Apprentissage actif en-ligne d'un classifieur évolutif, application à la reconnaissance de commandes gestuelles

Manuel Bouillon

▶ **To cite this version:**

Manuel Bouillon. Apprentissage actif en-ligne d'un classifieur évolutif, application à la reconnaissance de commandes gestuelles. Intelligence artificielle [cs.AI]. INSA de Rennes, 2016. Français. NNT : 2016ISAR0019 . tel-01730548

HAL Id: tel-01730548

https://theses.hal.science/tel-01730548v1

Submitted on 13 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

UNIVERSITE
BRETAGNE
LOIRE

THÈSE INSA Rennes
sous le sceau de l'Université Européenne de Bretagne
pour obtenir le grade de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Manuel Bouillon
ÉCOLE DOCTORALE : MATISSE
LABORATOIRE : IRISA – UMR 6074

Apprentissage actif en-ligne d'un classifieur évolutif, application à la reconnaissance de commandes gestuelles

Thèse soutenue le 18 mars 2016
devant le jury composé de :

Réjean Plamondon
Professeur à l'école Polytechnique de Montréal
Président

Jean-Yves Ramel
Professeur à l'école Polytechnique de l'université de Tours
Rapporteur

Muriel Visani
Maître de conférence HDR à l'université de La Rochelle
Rapporteuse

Vincent Lemaire
Docteur HDR à Orange Labs
Examinateur

Eric Anquetil
Professeur à l'INSA de Rennes
Directeur de thèse

Apprentissage actif en-ligne d'un classifieur évolutif, application à la reconnaissance de commandes gestuelles

Manuel Bouillon

En partenariat avec

Quelle est la différence entre un canard ?
Aucune, surtout la patte avant.

Remerciements

Je remercie tout d'abord Réjean Plamondon, professeur à l'école Polytechnique de Montréal, d'avoir accepté de faire le déplacement pour présider mon jury de thèse. Je remercie également Jean-Yves Ramel, professeur à l'école Polytechnique de l'université de Tour, et Muriel Visani, maître de conférences HDR à l'université de La Rochelle, d'avoir accepté la charge de rapporteur. Je les remercie en particulier d'avoir trouvé le temps de lire ce manuscrit, et ce malgré le délai plus court que je ne l'aurais souhaité. Je remercie de même Vincent Lemaire, docteur HDR à Orange Labs, de bien avoir voulu prendre le temps de venir participer à mon jury de thèse.

Je remercie ensuite Eric Anquetil, professeur à l'INSA de Rennes, pour m'avoir offert l'opportunité de faire cette thèse, et de m'avoir guidé pendant ces trois années. Je suis en particulier reconnaissant pour ses conseils qui m'ont permis d'améliorer toutes mes publications, et merci également pour m'avoir permis d'aller presenter mes travaux aux conférences associées.

Je remercie chaleureusement tous les membres de l'équipe IntuiDoc pour les pauses café, les déjeuners, les séminaires au vert/gris et autres moments partagés. Je remercie en particulier Cérès Carton, ma super collègue de bureau, pour son enthousiasme et tous ces moments partagés pendant ces trois années. Merci pour tout, et plus encore. Je remercie également tous les collègues de l'IRISA avec qui j'ai eu l'occasion d'échanger, en particulier les coureurs de courir@irisa.fr pour toutes ces courses passées à papoter.

Je remercie ma famille, en particulier « la maman de Manuel » pour toutes ses questions et corrections qui m'ont permit d'améliorer ce manuscrit, mais pour le reste aussi. Je remercie tous mes différents colloc avec qui j'ai partagé cette thèse à la maison, mais pas seulement. Je remercie tous les grimpeurs et tous les danseurs avec qui je n'ai pas partagé cette thèse, mais plein d'autre choses, car ça m'était nécessaire pour faire cette thèse !

Je remercie enfin Marcus Liwicki et Rolf Ingold de m'avoir accepté en postdoc à l'université de Fribourg, en Suisse, et ce avant même que je finisse ma thèse !

Sommaire

Introduction

Introduction générale

Avant-propos

Depuis son invention en Mésopotamie il y a plus de six millénaires, l'écriture s'est développée et répandue jusqu'à devenir universelle. D'abord utilisée comme système d'enregistrement, l'écriture est ensuite devenue un outil au service de la langue, créant ainsi le pendant écrit de la communication orale. L'écriture s'est depuis développée comme moyen de communication et d'interaction, toujours omniprésent dans notre société actuelle.

Si la production d'écriture a commencée à être automatisée il y a plus de cinq siècles avec la typographie, la reconnaissance automatique de l'écriture est en revanche très récente. C'est grâce à l'essor des technologies informatiques et des méthodes d'apprentissage automatique que la reconnaissance d'écriture est devenue possible il y a quelques décennies. Ces techniques ont profité des quantités de données disponibles toujours plus importantes pour réussir des prouesses jusque-là inenvisageables. La reconnaissance d'écriture est maintenant une réalité, impressionnante de précision sur texte typographié.

Même si la majorité des textes sont maintenant typographiés, l'écriture manuscrite reste facile et naturelle. Sa reconnaissance représente aujourd'hui encore un défi de par sa grande variabilité d'un scripteur à un autre. Plutôt que l'utilisateur soit obligé de s'adapter au système de reconnaissance, une autre approche est de construire un système de reconnaissance qui va s'adapter spécifiquement à son utilisateur. Dans cette approche, l'apprentissage du système est autant un défi que l'obtention de bonnes performances de reconnaissance. Cette thèse explore les systèmes d'apprentissage et de reconnaissance dans cette direction, qui vont interagir avec l'utilisateur pour apprendre et évoluer à ses cotés.

Contexte applicatif

Avec la démocratisation des interfaces tactiles (montres connectées, *smartphones*, tablettes, écrans, etc.), les interactions Homme-Machine évoluent. Ces nouvelles interfaces tactiles ouvrent de nouveaux champs de recherche pour des interactions plus intuitives et plus naturelles. Plutôt que l'utilisateur soit obligé de s'adapter à la machine, à l'utilisation codifiée du clavier et de la souris, les interactions tendent à devenir tactiles et évolutives. L'utilisateur va alors pouvoir interagir librement et c'est la machine qui va évoluer pour s'adapter aux particularités de son utilisateur. De nouvelles méthodes d'interaction ont été inventées en suivant cette philosophie, et parmi elles, les commandes gestuelles.

De la même manière qu'il existe des raccourcis clavier pour faciliter l'interaction avec un ordinateur disposant d'un clavier, les commandes gestuelles sont des « raccourcis graphiques » qui facilitent les interactions sur interface tactile. Les commandes gestuelles sont des associations entre des symboles/gestes et des commandes, qui permettent à l'utilisateur d'exécuter de nombreuses actions simplement en traçant les symboles associés.

Bien que simplifiant les interactions, les commandes gestuelles impliquent certaines contraintes comme la mémorisation du jeu de symboles, et des associations symboles commandes. Pour mémoriser facilement une douzaine de commandes, il est essentiel de laisser l'utilisateur choisir lui-même ses propres symboles. L'utilisation de commandes gestuelles personnalisées est une situation d'apprentissage croisé. D'un coté l'utilisateur doit apprendre les symboles et mémoriser les associations symboles commandes. De l'autre, le système doit modéliser les symboles et apprendre à les reconnaitre pour exécuter les commandes associées. Cette thèse se situe donc à l'interface des domaines de l'apprentissage automatique, de la reconnaissance d'écriture et de l'interaction avec l'utilisateur.

Le contexte applicatif des commandes gestuelles personnalisées met en exergue un certain nombre de besoins vis-à-vis du système de reconnaissance d'une part, et d'autre part la nécessité d'une méthode de supervision optimisant les interactions avec l'utilisateur.

Tout d'abord, le système d'apprentissage doit être capable d'**apprendre à partir de peu de données**. Pour être personnalisable, le système doit pouvoir apprendre à partir des quelques exemples tracés par l'utilisateur lors de la définition des commandes. Comme il n'est pas envisageable de demander à l'utilisateur de saisir plus de deux ou trois fois chaque symbole, le système doit être capable d'apprendre un modèle de connaissance initial à partir de ces premiers exemples.

Il est ensuite souhaitable que le système continue d'**apprendre pendant son utilisation**, et ce pour deux raisons. Comme le système ne dispose que de peu d'exemples à la création d'une commande gestuelle, apprendre pendant l'utilisation permet d'obtenir davantage d'exemples et d'améliorer le système. De plus, les symboles de l'utilisateur vont évoluer avec le temps, et il faut que le système suive cette évolution.

Il est également nécessaire que l'utilisateur puisse **ajouter de nouvelles commandes gestuelles « à la volée »**, dès qu'il en ressent le besoin pendant son utilisation. Il faut alors que le système soit capable d'apprendre de nouvelles classes à n'importe quel moment de son apprentissage, et pour autant **sans dégrader la reconnaissance** des classes déjà présentes.

Enfin, en plus de ce système de reconnaissance, il est nécessaire de disposer d'un système de supervision de l'apprentissage. Dans le contexte des commandes gestuelles, l'utilisateur est dans la boucle d'apprentissage du système. Le seul moyen d'étiqueter les données pour l'apprentissage est d'interagir avec l'utilisateur, directement ou indirectement. Il faut alors optimiser le processus de supervision pour **minimiser les sollicitations de l'utilisateur** tout en **maximisant l'apprentissage du système**.

Problématique scientifique

Ces travaux se basent sur le contexte applicatif de l'utilisation de commandes gestuelles personnalisées pour soulever deux axes de recherche : celui des classifieurs évolutifs et celui de leur apprentissage actif en-ligne.

Le premier axe de recherche se concentre sur les classifieurs évolutifs sur flux de petite échelle. L'objectif est d'obtenir un **système de classification réactif et dynamique**, qui puisse **apprendre à partir de peu de données**, y compris de **nouvelles classes en cours d'utilisation**, et **suivre indéfiniment toutes les évolutions** des données. Pour cela, il faut que le système soit capable de mettre à jour son modèle de connaissance à l'arrivée de chaque donnée d'apprentissage avec une complexité faible et bornée. Afin de rester évolutif indéfiniment, il est indispensable que l'algorithme d'apprentissage **intègre de l'oubli** pour maintenir le gain de l'apprentissage au cours du temps, et oublier les connaissances devenues obsolètes lors de l'évolution des données.

Le deuxième axe de recherche vise quant à lui à **optimiser la supervision de l'apprentissage** d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage. Dans ce cas, l'étiquetage des données est coûteux, puisqu'il ne peut être réalisé sans erreur que par l'utilisateur lui-même. Il faut alors choisir soigneusement les données qui seront utilisées pour l'apprentissage, afin de minimiser les sollicitations de l'utilisateur, tout en maximisant l'apprentissage du classifieur. Pour cela, il est nécessaire d'évaluer l'étendue et la qualité du modèle de connaissance du classifieur, afin de sélectionner les données qui seront les plus bénéfiques à son apprentissage.

L'originalité de cette thèse est le fait d'apprendre à partir de très peu de données, et avec l'utilisateur dans la boucle d'apprentissage. Contrairement à tous les classifieurs conçus pour travailler à grande échelle sur des flux de données de très grandes tailles, il nous faut ici un classifieur beaucoup plus rapide, et avec une grande réactivité. La présence de l'utilisateur dans la boucle implique d'avoir une méthode de supervision spécifique pour optimiser la coopération entre le système et l'utilisateur.

Contributions de cette thèse

Cette thèse présente deux contributions principales : le classifieur évolutif en-ligne « ***Evolve*** ∞ » (textit(Evolve infinity), et le superviseur actif en-ligne « ***IntuiSup*** » *(Intuitive Supervisor)*. Ces deux systèmes ont été développés en synergie mais sont complètement indépendants. Le classifieur *Evolve* ∞ peut être utilisé pour n'importe quelle tâche d'apprentissage, avec d'autres systèmes de supervision. De même, le superviseur *IntuiSup* peut être utilisé pour superviser d'autres systèmes de classification évolutifs lorsque l'utilisateur est dans la boucle d'apprentissage.

Le classifieur évolutif « ***Evolve*** ∞ » *(Evolve infinity)* est une évolution du système d'inférence floue *Evolve*, qui le rend notamment capable d'apprendre et d'évoluer indéfiniment. *Evolve* est un système d'inférence floue d'ordre un – dit de Takagi-Sugeno – qui a été développé par le passé dans l'équipe de recherche IntuiDoc (laboratoire IRISA, INSA Rennes, France).

L'utilisation d'une architecture sous la forme d'un ensemble de règles permet d'apprendre facilement de nouvelles informations et de **nouvelles classes**, sans pour autant détériorer la connaissance existante. *Evolve* ∞ est capable d'**apprendre rapidement** grâce aux capacités génératrices des prémisses des règles, tout en permettant d'obtenir une **précision élevée** grâce aux capacités discriminantes des conclusions d'ordre un. Son algorithme d'apprentissage incrémental d'une faible complexité lui permet d'être facilement appris en-ligne.

Evolve ∞ intègre notamment de l'oubli dans son processus d'apprentissage. Cela permet de maintenir le gain de l'apprentissage indéfiniment et ainsi de garantir son **évolutivité « à vie »**. Cette capacité d'oubli lui permet également de s'adapter aux changements de concepts du flux de données en oubliant les anciennes données lorsqu'elles deviennent obsolètes.

Evolve ∞ dispose également d'une capacité de rejet des données, afin de limiter les erreurs de reconnaissance, qui est également évolutive. Elle rassemble une option de rejet de distance et une option de rejet de confusion à seuils multiples. Le rejet de distance permet de rejeter les données qui ne correspondent pas aux exemples appris jusque-là. Le rejet de confusion permet de **limiter les erreurs** de reconnaissance du classifieur lorsque des données semblent correspondre à plusieurs classes. Ces capacités de rejet sont également apprises en-ligne, afin de les faire évoluer en association avec le système de reconnaissance.

Le superviseur actif en-ligne « ***IntuiSup*** » *(Intuitive Supervisor)* est un système permettant de superviser l'apprentissage d'un classifieur évolutif avec l'utilisateur dans la boucle. Il est lui-même évolutif afin de s'adapter au système qu'il supervise et à son environnement.

Il permet de sélectionner activement les données qui sont les plus intéressantes pour l'apprentissage du classifieur, pour les faire étiqueter par l'utilisateur. Pour cela, il se base sur une mesure de confiance du classifieur lors de la reconnaissance des données, pour évaluer leur intérêt par rapport au modèle de connaissance actuel du système. Ce processus fonctionne de manière similaire à une option de rejet, en sélectionnant les données pour lesquelles le classifieur est le plus incertain.

IntuiSup est un superviseur évolutif, qui va s'adapter à l'apprentissage du classifieur supervisé. Comme l'utilisateur est dans la boucle d'apprentissage, les erreurs de reconnaissance et l'étiquetage des données pour l'apprentissage ont un impact fort en terme d'interaction avec l'utilisateur. La quantité de données étiquetées va donc être adaptée à l'évolution du classifieur pour maintenir le compromis erreur/rejet choisi au cours du temps.

L'utilisation d'une méthode de dopage *(boosting)* de l'apprentissage permet d'optimiser la répartition de ces interactions avec l'utilisateur pour maximiser leur impact sur l'apprentissage. *IntuiSup* permet de faire évoluer la proportion de données qui sont étiquetées en fonction de la difficulté du problème et de l'évolution de l'environnement. Cela permet d'accélérer l'évolution du classifieur supervisé, à la fois lors de l'apprentissage initial et lors des changements de concepts.

Description par chapitre

Ce manuscrit est organisé en huit chapitres, qui sont répartis dans trois parties.

La première partie regroupe les chapitres introductifs qui présentent l'état de l'art et les fondements de cette thèse. La littérature est passée en revue et les travaux connexes de l'état de l'art sont présentés afin de justifier et de bien positionner ceux que nous allons présenter ensuite. Les travaux qui ont servi de base à cette thèse seront en particulier détaillés afin de bien mettre en évidence leurs limites et la nécessité de leur approfondissement.

- **Le premier chapitre** présente et clarifie plusieurs définitions de l'état de l'art qui sont nécessaires à la bonne compréhension de cette thèse. Il précise le contexte applicatif de ces travaux : l'utilisation de commandes gestuelles personnalisées et les besoins que cela implique, tant en matière de système de classification que de supervision. Plusieurs typologies sont établies, en partant des différents problèmes d'apprentissage automatique *(machine learning)*, puis en détaillant les différents types de classifieurs pouvant être utilisés pour y répondre. Une typologie des différents types d'apprentissage pouvant être mis en œuvre pour entraîner ces classifieurs est également présentée, ainsi que celle des différents types de supervision possibles pour cet apprentissage. Enfin, les différents scénarios d'apprentissage actif et les différents types de rejet sont clarifiés pour une meilleure compréhension de cette thèse.
- **Le deuxième chapitre** expose les principes et les méthodes de supervision active en-ligne de l'apprentissage *(active learning)*. Le fait que l'utilisateur soit dans la boucle d'apprentissage implique que la supervision soit effectuée en temps réel, donnée par donnée. Différentes méthodes d'échantillonnage pour flux de données sont présentées, en particulier la méthode de l'incertitude maximale *(uncertainty sampling)*. L'utilisation de l'option de rejet du classifieur rentre dans cette catégorie et différentes architectures d'option de rejet sont donc détaillées. Les principes de l'apprentissage semi-supervisé sont également exposés, afin de pouvoir tirer parti des données non-étiquetées par le processus de supervision.
- **Le troisième chapitre** présente un panorama des différents classifieurs incrémentaux qui existent dans la littérature. Nous nous attachons en particulier à définir dans quelle mesure ils sont évolutifs. Si de nombreux classifieurs peuvent être appris en-ligne *(online learning)*, un certain nombre ne sont pas évolutifs pour autant. Leur algorithme d'apprentissage est incrémental en temps réel, mais le gain de l'apprentissage diminue avec le temps. Ces classifieurs voient alors leur modèle de connaissance converger avec le temps, et leur capacité d'adaptation aux changements du flux de données disparaître. Un intérêt particulier est donc porté à la capacité d'oubli de ces classifieurs, pour maintenir le gain de l'apprentissage dans le temps et leur permettre de suivre les changements de concepts *(concept drifts)*.
- **Le quatrième chapitre** décrit en détails le classifieur incrémental *Evolve*, qui a servi de base à ces travaux. Son architecture et son algorithme d'apprentissage sont présentés en détails. C'est un système d'inférence floue dont la structure est évolutive, et dont les paramètres sont appris incrémentalement. Les limites d'*Evolve* sont

mises en évidence, notamment vis-à-vis de l'ajout de classes en cours d'utilisation et du suivi des changements de concepts. Différentes alternatives existantes sont également étudiées.

La deuxième partie présente les contributions de cette thèse, à savoir le classifieur évolutif *Evolve* ∞ et le superviseur actif en-ligne *IntuiSup*. Bien que développées en synergie, ces deux contributions sont complètement indépendantes. Le classifieur évolutif *Evolve* ∞ peut tout à fait être utilisé pour d'autres tâches d'apprentissage statistique. De même, le superviseur *IntuiSup* peut également servir à superviser l'apprentissage actif en-ligne d'autres classifieurs évolutifs. Le couplage du classifieur et du superviseur s'effectue grâce à la fonction de confiance du classifieur. Celui-ci doit donc être capable d'évaluer les limites de son modèle de connaissance. Le superviseur *IntuiSup* met alors à profit les informations fournies par la mesure de confiance du classifieur pour adapter l'apprentissage aux difficultés du système et à l'évolution de son environnement.

— **Le cinquième chapitre** présente les améliorations et spécificités du classifieur évolutif *Evolve* ∞ par rapport au classifieur incrémental *Evolve*. L'intégration d'oubli dans le processus d'apprentissage du système est notamment présenté, afin de permettre au système de rester évolutif indéfiniment. Cette capacité d'oubli le rend ainsi capable d'apprendre de nouvelles classes à n'importe quel moment de son utilisation et de suivre les changements de concepts du flux de données. *Evolve* ∞ dispose également d'une capacité d'auto-évaluation de la qualité de son modèle, qui lui permet d'évaluer les limites de ses connaissances. Cette capacité d'évaluation permet d'obtenir la fonction de confiance qui est utilisée par *IntuiSup* pour superviser l'apprentissage lorsque l'utilisateur est dans la boucle. Elle permet également d'obtenir une option de rejet évolutive pour réduire son taux d'erreur dans les contextes où les mauvaises reconnaissances sont coûteuses.

— **Le sixième chapitre** décrit le superviseur *IntuiSup* que nous avons créé pour superviser l'apprentissage actif en-ligne d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage. Il permet d'optimiser la coopération entre l'utilisateur et le système en minimisant les sollicitations tout en maximisant leur impact sur l'apprentissage. Ce superviseur est évolutif et permet de maintenir au cours du temps le compromis erreur/sollicitation choisi, en s'adaptant à l'évolution du classifieur et aux changements du flux de données. Il permet également de doper l'apprentissage en ajustant la répartition des sollicitations pour les concentrer aux moments où elles sont les plus importantes : pendant l'apprentissage initial et durant les changements de concepts.

La troisième partie donne les résultats expérimentaux obtenus avec le classifieur *Evolve* ∞ et le superviseur *IntuiSup*. Différents scénarios de test permettent d'évaluer et de valider expérimentalement leurs capacités et leurs performances. Leur sensibilité au contexte applicatif, leur paramétrage et leur positionnement par rapport à l'état de l'art sont également évalués dans différentes situations d'apprentissage.

— **Le septième chapitre** présente les performances obtenues avec *Evolve* ∞ pour

la reconnaissance de gestes manuscrits, et plus généralement pour différents problèmes de classification de l'état de l'art. *Evolve* ∞ est ainsi comparé et positionné, en matière de rapidité d'apprentissage et de taux de reconnaissance à convergence, par rapport à différents systèmes de l'état de l'art. Ses capacités d'apprentissage et d'évolution sont ensuite mises en avant, au travers de l'ajout de nouvelles classes en cours d'utilisation. L'apprentissage et la reconnaissance de flux de données comportant des changements de concepts met en évidence l'intérêt de la capacité d'oubli d'*Evolve* ∞. L'efficacité de sa capacité d'auto-évaluation et de rejet est également mise en avant lors de l'apprentissage à partir de très peu de données.

