « suppression de classes » (décrit section 4.1.4.2 page 29) sont présentées figure 4.8.

Suite à une suppression de classes, *Evolve++* et *Evolve VFR* se comportent de manière très similaire. *Evolve W* semble converger légèrement plus vite.

Les performances d'*Evolve W* pourraient être probablement améliorées si l'on offre à l'utilisateur la possibilité de supprimer explicitement une classe. Nous pourrions alors enlever tous les exemples de cette classe de la fenêtre pour accélérer l'amélioration des performances liée à la diminution du nombre de classes.

4.3.3 Alternance de classes

Les performances obtenues par les différents systèmes sur le scénario « alternance de classes » (décrit section 4.1.4.3 page 30) sont présentées figure 4.9.

Evolve VFR converge plus vite que *Evolve++* et obtient ainsi des résultats légèrement meilleurs malgré des augmentations du taux d'erreur aux changements de classes.

La fenêtre d'*Evolve W* est ici configurée sur 240 exemples, ce qui correspond à 5 exemples des 24 classes des deux groupes. Cela permet de simuler une alternance rapide de classe où la fenêtre contient encore des exemples de l'ancien groupe lorsque l'on alterne de nouveau.

Evolve W obtient de bons résultats car l'augmentation du taux d'erreur après les changements est modérée puisque la fenêtre contient encore des données des anciennes classes. Si la durée entre les changements était plus longue, la fenêtre ne contenait plus d'anciens exemples et on observerait un fort taux d'erreur après le changement et une convergence similaire à la convergence initiale.

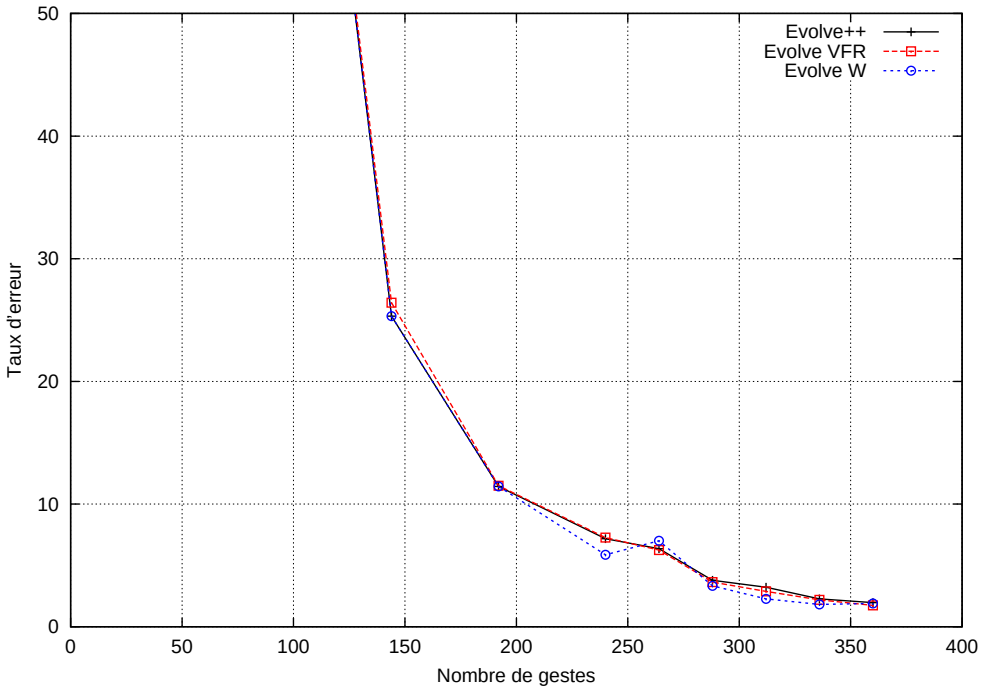

FIGURE 4.8 – Suppression de classes

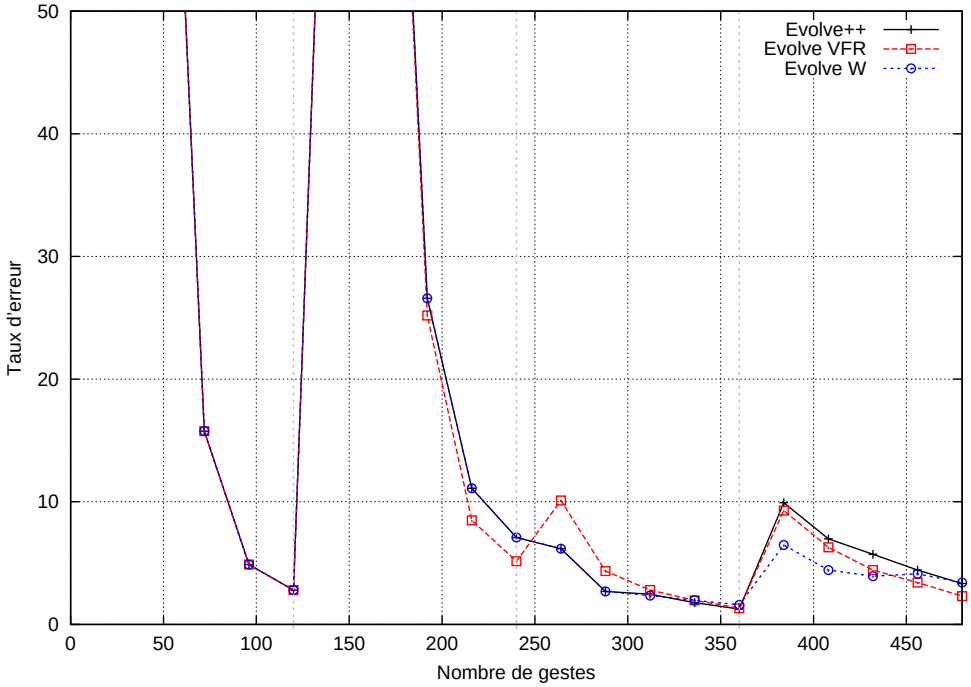

FIGURE 4.9 – Alternance de classes

4.3.4 Alternance de classes– test *batch* par groupe

Les performances obtenues par les différents systèmes sur le scénario « alternance de classes – test *batch* » (décrit section 4.1.4.4 page 30) sont présentées figure 4.10.

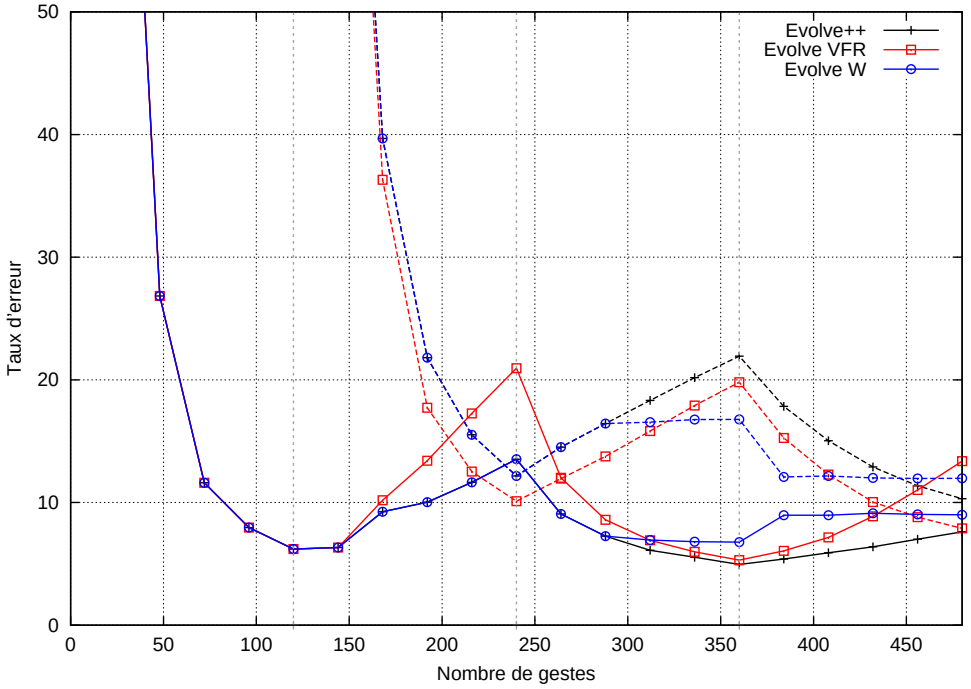

FIGURE 4.10 – Alternance de classes– test *batch* par groupe

On peut remarquer que *Evolve++* à déjà une tendance à oublier les classes que l'on arrête de lui présenter. Cette tendance est renforcée avec *Evolve VFR* qui va oublier plus vite les classes lorsque l'on arrête de les présenter. Cependant, *Evolve VFR* reconverge et réapprend vite les anciennes classes lorsque l'on recommence à les présenter. *Evolve VFR* est donc nettement plus réactif que *Evolve++* .

4.4 Discussion

Nos deux approches d'apprentissage décrémental ont bien l'effet voulu, elles permettent au système d'apprentissage incrémental de rester dynamique et réactif à long terme. Elle permettent également d'améliorer légèrement la réponse du système aux perturbations.

L'intérêt de nos approches pour le passage d'un utilisateur novice à un expert n'a pas pu être montré expérimentalement. Il faudrait une campagne de saisie plus longue pour disposer de suffisamment de données non obsolètes afin que l'on ait intérêt à oublier celles qui le sont.