— **Le huitième chapitre** montre l'efficacité du superviseur évolutif *IntuiSup* pour réaliser l'apprentissage actif en-ligne d'un classifieur évolutif. Son application à l'apprentissage de commandes gestuelles confirme sa capacité à optimiser les interactions avec l'utilisateur pour superviser l'apprentissage lorsque l'utilisateur est dans la boucle, en tirant parti de la capacité d'auto-évaluation du classifieur. Ses capacités d'évolution sont ensuite évaluées pour s'adapter à l'évolution du classifieur et aux changements du flux de données, et pour maintenir le compromis erreur/rejet choisi au cours du temps. Enfin, l'intérêt de la répartition et du dopage *(boosting)* des interactions est mis en évidence pour accélérer l'apprentissage initial du classifieur et son adaptation aux changements de concepts.

La conclusion générale termine ce manuscrit, en synthétisant les contributions présentées – le classifieur *Evolve* ∞ et le superviseur *IntuiSup* – ainsi que leurs implications. Les différentes perspectives ouvertes par cette thèse sont également discutées.

Première partie

État de l'art

Chapitre 1

Contexte et définitions

1.1 Introduction

L'utilisation de commandes gestuelles est un nouveau paradigme d'interaction sur interface tactile. Il consiste à associer des commandes à des symboles afin de faciliter leur utilisation. Pour plus de flexibilité, il est souhaitable que le jeu de symboles soit personnalisable, et que le système s'adapte à l'utilisateur final et à son évolution. Ce contexte applicatif induit une situation d'apprentissage croisé, où l'utilisateur doit mémoriser le jeu de symboles et le système doit apprendre à les reconnaître. Les implications de cette situation d'apprentissage croisé sont détaillées dans la section 1.2.

Pour l'utilisation de commandes gestuelles, plusieurs types de tracés existent et peuvent être utilisés. Ils sont donc détaillés en section 1.3.

La reconnaissance de symboles, comme dans le cas des commandes gestuelles, est un problème de classification, où il faut reconnaître l'étiquette qui doit être associée à des données inconnues. Pour les problèmes de classification, un système de reconnaissance est entraîné par apprentissage automatique *(machine learning)* à partir d'une base d'exemples. Nous présentons également les autres types de problème d'apprentissage automatique dans la section 1.4, dont certains aspects et solutions peuvent être similaires à ceux des problèmes de classification.

Les systèmes d'apprentissage et de reconnaissance – appelés classifieurs – sont composés de plusieurs éléments qui sont présentés sur la figure 1.1. Ces différents éléments peuvent être de différents types suivant les besoins et les objectifs souhaités. Dans le contexte de la reconnaissance de commandes gestuelles, il est nécessaire que le système d'apprentissage et de reconnaissance soit dynamique. Le système doit pouvoir apprendre et évoluer pendant son utilisation, avec l'utilisateur dans la boucle d'apprentissage.

Les classifieurs sont basés sur un algorithme de reconnaissance, qui utilise un modèle de connaissance pour reconnaître les données inconnues. Nous établissons donc une taxonomie des modèles de connaissance sur lesquels peut être basé un classifieur dans la section 1.5. Nous introduisons ensuite une taxonomie des algorithmes d'apprentissage qui peuvent être utilisés pour créer le modèle de connaissance du classifieur dans la section 1.6. Nous présentons enfin une taxonomie des méthodes de supervision possibles de l'algorithme

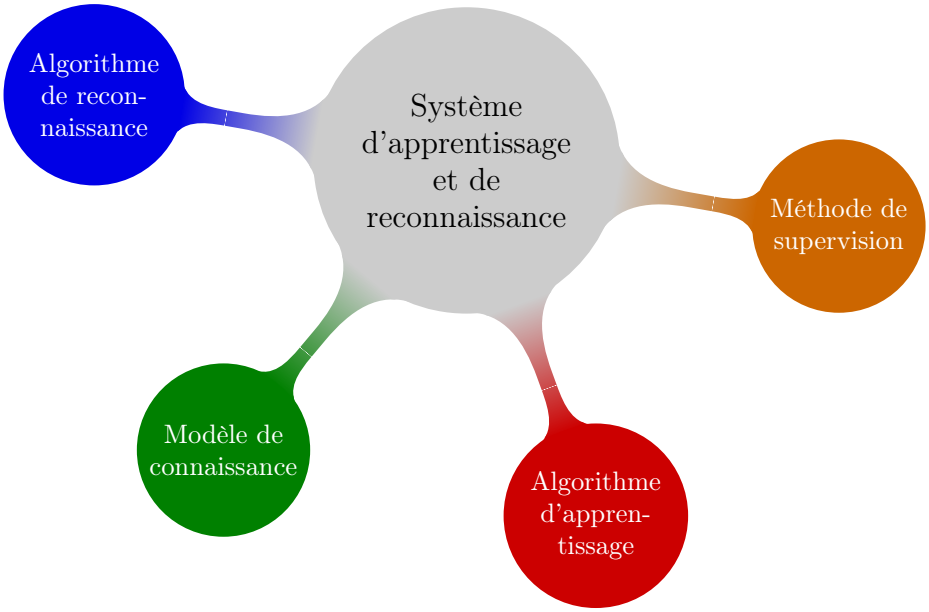


FIGURE 1.1 – Éléments constitutifs d'un système d'apprentissage et de reconnaissance.

d'apprentissage dans la section 1.7.

1.2 Contexte et problématique

Cette section présente le contexte applicatif de ces travaux, et les différentes contraintes qui en découlent. Ces travaux ont pour but la définition et l'apprentissage d'un classifieur permettant l'utilisation de commandes gestuelles. Ce contexte d'utilisation induit un certain nombre de contraintes que nous détaillons dans cette section :

— apprendre à partir de peu de données pour avoir un système personnalisable ;
— continuer à apprendre pendant l'utilisation du système pour améliorer ses performances et s'adapter à l'évolution de l'utilisateur ;
— interagir à bon escient avec l'utilisateur pour superviser l'apprentissage ;
— pouvoir oublier quand cela est nécessaire pour évoluer indéfiniment.

1.2.1 Reconnaissance de commandes gestuelles

Avec la démocratisation des écrans tactiles, les interactions Homme-Machine évoluent. De nouvelles méthodes d'interaction ont été inventées pour tirer parti du nouveau potentiel d'interaction offert par ces nouvelles interfaces. Parmi elles, un nouveau concept est apparu récemment : associer des commandes à des gestes/symboles. Ces commandes gestuelles [1] [Wobbrock et al., 2009] [Yang et al., 2011] permettent à l'utilisateur d'exécuter

1. Voir `http://youtu.be/qOx4IY6uYf8` pour une démonstration de commandes gestuelles

de nombreuses actions simplement en traçant les symboles associés. Pour mémoriser facilement une douzaine de commandes, il est essentiel de laisser l'utilisateur choisir lui-même ses propres gestes [Li 13]. L'utilisation de commandes gestuelles requiert un système de reconnaissance de symboles manuscrits. De plus, pour que les commandes gestuelles soient personnalisables, ce classifieur doit être capable d'apprendre à partir de peu de données et d'évoluer pendant son utilisation.

L'utilisation de commandes gestuelles est un problème de classification, où il faut reconnaître des symboles inconnus. Si la reconnaissance de symboles manuscrits est un domaine largement étudié, le contexte des commandes gestuelles implique certaines contraintes particulières, que nous allons détailler.

1.2.2 Apprentissage à partir de peu de données

L'utilisation de commandes gestuelles donne lieu à une situation d'apprentissage croisé. L'utilisateur doit mémoriser les différents gestes, et apprendre à quelles commandes ils sont associés. Le système doit de son côté apprendre à reconnaître les différents symboles.

Mémoriser plus d'une douzaine de commandes est une tâche non triviale pour les utilisateurs. Un bon moyen pour faciliter la mémorisation des symboles, et des associations symboles-commandes, est de permettre aux utilisateurs de définir eux-mêmes leur propre jeu de symboles, et de choisir quel symbole est associé à quelle commande [Li et al., 2012].

Le système de reconnaissance doit alors être capable d'apprendre à partir de peu de données. En effet, il est difficilement envisageable de demander à l'utilisateur de saisir plus de deux ou trois exemples de chaque symbole. Par conséquent, le classifieur doit être capable d'apprendre un modèle de connaissance, avec peu de données, qui soit suffisamment robuste pour permettre son utilisation.

De plus, permettre aux utilisateurs de personnaliser leur jeu de commandes donne lieu à une très grande diversité de formes à reconnaître. La figure 1.2 illustre la diversité et l'originalité des symboles qui peuvent être obtenues lors de la personnalisation de commandes gestuelles. Avec cette diversité vient également le problème des symboles similaires, ou mal différenciés par le classifieur. Il est préférable de guider les utilisateurs lors de l'étape de définition du jeu de symboles, en donnant un retour sur le degré de confusion du classifieur entre les différents symboles choisis. Ce guidage des utilisateurs nécessite une capacité d'auto-évaluation de la part du classifieur, qui soit utilisable à un stade très précoce de l'apprentissage, comme celle présentée dans la section 5.5.3. Son efficacité pour guider le choix du jeu de gestes des utilisateurs est montré en section 7.6.

1.2.3 Apprentissage pendant l'utilisation du système

Puisque le système ne dispose que de quelques exemples pour inférer son modèle initialement, celui-ci est forcément limité. Il est donc nécessaire que le système continue d'apprendre pendant son utilisation pour améliorer ses performances.

Qui plus est, il est également souhaitable que le système s'adapte de manière fine à l'utilisateur. En effet, l'utilisateur novice va sans doute tracer ses symboles avec lenteur et précision, mais va progressivement accélérer. Au bout d'un certain temps, l'utilisateur

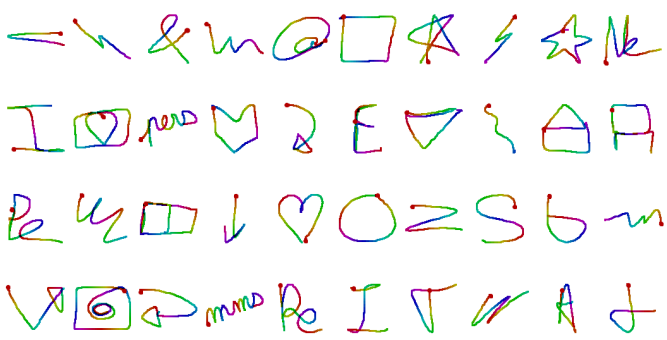


FIGURE 1.2 – Exemples de symboles inventés par les scripteurs du groupe 1 de la base de données de commandes gestuelles personnalisées ILGDB [Renau-Ferrer et al., 2012].

confirmé va ainsi tracer ses symboles de plus en plus rapidement et, à la manière d'une signature, les tracés seront de plus en plus déformés. Il faut alors que le système soit capable de s'adapter, de suivre l'évolution de l'utilisateur, et non pas que l'utilisateur se restreigne pour s'adapter à un système statique.

Il faut donc un système dynamique avec un algorithme d'apprentissage incrémental, pour pouvoir mettre à jour le modèle de connaissance du classifieur. De plus, cette mise à jour doit être faite entre deux utilisations du système. L'algorithme d'apprentissage doit donc être incrémental en temps réel, c'est-à-dire être « en-ligne », comme nous allons le définir en section 1.6.

1.2.4 Interactions avec l'utilisateur

L'algorithme d'apprentissage d'un classifieur est, la majorité du temps, supervisé : les données d'apprentissage sont étiquetées, chaque symbole est associé à une commande gestuelle. Il est donc nécessaire d'interagir avec l'utilisateur pour superviser l'apprentissage du système pendant son utilisation. Cependant, solliciter trop souvent l'utilisateur pour superviser l'apprentissage peut vite rendre l'utilisation du système de commandes gestuelles pénible.

D'un côté, la supervision est gênante pour l'utilisateur, mais d'un autre coté cette supervision est nécessaire pour améliorer le classifieur, et donc réduire son taux d'erreur. Or les erreurs sont également coûteuses puisqu'elles obligent l'utilisateur à intervenir pour les corriger, puis à re-tenter les commandes qui ont été mal reconnues. Il y a donc un compromis à trouver dans les interactions avec l'utilisateur, entre les interactions de supervision et celles de correction, comme nous le montrons en section 2.4.3.

1.2.5 Nécessité d'oublier

Nous avons vu qu'il est nécessaire que le système s'adapte à l'utilisateur, et non l'inverse ! Lorsque le style d'écriture de l'utilisateur évolue, la connaissance issue des anciennes données, correspondant à l'ancien style d'écriture, n'est plus pertinente. Il est alors inté-

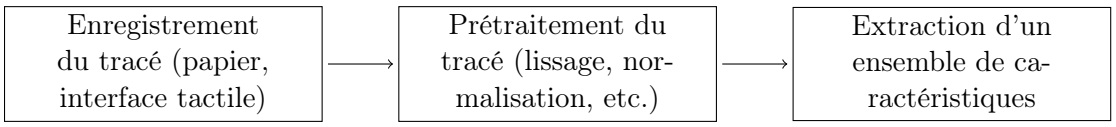


FIGURE 1.3 – Processus d'acquisition des données.

ressant de pouvoir l'oublier pour pouvoir mieux apprendre le nouveau style d'écriture, et la nouvelle connaissance associée.

Le système évolue, notamment pour s'adapter à l'utilisateur, et met à jour son modèle de connaissance. De manière plus générale, afin de pouvoir continuer à évoluer, à apprendre indéfiniment, il est forcément nécessaire d'oublier les connaissances les plus anciennes. Si le système attribue un poids relatif équivalent à toutes les données, alors ce poids va décroitre progressivement au fur et à mesure que la quantité de données va augmenter. La dixième donnée se verra attribuer un poids d'un dixième, la centième donnée un poids d'un centième, et la millième un poids d'un millième. La millième donnée n'aura alors plus beaucoup d'effet sur l'apprentissage du système. Il est donc nécessaire de limiter le poids des anciennes données, c'est-à-dire disposer d'un mécanisme d'oubli.

L'intégration d'oubli dans le processus d'apprentissage est une des principales contributions de cette thèse, comme nous le présentons chapitre 5.

1.3 Typologie de tracés manuscrits

Ces travaux portent sur la reconnaissance de commandes gestuelles, qui sont un nouveau moyen d'interaction où des symboles sont associés à des commandes. Les symboles manuscrits en général peuvent avoir différentes caractéristiques. Ils peuvent être soit en-lignes soit hors-lignes, suivant le mode d'acquisition, et mono-strokes, multi-strokes ou encore multi-points.

Une fois l'acquisition des symboles réalisée, il est nécessaire d'en extraire un vecteur de caractéristiques qui puisse être utilisé par le système d'apprentissage. Ces caractéristiques *(features)*, aussi appelées descripteurs ou attributs, caractérisent différents aspects des tracés afin de permettre leur apprentissage et leur reconnaissance. La figure 1.3 synthétise le processus d'acquisition des données.

L'étape d'extraction des caractéristiques est hors du champ de cette thèse. Nous utilisons donc le jeu de caractéristiques standard et universel HBF49 [Delaye and Anquetil, 2013]. Cela permet de garantir de bonnes performances pour la reconnaissance de symboles personnalisés. Il est en effet impossible de prévoir à l'avance les symboles que l'utilisateur va définir, d'où l'intérêt d'un jeu de caractéristiques polyvalent. L'utilisation de ce jeu de caractéristiques standard permet de fournir une référence pour comparer les performances de différents systèmes de reconnaissance.

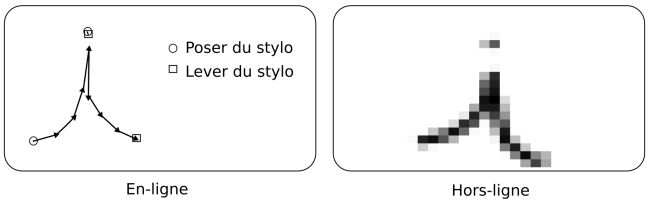


FIGURE 1.4 – Différence entre tracés en-lignes et tracés hors-lignes. Un tracé en-ligne est une série temporelle de points dans l'espace. Un tracé hors-ligne est une matrice de pixels. – Illustration : Wikipédia.

1.3.1 Tracés hors-lignes

Le terme symbole hors-ligne, ou plus généralement tracé hors-ligne, est utilisé lorsque seule l'image du tracé est disponible. C'est notamment le cas lors de la reconnaissance de documents imprimés, ou de textes manuscrits, qui ont été numérisés.

Dans ce cas, le système d'apprentissage et de reconnaissance dispose en entrée de l'image du tracé, au sens d'un tableau de pixels. Le plus souvent, des caractéristiques en sont extraites plutôt que de travailler directement sur ce tableau de pixels. Ces caractéristiques sont alors elles aussi hors-ligne dans le sens ou elles caractérisent l'image du tracé. Les informations dynamiques telles que le sens ou la vitesse du tracé ne sont pas disponibles dans le cas de tracés hors-lignes.

Le terme tracés hors-lignes caractérise seulement le type de symboles utilisés en entrée du système. Ce type de symboles n'a aucun rapport avec le fait que l'apprentissage soit hors-ligne ou en-ligne (voir section 1.6).

1.3.2 Tracés en-lignes

Le terme tracés en-ligne est utilisé lorsque la trajectoire du tracé est disponible. C'est le cas lors de la reconnaissance d'écriture sur interface tactile (smartphone, tablette, etc.). La figure 1.4 illustre la différence entre un tracé en-ligne et un tracé hors-ligne.

Dans le cas des tracés en-lignes, le système d'apprentissage dispose en entrée de la trajectoire du tracé, au sens d'une suite temporelle de points. Les caractéristiques extraites peuvent alors être en-ligne, si elles tirent parti des informations dynamiques du tracé. Il est également possible d'extraire des caractéristiques hors-lignes à partir des tracés en-lignes, si les aspects dynamiques ne sont pas utilisés.

On utilise toujours le terme « symbole » lorsque les tracés sont hors-lignes. Dans le cas en-ligne, le terme « geste » est aussi parfois employé puisque la dynamique est prise en compte, et que la manière de réaliser le tracé compte autant que le symbole obtenu.

Il faut noter que tracés en-lignes et apprentissage en-ligne (voir section 1.6) sont deux choses distinctes et sans rapport.

(a) Zoom multi-point command (b) Rotate multi-point command

FIGURE 1.5 – Exemples de tracés multi-points – Illustration : Wikipedia.

1.3.3 Tracés mono-strokes

Le terme mono-strokes désigne des tracés qui sont chacun constitués d'un seul trait continu. C'est le cas de la plupart des tracés manuscrits des lettres minuscules cursives, qui sont composées d'un seul et unique trait.

C'est également le cas de la majorité des systèmes de commandes gestuelles, et en particulier de la base de données ILGDB [Renau-Ferrer et al., 2012] que nous utilisons pour valider expérimentalement ces travaux (voir section 7.2.2.1).

1.3.4 Tracés multi-strokes

Le terme multi-stroke désigne quant à lui un tracés composé de plusieurs segments. C'est le cas de la plupart des tracés manuscrits des lettres en capitales d'imprimeries, qui sont composées de plusieurs traits.

L'utilisation de symboles multi-strokes dans un système de commandes gestuelles est possible, mais il implique une certaine latence dans l'acquisition. Il faut alors attendre un petit laps de temps après chaque levé de stylo pour laisser la possibilité à l'utilisateur d'ajouter des strokes à son symbole.

1.3.5 Tracés multi-points

Les tracés multi-points sont des symboles multi-strokes en-ligne, où les différents traits *(strokes)* sont tracés en même temps, avec plusieurs doigts [Chen et al., 2014] [Chen et al., 2015]. Il faut prendre en compte la synchronisation des différents traits *(strokes)* qui peuvent être tracés en parallèle, et non juste en séquence.

C'est un type de tracé spécifique aux interfaces tactiles, qu'il est difficile de retranscrire sur papier. Les exemples de tracés multi-points les plus courants sont les commandes de zoom et de rotation, qui sont rappelées figure 1.5, sur *smartphones* et tablettes tactiles.

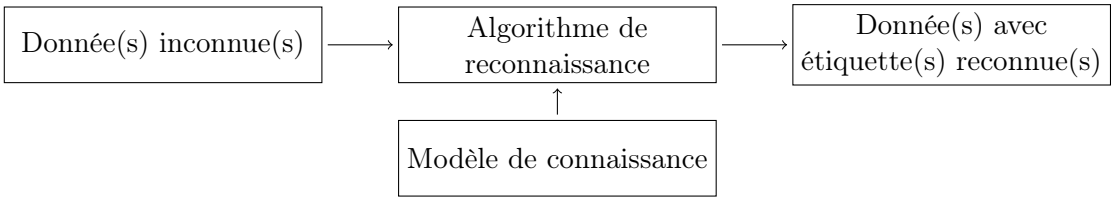


FIGURE 1.6 – Processus de reconnaissance des données.

1.4 Typologie des problèmes d'apprentissage

Cette section présente les principaux types de problèmes d'apprentissage automatique – aussi appelé apprentissage artificiel [Cornuéjols and Miclet, 2010] – qui existent :

- les problèmes de classification ;
- les problèmes de régression ;
- les problèmes de regroupement *(clustering)* ;
- les problèmes de détection d'erreurs *(outliers)* ;
- les problèmes d'inférence de règles.

La reconnaissance de commandes gestuelles est un problème de classification, de catégorisation de données inconnues. Cependant, nous verrons que certaines approches et techniques utilisées pour d'autres types de problèmes peuvent être intéressantes et utiles dans notre cas.

1.4.1 Problème de classification

Un problème de classification est un problème d'étiquetage de données inconnues. Une donnée peut prendre différentes formes suivant le contexte. Dans le cas des systèmes d'apprentissage statistique, une donnée est un point en dimension n, c'est-à-dire un vecteur de n valeurs, associé à une étiquette, appelée classe, qui est une commande gestuelle dans notre cas. Dans d'autres cas, les données peuvent prendre la forme de suites temporelles, également associées à des classes. Ce sont ces classes que le système doit apprendre à reconnaître.

Le système doit modéliser les différentes classes pour pouvoir ranger, classer, toute nouvelle donnée dans une de ces classes. La classe d'une donnée est le plus souvent unique, mais il existe également des problèmes de classification multi-classes, où chaque donnée peut appartenir à plusieurs classes en même temps.

Un système de reconnaissance est appris sur une base de données d'apprentissage, dont les données sont étiquetées. Ensuite, le système peut reconnaître des données inconnues, leur attribuer une étiquette, les classer dans une catégorie comme représenté figure 1.6.

Notre problème de reconnaissance de commandes gestuelles est un problème de classification. Il faut en effet reconnaître la classe, l'étiquette, des commandes dessinées par l'utilisateur.

1.4.2 Problème de régression

Un problème de régression est un problème de prédiction de valeurs inconnues. Les données sont également des points en dimension n, mais elles sont cette fois associées à une valeur numérique (et non plus une classe).

Le système doit modéliser la fonction donnant la relation entre les données en entrée et les valeurs en sortie.

Le système de reconnaissance est également appris sur une base d'apprentissage étiquetée. Le système peut ensuite prédire les valeurs associées à des données inconnues.

La reconnaissance de commandes gestuelles n'est pas un problème de régression, mais un problème de classification. Cependant, les méthodes et systèmes utilisés pour des problèmes de régression peuvent souvent être adaptés et transférés aux problèmes de classification (comme c'est le cas pour les systèmes d'inférence floue [Angelov and Zhou, 2008] par exemple).

1.4.3 Problème de regroupement *(clustering)*

Un problème de regroupement est un problème de partitionnement de données inconnues, sans référence.

Le système doit créer des groupes de données, en rassemblant les données les plus similaires, tout en ayant des groupes les plus différents possibles. Dans le cas général, les données n'ont pas d'étiquettes, et il n'y a pas de base de données d'apprentissage. Dans certains cas, le nombre de groupes recherchés est connu, dans d'autres cas c'est à l'algorithme de le déterminer en fonction des données.

La reconnaissance de commandes gestuelles n'est pas un problème de regroupement. Toutefois, certains systèmes de classification, comme les réseaux de neurones RBF *(Radial Basis Function)* [Pedrycz, 1998] et les systèmes d'inférence floue [Chiu, 1994], peuvent utiliser des algorithmes de regroupement lors de leur apprentissage, pour définir la structure du système.

1.4.4 Problème de détection d'erreurs/de nouveautés

Un problème de détection d'erreurs, aussi appelé détection de nouveautés, est un problème de détection de données aberrantes par rapport à la distribution des données, ou de détection de changements dans celle-ci.

Le système doit modéliser la distribution des données « normales ». Le système doit ensuite être capable de détecter les données aberrantes par rapport à cette distribution. De la même manière, le système doit détecter tout changement dans la distribution des données.

La reconnaissance de commandes gestuelles reste un problème de classification, mais la détection d'erreurs peut néanmoins être utile. Des techniques de détection de nouveautés peuvent être utilisées pour détecter les symboles ne correspondant à aucune classe. La détection de nouveautés peut aussi servir pour détecter des évolutions dans le style d'écriture de l'utilisateur – des changements de concepts (voir section 1.6.2.3) – pour s'y adapter.

1.4.5 Problème d'inférence de règles

Il existe deux principaux types de problèmes d'inférence de règles : l'inférence de règles d'associations et l'inférence de règles grammaticales.

Un problème d'inférence de règles d'associations est un problème de fouille de données. Le but est trouver des relations de corrélation entre plusieurs variables, qui sont fréquemment présentes ensemble. L'inférence de règles d'associations se fait à partir des grandes bases de données, dans lesquelles une structure est recherchée.

Un problème d'inférence grammaticale est un problème d'induction d'une description formelle d'un langage. Une grammaire est une description formelle d'un langage, c'est-à-dire d'un ensemble de séquences. L'inférence grammaticale consiste donc à retrouver la grammaire la plus simple qui a pu engendrer un ensemble d'échantillons positifs (données de la classe qui est modélisée). Un ensemble d'échantillons négatifs est souvent utilisé pour obtenir une grammaire suffisamment précise.

1.5 Taxonomie de modèles de connaissance des classifieurs

Cette section définie une taxonomie des modèles de connaissance des systèmes de classification, présenté figure 1.7, suivant le caractère dynamique de leur modèle de connaissance. Nous distinguons ainsi :

— les systèmes avec modèle statique ;
— les systèmes avec modèle adaptatif ;
— les systèmes avec modèle évolutif.

1.5.1 Classifieur à modèle statique

Un classifieur est un système permettant de reconnaître des données inconnues, à partir d'un modèle de connaissances. Ce modèle est créé par un algorithme d'apprentissage à partir des données d'entrainement. Le classifieur utilise ensuite ce modèle pour reconnaître les données inconnues pendant l'utilisation du système.

Le plus souvent, le modèle du classifieur est statique, c'est-à-dire que ce modèle n'est pas modifiable pendant l'utilisation du système. Si de nouvelles données deviennent disponibles, il est nécessaire de ré-apprendre un nouveau modèle depuis zéro pour pouvoir en tirer profit. De même, si certaines données utilisées lors de l'apprentissage du modèle deviennent obsolètes, il n'est pas possible de les oublier, il est nécessaire de ré-apprendre un nouveau modèle.

Plus de flexibilité requiert un classifieur avec un modèle dynamique, c'est-à-dire un classifieur adaptatif ou évolutif.

1.5.2 Classifieur à modèle adaptatif

Un classifieur adaptatif est un système dont le modèle peut être modifié a posteriori, après son apprentissage. Le modèle doit quand même être créé initialement par un al-

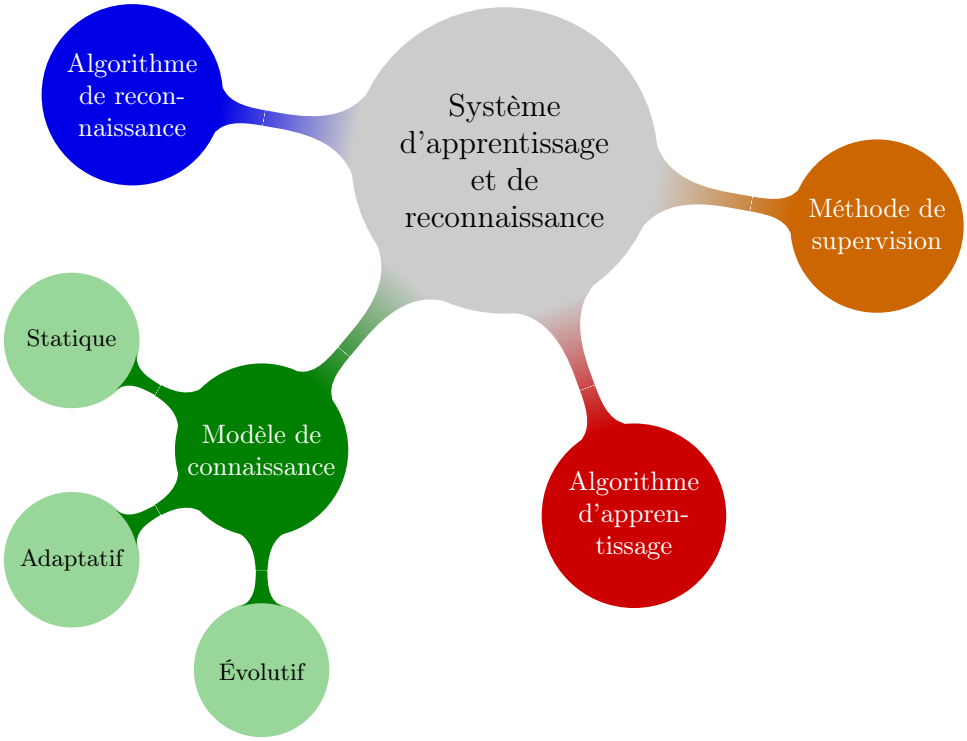


FIGURE 1.7 – Détails des différents types de modèles de connaissance possibles pour un classifieur.

gorithme d'apprentissage sur une base de données d'entrainement, mais ses paramètres peuvent ensuite être adaptés [Mouchère et al., 2007b].

Les différents paramètres du modèle peuvent être modifiés pour prendre en compte de nouvelles données. Un classifieur adaptatif peut notamment être appris incrémentalement, en construisant son modèle au fur et à mesure du traitement des exemples d'apprentissage.

Si les paramètres du modèle peuvent évoluer, sa structure reste fixe, et en particulier le nombre de classes ne peut pas changer. Un système dont les paramètres et la structure peuvent évoluer est appelé un système évolutif.

1.5.3 Classifieur à modèle évolutif

Un classifieur évolutif est un système dont le modèle est entièrement dynamique. Il peut être non seulement adapté de manière dynamique, mais également modifié structurellement.

Un classifieur évolutif supporte l'ajout et la suppression de classes dynamiquement, sans nécessiter un ré-apprentissage du modèle. Un tel système peut ainsi être appris dynamiquement sur un flux de données, en faisant évoluer son modèle à l'arrivée de chaque nouvelle donnée, tout en restant utilisable entre temps.

Afin de conserver ses capacités d'évolution, l'algorithme d'apprentissage d'un modèle évolutif doit intégrer de l'oubli, de manière explicite ou implicite [Aggarwal et al., 2006].

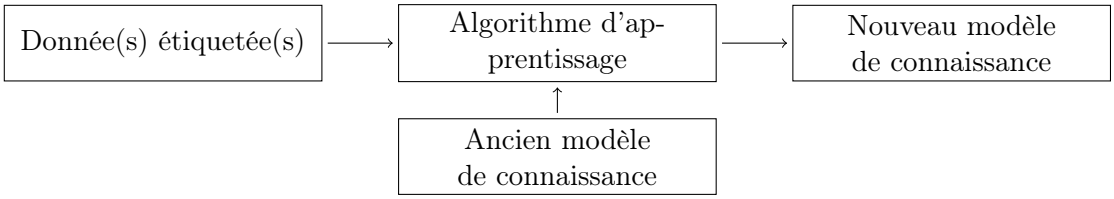


FIGURE 1.8 – Processus d'apprentissage du modèle de connaissance à partir des données.

Sans oubli, le modèle va prendre en compte toutes les données d'apprentissage. L'impact des nouvelles données diminue avec le temps et le modèle finit par ne plus évoluer.

Une capacité spécifique aux systèmes évolutifs, dont l'apprentissage intègre de l'oubli, est le suivi des changements de concepts [Gama et al., 2014]. Avec l'utilisation d'oubli, le modèle ne prend en compte que les dernières données et suit indéfiniment toutes les évolutions du flux.

Ces notions d'oubli et de changements de concepts sont détaillées dans la section 1.6.2.1 qui est consacrée à l'apprentissage en-ligne de systèmes évolutifs.

1.5.3.1 Compromis plasticité/stabilité

Le compromis plasticité/stabilité désigne l'équilibre à trouver en la capacité d'un système évolutif à intégrer de la nouveauté dans son modèle, sans pour autant perturber la connaissance existante et compromettre sa stabilité.

C'est un aspect important des systèmes évolutifs, qui doivent être capables d'apprendre rapidement de nouvelles informations, comme des nouvelles classes, en cours d'utilisation. Cet apprentissage de nouvelles connaissances ne doit pas se faire au détriment des connaissances existantes, ce qui limiterait les capacités de modélisation du système.

Le seul cas où l'apprentissage de nouvelles connaissances peut se faire au détriment de celles existantes est celui où ces connaissances sont contradictoires. Il est alors nécessaire de mettre à jour le modèle en oubliant complètement les anciennes connaissances devenues obsolètes.

1.6 Taxonomie des méthodes d'apprentissage

Cette section présente les différents types d'apprentissage pouvant être utilisés pour entrainer un classifieur, qui sont résumés sur la figure 1.9. Le processus d'apprentissage est présenté figure 1.8. Nous détaillons ainsi :

- l'apprentissage hors-ligne ;
  - l'apprentissage « batch » ;
  - l'apprentissage incrémental ;
- l'apprentissage en-ligne ;
  - l'apprentissage incrémental sur flux ;
  - l'apprentissage « permanent » *(anytime learning)*.

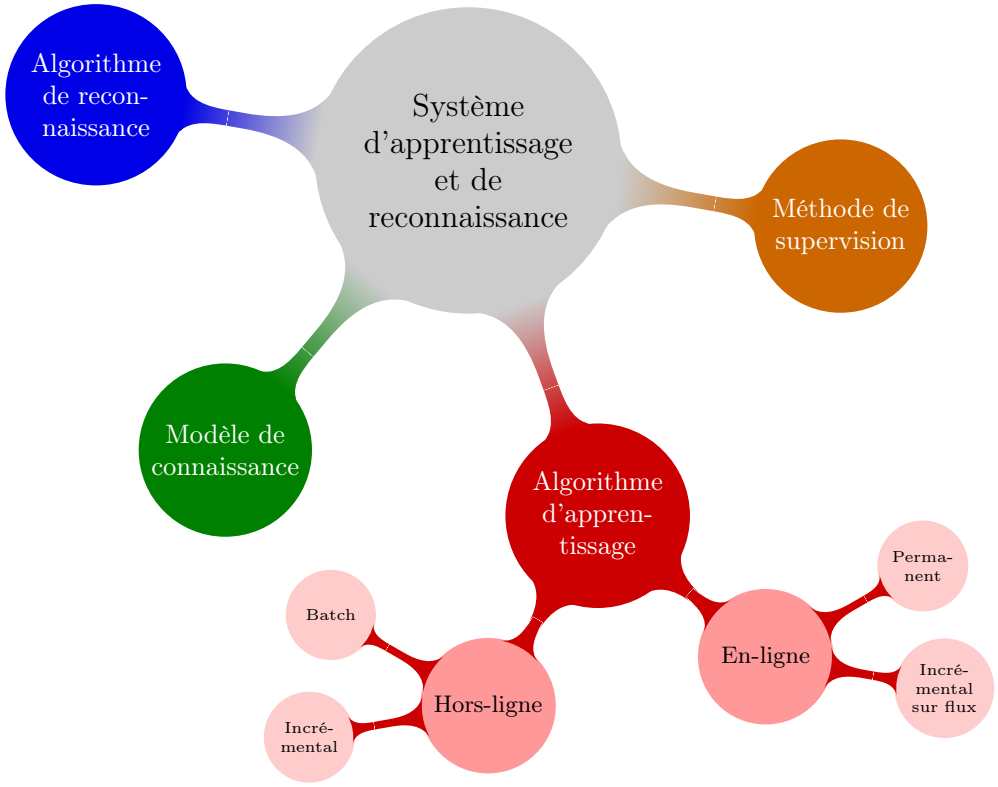


FIGURE 1.9 – Détails des différents types d'apprentissage possible pour un classifieur.

1.6.1 Apprentissage hors-ligne

L'apprentissage hors-ligne d'un système d'apprentissage automatique est la manière la plus courante de procéder [Cornuéjols and Miclet, 2010]. Cela consiste à séparer les phases d'apprentissage et d'utilisation du système comme représenté sur la figure 1.10a.

Dans un premier temps, le système est entrainé sur une base d'apprentissage. L'apprentissage se poursuit aussi longtemps que nécessaire pour optimiser le modèle du classifieur à partir de cette base d'apprentissage.

Dans un second temps, une fois l'apprentissage du modèle terminé, celui-ci peut être utilisé par l'algorithme de reconnaissance. Le modèle obtenu à l'issu de la phase d'apprentissage est un modèle statique, qui ne changera pas pendant la phase d'utilisation.

Cette manière de procéder comporte plusieurs inconvénients. Tout d'abord, il faut que la base d'apprentissage soit la plus grande possible, pour garantir de bonnes performances. Ensuite, il faut que les données d'apprentissage soient représentatives des données d'utilisation, car le modèle étant statique, il ne s'adapte pas en cas de changement dans la distribution des données. Enfin, les classes sont prédéfinies lors de l'apprentissage et il n'est pas possible d'en ajouter lors de l'utilisation.

L'apprentissage hors-ligne s'oppose à l'apprentissage en-ligne, où cette fois le système apprend pendant son utilisation. L'apprentissage hors-ligne d'un système est le plus souvent

réalisé de manière « batch », mais il peut également être réalisé de manière incrémentale.

1.6.1.1 Apprentissage « batch »

Le terme « batch » désigne un paquet de données. L'apprentissage « batch » désigne le fait d'apprendre le système, de construire son modèle de connaissance à partir de l'ensemble des données, en une étape.

Soit $\mathbf{x}_i$ $(i = 1..t)$ la i$^{\text{e}}$ donnée, M_t le modèle du classifieur à l'instant t, et f l'algorithme d'apprentissage. Le processus d'apprentissage « batch » correspond à :

$$M_t = f(\mathbf{x}_1, \ldots, \mathbf{x}_t) \tag{1.1}$$

L'apprentissage « batch » nécessite donc d'avoir accès à l'ensemble du jeu de données d'apprentissage. Cette manière de procéder s'oppose à l'apprentissage incrémental, où cette fois les données d'apprentissage sont considérées une à une, et non dans leur ensemble.

1.6.1.2 Apprentissage incrémental

L'apprentissage incrémental est le fait d'apprendre en parcourant les données une par une, sans jamais considérer le jeu de données d'apprentissage dans son ensemble.

Soit $\mathbf{x}_i$ $(i = 1..t)$ la i$^{\text{e}}$ donnée, M_t le modèle du classifieur à l'instant t, et f l'algorithme d'apprentissage. Le processus d'apprentissage incrémental peut être défini comme suit :

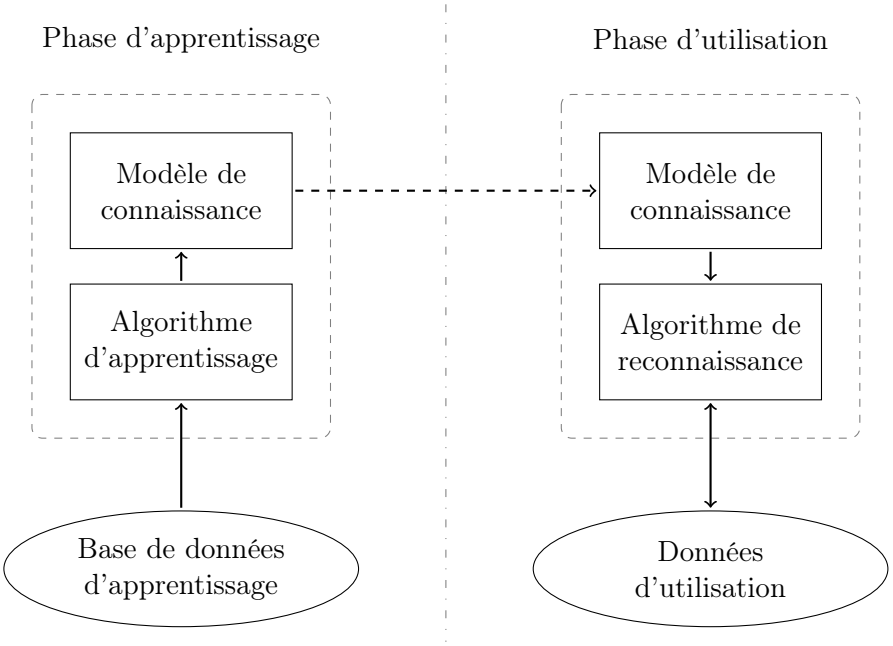
$$M_t = f(M_{t-1}, \mathbf{x}_t) \tag{1.2}$$

Un algorithme d'apprentissage incrémental est nécessaire lors de l'apprentissage d'un système en-ligne. Plus précisément, l'apprentissage en-ligne nécessite un algorithme incrémental en temps réel (et avec de l'oubli pour suivre les changements de concepts, voir section suivante). Cette notion de temps réel varie fortement suivant le contexte applicatif. Dans le cadre de la reconnaissance de commandes gestuelles, l'algorithme d'apprentissage ne doit pas nécessiter plus de quelques dixièmes de secondes pour mettre le modèle du classifieur à jour. Le système doit en effet être prêt à reconnaître la commande suivante en moins d'une fraction de seconde. Dans le cadre de la régulation et du contrôle de systèmes physiques complexes, les flux de données sont souvent plus lents, il n'est pas rare que les données n'arrivent que toutes les heures.

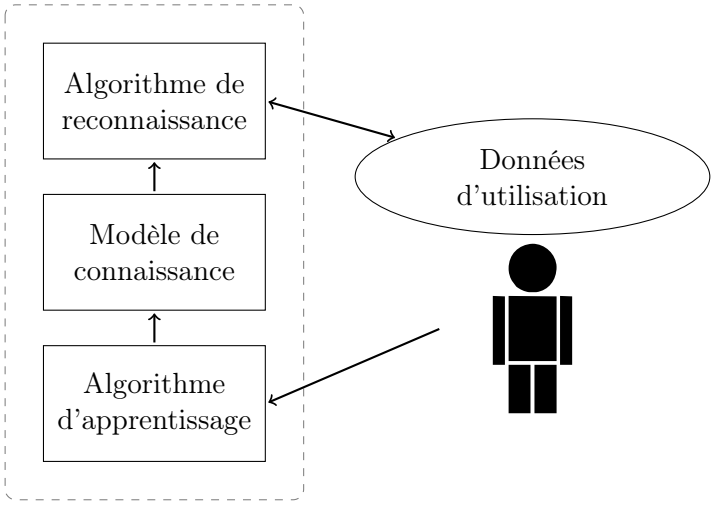
De l'autre côté, l'apprentissage hors-ligne du modèle d'un système peut très bien être également réalisé de manière incrémentale. La raison la plus courante est que la taille du jeu de données d'apprentissage est trop importante pour que celui-ci puisse être considéré dans son ensemble. Certains algorithmes d'apprentissage sont également incrémentaux par nature, c'est le cas de l'apprentissage d'un réseau de neurones par rétro-propagation du gradient de l'erreur.

1.6.2 Apprentissage en-ligne

À l'apprentissage hors-ligne s'oppose l'apprentissage en-ligne, où cette fois les phases d'apprentissage et d'utilisation sont mélangées. La figure 1.10b représente ce mode de fonctionnement où l'utilisateur est dans la boucle d'apprentissage.