Le principal inconvénient de l'apprentissage décrémental est qu'il ne faut pas trop oublier sous peine de dégrader les performances générales du système.

Les différents tests ont montrés que l'approche décrémental sur une fenêtre est plus stable que l'approche décrémentale progressive. Par contre, une classe trop peu fréquente sera oubliée si la dernière donné de cette classe sort de la fenêtre avant l'arrivée de la suivante.

Chapitre 5

Conclusion et perspectives

Dans ce mémoire nous avons étudié la mise au point d'un système d'apprentissage incrémental et décrémental pour la classification de gestes manuscrits.

Nous avons présenté différents systèmes d'apprentissages existants dans la littérature, des machines à vecteurs supports, des systèmes à ensembles et des systèmes d'inférence floue (SIF). Nous avons vu que ces derniers sont bien adaptés à notre contexte applicatif, car ils sont flexibles et supportent bien l'ajout de classes. L'intérêt de l'apprentissage décrémental a ensuite été étudié. Nous avons notamment montré que l'oubli est indispensable pour conserver un système dynamique et réactif à long terme.

Nous avons proposé deux approches pour l'apprentissage décrémental des systèmes d'inférence floue. Ces deux approches se focalisent sur l'apprentissage des conclusions des règles d'inférence par l'algorithme des moindres carrés récursifs. La première est basée sur une fenêtre glissante et la seconde sur un oubli progressif des anciennes données. Ces deux nouvelles approches ont enfin été validées expérimentalement sur plusieurs jeux de données connus de reconnaissance de gestes manuscrits. Nous avons mis au point plusieurs scénarios de test afin de mettre en valeur les intérêts de l'oubli ainsi que ses limites.

Ces travaux sont donc encourageants et montrent que l'apprentissage décrémental est nécessaire pour l'utilisation d'un système d'apprentissage dynamique à long terme. Les deux approches explorées ici se concentrent sur l'optimisation des conclusions des règles d'inférence par l'algorithme des moindres carrés récursif. On peut également optimiser ces conclusions par l'algorithme des vastes marges auquel on pourrait peut-être également intégrer de l'oubli.

Il est également possible d'intégrer de l'oubli à d'autres niveaux. L'apprentissage des prototypes des règles pourrait aussi en intégrer pour suivre les dérives des données. De plus, l'ensemble des règles est pour l'instant géré de manière incrémentale, mais on pourrait également en supprimer lorsqu'elles correspondent à des classes supprimées et qu'elles ne sont plus utilisées.

Par ailleurs, d'autres approches que l'apprentissage décrémental peuvent être envisagées pour répondre aux mêmes besoins. On pourrait par exemple recréer régulièrement un nouveau classifieur en arrière-plan et commencer à l'utiliser lorsque ses performances dépassent celles du classifieur courant. Cette approche peut être considérée comme de la combinaison de classifieurs forts.

Ces travaux ne font donc que commencer et les nouvelles interfaces tactiles promettent encore de nombreux défis en reconnaissance de forme.

Bibliographie

[Almaksour, 2011] Almaksour, A. (2011). *Incremental learning of evolving fuzzy inference systems : application to handwritten gesture recognition.* PhD thesis, Institut National des Sciences Appliquées (INSA) de Rennes.

[Almaksour and Anquetil, 2009] Almaksour, A. and Anquetil, E. (2009). Fast incremental learning strategy driven by confusion reject for online handwriting recognition. In *Tenth International Conference on Document Analysis and Recognition (ICDAR 2009)*, pages 81–85.

[Almaksour and Anquetil, 2010] Almaksour, A. and Anquetil, E. (2010). Improving premise structure in evolving takagi-sugeno neuro-fuzzy classifiers. In *Proceedings of the Ninth International Conference on Machine Learning and Applications (ICMLA 2010).*

[Angelov and Filev, 2004] Angelov, P. and Filev, D. (2004). An approach to online identification of takagi-sugeno fuzzy models. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on*, 34(1) :484 – 498.

[Angelov and Zhou, 2008] Angelov, P. and Zhou, X. (2008). Evolving fuzzy-rule-based classifiers from data streams. *Fuzzy Systems, IEEE Transactions on*, 16(6) :1462 –1475.

[Boukharouba et al., 2009] Boukharouba, K., Bako, L., and Lecoeuche, S. (2009). Incremental and decremental multi-category classification by support vector machines. In *Machine Learning and Applications, 2009. ICMLA '09. International Conference on*, pages 294 –300.

[Cornuéjols and Miclet, 2002] Cornuéjols, A. and Miclet, L. (2002). *Apprentissage artificiel : concepts et algorithmes.* Algorithmes (Paris). Eyrolles.

[Fortescue et al., 1981] Fortescue, T., Kershenbaum, L., and Ydstie, B. (1981). Implementation of self-tuning regulators with variable forgetting factors. *Automatica*, 17(6) :831 – 835.