(a) Apprentissage hors-ligne : séparation de la phase d'apprentissage et de la phase d'utilisation du système.



(b) Apprentissage en-ligne : l'apprentissage se fait conjointement avec l'utilisation du système.

FIGURE 1.10 – Différence entre apprentissage hors-ligne et apprentissage en-ligne.

1.6.2.1 Apprentissage incrémental sur flux

Lors de l'apprentissage en-ligne d'un système, les données arrivent sous forme de flux. C'est donc un environnement non-stationnaire, qui va pouvoir évoluer au cours du temps. Un algorithme d'apprentissage en-ligne, c'est-à-dire incrémental sur flux, fait évoluer le modèle de connaissance appris pour suivre les évolutions de l'environnement [Choy et al., 2006]. Le modèle de connaissance modélise alors l'état actuel du flux, quitte à ne plus modéliser les données issues de l'ancien état du flux. C'est la principale différence avec un algorithme d'apprentissage incrémental hors-ligne, qui construit un modèle de connaissance modélisant l'ensemble des données d'apprentissage.

Lorsqu'une nouvelle donnée arrive, le système essaye de la reconnaître avec la version actuelle de son modèle. Ensuite, le système récupère l'étiquette de cette donnée, jusque-là inconnue, grâce à une méthode de supervision. Le système utilise alors cette nouvelle donnée, maintenant étiquetée, pour améliorer son modèle à l'aide d'un algorithme d'apprentissage incrémental sur flux, c'est-à-dire intégrant de l'oubli.

Les avantages de cette manière de procéder sont multiples. La base d'apprentissage initiale peut être minimale, puisque le système apprend et s'améliore pendant son utilisation. Le système s'adapte à toute modification dans la distribution des données – appelée changement de concepts – de manière transparente. Enfin, l'apprentissage en-ligne d'un système évolutif permet de pouvoir ajouter et supprimer des classes « à la volée », pendant l'utilisation du système.

L'apprentissage en-ligne nécessite deux choses :

- un algorithme d'apprentissage en-ligne :
  - incrémental,
  - en temps réel,
  - avec de l'oubli ;
- une méthode de supervision de l'apprentissage en-ligne, pour étiqueter les données pendant l'utilisation du système.

1.6.2.2 Notion d'oubli

Une composante importante des algorithmes d'apprentissage en-ligne est l'intégration d'oubli dans le processus d'optimisation [Gent et al., 2012].

Il est en effet nécessaire de limiter le poids des anciennes données afin de maintenir le gain de l'apprentissage. Si toutes les données ont un poids relatif équivalent, alors le gain de l'apprentissage tend vers zéro avec le temps, et le modèle de connaissance finit par ne plus évoluer. Or c'est justement le rôle d'un algorithme d'apprentissage en-ligne de faire évoluer continuellement le modèle de connaissance du système.

Bien qu'il soit indispensable d'intégrer de l'oubli dans le processus d'apprentissage, il ne faut pas non plus trop oublier. Le fait de trop oublier limite en effet les performances du système lorsque son environnement est stationnaire. Trop d'oubli peut même causer l'effondrement du système, c'est le phénomène d'oubli catastrophique *(catastrophic forgetting)* [French, 1999].

La notion d'oubli dans un algorithme d'apprentissage exprime le fait de ne pas accorder

le même poids à toutes les données, voire de ne plus prendre en compte certaines données au bout d'un certain temps [Bifet and Gavalda, 2007]. L'intégration d'oubli dans l'algorithme d'apprentissage peut être fait de manière explicite ou implicite.

1.6.2.3 Notion de changement de concepts

Toute évolution de la distribution des données au cours du temps est appelée « changement de concepts » [Widmer and Kubat, 1996]. Les changements de concepts peuvent être caractérisés par leur vitesse : abrupt, léger, ou encore progressif.

Dans le contexte de la reconnaissance de commandes gestuelles, l'évolution du style d'écriture de l'utilisateur fait lentement changer les concepts des différentes classes, et ainsi créer des changements de concepts lents. Toute modification soudaine d'un symbole, pour le simplifier par exemple, crée un changement de concept abrupt. Il faudra alors que le système s'y adapte pour maintenir de bonnes performances de reconnaissance.

Un algorithme d'apprentissage en-ligne doit pouvoir faire évoluer le modèle de connaissance appris pour s'adapter à la distribution courante des données. Afin de pouvoir suivre les changements de concepts efficacement, il est nécessaire d'intégrer de l'oubli au processus d'apprentissage [Žliobaite, 2010].

1.6.3 Apprentissage « permanent » *(anytime learning)*

L'apprentissage permanent *(anytime learning)* est une forme d'apprentissage en-ligne avancé, où l'apprentissage est continu tout en restant interruptible à tout moment [Seidl et al., 2009]. Un algorithme d'apprentissage permanent tire ainsi profit du temps entre deux reconnaissances pour continuer à améliorer le modèle du classifieur. Plus le temps entre deux reconnaissances est long, plus le système peut s'améliorer et être performant lorsque la prochaine reconnaissance devra être faite.

L'apprentissage permanent est également appelé apprentissage interruptible, car le processus d'apprentissage peut être interrompu à tout moment, lorsqu'une reconnaissance doit être effectuée. L'apprentissage est alors stoppé, pendant l'utilisation du système, et reprend ensuite.

Lorsque l'apprentissage du système est potentiellement trop long, ou l'utilisation du système est potentiellement trop fréquente, le processus d'apprentissage et celui de reconnaissance peuvent être décorrélés pour avoir le temps d'apprendre le système entre deux reconnaissances. Ce mode de fonctionnement n'est cependant pas de l'apprentissage permanent, mais juste une désynchronisation des processus. L'apprentissage est alors réalisé en tâche de fond, en parallèle des reconnaissances.

L'apprentissage permanent consiste à tirer profit du temps entre deux reconnaissances pour améliorer le système au maximum. La synthèse de données artificielles [Mouchère et al., 2007a] [Almaksour et al., 2011] à partir des derniers tracés de l'utilisateur permet par exemple de créer davantage de données d'apprentissage. L'utilisation de ces données artificielles permet de continuer à entraîner le système, en attendant la prochaine donnée réelle produite par l'utilisateur.

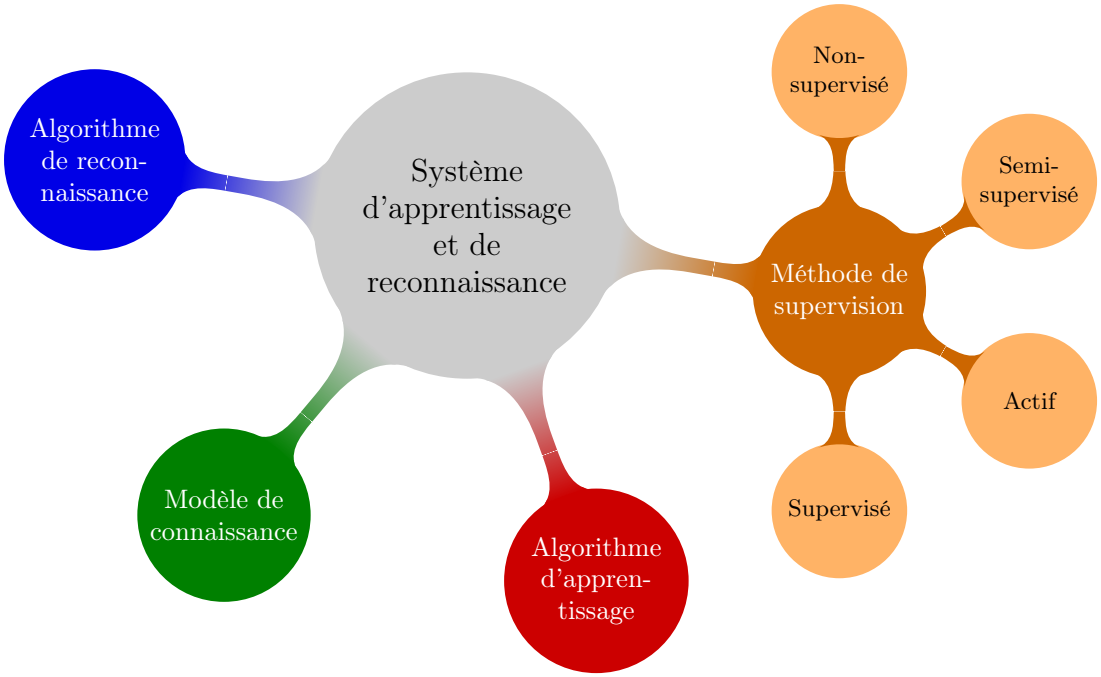


FIGURE 1.11 – Détails des différents types de supervision de l'apprentissage d'un classifieur.

1.7 Taxonomie de supervision de l'apprentissage

La supervision de l'apprentissage est un enjeu majeur dans beaucoup de contextes. L'apprentissage supervisé consiste à apprendre à partir de données étiquetées, c'est-à-dire associés à une classe. Cependant, cet étiquetage des données est souvent coûteux et ne peut donc pas être réalisé pour toutes les données. Le processus de supervision de l'apprentissage, pour étiqueter les données, est présenté figure 1.12. L'apprentissage automatique peut être divisé en plusieurs catégories, suivant la supervision de cet apprentissage, comme présenté figure 1.11. Nous distinguons ici trois types d'apprentissage :

— l'apprentissage non-supervisé ;
— l'apprentissage supervisé ;
— l'apprentissage actif ;
— l'apprentissage semi-supervisé.

1.7.1 Apprentissage non-supervisé

L'apprentissage non-supervisé consiste à chercher une structure dans des données non-étiquetées [Cornuéjols and Miclet, 2010]. Comme les données ne sont pas étiquetées, elles ne sont pas rangées dans des classes prédéfinies. Les algorithmes d'apprentissage non-supervisés cherchent alors à modéliser la distribution des données. La tâche d'apprentissage non-supervisé la plus connue est le regroupement *(clustering)*, que nous avons présenté en section 1.4.2.

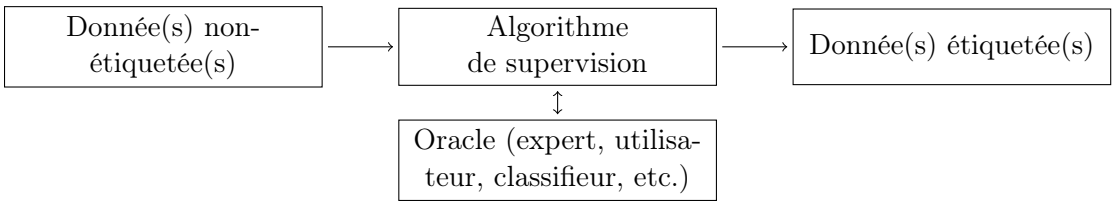


FIGURE 1.12 – Processus de supervision de l'apprentissage.

1.7.2 Apprentissage supervisé

L'apprentissage supervisé consiste à inférer la fonction d'association entre des données et leur étiquette ou valeur associée [Cornuéjols and Miclet, 2010]. Cette étiquette ou valeur associée peut être numérique, dans le cas des problèmes de régression, ou discrète dans le cas des problèmes de classification. L'apprentissage supervisé consiste à apprendre comment associer les données à leur étiquette à partir d'un ensemble d'apprentissage étiqueté servant d'exemple. Cet ensemble de données d'apprentissage doit avoir été étiqueté par un expert, qui supervise l'apprentissage. L'objectif du système est ensuite de prédire l'étiquette qui doit être associée aux exemples inconnus.

1.7.3 Apprentissage semi-supervisé

Le principe de l'apprentissage semi-supervisé est d'apprendre à la fois à partir d'exemples étiquetés et d'exemples non-étiquetés [Bouchachia et al., 2010]. L'étiquetage des données est en effet souvent difficile, coûteux ou chronophage, et cela requiert l'intervention d'experts humains. Les données non-étiquetées sont quant à elle faciles à collecter, mais sont par contre beaucoup moins évidentes à utiliser. L'intérêt de l'apprentissage semi-supervisé est donc de réduire les efforts d'annotation [Tomanek and Hahn, 2009], tout en améliorant les performances de généralisation. Il est important de noter que le système ne peut pas choisir quelles sont les données qui sont étiquetées. L'apprentissage semi-supervisé se concentre sur ce problème d'utilisation des données non-étiquetées, en synergie avec celles qui le sont, pour améliorer le modèle de connaissance du classifieur.

L'apprentissage semi-supervisé est très utilisé sur les très grandes bases de données. Classiquement, les données étiquetées sont peu nombreuses par rapport aux données non-étiquetées. Les données non-étiquetées permettent alors de déduire de l'information sur la distribution des données, et de propager efficacement l'information contenue dans les données étiquetées [Carlson et al., 2010]. Il est aussi possible d'utiliser l'apprentissage semi-supervisé sur pour des systèmes évolutifs sur flux de données [Bouchachia et al., 2010]. L'apprentissage semi-supervisé regroupe toutes les formes d'apprentissage entre l'apprentissage supervisé, où toutes les données sont étiquetées, et l'apprentissage non-supervisé, où aucune donnée n'est étiquetée.

L'utilisation de données non-étiquetées en conjonction avec une petite quantité de données étiquetées peut permettre d'améliorer considérablement les capacités de généralisation du modèle de connaissance appris par le système. La figure 1.13, extraite des travaux de

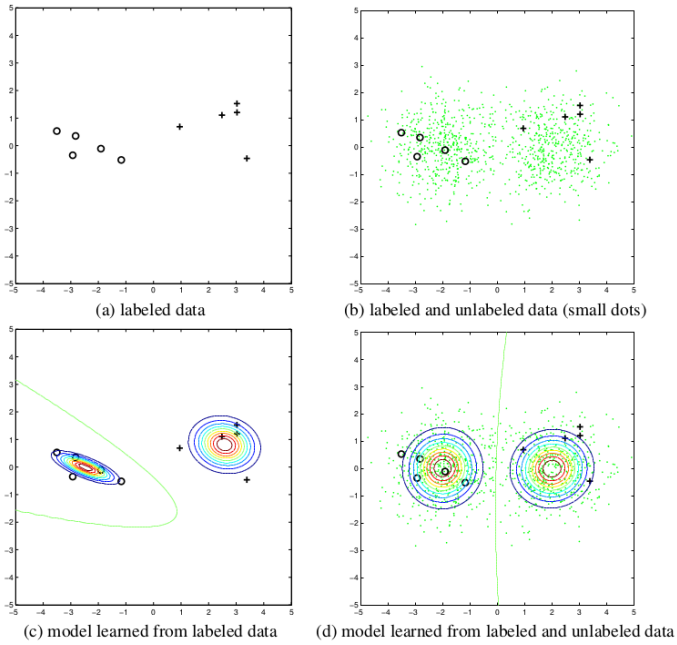


FIGURE 1.13 – Exemple de l'influence de données non-étiquetées sur la modélisation de la distribution – Illustration : [Settles, 2010].

Settles [Settles, 2010], montre un exemple de l'influence possible de données non-étiquetées sur la modélisation de la distribution.

Il existe de nombreuses techniques d'apprentissage semi-supervisé : l'auto-apprentissage, le co-apprentissage, l'algorithme espérance-maximisation *(expectation maximization)* pour les mélanges de modèles génératifs, les machine à vecteur support transductive, etc. [Zhu, 2005]. En particulier, l'auto-apprentissage est une technique d'apprentissage semi-supervisé générique, qui peut être utilisée avec n'importe quel algorithme d'apprentissage. Elle consiste à entrainer le système sur les données étiquetées, puis à classifier les données inconnues. Les données classifiées pour lesquelles le classifieur est le plus confiant sont alors ajoutées à l'ensemble des données étiquetées. Le système est ré-entrainé et le processus est répété. Dans cette technique, le système utilise ses propres prédictions pour s'auto-améliorer.

1.7.4 Apprentissage actif

L'apprentissage actif est une forme d'apprentissage supervisé où c'est le système qui choisit quelles sont les données qui sont étiquetées. Les données non-étiquetées ne sont cette fois pas utilisées, contrairement au cas de l'apprentissage semi-supervisé.

L'apprentissage supervisé nécessite une base de données d'apprentissage étiquetée. Cet étiquetage – ou vérité terrain – est souvent coûteux à obtenir. Dans certains contextes, il serait trop coûteux d'étiqueter toute la base d'apprentissage, et il faut alors choisir quelles sont les données qui doivent être étiquetées. L'apprentissage actif est le fait de choisir les

données à étiqueter pour être utilisées comme données d'apprentissage [Settles, 2010].

Une approche intéressante est la combinaison des techniques d'apprentissage semi-supervisé avec les techniques d'apprentissage actif. Certaines données stratégiques sont alors étiquetées manuellement et les autres sont utilisées pour propager l'information [Tur et al., 2005] [Grira et al., 2005].

L'apprentissage avec l'utilisateur dans la boucle est en particulier un cas où l'étiquetage des données est coûteux, car il nécessite de solliciter l'utilisateur final. Il est donc nécessaire de mettre en place une stratégie d'apprentissage actif pour limiter le coût du processus d'apprentissage. En particulier, l'apprentissage de commandes gestuelles est un cas où l'étiquetage des données est coûteux, puisqu'il requiert l'intervention de l'utilisateur. Il faut donc sélectionner avec soin les données qui sont étiquetées pour servir à l'apprentissage du système, c'est de l'apprentissage actif. Nous détaillons donc la supervision active de l'apprentissage dans le prochain chapitre.

1.8 Conclusion

Nous avons présenté le contexte de ces travaux : l'utilisation de commandes gestuelles personnalisables. Ce contexte induit une situation d'apprentissage croisé, où l'utilisateur doit mémoriser le jeu de symboles et le système doit apprendre à reconnaître les différents symboles. Cela implique un certain nombre de contraintes, comme le fait de devoir apprendre à partir de peu de données, et de continuer à apprendre pendant l'utilisation du système afin d'améliorer ses performances et de suivre l'évolution du style d'écriture de l'utilisateur. Il faut également intégrer de l'oubli au processus d'optimisation du système pour maintenir sa capacité d'apprentissage avec le temps et permettre son utilisation « à vie ».

Nous avons commencé par voir la différence entre les symboles en-lignes, qui correspond au cas des commandes gestuelles, et les symboles hors-lignes. Nous avons également défini les termes mono-stroke, multi-strokes et multi-points, et avons vu que tous peuvent être utilisés pour des commandes gestuelles. Les deux derniers engendrent cependant des problématiques de segmentation, pour regrouper les traits *(strokes)* appartenant au même symbole, qui peuvent devenir très complexes [Zhaoxin et al.,] et sont hors du champ de ces travaux. Dans tous les cas, indépendamment du type de tracé utilisé, il est nécessaire d'extraire des caractéristiques des données afin de réaliser l'apprentissage du système de classification.

Plus généralement, la reconnaissance de commandes gestuelles est un problème de classification. Il faut en effet reconnaître à quelle catégorie, quelle commande gestuelle, correspondent des données inconnues. Nous avons également présenté rapidement les autres types de problème d'apprentissage automatique qui existent : les problèmes de régression, de regroupement, de détection d'erreur et d'inférence de règles.

Pour reconnaître les symboles dessinés, il faut apprendre un classifieur. Il est ici nécessaire que ce classifieur soit évolutif afin de s'adapter à l'évolution des symboles dessinés par l'utilisateur. Cela permet également à l'utilisateur d'ajouter de nouvelles commandes gestuelles en cours d'utilisation. Nous avons vu la différence entre un classifieur évolutif,

un classifieur adaptatif, et un classifieur statique.

Un classifieur évolutif est donc appris en même temps qu'il est utilisé, ce qui est notamment le cas lorsque l'utilisateur est dans la boucle d'apprentissage. C'est ce qui est appelé apprentissage en-ligne, par opposition à l'apprentissage hors-ligne. Nous avons également clarifié les définitions d'apprentissage batch et incrémental.

Afin de pouvoir apprendre pendant l'utilisation du système, il est nécessaire d'interagir avec l'utilisateur pour obtenir les étiquettes des données. L'apprentissage est en effet supervisé, il est réalisé à partir de symboles dont les étiquettes sont connues, par opposition à l'apprentissage non-supervisé. Cependant, interagir constamment avec l'utilisateur réduit considérablement l'intérêt des commandes gestuelles. Il faut donc choisir les données les plus intéressantes pour le processus d'apprentissage : c'est de l'apprentissage actif.

Cette thèse se focalise sur l'apprentissage actif en-ligne d'un classifieur évolutif, pour les problèmes de classification avec l'utilisateur dans la boucle d'apprentissage. Il est donc nécessaire d'avoir d'une part un classifieur évolutif, et d'autre part un système de supervision actif pour l'apprentissage du classifieur. Nous présentons dans le prochain chapitre les principes et les méthodes de supervision active de l'apprentissage, en particulier l'utilisation de l'option de rejet du classifieur.

Chapitre 2

Supervision active en-ligne

2.1 Introduction

Dans le contexte d'utilisation de commandes gestuelles personnalisées, la supervision de l'apprentissage est un enjeu majeur. Le seul moyen d'obtenir de façon certaine l'étiquette d'un symbole est de poser la question à l'utilisateur. On peut aussi essayer de déduire la valeur de l'étiquette à partir du comportement de l'utilisateur, comme nous l'expliquons en section 6.2.

Dans tous les cas, les interactions avec l'utilisateur restent indispensables, et doivent être en nombre suffisant pour permettre l'apprentissage du classifieur. Ces sollicitations doivent cependant rester suffisamment rares pour ne pas ennuyer l'utilisateur et ne pas le décourager d'utiliser le système de commandes gestuelles. Il est donc nécessaire de choisir soigneusement les données pour lesquelles l'utilisateur sera sollicité pour les étiqueter. Qui plus est, dans le cadre des commandes gestuelles, l'apprentissage est réalisé en-ligne, et cette sélection doit donc être faite en-ligne également.

L'apprentissage actif part du constat que les algorithmes d'apprentissage peuvent obtenir d'aussi bonnes performances avec beaucoup moins d'exemples d'apprentissage, si ces derniers sont bien choisis, qu'en utilisant toutes les données. Le principe de l'apprentissage actif est donc de choisir le mieux possible les données qui seront étiquetées pour optimiser l'apprentissage. Il faut alors déterminer quelles données contiennent le plus d'informations nouvelles pour le modèle de connaissance du classifieur. D'une manière générale, cette sélection des données utilisées pour l'apprentissage est motivée par le fait que les données réelles sont souvent abondantes, mais que le processus d'étiquetage par l'utilisateur est long et coûteux.

Il existe plusieurs scénarios d'apprentissage actif que nous présentons en section 2.2, mais nous nous concentrons ici sur le scénario d'échantillonnage de flux de données, qui correspond au cas de l'apprentissage en-ligne. Différentes méthodes pour effectuer ce choix des données sont possibles pour un échantillonnage en-ligne, comme l'incertitude maximale, la minimisation de l'erreur et la réduction de la variance par exemple.

Une manière simple et efficace de sélectionner les données selon le principe de l'incertitude maximale est d'utiliser un classifieur avec une option de rejet des données. Nous

définissons les deux types de rejet possibles, ainsi que le compromis entre le taux d'erreur et le taux de rejet que cela implique, dans la section 2.4.

Ce chapitre présente les principes et les méthodes de supervision actives de l'apprentissage en-ligne. Nous commençons par définir les différents scénarios d'apprentissage actif dans la section 2.2 et en particulier le problème d'échantillonnage de flux de données qui est induit par le contexte des commandes gestuelles. Nous présentons ensuite les différentes méthodes d'échantillonnage qui existent pour réaliser la sélection des données d'apprentissage dans la section 2.3. Finalement, nous détaillons l'utilisation de l'option de rejet comme méthode d'échantillonnage dans la section 2.4.

2.2 Scénarios d'apprentissage actif

Lorsque l'utilisateur est dans la boucle d'apprentissage, la supervision de l'apprentissage est coûteuse puisqu'elle nécessite l'intervention de l'utilisateur final. Il faut limiter le nombre de données à étiqueter, et donc choisir soigneusement celles qui sont utilisées pour l'apprentissage du classifieur : c'est de l'apprentissage actif [Settles, 2010].

L'idée de l'apprentissage actif est que, lorsque l'étiquetage de toutes les données est trop coûteux, c'est le système apprenant lui-même qui choisit à partir de quelles données il apprend. Un classifieur peut atteindre des performances équivalentes à celles obtenues en utilisant toutes les données d'apprentissage, mais avec seulement une partie de ces données, si elles ont été correctement choisies.

L'apprentissage actif est motivé par le fait qu'il est souvent coûteux d'obtenir des données étiquetées pour l'apprentissage, ce qui est complètement notre cas lors de l'apprentissage de commandes gestuelles. En effet, le seul moyen pour obtenir l'étiquette d'un symbole dessiné par l'utilisateur avec certitude est de lui poser la question. Or demander à l'utilisateur quelle commande voulait-il faire est fastidieux et enlève beaucoup de l'intérêt des commandes gestuelles. Les interactions avec l'utilisateur ont ainsi un coût non-négligeable.

Les problèmes d'apprentissage actif peuvent être répartis en trois catégories :
— les problèmes de génération d'échantillons ;
— les problèmes d'échantillonnage de bases de données ;
— les problèmes d'échantillonnage de flux de données.

2.2.1 Scénario de génération d'échantillons

Dans le cas des scénarios de génération d'échantillons, c'est le système lui-même qui génère les données à étiqueter. Ces données peuvent être situées n'importe où dans l'espace d'entrée, c'est-à-dire qu'elles sont construites par juxtaposition de n'importe quelles valeurs de chacune des caractéristiques sur leur domaine respectifs.

Ce scenario ne convient pas aux problèmes de reconnaissance de caractères et de formes manuscrites en général. En effet, les combinaisons de caractéristiques générées ne correspondent souvent pas à un caractère sémantique (donnée réelle), et il n'est alors pas possible de les étiqueter.

Les données sont générées à partir du modèle de connaissance du système, pour l'améliorer dans les zones imprécises de l'espace de représentation utilisé. Ce processus est complètement différent de la synthèse de données artificielles, qui utilise cette fois des données réelles modifiées afin d'améliorer la capacité de généralisation du classifieur.

2.2.2 Scénario d'échantillonnage de bases de données

Les scénarios d'échantillonnage de bases de données sont ceux où le système doit choisir les données à étiqueter parmi une base de données existante. Le système peut alors parcourir et étudier toute la base de données pour effectuer son échantillonnage.

Lorsque l'apprentissage actif est combiné à l'apprentissage semi-supervisé, les données non choisies, et qui ne seront donc pas étiquetées, sont également utilisées conjointement à celles qui ont été étiquetées.

Ce type de problème, d'échantillonnage de base de données, correspond aux tâches d'apprentissage hors-ligne, mais ne s'applique pas aux tâches d'apprentissage en-ligne.

2.2.3 Scénario d'échantillonnage de flux de données

L'apprentissage en-ligne sur un flux de données induit un scénario d'échantillonnage de flux de données. Le système doit cette fois décider quelles sont les données qui seront étiquetées alors que les données arrivent une à une. La décision d'échantillonnage est prise sans avoir connaissance des prochaines données qui arriveront, et sans garder en mémoire les données passées.

C'est ce type de problème qui se pose lors de la supervision de l'apprentissage de commandes gestuelles. Il faut décider après chaque commande gestuelle s'il est intéressant d'apprendre à partir de ce symbole, et si besoin faire vérifier par l'utilisateur l'étiquette reconnue par le système.

2.3 Méthodes d'échantillonnage

Dans tous les types de problèmes mentionnés ci-dessus, le système doit choisir quels sont les exemples dont il veut l'étiquette pour pouvoir apprendre. De nombreuses méthodes existent pour prendre cette décision :

— la méthode de l'incertitude maximale ;
— la méthode du comité de sélection ;
— la méthode de l'impact maximal ;
— la méthode de minimisation de l'erreur ;
— la méthode de réduction de la variance ;
— la méthode de la densité.

2.3.1 Méthode de l'incertitude maximale

La méthode la plus courante pour prendre cette décision d'échantillonnage est la méthode de l'incertitude maximale [Settles, 2010]. Son principe est simple : il s'agit de choisir

les données que le système est le moins sûr de savoir reconnaître. Ces données, pour lesquelles les incertitudes de classification sont les plus grandes, sont celles qui correspondent le moins bien à la connaissance actuelle du classifieur. Elles vont donc permettre d'améliorer le modèle du classifieur en apportant de l'information dans les zones inconnues ou à fort potentiel de confusion.

Pour utiliser la méthode de l'incertitude maximale, le plus simple est d'avoir un système qui dispose d'une option de rejet des données, comme nous le détaillons en section 2.4. En effet, l'option de rejet permet de sélectionner les données pour lesquelles le classifieur a le plus de chance de commettre une erreur, c'est-à-dire les données avec les plus fortes incertitudes de reconnaissance.

2.3.2 Méthode du comité de sélection *(query by committee)*

La méthode du comité de sélection consiste à maintenir un groupe (comité) de modèles représentant des hypothèses concurrentes. Chaque membre du comité tente de reconnaître les données, comme dans un ensemble de classifieurs simplistes – dit faibles. La décision d'échantillonnage est ensuite prise en sélectionnant les exemples pour lesquels les membres du comité sont les moins d'accord, c'est-à-dire pour lesquels les étiquettes mises par les différents modèles sont les plus variées.

Pour les systèmes modélisant la distribution des données (modèles génératifs), le comité peut être créé en échantillonnant aléatoirement la distribution postérieure. Par exemple, pour un classifieur Bayésien naïf, le comité peut être généré en échantillonnant les paramètres du modèle avec une distribution de Dirichlet [McCallum and Nigamy, 1998]. De même, pour les chaines de Markov cachées, le comité peut être réalisé en échantillonnant leurs paramètres avec une loi Normale [Dagan and Engelson, 1995]. Pour les modèles discriminatifs ou non-probabilistes, il est possible d'utiliser les méthodes de *boosting* et de *bagging* pour construire le comité [Abe and Mamitsuka, 1998]. Muslea et al. [Muslea et al., 2000] construisent un comité de deux modèles en partitionnant l'espace des caractéristiques.

Dans notre contexte d'apprentissage de commandes gestuelles, il est difficilement envisageable de générer un comité pour prendre la décision d'échantillonnage en temps réel.

2.3.3 Méthode de l'impact maximal *(expected model change)*

Une autre méthode d'échantillonnage est la méthode de l'impact maximal, qui consiste à faire étiqueter l'exemple qui aura le plus d'impact sur le modèle de connaissance du classifieur.

La méthode de l'impact maximal la plus répandue est celle de la norme du gradient attendu *(expected gradient length)* [Settles et al., 2008]. Cette méthode peut s'appliquer à tous les systèmes optimisés par une descente de gradient. En particulier, elle s'applique très bien sur la majorité des systèmes probabilistes discriminatifs. Dans cette approche, le changement est quantifié comme la norme du vecteur du gradient d'apprentissage. Les données qui doivent être étiquetées sont donc choisies comme celles qui engendreraient le gradient d'apprentissage le plus grand.

Cette approche a montré de bons résultats empiriquement, mais elle peut se révéler coûteuse en matière de calculs si à la fois la dimension de l'espace d'entrée (nombre de caractéristiques des données) et la dimension de l'espace de sortie (nombre de classes) sont grandes.

2.3.4 Méthode de minimisation de l'erreur *(expected error reduction)*

Cette approche consiste à évaluer non pas combien le modèle va changer, mais combien son erreur de généralisation est susceptible de diminuer. Pour cela, l'erreur future est estimée sur le reste des données parmi lesquelles l'échantillonnage est à réaliser. Les données qui sont étiquetées sont alors celles qui permettent d'obtenir l'estimation de l'erreur future la plus basse.

Cette méthode a été utilisée avec succès pour de nombreux systèmes comme les systèmes Bayésiens naïfs [Roy and McCallum, 2001], des champs aléatoires gaussiens [Zhu et al., 2003], des régressions logistiques [Guo and Greiner, 2007], des machines à vecteurs supports [Moskovitch et al., 2007] [Nissim et al., 2012] [Nissim et al., 2014].

Le principal inconvénient de cette approche est son importante complexité, car non seulement l'erreur doit être ré-estimée pour chaque exemple d'apprentissage potentiel, mais pour cela le modèle doit être mis à jour à chaque fois. Enfin, cette méthode ne s'applique qu'aux problèmes d'échantillonnage de bases de données, alors que notre contexte applicatif implique une méthode d'échantillonnage de flux de données.

2.3.5 Méthode de réduction de la variance

Minimiser l'estimation d'une fonction de coût directement est complexe. Cependant, l'erreur de généralisation peut être réduite indirectement en minimisant la variance de la sortie du système.

Cette approche est par exemple utilisée dans [Cohn, 1994] pour générer des exemples à échantillonner.

L'avantage de cette approche par rapport aux méthodes de minimisation de l'erreur est que le modèle n'a pas besoin d'être mis à jour, ce qui réduit le coût en matière de calcul. Cette méthode reste cependant plus coûteuse que les méthodes d'échantillonnage par incertitude maximale par exemple.

2.3.6 Méthode utilisant la densité

Le problème des méthodes d'échantillonnage simples, comme les méthodes d'incertitude maximales, de l'impact maximal et du comité de sélection, est qu'elles ont tendance à se concentrer sur les données atypiques, et potentiellement aberrantes. Les données aberrantes sont en effet loin du modèle de connaissance de classifieur, et ont donc une forte incertitude et un impact potentiel sur le modèle important. Cependant, ces exemples ne sont pas toujours représentatifs du reste des données, et leur étiquetage n'améliorera probablement pas la qualité générale du modèle. L'idée des méthodes utilisant la densité est que les données intéressantes à faire étiqueter sont celles qui sont incertaines et qui sont

représentatives de la distribution des données, c'est-à-dire situées dans les zones de fortes densités.

Plusieurs approches de la littérature considèrent la densité et la représentativité des données pour prendre la décision d'échantillonnage [Settles and Craven, 2008].

Par exemple, des méthodes utilisant ce principe ont été proposées pour les classifieurs Bayésiens naïfs [McCallum and Nigamy, 1998] et pour les classifieurs des plus proches voisins [Fujii et al., 1998].

L'inconvénient de cette approche par rapport à notre contexte de reconnaissance de commandes gestuelles est qu'il est difficile d'estimer la densité sur un flux de données.

2.4 Échantillonnage par une option de rejet des données

L'utilisation de commandes gestuelles crée une situation d'apprentissage croisé, où l'utilisateur et le système doivent interagir. D'un côté l'utilisateur doit mémoriser les symboles, et apprendre lequel est associé à quelle commande. De l'autre côté, le système doit modéliser les différentes classes pour pouvoir reconnaître les symboles dessinés par l'utilisateur.

Lorsque l'utilisateur est dans la boucle d'apprentissage, il est trop coûteux de lui faire étiqueter toutes les données. Il est nécessaire de choisir seulement les données les plus intéressantes pour l'apprentissage du classifieur : c'est de l'apprentissage actif. La manière la plus courante d'effectuer cet échantillonnage est d'utiliser l'option de rejet des données du classifieur. Deux types d'option de rejet existent [Mouchere and Anquetil, 2006b] : les options de rejet de distance et les options de rejet de confusion.

2.4.1 Rejet de distance

Le principe du rejet de distance est de délimiter le domaine de connaissance du classifieur. Ce type de rejet est utilisé dans les contextes où il peut y avoir des données de type inconnu, qui n'appartiennent à aucune des classes vues lors de l'apprentissage. Le rejet de distance permet par exemple à un classifieur de lettres de ne pas essayer de reconnaître des chiffres, symboles sur lesquels il n'a pas été entrainé.

L'option de rejet de distance permet de rejeter les données qui sont trop loin du modèle de connaissance du classifieur. Les données ayant de faibles probabilités d'appartenir à chacune des classes sont ainsi rejetées, car elles ne semblent correspondre à aucune classe connue du classifieur.

2.4.2 Rejet de confusion

Le principe du rejet de confusion est d'évaluer les zones d'incertitude du classifieur. Ce type de rejet est utilisé quand il est très coûteux de commettre une erreur de reconnaissance. Le rejet de confusion est par exemple utilisé dans la reconnaissance de chèques, où il faut à tout prix éviter de se tromper dans la reconnaissance du montant. Un opérateur humain vient alors prendre le relais du classifieur.

L'option de rejet de confusion permet de rejeter les données qui sont entre les modèles de plusieurs classes du classifieur. Les données ayant de fortes probabilités d'appartenir à plusieurs classes sont ainsi rejetées, car elles semblent correspondre à plusieurs modèles connus du classifieur.

2.4.3 Compromis erreur/rejet

Dans tous les cas, que l'option de rejet soit de distance ou de confusion, il y un compromis à faire entre la quantité d'erreurs et la quantité de rejet [Chow, 1970] [Hansen et al., 1997]. Plus la quantité de rejet est importante, moins il reste d'erreurs de classification, mais moins il y aura de données classifiées par le système. Inversement, moins il y a de données rejetées, plus il y a de données classifiées par le système, mais plus il reste des erreurs de classification. L'équilibre choisi dépend du coût associé à une erreur et de celui associé au rejet d'une donnée, que la décision de rejet soit correcte ou non.

Dans le contexte de la reconnaissance de commandes gestuelles, un rejet oblige l'utilisateur à valider la commande souhaitée dans un menu avant son exécution. Une erreur de reconnaissance engendre quant à elle l'exécution d'une mauvaise commande que l'utilisateur est obligé d'annuler, avant de refaire la commande voulue.

Nous avons réalisé un sondage auprès des 62 sujets qui ont participé à notre campagne de test [Boui 13c]. Un rejet a été évalué gênant avec un score moyen de 3.96, sur une échelle de 0 à 10, alors qu'une erreur a été jugée ennuyeuse avec une score moyen de 6.71. Il semble raisonnable de supposer qu'une erreur est deux fois plus ennuyeuse qu'un rejet, étant donné qu'une erreur oblige l'utilisateur à annuler la commande erronée et à retenter la commande voulue, alors qu'un rejet ne nécessite qu'une validation ou correction dans un menu. Par conséquent, nous n'utiliserons pas l'équilibre ERR (Equal Error Rate), où le taux de fausses acceptations *(False Acceptance Rate : FAR)* égale le taux de faux rejets *(False Rejection Rate : FRR)*. À la place, nous souhaitons avoir deux fois plus de rejets que d'erreurs, ce que nous appellerons le E2RR : *1 Error for 2 Rejections Rate*.

Lorsque le classifieur est dynamique (adaptatif ou évolutif comme vu en section 1.5) et qu'il est appris en-ligne, le système s'améliore et le compromis erreur/rejet change. Il est alors nécessaire de faire évoluer l'option de rejet en accord avec le classifieur afin de maintenir le compromis erreur/rejet souhaité.

2.4.4 Fonction de confiance

Le but d'une fonction de confiance est d'évaluer dans quelle mesure le classifieur risque de commettre une erreur lors du processus de reconnaissance d'un exemple.

Il est possible de construire différentes fonctions de confiance en fonction du besoin. Ainsi, pour faire du rejet de distance, il faut prendre une mesure de confiance évaluant l'adéquation entre une donnée et le modèle de connaissance du classifieur. Pour faire du rejet de confusion, il faut choisir une mesure de confiance qui évalue la différence entre la classe reconnue et les autres classes connues du classifieur. Afin de prendre la décision d'échantillonnage, il faut une mesure de confiance générale, qui combine les notions de rejet de distance et de confusion.

2.4.5 Architectures des options de rejet

Cette section présente les différentes architectures possibles pour avoir une option de rejet. Ces architectures peuvent aussi bien être utilisées pour du rejet de distance que du rejet de confusion (voir section 2.4). Il faut alors choisir les fonctions de confiance, ou les classes de rejet, adéquates. Une option de rejet peut être basée sur :

— un seuil de rejet sur un score de confiance ;
— une classe de rejet intégrée à l'apprentissage ;
— un classifieur de rejet séparé.

2.4.5.1 Seuil de rejet sur un score de confiance

Utiliser un seuil de rejet sur un score de confiance est l'architecture la plus répandue pour avoir une option de rejet [Chow, 1970] [Dubuisson and Masson, 1993]. Le score de confiance peut simplement correspondre à la probabilité d'appartenance à la meilleure classe, ou être une mesure de confiance plus complexe [Frélicot et al., 1995].

L'avantage de l'utilisation d'un seuil de rejet est sa simplicité, qui permet d'apprendre le seuil avec peu de données. Cette approche peut être étendue par l'utilisation de plusieurs seuils (généralement un seuil par classe) [Fumera et al., 2000], éventuellement sur plusieurs fonctions de confiances. L'utilisation de plusieurs seuils permet d'affiner la précision de l'option de rejet, mais rend son optimisation beaucoup plus complexe.

2.4.5.2 Classe de rejet intégrée à l'apprentissage

Utiliser une classe de rejet, aussi appelée « modèle de rebut » *(garbage model)* [Lethelier et al., 1995] [Wilpon et al., 1990], implique d'ajouter une autre classe au problème de classification initial. Cela complique l'apprentissage car la classe de rejet doit entourer le modèle de connaissance du classifieur et s'insérer sur les frontières de décisions entre les classes initiales. Cette classe de rejet est donc relativement complexe à modéliser, et nécessite un certain nombre de données d'apprentissage pour pouvoir être correctement apprise.

2.4.5.3 Classifieur de rejet séparé

Utiliser un classifieur de rejet séparé [Pitrelli et al., 2006] permet de ne pas compliquer la tâche de classification initiale en ayant un classifieur dédié à la tâche de rejet. Ce classifieur dédié peut ainsi se concentrer sur les zones d'incertitude et les limites du classifieur principal.

Le classifieur de rejet peut soit utiliser le même jeu de caractéristiques que le classifieur principal, soit utiliser des fonctions de confiance issues du classifieur principal. L'utilisation d'un classifieur sur des fonctions de confiance se rapproche de l'utilisation de seuils multiples, mais permet d'avoir des frontières de rejet plus complexes et plus précises.

Enfin, d'autres architectures sont également possibles, comme l'utilisation de techniques de détection de nouveauté pour faire du rejet de distance [Markou and Singh, 2003b] [Markou and Singh, 2003a], ou dans le cas de l'utilisation de l'architecture spécifique du classifieur [Fumera and Roli, 2002] [Tax and Duin, 2004]. Cependant, construire un modèle de classe poubelle, apprendre un classifieur dédié au rejet, ou utiliser une architecture plus complexe, est assez difficilement envisageable avec le peu de données disponibles dans notre contexte.

Dans le cas où l'option de rejet sert de méthode d'échantillonnage pour l'apprentissage actif, le compromis erreur/rejet dépend également du nombre de données nécessaires pour l'apprentissage et du coût d'étiquetage des données. De plus, lorsque le système évolue dans un environnement dynamique, le compromis erreur/rejet est modifié avec l'évolution du classifieur. Il est donc nécessaire de faire également évoluer l'option de rejet pour maintenir le compromis souhaité. Nous y revenons et le détaillons en section 6.3 puisque c'est vers cette approche que nous allons nous orienter en utilisant une option de rejet évolutive pour prendre la décision d'échantillonnage de l'apprentissage actif.

2.5 Conclusion

L'apprentissage actif est le fait de choisir les données qui seront étiquetées pour être utilisées pour l'apprentissage. Nous avons présenté les différents scénarios d'apprentissage actif possibles : les scénarios de génération d'échantillons, d'échantillonnage de bases de données et d'échantillonnage de flux de données. C'est ce dernier cas qui correspond à notre problème d'apprentissage de commandes gestuelles, où le choix des données à échantillonner doit être effectué en-ligne, donnée par donnée. Nous avons vu plusieurs méthodes d'échantillonnage différentes pour effectuer ce choix, en particulier la méthode de l'incertitude maximale avec l'utilisation de l'option de rejet.

Nous avons défini la notion de confiance du processus de reconnaissance, et l'utilisation d'une option de rejet, qui est souvent utilisée dans les scénarios d'apprentissage actif. Nous avons présenté les deux types de rejet possibles, le rejet de distance et le rejet de confusion, qui sont deux approches complémentaires. Nous avons également vu qu'utiliser une option de rejet implique de faire un compromis entre le taux d'erreur et le taux de rejet.