[Hunt et al., 1996] Hunt, K., Haas, R., and Murray-Smith, R. (1996). Extending the functional equivalence of radial basis function networks and fuzzy inference systems. *Neural Networks, IEEE Transactions on*, 7(3) :776 –781.

[Kasabov, 2001] Kasabov, N. (2001). Evolving fuzzy neural networks for supervised/unsupervised on-line knowledge-based learning.

[Kasabov and Song, 2002] Kasabov, N. and Song, Q. (2002). Denfis : Dynamic evolving neural-fuzzy inference system and its application for time-series prediction.

[Klinkenberg and Joachims, 2000] Klinkenberg, R. and Joachims, T. (2000). Detecting concept drift with support vector machines. In *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 487–494. Morgan Kaufmann.

[LaViola Jr. and Zeleznik, 2007] LaViola Jr., J. J. and Zeleznik, R. C. (2007). A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(11) :1917–1926.

[Lühr and Lazarescu, 2009] Lühr, S. and Lazarescu, M. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data & Knowledge Engineering*, 68(1) :1 – 27.

[Liavas and Regalia, 1999] Liavas, A. and Regalia, P. (1999). On the numerical stability and accuracy of the conventional recursive least squares algorithm. *Signal Processing, IEEE Transactions on*, 47(1) :88 –96.

[Lughofer, 2008a] Lughofer, E. (2008a). *Evolving fuzzy models : incremental learning, interpretability, and stability issues, applications*. VDM Verlag Dr. Müller.

[Lughofer, 2008b] Lughofer, E. (2008b). Flexfis : A robust incremental learning approach for evolving takagi-sugeno fuzzy models. *Fuzzy Systems, IEEE Transactions on*, 16(6) :1393 –1410.

[Lughofer and Angelov, 2011] Lughofer, E. and Angelov, P. (2011). Handling drifts and shifts in on-line data streams with evolving fuzzy systems. *Appl. Soft Comput.*, 11(2) :2057–2068.

[Maloof and Michalski, 2004] Maloof, M. A. and Michalski, R. S. (2004). Incremental learning with partial instance memory. *Artif. Intell.*, 154(1-2) :95–126.

[Mamdani, 1977] Mamdani, E. (1977). Application of fuzzy logic to approximate reasoning using linguistic synthesis. *Computers, IEEE Transactions on*, C-26(12) :1182 –1191.

[Minku et al., 2009] Minku, F. L., Inoue, H., and Yao, X. (2009). Negative correlation in incremental learning. *Natural Computing : an international journal*, 8 :289–320.

[Moustakides, 1997] Moustakides, G. (1997). Study of the transient phase of the forgetting factor rls. *Signal Processing, IEEE Transactions on*, 45(10) :2468 –2476.

[Polikar et al., 2001] Polikar, R., Udpa, L., Udpa, S., Member, S., Member, S., and Honavar, V. (2001). Learn++ : An incremental learning algorithm for supervised neural networks. *IEEE Transactions on System, Man and Cybernetics (C), Special Issue on Knowledge Management*, 31 :497–508.

[Ramezani et al., 2008] Ramezani, R., Angelov, P., and Zhou, X. (2008). A fast approach to novelty detection in video streams using recursive density estimation. In *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, volume 2, pages 14–2 –14–7.

[Rao Sripada and Grant Fisher, 1987] Rao Sripada, N. and Grant Fisher, D. (1987). Improved least squares identification. *International Journal of Control*, 46(6) :1889–1913.

[Ruping, 2001] Ruping, S. (2001). Incremental learning with support vector machines. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 641 –642.

[Salgado et al., 1988] Salgado, M. E., Goodwin, G. C., and Middleton, R. H. (1988). Modified least squares algorithm incorporating exponential resetting and forgetting. *International Journal of Control*, 47(2) :477–491.

[Syed et al., 1999] Syed, N. A., Liu, H., and Sung, K. K. (1999). Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 317–321, New York, NY, USA. ACM.

[Takagi and Sugeno, 1985] Takagi, T. and Sugeno, M. (1985). Fuzzy Identification of Systems and Its Applications to Modeling and Control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1) :116–132.

[Vahidi et al., 2005] Vahidi, A., Stefanopoulou, A., and Peng, H. (2005). Recursive least squares with forgetting for online estimation of vehicle mass and road grade : theory and experiments. *Vehicle System Dynamics*, 43(1) :31–55(25).

[Viard-Gaudin et al., 1999] Viard-Gaudin, C., Lallican, P. M., Binter, P., and Knerr, S. (1999). The ireste on/off (ironoff) dual handwriting database. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, ICDAR '99, pages 455–, Washington, DC, USA. IEEE Computer Society.

[Zadeh, 1965] Zadeh, L. (1965). Fuzzy sets. *Information Control*, 8 :338–353.