La méthode de l'incertitude maximale consiste à supposer que les données pour lesquelles le classifieur est le plus incertain sont celles qui lui apporteront le plus d'information. L'option de rejet du classifieur permet de sélectionner « à la volée » les données pour lesquelles l'incertitude du classifieur est élevée. Il faut alors faire un compromis entre le taux d'erreur et le taux d'interaction pour l'étiquetage. Pour améliorer le système efficacement et réduire le taux d'erreur, il est nécessaire de rejeter suffisamment de données, mais cela engendre de nombreuses sollicitations pour l'utilisateur.

L'apprentissage actif est au cœur de la problématique de la reconnaissance de commandes gestuelles. Il est nécessaire de limiter au minimum les sollicitations utilisateur pour étiqueter les données, tout en permettant au classifieur d'apprendre efficacement. Il faut choisir soigneusement les quelques données qui seront étiquetées par l'utilisateur. Plus encore qu'un classifieur performant, il faut donc une méthode d'apprentissage actif précise

et efficace.

Si la plupart des cas d'utilisation de méthodes d'apprentissage actif en-ligne sont sur des flux de données de grandes tailles, le contexte des commandes gestuelles est différent. Dans notre cas, il faut apprendre le plus rapidement possible à partir de très peu de données. Le processus de sélection des données d'apprentissage est alors critique pour le bon fonctionnement du système. Le classifieur utilisé pour reconnaître les commandes gestuelles doit donner accès à suffisamment d'information sur son modèle de connaissance pour permettre un apprentissage actif en-ligne efficace.

Le prochain chapitre passe en revue la littérature dédiée aux classifieurs évolutifs. Nous étudions en particulier les systèmes donnant accès à suffisamment d'information sur leur modèle de connaissance pour permettre la mise en place d'une option de rejet.

Chapitre 3

Panorama des classifieurs évolutifs en-ligne

3.1 Introduction

Pour l'apprentissage de commandes gestuelles, il n'y a que très peu d'exemples disponibles pour l'apprentissage initial. Il faut donc un système en-ligne, qui puisse apprendre incrémentalement en temps réel, pour s'améliorer pendant son utilisation, et donc un classifieur évolutif.

Plusieurs approches sont possibles pour répondre aux problèmes de classification en-ligne, c'est-à-dire sur flux de données [Gama, 2012], et avec des changements de concepts [Gama et al., 2014]. Le classifieur peut être ré-appris intégralement à l'arrivée de nouvelles données, si les nouvelles données arrivent peu souvent ou en groupes, et s'il n'y a pas de changements de concepts. Si les données du flux évoluent, une fenêtre glissante peut alors être utilisée pour que le système s'adapte aux changements de concepts. La solution la plus courante est l'utilisation d'un classifieur évolutif appris en-ligne, c'est-à-dire avec un algorithme d'apprentissage incrémental en temps réel, et une capacité d'oubli pour suivre les changements de concepts. Il est également possible de détecter les changements de concepts, et de déclencher un mécanisme d'adaptation lorsque cela est nécessaire. Si toutes ces approches sont intéressantes, et ont différents intérêts, toutes ne conviennent pas au contexte de l'utilisation de commandes gestuelles.

Une première approche, simple et naïve, est de ré-apprendre intégralement le modèle de connaissance du classifieur lorsque de nouvelles données sont disponibles. Par conséquent, cette approche est très adaptée aux flux où les données arrivent par paquets, ou alors quand peu de reconnaissances sont faites et que le modèle peut être mis à jour juste avant chaque reconnaissance *(lazy learning)* [Aggarwal et al., 2004]. Par contre, cette approche est complètement inadaptée aux flux rapides, qui nécessitent une adaptation efficace et continue. Cette méthode est en effet très coûteuse en matière de complexité spatiale, car il faut stocker toutes les données, et en matière de complexité temporelle, car il faut reconstruire intégralement le modèle du classifieur à chaque fois. Cette approche n'est donc pas du tout adaptée au contexte des commandes gestuelles, qui implique un apprentissage

rapide et continu. De plus, mémoriser toutes les données n'est pas pertinent dans notre situation d'apprentissage en-ligne, avec l'utilisateur dans la boucle, où les données évoluent avec l'apprentissage de l'utilisateur.

Une solution simple pour que le système s'adapte aux changements de concepts est d'utiliser une fenêtre glissante qui contienne les données les plus récentes [Last, 2002]. L'utilisation d'une fenêtre glissante a également l'avantage de limiter à la fois la complexité spatiale et la complexité temporelle. Lorsqu'une fenêtre glissante est utilisée, il faut faire un compromis entre la réactivité du système et sa précision [Bifet and Gavalda, 2007] (voir également section 7.5.2). Une fenêtre courte permet une adaptation rapide aux changements de concepts, mais peut limiter les performances du classifieur, notamment pendant les périodes stationnaires du flux de données. Une fenêtre longue permet de conserver assez d'exemples d'apprentissage pour obtenir de bonnes performances, mais rallonge le temps d'adaptation du classifieur lors des changements de concepts. Dans notre contexte, l'utilisation d'une fenêtre glissante est possible, mais reconstruire le modèle de données du classifieur à chaque fois qu'une nouvelle donnée arrive n'est définitivement pas la manière la plus efficace d'apprendre des commandes gestuelles. Si les ressources disponibles sont faibles (smartphone par exemple) et que les classes sont nombreuses ou complexes, cette méthode ne pourra garantir un apprentissage en temps réel.

Une deuxième approche est d'analyser le flux de données pour détecter les changements de concepts, et de déclencher un mécanisme d'adaptation pour adapter le classifieur le cas échéant. Cette méthode est efficace lorsque les changements sont brusques et occasionnels, et que les périodes stationnaires sont relativement longues entre ces changements de concepts. Différents détecteurs de changements existent, certains sont basés sur une analyse de la distribution des données [Markou and Singh, 2003b], d'autres sont basés sur une analyse des performances du classifieur [Widmer and Kubat, 1996]. Dans notre cas, cette approche n'est pas appropriée car même s'il faut que le système puisse s'adapter aux changements de concepts brusques, il faut surtout qu'il s'adapte à l'évolution progressive et continue du style d'écriture de l'utilisateur.

Un classifieur évolutif a l'avantage de s'adapter continuellement au flux de données, et de suivre les changements de concepts de manière transparente. Les principaux critères auxquels doit répondre un classifieur évolutif ne font pas l'unanimité dans la littérature (voir table 3.1), les plus courants sont présentés ci-dessous :

1 les données ne doivent être utilisées qu'une seule fois, et dans leur ordre d'arrivée,
2 les complexités spatiales et temporelles doivent être faibles et fixées a priori, et en accord avec la vitesse du flux,
3 le modèle doit pouvoir suivre les changements de concepts,
4 le modèle de connaissance obtenu doit être proche de celui qui aurait été obtenu avec un apprentissage batch,
5 le système doit pouvoir être interrogé n'importe quand.

Les critères 1, 2 et 3 sont la définition même de l'apprentissage en-ligne comme nous l'avons présentée en section 1.6. Le critère 4 a peu de sens lorsqu'il y a des changements de concepts puisque le classifieur doit alors évoluer avec le flux. Le modèle obtenu est forcément différent de celui obtenu par apprentissage *batch*, mais peu importe s'il donne de meilleurs résultats.

Critère	1	2	3	4	5
[Fayyad et al., 1996]	✓	✓		✓	
[Hulten et al., 2001]	✓	✓	✓		✓
[Domingos and Hulten, 2001]	✓	✓	✓	✓	
[Stonebraker et al., 2005]	✓	✓	✓		✓

TABLE 3.1 – Comparaison des caractéristiques d'un classifieur évolutif

Le critère 4 correspond en réalité à l'apprentissage incrémental hors-ligne. Le critère 5 correspond quant à lui à la notion d'apprentissage permanent.

L'apprentissage et la reconnaissance de commandes gestuelles nécessitant un classifieur évolutif, nous présentons donc dans ce chapitre les différents systèmes évolutifs en-lignes qui existent dans la littérature. Nous étudions dans quelle mesure ils sont évolutifs et ils peuvent être appris en-ligne. Nous serrons en particulier attentifs à la capacité d'oubli de ces classifieurs, et donc à leur faculté à suivre les changements de concepts. Certains classifieurs simples sont incrémentaux par nature, comme les classifieurs probabilistes que nous détaillons dans la section 3.2, et les systèmes d'appariement de modèles que présenterons en section 3.3. Des classifieurs populaires comme les réseaux de neurones et les séparateurs à vastes marges ont également leurs versions incrémentales que nous présentons en section 3.4 et en section 3.5. Nous présentons également les arbres de décisions incrémentaux en section 3.6, puis les ensembles de classifieurs faibles en section 3.7. Enfin, nous finissons par détailler les classifieurs à base de règles, et en particulier les systèmes flous évolutifs, dans la section 3.8.

3.2 Systèmes linéaires

Cette section présente deux systèmes de classification linéaire très connus : le classifieur bayésien naïf et la régression binomiale. Le premier est un système modélisant la distribution des données (modèle génératif), qui apprend très vite. Le deuxième est une système modélisant les frontières de décision (modèle discriminant), qui permet d'obtenir de meilleures performances avec suffisamment d'exemples d'apprentissages [Ng and Jordan, 2002].

3.2.1 Classifieurs bayésiens naïfs

Le classifieur bayésien naïf [Langley et al., 1992] est un système de classification basé sur le théorème de Bayes qui suppose une indépendance des variables explicatives, c'est-à-dire des caractéristiques des données. C'est un système génératif qui modélise les probabilités des différentes classes ainsi que les probabilités conditionnelles des caractéristiques en fonction des différentes classes. C'est un classifieur simple, mais qui donne de bonnes performances [Hand and Yu, 2001].

L'apprentissage d'un classifieur bayésien naïf consiste à estimer les probabilités conditionnelles des caractéristiques en fonction des différentes classes. Ces probabilités condition-

nelles sont faciles à estimer lorsque les caractéristiques des données sont discrètes, mais plus complexes pour des caractéristiques numériques continues. Une première méthode pour les caractéristiques continues consiste à les discrétiser afin de pouvoir facilement estimer les probabilités conditionnelles [Yang and Webb, 2002]. Une autre méthode est de supposer que les caractéristiques numériques continues suivent une distribution gaussienne, ou parfois une distribution de Cauchy [Sebe et al., 2002]. Des méthodes plus complexes existent également, par exemple celle des noyaux multiples [John and Langley, 1995] qui estime ces distributions de manière non-paramétrique.

L'apprentissage peut facilement être réalisé de manière incrémentale en mettant à jour les statistiques utilisées pour estimer les probabilités conditionnelles. Pour les caractéristiques discrètes, ou discrétisées, il suffit d'incrémenter les compteurs correspondants à l'arrivée de chaque nouvelle donnée d'apprentissage.

Le principal avantage du classifieur bayésien naïf est sa capacité de généralisation et ses bonnes performances prédictives. Un autre avantage, très intéressant dans notre contexte, est sa capacité d'apprentissage à partir de très peu de données. Le classifieur bayésien naïf donne de meilleures performances que de nombreuses méthodes plus complexes quand très peu de données d'apprentissage sont disponibles [Domingos and Pazzani, 1997].

Une version incrémentale sur flux de données, qui utilise des résumés des quantiles pour suivre les évolutions de la distribution statistique des données [Salperwyck et al., 2015], a été conçue pour travailler sur des flux de grandes tailles. Cette méthode utilise également une pondération des caractéristiques, qui est calculée de manière stochastique.

Dans le cas de caractéristiques discrétisées, la principale difficulté pour le rendre en-ligne, c'est-à-dire incrémental sur flux de données, est de rendre le processus de discrétisation en-ligne. IFFD *(Incremental Flexible Frequency Discretization)* [Lu et al., 2006] est une méthode de discrétisation incrémentale permettant de ne pas recalculer toutes les probabilités conditionnelles à l'arrivée de chaque donnée. Le nombre et la taille des intervalles sont flexibles et adaptés incrémentalement. De nouveaux intervalles peuvent également être insérés, et ceux existants peuvent être éclatés au besoin.

3.2.2 Régression binomiale

Contrairement au classifieur bayésien naïf qui est un modèle génératif, la régression binomiale est un modèle discriminant. Le plus connu des modèles de régression binomiale est le modèle logistique, basée sur une distribution de Bernouilli, mais il existe également le modèle probit, basé sur une distribution normale.

La régression binomiale permet d'obtenir un hyperplan séparateur entre deux classes. De nombreux autres systèmes linéaires permettent d'obtenir des hyper-plans séparateurs, comme les perceptrons mono-couche et les machines à vecteurs supports sans noyau par exemple [Yuan et al., 2012].

La régression logistique [Cox, 1958] [Walker and Duncan, 1967] est un modèle de régression où la variable dépendante est catégorielle, ce qui permet d'estimer la probabilité d'une réponse binaire à partir de variables explicatives. C'est donc un système de classification, malgré son appellation de régression binomiale. La régression logistique est un modèle de

faible complexité qui permet l'apprentissage sur de grandes bases de données [Fan et al., 2008]. Cet algorithme n'est cependant pas utilisable pour faire de la classification en-ligne.

Il existe par contre différents algorithmes incrémentaux, dans la littérature dédiée à la régression et au contrôle principalement, qui peuvent être utilisés pour apprendre incrémentalement les coefficients d'une régression binomiale. Le principe du filtrage de Kalman [Kalman, 1960] peut par exemple être utilisé.

Les systèmes d'inférence floue d'ordre un (voir section 3.8.2) utilisent le plus souvent l'algorithme des moindres carrés récursifs [Haykin, 2001] pour optimiser les fonctions linéaires définissant les hyper-plans séparateurs des conclusions des règles. Cet algorithme est incrémental, mais pas en-ligne dans le sens où il n'est pas capable de suivre les changements de concepts.

Suivre les changements de concepts nécessite alors d'intégrer de l'oubli dans le processus d'optimisation des moindres carrés récursifs (RLS). L'intégration d'oubli dans cet algorithme est un problème qui a beaucoup été étudié dans la littérature dédiée au contrôle de systèmes physiques complexes [Lughofer and Angelov, 2011], mais qui reste sans solution universelle. Le principal problème est de ne pas faire diverger la matrice de covariance utilisée par l'algorithme *(covariance wind-up problem)*, ce qui cause un oubli catastrophique et l'effondrement du système. La principale propriété recherchée dans une méthode d'oubli pour l'algorithme des moindres carrés récursifs est que sa matrice de covariance soit bornée [Salgado et al., 1988] [Parkum et al., 1992]. La borne supérieure permet de garantir la stabilité du système en évitant le problème d'explosion de la matrice de covariance souvent rencontré. La borne inférieure permet quant à elle de garantir l'évolutivité du système en maintenant le gain de l'apprentissage. Nous détaillons davantage ce problème dans la section 4.5, lorsque nous détaillons l'apprentissage des conclusions du système d'inférence floue *Evolve* avec l'algorithme des moindres carrés récursifs.

3.3 Systèmes d'appariement de modèles

3.3.1 Méthode des k-plus-proches-voisins

Les algorithmes de classification basés sur la méthode des k-plus-proches-voisins sont connus depuis très longtemps [Cover and Hart, 1967] [Aha et al., 1991]. Leur apprentissage est passif, ils ne font que stocker les exemples d'apprentissage, en faisant éventuellement une sélection des exemples conservés [Brighton and Mellish, 2002], et diffèrent tout autre traitement à l'étape de reconnaissance. Ces méthodes ne construisent pas de modèle de connaissance global, mais se contentent d'accumuler de l'information localement en stockant les exemples d'apprentissage. La reconnaissance de données inconnues se fait ensuite en cherchant les k exemples d'apprentissage les plus proches, et en sélectionnant l'étiquette majoritaire.

Les principales différences entre les différentes méthodes des k-plus-proches-voisins résident principalement dans l'algorithme de recherche des plus proches voisins, et en particulier de la métrique utilisée. Plusieurs métriques sont possibles, et la métrique peut même être apprise au fur et à mesure [Robnik-Šikonja and Kononenko, 2003], ou alors en

la formulant comme un problème d'optimisation où les exemples de même classe doivent être les plus proches possibles et en même temps les plus loin possible de ceux des autres classes [Globerson and Roweis, 2005] [Weinberger and Saul, 2009]. Différents algorithmes de recherche des plus proches voisins existent également afin d'accélérer le processus de recherche comme AESA *(Approximation Elimination Search Algorithm)* [Vidal, 1994] et ses extensions [Moreno-Seco et al., 2002].

L'avantage de la méthode des k-plus-proches-voisins est qu'elle est par définition incrémentale. Sur de très grandes bases de données, l'apprentissage n'est pas complexe, puisqu'il est passif, mais il nécessite par contre beaucoup de mémoire. La reconnaissance devient vite problématique lorsqu'il faut trouver les plus proches voisins d'un exemple au milieu d'une grande base de données [Sankaranarayanan et al., 2007]. Le deuxième inconvénient de la méthode des k-plus-proches-voisins est sa sensibilité aux données aberrantes. Les performances décroissent rapidement avec le niveau de bruit. Ces deux problèmes peuvent facilement être résolus en supprimant des exemples, c'est-à-dire en intégrant de l'oubli dans le processus d'apprentissage [Brighton and Mellish, 2002].

L'utilisation d'oubli est primordiale pour limiter les besoins en mémoire, et les temps de réponse lors d'une demande de classification. Lors de la reconnaissance de flux de données, le temps de réponse peut en effet être une valeur critique suivant les contextes. L'oubli garantit en même temps la capacité de la méthode des k-plus-proches-voisins à suivre les changements de concepts.

L'intégration d'oubli dans ce type de méthode se fait en ne conservant qu'une partie des exemples d'apprentissage. Différentes options sont alors possibles, la plus simple étant l'utilisation d'une fenêtre glissante ne conservant que les données les plus récentes. D'autres approches effectuant une sélection plus fine des exemples conservés donnent de meilleures performances. Elles prennent en considération non seulement la pertinence temporelle mais également la pertinence spatiale et la pertinence vis-à-vis des concepts courant [Widmer, 1994]. Dans tous les cas, il faut faire un compromis entre la précision et l'efficacité du système *(accuracy-efficiency trade-off)* [Beringer and Hullermeier, 2007].

D'autre approches sont également possibles pour répondre aux contraintes de classification sur flux, comme par exemple la discrétisation de l'espace d'entrée pour réduire la taille de l'espace de recherche [Law and Zaniolo, 2005]. L'utilisation de plusieurs classifieurs avec différentes discrétisations permet ensuite d'obtenir de bonnes performances tout en réduisant les besoins de stockage.

L'utilisation d'oubli rend la méthode des k-plus-proches-voisins davantage sensible au bruit. Néanmoins, l'utilisation de tests statistiques pour détecter les données aberrantes *(outliers)* peut permettre de maintenir les performances malgré la présence de bruit [Aha et al., 1991].

3.3.2 Alignement de séquences

Le principe de l'alignement de séquences est d'obtenir une mesure de différence entre deux séquences, et ainsi de faire de la reconnaissance en comparant une séquence inconnue à des références étiquetées. La méthode d'alignement de séquence la plus connue est

l'alignement temporel dynamique *(Dynamic Time Warping : DTW)*, qui est une méthode d'alignement élastique, c'est-à-dire non-linéaire. Cette technique a originellement été introduite pour faire de la reconnaissance de la parole [Sakoe and Chiba, 1978], mais elle s'applique très bien à la reconnaissance d'écriture cursive [Tappert, 1982] [Uchida and Sakoe, 2005].

Le point le plus délicat est le choix des prototypes de symboles, qui servent de références étiquetées pour la reconnaissance. Ces prototypes doivent être les plus représentatifs possibles pour garantir de bonnes performances de reconnaissance.

L'alignement de séquences est une technique qui peut être utilisée en-ligne lorsque le nombre de prototypes reste raisonnable. Différentes techniques d'optimisation de l'algorithme d'alignement temporel dynamique *(DTW)* ont été proposées, comme l'utilisation d'une modélisation éparse [Al-Naymat et al., 2012]. Il suffit alors d'adapter les prototypes de manière en-ligne pour que le système suive les changements de concepts lorsque le style d'écriture de l'utilisateur évolue. Malgré cela, l'utilisation de l'alignement temporel dynamique pour la reconnaissance d'écriture manuscrite en-ligne n'est que très peu présente dans la littérature.

Un système de reconnaissance adaptatif, utilisant six mesures de distances basées sur l'alignement temporel dynamique, a été proposé [Vuori et al., 2001]. Ce système s'adapte à l'utilisateur final, et passe progressivement d'un système omni-utilisateur à un système mono-utilisateur. Le système s'adapte en modifiant les prototypes incrémentalement par quantification vectorielle *(vector quantization)*. De nouveaux prototypes peuvent être créés et les prototypes existants peuvent être désactivés. Cette adaptation du système est réalisée de manière en-ligne, et auto-supervisée pour ne pas déranger l'utilisateur.

Le « 1$ classifier » [Wobbrock et al., 2007] est un autre classifieur basé sur de l'alignement de séquences. C'est un système extrêmement simple, dont l'algorithme tient en une centaine de lignes de code, mais qui donne d'aussi bonnes performances que l'alignement temporel dynamique *(DTW)* sous certaines conditions. L'alignement n'est pas élastique dans ce système, et est donc réalisé après un certain nombre de normalisations, ce qui limite son champ d'application.

Si l'alignement de séquences est généralement fait entre les séquences de points, il peut également être fait entre d'autres séries temporelles comme la courbure [Ahn et al., 2009]. D'autres mesures de distance que l'alignement dynamique temporel peuvent également être utilisées pour faire la mise en correspondance avec les prototypes [Golubitsky and Watt, 2010]. Les distances obtenues avec ces différents types de mise en correspondance peuvent aussi servir de caractéristiques pour un classifieur statistique comme une régression binomiale [Mitoma et al., 2005].

3.4 Réseaux de neurones

Les réseaux de neurones artificiels sont une famille de modèles inspirés des neurones biologiques [Rosenblatt, 1958]. Ils sont composés d'un ensemble de neurones, organisés en différentes couches, qui laissent plus ou moins passer l'information en fonction de leurs entrées. Les réseaux de neurones multi-couches sont des approximateurs universels [Hornik

et al., 1989] qui sont utilisés en classification depuis de nombreuses années [Zhang, 2000].

Cette section présente trois types de réseaux de neurones : les perceptrons, les réseaux à fonction de base radiale, et les réseaux auto-organisés.

3.4.1 Perceptrons

L'apprentissage des perceptrons multi-couches, le plus souvent par rétro-propagation du gradient de l'erreur, est par définition incrémental [Bishop, 1995]. Le problème réside dans la vitesse d'apprentissage de ce type de modèle, qui nécessite souvent plusieurs « époques » d'apprentissage (passage des données). Une époque représente une présentation complète de l'ensemble du jeu de données d'apprentissage. Dans notre cas, l'utilisation de plusieurs époques est impossible car l'apprentissage en-ligne implique une seule présentation des données.

Des réseaux de neurones ont cependant été utilisés en-ligne (une seule époque) avec succès pour des problèmes de contrôle de systèmes physiques complexes. L'algorithme *backpropagation-decorrelation* permet l'apprentissage en-ligne de réseaux de neurones récurrents, avec une complexité linéaire, pour faire de la prédiction de séries temporelles [Steil, 2004]. Un réseau hybride avec un algorithme d'apprentissage à plusieurs étages a donné d'excellents résultats sur un problème de contrôle de feux de signalisation pour optimiser la circulation [Choy et al., 2006].

Les rares perceptrons en-lignes a avoir été utilisés pour des problèmes de classification sont en fait des perceptrons mono-couches [Saad, 1998]. Les perceptrons mono-couches sont des séparateurs linéaires, et leurs performances sont donc limitées sur des problèmes de classification complexes.

3.4.2 Réseaux à fonction de base radiale

Les réseaux de neurones à fonction de base radiale sont des réseaux de neurones utilisant des fonctions de bases radiales comme fonctions d'activation. Ces réseaux, utilisant souvent la logique floue, sont très proches des systèmes d'inférence floue qui sont présentés en section 3.8.2.

Le plus courant dans les réseaux à fonction de base radiale est l'utilisation d'hyper-sphères pour représenter les données dans l'espace d'entrée. EFuNN *(Evolving Fuzzy Neural Networks)* [Kasabov, 1998] est un réseau de neurones évolutif, basé sur des hyper-sphères et utilisant la logique floue, destiné au contrôle et à la prédiction en-ligne. Sa vitesse d'apprentissage est trois à six fois plus rapide que celle d'un perceptron multi-couches utilisant l'algorithme de rétro-propagation du gradient de l'erreur. EFuNN est un des premiers systèmes évolutifs, qui puisse être appris en-ligne en adaptant ses paramètres et sa structure, et qui supporte l'ajout de classes et de caractéristiques en-ligne.

Une autre possibilité est l'utilisation d'hyper-rectangles comme dans le modèle NGE *(Nested Generalized Exemplar)* [Salzberg, 1991]. GFMM *(General Fuzzy Min-Max Neural Network)* [Gabrys and Bargiela, 2000] est une sorte de réseau de neurones où les neurones d'entrée sont aussi reliés à ceux de sortie par des hyper-rectangles. Une fonction d'activation

floue est ensuite utilisée pour évaluer l'appartenance des données aux différents hyper-rectangles.

3.4.3 Réseaux auto-organisés

Les réseaux auto-organisés sont une forme particulière de réseaux de neurones. Ils se construisent de manière à modéliser l'ensemble des données. Chaque neurone représente un sous-ensemble particulier des données, réunies par un certain nombre de caractéristiques communes. Les réseaux auto-organisés permettent de discrétiser l'espace de représentation des données, en le séparant en différentes zones qui sont chacune représentée par un vecteur référent, comme une technique de quantification vectorielle. Les deux principaux types de réseaux auto-organisés sont les cartes auto-organisatrices SOM *(Self Organizing Maps)* [Kohonen, 1990] et les réseaux ART *(Adaptive Reasonance Theory)* [Carpenter and Grossberg, 1988].

Les cartes auto-organisatrices (SOM) produisent une discrétisation de l'espace des données, qui réduit sa dimension tout en conservant les propriétés topologiques initiales [Kohonen, 1990]. Ces cartes sont apprises par apprentissage compétitif, par opposition à l'apprentissage par correction d'erreur des réseaux de neurones classiques. L'apprentissage compétitif est une forme d'apprentissage, supervisé ou non, au cours duquel les différents neurones rivalisent pour répondre à des sous-ensembles de données. L'algorithme de quantification vectorielle LVQ *(Learning Vector Quantization)* [Kohonen, 1997] est également une forme de carte auto-organisatrice. D'autres versions de réseaux auto-organisés existent également comme ESOINN (Enhanced Self-Organizing Incremental Neural Network) [Furao et al., 2007] par exemple.

Les cartes auto-organisatrices peuvent en particulier être utilisées en-ligne. La version évolutive ESOM *(Evolving SOM)* [Deng and Kasabov, 2003] donne de très bonnes performances de reconnaissance en-ligne, avec un temps d'apprentissage très court. Les cartes auto-organisatrices peuvent aussi servir à faire de la détection de nouveauté en-ligne [Feng et al., 2010], où elles permettent de réduire le taux de fausses alarmes. Un classifieur incrémental, basé sur une carte auto-organisatrice SOM, a notamment été appliqué à la reconnaissance d'écriture cursive [Morasso et al., 1993].

La théorie de la résonance adaptative ART *(Adaptive Reasonance Theory)* décrit une famille de réseaux de neurones, pour des problèmes de régression et de classification, avec un apprentissage supervisé ou non. ARTMAP est un système d'apprentissage et de classification en temps réel, basé sur une paire de modules ART, destiné aux flux de données [Carpenter et al., 1991b]. Ces algorithmes ont été étendus avec les principes de la logique floue [Carpenter et al., 1991c] [Carpenter et al., 1992], ce qui permet d'améliorer leur capacité de généralisation. Plus récemment, une hybridation des réseaux ART avec les réseaux de neurones GRNN *(Generalized Regression Neural Network)* a montré de bonnes performances de classification en-ligne [Yap et al., 2008].

L'avantage des réseaux de neurones ART est qu'ils sont capables d'apprendre de nouveaux modèles (plasticité), tout en conservant les connaissances précédemment apprises (stabilité). Par rapport aux perceptrons multi-couches, ils ont l'avantage de converger plus

rapidement, d'obtenir un meilleur taux de reconnaissance, et surtout de pouvoir être appris en-ligne [Tay and Khalid, 1997]. Un inconvénient des réseaux ART est le fort impact de l'ordre de présentation des données sur le modèle obtenu.

3.5 Séparateurs à Vaste Marge (SVM)

En apprentissage automatique, les séparateurs à vaste marge (SVM), ou machines à vecteurs supports, sont connus de tous. Ce sont des classifieurs très puissants, issus de la théorie de l'apprentissage statistique [Vapnik, 1995], qui permettent de séparer deux catégories avec la plus grande marge possible. Les SVM sont notamment très efficaces pour modéliser les problèmes non-linéaires grâce à l'astuce des noyaux *(kernel trick)* [Boser et al., 1992], qui permet de linéariser le problème en travaillant dans un espace de dimension supérieure. L'algorithme original des SVM a été proposé avec des marges dures, et a ensuite été étendu avec des marges souples [Cortes and Vapnik, 1995].

Les SVM ne sont que peu utilisés sur les flux de données car ce ne sont pas des classifieurs incrémentaux. L'apprentissage du modèle d'un SVM nécessite en effet la résolution d'un problème d'optimisation quadratique, ce qui requiert l'accès à toutes les données et qui est très coûteux en matière de temps. Pour un ensemble d'apprentissage de n données, la complexité classique de l'apprentissage d'un SVM est en $o(n^3)$ en matière de temps, et en $o(n^2)$ en matière d'espace.

La mise à jour du modèle d'un SVM étant coûteuse, cette opération n'est pas réalisée à l'arrivée de chaque nouvelle donnée. Il existe trois méthodes principales de sélection des données pour l'apprentissage incrémental :

— la méthode du partitionnement ;
— la méthode des erreurs ;
— la méthode des marges.

Dans tous les cas, une fois les données sélectionnées pour mettre à jour le modèle, elles sont utilisées avec l'ensemble des vecteurs supports, pour apprendre le nouveau modèle. Les nouveaux vecteurs supports sont alors sauvegardés, et les autres données sont supprimées.

La méthode du partitionnement [Mitra et al., 2000] s'applique naturellement aux données arrivant par paquets, mais peut éventuellement être utilisée en créant des paquets artificiellement. Elle consiste simplement à utiliser tout le nouveau paquet de données pour mettre à jour le modèle.

La méthode des erreurs [Syed et al., 1999a] consiste à ne conserver que les données qui sont mal classifiées par le modèle actuel. La mise à jour du modèle est ensuite faite périodiquement, lorsqu'il y a suffisamment de nouvelles données.

La méthode des marges consiste à conserver toutes les données qui tombent dans les marges définies par les vecteurs supports du modèle courant. Le modèle est là encore mis à jour périodiquement.

3.5.1 Méthodes incrémentales paquet par paquet

Plusieurs approches ont été proposées afin de limiter la complexité et d'apprendre des SVM sur de très grands jeux de données. La méthode « diviser pour régner » permet par exemple d'apprendre un SVM à partir de jeux de données plus conséquents en séparant le problème en sous problèmes [Dong et al., 2005]. La *Core Vector Machine* (CVM) utilise la méthode des *Minimum Enclosing Balls* (MEB) qui permet d'approximer la solution optimale [Tsang et al., 2005]. La complexité spatiale est fixée a priori, et est indépendante de la taille du jeu de données, mais il faut alors faire un compromis entre le temps d'apprentissage du modèle et sa précision. Ces approches ne sont cependant pas incrémentales.

Des algorithmes d'apprentissage incrémental par paquets de SVM ont d'abord été proposés, notamment pour les très grands jeux de données [Osuna et al., 1997] [Domeniconi and Gunopulos, 2001]. L'approche *SV-incremental* construit un nouveau modèle de connaissance à partir des nouvelles données et des vecteurs supports de l'ancien modèle [Syed et al., 1999b] [Syed et al., 1999a]. Cette approche est incrémentale paquet-par-paquet *(block-by-block)*, et les anciennes données ne sont pas reconsidérées. Elle est efficace lorsque les données arrivent par paquets, mais pas pour les flux où les données arrivent une par une.

L'algorithme *SV-L-incremental* étend cette approche en ajoutant une pondération des anciens vecteurs supports. Cela donne un modèle de connaissances final plus proche du modèle obtenu avec un apprentissage batch [Ruping, 2001]. Cette approche limite les effets sur le modèle final qu'auraient d'éventuels changements de concepts non-désirés lors d'un apprentissage incrémental sur une grande base de données. Cette caractéristique n'est cependant pas du tout intéressante dans notre contexte où il est au contraire souhaitable de suivre tous les changements de contexte.

Ces approches ne sont cependant pas incrémentales donnée par donnée et ne sont donc pas compatibles avec la problématique d'apprentissage sur flux où les données doivent être apprises une par une.

3.5.2 Méthodes incrémentales donnée par donnée

Un algorithme incrémental (et décrémental) donnée par donnée, et non plus paquet par paquet, a ensuite été proposé [Cauwenberghs and Poggio, 2001]. Cette technique permet d'obtenir un modèle exact, c'est-à-dire identique au modèle obtenu de manière batch, mais appris donnée par donnée. L'idée de cette approche est de maintenir les conditions de Kuhn-Tucker sur toutes les données déjà vues, et d'ajouter de manière « adiabatique » le nouvel exemple à la solution courante (voir [Cauwenberghs and Poggio, 2001] pour plus de détails).

Une extension de cet algorithme permet d'améliorer l'efficacité de l'apprentissage [Laskov et al., 2006]. Les auteurs testent également une stratégie d'apprentissage actif [Warmuth et al., 2003] pour accélérer davantage l'apprentissage. Cependant, la complexité de l'apprentissage d'un nouvel exemple est en $o(n^2)$, où n est le nombre d'exemples d'apprentissage passés. Cet algorithme devient donc de plus en plus coûteux avec le temps, et de manière plus générale n'intègre pas d'oubli, ce qui le rend inadapté à l'apprentissage

en-ligne sur flux.

LASVM [Bordes and Bottou, 2005] est une méthode incrémentale d'apprentissage de SVM, qui permet de mettre à jour le modèle à l'arrivée d'une nouvelle donnée. Cette méthode a été conçue comme une manière d'approximer rapidement la solution optimale en matière de plus vaste marge. LASVM permet d'obtenir, en seulement une époque, c'est-à-dire en un seul passage sur les données, un modèle de connaissance compétitif avec les algorithmes classiques avec plusieurs époques quant à sa précision. Son avantage est que le temps de calcul nécessaire est largement inférieur aux algorithmes classiques. La complexité de LASVM est en $o(n)$, où n est le nombre d'exemples d'apprentissage, et des bornes théoriques permettent de garantir la qualité de l'approximation obtenue [Usunier et al., 2010]. LASVM est donc un algorithme d'apprentissage incrémental de SVM, mais difficilement utilisable en-ligne puisque sa complexité augmente avec le temps. Une stratégie d'apprentissage actif a également été présentée pour réduire drastiquement le temps d'apprentissage de LASVM [Bordes et al., 2005]. Cette stratégie permet également d'obtenir des modèles de connaissances plus compacts, et avec une précision équivalente, voire des capacités de généralisation supérieures.

L'algorithme StreamSVM [Rai et al., 2009] est un algorithme d'apprentissage en-ligne de SVM, avec une complexité poly-logarithmique pour l'apprentissage de chaque nouvelle donnée. StreamSVM est basée sur le principe des *minimum enclosing balls* (MEB), mais apprend en une seule passe sur les données. StreamSVM permet d'obtenir des taux de reconnaissance similaires à ceux des algorithmes d'apprentissage *batch*, mais avec une complexité poly-logarithmique.

Bien qu'incrémentales, ces différentes approches restent calculatoirement coûteuses, et leur utilisation en-ligne dépend de la vitesse du flux de données. Il manque également une caractéristique essentielle pour que ces systèmes soient réellement évolutifs en-lignes : la capacité d'oubli pour suivre les changements de concepts. L'utilisation d'une fenêtre glissante pour détecter les changements de concepts est bien sûr possible [Klinkenberg and Joachims, 2000], mais loin d'être optimale.

L'ajout de classes est également complexe puisque les SVM multi-classes sont en fait des combinaisons de SVM bi-classes. À l'ajout d'une nouvelle classe, il faut alors créer autant de nouveaux classifieurs qu'il y avait de classes dans le système. Ces nouveaux systèmes ne commencent à être opérationnels qu'après l'apprentissage de plusieurs exemples de la nouvelle classe et des anciennes. Pour que la nouvelle classe soit utilisable dès son ajout, il faut maintenir une mémoire des données des anciennes classes.

3.6 Arbres de décision

Les arbres de décision sont des outils d'aide à la décision qui permettent d'aboutir à une décision à partir d'une succession de choix. En apprentissage automatique, il existe des algorithmes permettant de construire des arbres de décision pour répondre aux problèmes de classification. L'arbre représente alors le modèle de connaissance du classifieur, où les nœuds sont des conjonctions de caractéristiques, et les feuilles sont les étiquettes des différentes classes. Les algorithmes d'apprentissage d'arbres de décision les plus connus sont

CART [Breiman et al., 1984] et C4.5 [Quinlan, 1993].

Le problème de construction d'un arbre de décision optimal est un problème NP-complet [Hyafil and Rivest, 1976], même pour des concepts simples. Les algorithmes d'apprentissage sont donc basés sur des heuristiques, ou des algorithmes gloutons. L'apprentissage incrémental d'un arbre de décision est donc complexe, et rien ne garantit l'optimalité du modèle obtenu. Les arbres sont des modèles qui sont propices au sur-apprentissage, il est donc nécessaire d'avoir un mécanisme d'élagage pour garantir une certaine capacité de généralisation. Un autre inconvénient des arbres est leur difficulté à modéliser certains concepts comme le XOR (ou exclusif) par exemple.

Il existe plusieurs algorithmes d'apprentissage incrémental d'arbres de décision. L'algorithme CART a été étendu pour supporter l'apprentissage incrémental [Crawford, 1989]. ITI *(Incremental Tree Inducer)* [Utgoff et al., 1997] est une méthode de la famille de l'algorithme C4.5 qui garantit l'obtention du même modèle peu importe l'ordre des données, que l'apprentissage soit incrémental ou non. Le coût de construction incrémentale du modèle est cependant plus important que lorsque toutes les données sont disponibles.

3.6.1 Arbres de décision incrémentaux

Il existe plusieurs classifieurs incrémentaux, destinés aux très grandes bases de données ou au flux, basés sur des arbres de décision. La majorité de ces systèmes n'est cependant pas réellement en-ligne, car incapable de gérer les changements de concepts.

Un des algorithmes de référence pour la construction incrémentale d'arbres de décision sur des flux de données est VFDT : *Very Fast Decision Trees* [Domingos and Hulten, 2000]. VFDT est un algorithme incrémental, sans mémoire des données, ce qui fait qu'il est adapté aux flux, mais il ne gère pas les changements de concepts. Il est conçu pour pouvoir apprendre à partir de très grandes bases de données, sur lesquelles l'algorithme C4.5 est impraticable. L'algorithme VFDT originel ne gère que les attributs catégoriels. VFDTc [Gama et al., 2003] est une version de VFDT qui accepte les attributs numériques continus, et qui intègre des classifieurs Bayésiens naïfs dans les feuilles pour améliorer la précision.

IADEM [Ramos-Jiménez et al., 2006] est un algorithme d'apprentissage incrémental d'arbre de décision utilisant la borne de Hoeffding. Cette méthode est basée sur un taux d'erreur maximum, qui est passé en paramètre, et une confiance associée. L'algorithme fait ensuite croitre l'arbre jusqu'à obtenir un modèle suffisamment précis. Par rapport à VFDT, IADEM donne en général des arbres de taille plus réduite, mais avec une précision légèrement moins bonne. IADEMc [del Campo-Avila et al., 2006] est, comme VFDTc, la version permettant de gérer les attributs numériques et intégrant des classifieurs Bayésiens naïfs dans les feuilles.

Un arbre bayésien [Seidl et al., 2009], utilisant une hiérarchie de densités de probabilités, a été proposé pour faire de l'apprentissage permanent *(anytime learning)*. Il utilise un algorithme d'apprentissage incrémental, qui exploite le temps disponible pour optimiser l'apprentissage, tout en restant interruptible à tout moment pour la reconnaissance. Cet arbre bayésien n'est cependant pas prévu pour suivre les changements de concept, ce qui

n'en fait pas un algorithme réellement en-ligne.

3.6.2 Arbres de décision évolutifs

Il existe également des classifieurs en-ligne, gérant les changements de concepts, basés sur des arbres de décision. Même si ces systèmes peuvent s'adapter aux changements de concepts, la restructuration d'un arbre lors d'un changement de concept reste relativement coûteuse.

CVFDT [Hulten et al., 2001] est une extension de VFDT qui gère les changements de concepts, et en fait donc un réel algorithme en-ligne. Lorsque qu'un changement de concepts est détecté, des sous-arbres alternatifs sont construits en parallèle, et ils remplacent le sous-arbre original lorsqu'ils donnent de meilleurs résultats.

UFFT *(Ultra Fast Forest Trees)* [Gama et al., 2005] est un algorithme d'induction d'un système de classification *one-versus-one*, basé sur des arbres de décision. UFFT est un système en-ligne, qui apprend à partir de nouveaux exemples en temps constant, et qui s'adapte aux changements de concepts. Des classifieurs bayésiens naïfs sont notamment utilisés dans les nœuds pour détecter les changements dans les distributions des données, et élaguer le sous-arbre correspondant.

L'utilisation d'une architecture *one-vesus-all* (OVA) est également possible [Hashemi and Yang, 2009]. OVA est une architecture optimisée pour l'apprentissage sur flux de données comportant des changements de concepts. Elle est présentée avec des arbres de décision, mais pourrait également être utilisée avec d'autres types de classifieur. Un des avantages de cette architecture est qu'elle permet l'ajout de nouvelles classes (plasticité) sans altérer la reconnaissance de celles existantes (stabilité). Par contre, ces méthodes permettent difficilement le suivi des changements de concepts, qui nécessitent souvent une restructuration de l'arbre.

3.7 Ensemble de classifieurs faibles

Le principe des ensembles de classifieurs faibles est de combiner plusieurs modèles, potentiellement simplistes, pour obtenir un modèle global plus performant. Le terme « ensemble de classifieurs » est utilisé pour désigner la combinaison de nombreux modèles issus du même type de classifieur simpliste – dit faible.

Les ensembles de classifieurs faibles sont intéressants lors de l'apprentissage en-ligne en présence de changements de concepts. Les ensembles passent facilement à l'échelle, et sont faciles à paralléliser, pour apprendre en-ligne sur des flux de données de grandes tailles. Ils peuvent s'adapter facilement lors des changements de concepts en supprimant de l'ensemble les classifieurs devenus obsolètes, tout en conservant ceux encore pertinents. La diversité des ensembles de classifieurs leur permet ainsi d'obtenir des meilleurs taux de reconnaissance lors des changements de concepts [Minku et al., 2010].

3.7.1 *Bagging* et *Boosting* en-ligne

Les ensembles de classifieurs faibles se sont démocratisés avec l'arrivée du *bagging* [Breiman, 1996] et du *boosting* [Freund et al., 1996], deux techniques d'apprentissage d'ensembles très puissantes. Ces méthodes possèdent des fortes garanties théoriques, et ont montré d'excellents résultats expérimentaux. Cependant, lors de l'apprentissage en-ligne, l'échantillonnage des données d'apprentissage est impossible, et il faut alors utiliser les données au fur et à mesure de leur arrivée.

Des versions en-lignes du *bagging* ont été développées pour les flux de données de grandes tailles en présence de changements de concepts [Bifet et al., 2009]. Le *leveraging bagging*, qui utilise une pondération aléatoire des données, donne de meilleurs performances de classification que les forêts aléatoires sur de grands jeux de données contenant des changements de concepts [Bifet et al., 2010].

Au contraire, des méthodes de *bagging* et de *boosting* légères et rapides ont été développées pour les flux de données rapides [Chu and Zaniolo, 2004]. Le boosting d'arbres de décision ID4 a notamment montré d'excellentes performances pour la prédiction de branches d'exécution sur les micro-processeurs, ce qui nécessite une grande efficacité et implique des ressources très limitées [Fern and Givan, 2003]. Bien que plus rapides qu'en version batch, le *bagging* et le *boosting* en-ligne restent cependant moins rapides qu'un classifieur Bayésien naïf et qu'un perceptron multi-couches [Oza, 2005].

Il intéressant de noter que l'algorithme ILClass [Almaksour and Anquetil, 2013], destiné à l'apprentissage de systèmes d'inférence floue en-lignes (voir section 3.8.2), s'inspire des principes du *boosting* pour améliorer la vitesse de l'apprentissage en-ligne.

3.7.2 Ensemble évolutifs

Il existe différentes façons d'obtenir un classifieur évolutif par combinaison d'un ensemble de classifieurs faibles [Fern and Givan, 2003] [Kuncheva, 2004]. Les classifieurs faibles peuvent être évolutifs, la méthode de combinaison peut être évolutive, et l'ensemble lui-même peut être évolutif. Ces différentes méthodes peuvent également être combinées entre elles [Bouchachia, 2011].

Tout d'abord, les classifieurs faibles eux-mêmes peuvent être évolutifs [Iglesias et al., 2013]. Les différents éléments de l'ensemble évoluent alors en parallèle lors de l'arrivée de nouvelles données. Il faut alors maintenir une certaine diversité dans l'ensemble pour garantir de bonnes performances de généralisation. Ce cas de figure se rapproche des systèmes de classifieurs multiples combinant des classifieurs forts.

Ensuite, la combinaison des différents classifieurs faibles peut être modifiée en-ligne [Elwell and Polikar, 2011]. Les poids des différents classifieurs faibles évoluent alors dynamiquement en fonction de l'évolution du contexte [Kolter and Maloof, 2003].

Enfin, l'ensemble lui-même peut être évolutif, en ajoutant et supprimant des classifieurs faibles. Parmi ce type d'ensembles évolutifs, il y a les ensembles « en-lignes par paquets », pour les très grand flux de données [Wang et al., 2003]. Ces méthodes construisent de nouveaux classifieurs à l'arrivée de nouveaux paquets de données, qui sont ensuite incorporés à l'ensemble, en utilisant éventuellement une heuristique de remplacement [Street and

Kim, 2001]. Il est également possible de conserver les classifieurs retirés de l'ensemble pour pouvoir les ré-utiliser dans le cas de concepts récurrents [Jackowski, 2013].

Le principal problème auquel doivent faire face les ensembles de classifieurs évolutifs est le problème de « majorité incompétente » *(outvoting problem)* lors de l'introduction de nouvelles classes. Les anciens classifieurs ne connaissant pas les nouvelles classes, ils vont perturber la reconnaissance des nouvelles classes s'ils ne sont pas écartés par un mécanisme de rejet de distance (comme présenté section 2.4) [Muhlbaier et al., 2009].

3.8 Systèmes à base de règles

Le principe des systèmes à base de règles est de regrouper un ensemble de règles de décision qui permettent de faire le bon choix suivant le contexte. Ces règles sont chacune composées d'une prémisse et d'une conclusion. À l'origine les règles étaient définies par des experts, mais il existe maintenant de nombreux algorithmes pour apprendre un ensemble de règles à partir d'exemples étiquetés (c'est-à-dire dont la classe est connue).

3.8.1 Systèmes experts évolutifs

Les systèmes à base de règles sont nombreux, mais ceux pouvant fonctionner en-ligne et suivre les changements de concepts sont plus rares.

Tout d'abord il y a le cadre *(framework)* FLORA [Widmer and Kubat, 1996], qui est une famille d'algorithmes capables de s'adapter aux changements de concepts. Ces algorithmes sont basés sur l'utilisation d'une fenêtre de données contenant des exemples décrivant le concept actuel. En particulier l'algorithme FLORA2 est capable d'ajuster dynamiquement la taille de la fenêtre, et l'algorithme FLORA3 stocke les descriptions des concepts passés pour pouvoir les ré-utiliser.

Ensuite, il y a les algorithmes de la famille AQ [Maloof and Michalski, 2000] avec notamment AQ11-PM+WAH *(Algorithm Quasi-optimal - Partial Memory + Window Adjustment Heuristic)* [Maloof and others, 2003]. AQ11-PM+WAH est la combinaison de AQ11, qui est une version incrémentale de l'algorithme d'apprentissage de règles AQ ; et de AQ-PM, qui est une version avec une mémoire partielle stockant les données proches des frontières de décisions des règles. WAH est une heuristique pour adapter dynamiquement la taille de la fenêtre mémorisant les données aux concepts actuels. AQ11-PM+WAH est donc un système incrémental avec à la fois une mémoire des concepts et une mémoire partielle (à taille variable) des exemples pour suivre efficacement les changements de concepts.

Il y a également l'algorithme FACIL (Fast and Adaptive Classifier by Incremental Learning) [Ferrer-Troyano et al., 2006] qui est aussi un algorithme incrémental avec une mémoire partielle des exemples en bordure des règles. Cette mémoire des exemples permet d'éviter les révisions inutiles des concepts décrits par les règles. Lorsque des exemples ne correspondent pas aux règles existantes, ils sont classifiés par l'algorithme du plus proche voisin. FACIL intègre une heuristique d'oubli qui permet d'enlever les exemples positifs et négatifs lorsqu'ils ne sont pas proches les uns des autres.

L'algorithme VFDR *(Very Fast Decision Rules)* [Gama et al., 2011], et son extension AVFDR *(Adaptive –)* [Kosina and Gama, 2012] pour les flux avec changements de concepts, permet d'induire des ensembles de règles de décision qui soient efficaces à la fois en matière de mémoire et de temps d'apprentissage. VFDR a été testé sur de très grandes bases de données et donne de meilleurs résultats que l'algorithme VFDTc (voir section 3.6).

Enfin, l'algorithme RILL *(Rule-based IncrementaL Learner)* [Deckert and Stefanowski, 2014] est le plus récent à avoir été proposé pour apprendre des règles de décision sur un flux de données. RILL combine également une mémoire d'instance en plus des règles de décision pour mieux modéliser les concepts complexes et leurs changements. L'utilisation d'une fenêtre glissante permet d'intégrer de l'oubli, et l'ensemble de règles est élagué grâce à une mesure de leurs performances.

3.8.2 Systèmes flous évolutifs

Les systèmes d'inférence floue sont un deuxième type de systèmes à base de règles, basés sur les principes de la logique floue [Zadeh, 1965]. Les systèmes d'inférence floue sont des approximateurs universels [Kosko, 1994], comme les réseaux de neurones, mais tout en conservant une forte interprétabilité [Ho et al., 2010].

Les systèmes d'inférence floue sont dit de Mamdani [Mamdani, 1977] – ou d'ordre zéro – lorsque les conclusions des règles sont constantes, et d'ordre un – ou de Takagi-Sugeno [Takagi and Sugeno, 1985] – lorsque les sorties sont obtenues par des fonctions linéaires des entrées.

Les systèmes d'inférence floue ont originellement été mis au point pour répondre à des besoins de régulation de systèmes physiques complexes. Il existe donc dans la littérature de nombreux systèmes pour faire face aux problèmes de régression en-ligne. Plus récemment, des systèmes d'inférence floue ont été proposés pour répondre à des problèmes de classification.

DENFIS *(Dynamic Evolving Neural-Fuzzy Inference System)* [Kasabov and Song, 2002] est un système d'inférence floue évolutif, de Takagi-Sugeno, qui a été développé pour la prédiction de séries temporelles. L'évolution des prémisses des règles est réalisée par la méthode de regroupement incrémentale ECM *(Evolving Clustering Method)*, qui est basée sur une mesure de distance comme nous le verrons en section 4.3.

eTS [Angelov and Lughofer, 2008] est aussi un système d'inférence floue d'ordre un, dont les prémisses et les conclusions sont adaptées incrémentalement. La création de nouvelles règles est réalisée par l'algorithme de regroupement incrémental *eClustering*, qui est basé sur une estimation récursive de la densité comme nous le verrons en section 4.3. *eTS* a été étendu pour répondre au problème de classification avec eClass [Angelov and Zhou, 2008].

FLEXFIS *(Flexible Fuzzy Inference System)* [Lughofer, 2008] est également un système d'inférence floue évolutif de Takagi-Sugeno. L'apprentissage des prémisses est réalisé par une extension incrémentale de la méthode de quantification vectorielle. Les paramètres des fonctions linéaires des conclusions sont optimisés avec l'algorithme des moindres carrés récursifs, comme pour DENFIS et eTS. Un seuil de vigilance est utilisé pour déclencher la

création de nouvelles règles, et une stratégie de fusion basée sur une mesure de pertinence des prototypes permet de simplifier l'ensemble de règles. *FLEXFIS-Class* [Angelov et al., 2008] est la version de FLEXFIS développée pour les problèmes de classification.

Evolve [Almaksour and Anquetil, 2011] est un système d'inférence floue destiné aux problèmes de classification, qui est similaire à eClass. Les prototypes dans l'espace d'entrée sont des hyper-ellipses orientées, afin d'améliorer la modélisation des données. Il est intéressant de noter que l'algorithme ILClass [Almaksour and Anquetil, 2013], destiné à l'apprentissage de systèmes d'inférence floue en-ligne, s'inspire des principes du *boosting* (voir section 3.7.1) pour améliorer la vitesse d'apprentissage du système.

Différentes améliorations ont été proposées sur la base des systèmes d'inférence floue d'ordre un. Ainsi les paramètres des fonctions linéaires peuvent être optimisés avec l'algorithme des vastes marges pour améliorer leur précision [Juang et al., 2007]. L'algorithme incrémental ILClass [Almaksour and Anquetil, 2013] permet l'apprentissage conjoint des prémisses et des conclusions. L'utilisation d'inférence transductive [Tencer et al., 2015] permet également d'améliorer les performances en ajoutant une mémoire partielle des données.

Une autre amélioration tirée de la structure des séparateurs à vaste marge est l'utilisation d'un ensemble de classifieur un-contre-un, plutôt que l'architecture classique tous-contre-tous [Lughofer and Buchtala, 2013]. Cette architecture permet améliorer la vitesse d'apprentissage du système en divisant la tâche d'apprentissage multi-classes en sous-tâches plus simples.

Les systèmes d'inférence floue suivent très bien les changements de concepts de par leur structure évolutive et leur apprentissage incrémental voire en-ligne [Lughofer and Angelov, 2011]. L'intégration d'oubli est alors primordiale dans l'apprentissage des prémisses et des conclusions. Une méthode d'élagage de l'ensemble de règles peut également être utilisée pour supprimer les règles obsolètes. Leur suppression n'est cependant pas obligatoire, il peut même être intéressant de les conserver lorsque des concepts peuvent réapparaitre. L'intégration d'oubli dans les prémisses est relativement simple car la représentation statistique des données permet une mise en œuvre simple de l'oubli. Le cas des conclusions est plus complexe car intégrer de l'oubli dans l'algorithme des moindres carrés récursifs est un problème sans solution universelle. L'intégration d'oubli dans l'optimisation des conclusions est le principal point critique pour l'utilisation « à vie » d'un système d'inférence floue.

Cette thèse est basée sur le système d'inférence floue évolutif *Evolve*, et présente un nouvel algorithme pour son apprentissage en-ligne, qui intègre de l'oubli comme nous le détaillons dans les sections 5.3 et 5.4.

3.9 Conclusion

Il existe différentes manières de répondre à la problématique de classification en-ligne, c'est-à-dire sur flux de données avec des changements de concepts. Dans le cadre de l'utilisation de commandes gestuelles, le réapprentissage périodique du système ne permet pas d'avoir une réactivité suffisante. Il est donc nécessaire d'avoir un classifieur réellement évo-

lutif qui puisse être appris en-ligne, donnée par donnée. Il faut également que le système dispose d'une capacité d'oubli pour permettre l'apprentissage « à vie » du système, et lui permettre de suivre les changements de concepts.

Nous avons présenté dans ce chapitre les différents classifieurs incrémentaux qui existent dans la littérature. Nous nous sommes attachés à évaluer dans quelle mesure ils sont évolutifs et ils peuvent être appris en-ligne. Nous avons en particulier été attentifs à la capacité d'oubli de ces classifieurs, et à leur faculté à suivre les changements de concepts.

Nous avons ainsi vu le classifieur bayésien naïf, qui est incrémental par nature, et qui peut être rendu en-ligne facilement en intégrant un facteur d'oubli dans le calcul des probabilités conditionnelles. Les systèmes d'appariement de modèles sont aussi incrémentaux par nature. Ils peuvent être rendus en-ligne, en utilisant une fenêtre glissante de données, bien que cela implique des besoins importants en mémoire. Nous avons également vu les versions incrémentales et en-ligne des classifieurs populaires que sont les réseaux de neurones, avec notamment les réseaux auto-organisés. Ces réseaux, très proches des systèmes flous évolutifs, sont facilement utilisables en-ligne et supportent très bien l'ajout de classes. Les séparateurs à vaste marge restent quant à eux difficilement utilisables en-ligne, du fait de leur complexité non-négligeable et de la difficulté à intégrer de l'oubli dans leur processus d'appentissage. Ensuite, nous avons présenté les arbres de décisions incrémentaux qui sont beaucoup utilisés sur les flux de données de grandes tailles. L'intégration d'oubli dans l'apprentissage d'arbres de décision, ainsi que leur restructuration lors des changements de concepts, reste cependant complexe. Les ensembles de classifieurs faibles peuvent également être utilisés de manière en-ligne, mais se prêtent mal à l'apprentissage avec peu de données. Enfin, nous avons fini par détailler les classifieurs à base de règles, avec notamment les systèmes experts conçus pour suivre les changements de concepts et les systèmes d'inférence floue évolutifs.

Les systèmes floues évolutifs correspondent tout à fait à la problématique de l'utilisation de commandes gestuelles. Ce sont des systèmes à l'apprentissage en-ligne rapide grâce aux capacités génératrices des prototypes, tout en restant évolutifs. Ils deviennent également très précis à convergence grâce au pouvoir discriminant des conclusions des règles. Ils supportent très bien l'ajout de nouvelles classes en cours d'apprentissage de par leur organisation sous forme d'ensembles de règles indépendantes.

Cette thèse est basée sur le système d'inférence floue *Evolve*, auquel ont été apportées plusieurs contributions, comme l'intégration d'oubli dans le processus d'apprentissage notamment. Nous détaillons donc les différentes architectures de systèmes d'inférence floue qui existent, et notamment celle d'*Evolve*, dans le prochain chapitre. Nous verrons également les différents algorithmes qui peuvent être utilisés pour l'apprentissage en-ligne de ce type de systèmes. Finalement, nous étudions les limites d'*Evolve* vis-à-vis de la problématique de la reconnaissance de commandes gestuelles.

Chapitre 4

Le classifieur « Evolve »

4.1 Introduction

Ce chapitre présente le classifieur évolutif *Evolve* [Almaksour and Anquetil, 2011] sur lequel ces travaux sont basés, ainsi que son algorithme d'apprentissage incrémental [Almaksour and Anquetil, 2013]. C'est un système d'inférence floue *(fuzzy inference system)* évolutif, car non seulement ses paramètres sont adaptés pendant l'apprentissage, mais sa structure aussi évolue : de nouvelles règles sont créées lorsque cela est nécessaire pendant son utilisation. Son algorithme d'apprentissage est incrémental en temps réel, mais n'est pas en-ligne dans le sens où il n'intègre pas d'oubli. En effet, ses algorithmes d'apprentissage sont purement incrémentaux, c'est-à-dire que ce sont les équivalents d'un apprentissage *batch* sur l'ensemble du flux de données.

Dans le cas d'un flux de données avec des changements de concepts, seules les dernières données arrivées après le changement de concepts sont pertinentes, les données plus anciennes sont alors obsolètes. En particulier, il ne faut pas que le poids des nouvelles données dans l'apprentissage soit inversement proportionnel au nombre total de données, sinon la vitesse d'apprentissage du système se met à décroitre et tendre vers zéro. Le gain de l'apprentissage diminue avec le temps, et le modèle de connaissance appris finit donc par se figer. *Evolve* est ainsi incapable de suivre les changements de concepts.

Les systèmes d'inférences floue (SIF) sont composés d'un ensemble de règles, ce qui leur permet d'obtenir un compromis plasticité/stabilité très intéressant. La création de nouvelles règles permet en effet l'apprentissage de nouveautés, sans pour autant oublier la connaissance existante. Les prémisses des règles sont des prototypes dans l'espace d'entrée. Elles forment un modèle génératif qui permet au système d'apprendre vite à partir de peu de données. Les conclusions d'un SIF d'ordre un sont composées de fonctions linéaires (régressions binomiales). Elles forment ainsi un modèle discriminatif qui permet d'obtenir de meilleures performances après un nombre d'exemples d'apprentissages suffisant.

Evolve est capable de commencer à partir de rien ou de très peu d'exemples, et il supporte l'ajout de nouvelles classes à-la-volée. Il apprend incrémentalement en temps réel sur le flux de données d'utilisation pour optimiser son modèle de connaissance et améliorer ses performances. C'est donc un système très bien adapté à l'apprentissage de commandes

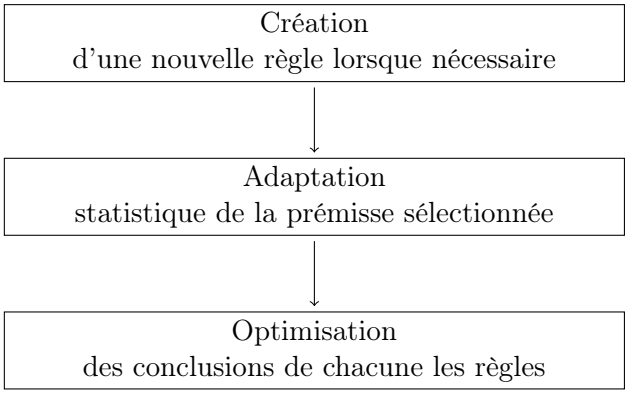


FIGURE 4.1 – Processus d'apprentissage en-ligne d'*Evolve* ∞.

gestuelles.

L'algorithme d'apprentissage d'*Evolve* se déroule en trois temps qui sont présentés figure 4.1 :

— l'algorithme de création de règles ;
— l'algorithme d'adaptation des prémisses ;
— l'algorithme d'optimisation des conclusions.

Nous commençons par décrire dans la section 4.2 les différentes architectures possibles pour les systèmes d'inférence floue. Nous étudions dans la section 4.3 les différents algorithmes possibles pour la création de nouvelles règles « à la volée ». Ensuite, nous détaillons l'algorithme d'adaptation des prémisses dans la section 4.4, et l'algorithme d'optimisation des conclusions dans la section 4.5. Nous concluons par leurs limites, ainsi que les différentes alternatives possibles.

4.2 Architecture des systèmes d'inférence floue

Evolve est un système d'inférence floue (SIF) d'ordre un – dit de Takagi-Sugeno [Takagi and Sugeno, 1985]. Les SIF ont démontré à de nombreuses reprises leurs excellentes performances sur les problèmes d'apprentissage en-ligne [Angelov and Zhou, 2008].

Un système d'inférence floue (SIF) est composé d'un ensemble de règles d'inférence comme suit :

$$\mathbf{Rule}^{(i)} : \mathbf{SI}\ \mathbf{x}\ \text{est proche de}\ C^{(i)} \tag{4.1}$$

$$\mathbf{ALORS}\ \hat{\mathbf{y}}^{(i)} = (\hat{\mathbf{y}}_1^{(i)} ; \ldots ; \hat{\mathbf{y}}_c^{(i)})^\top \tag{4.2}$$

Où $\mathbf{x} \in \mathbb{R}^n$ est le vecteur de caractéristiques d'une donnée, $C^{(i)}$ le prototype flou associé à la i-ème règle et $\hat{\mathbf{y}}^{(i)\top} \in \mathbb{R}^c$ le vecteur de résultat. Les prémisses des règles sont les degrés d'appartenance floue aux prototypes des règles, qui sont des groupes *(clusters)* dans l'espace d'entrée. Les conclusions des règles donnent les degrés d'appartenances floues à toutes les classes. Les sorties des différentes règles sont combinées pour produire le vecteur résultat global.

4.2.1 Structure des prémisses

Evolve utilise des prototypes $C^{(i)}$ de type hyper-elliptiques orientées, qui sont chacun défini par un centre $\boldsymbol{\mu}^{(i)} \in \mathbb{R}^n$ et une matrice de covariance $\Sigma^{(i)} \in \mathbb{R}^{n \times n}$ (où n est le nombre de caractéristiques des données et i représente le i$^{\text{e}}$ prototype).

$$\Sigma^{(i)} = \begin{bmatrix} \sigma_1^2 & \dots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \dots & \sigma_n^2 \end{bmatrix} \tag{4.3}$$

Le fait de prendre en considération les covariances $c_{i,j}$ entre les caractéristiques, en plus des variances σ_i^2, permet aux hyper-ellipses d'être orientées librement dans l'espace d'entrée, et pas forcément dans les directions des axes.

Le degré d'activation $\alpha^{(i)}(\mathbf{x})$ de chaque prototype $C^{(i)}$ est calculé pour la donnée $\mathbf{x}$ en utilisant une distribution normale multivariée :

$$\alpha^{(i)}(\mathbf{x}) = \frac{\exp\left(-\frac{1}{2} \cdot (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top \cdot (\Sigma^{(i)})^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top\right)}{(2\pi)^{n/2} \cdot \sqrt{|\Sigma^{(i)}|}} \tag{4.4}$$

$\top$ représente l'opération de transposition.

4.2.2 Structure des conclusions

Les conclusions des règles d'inférence donnent les degrés d'appartenance floue à toutes les classes. La structure de ces conclusions peut être soit d'ordre zéro [Mamdani, 1977], soit d'ordre un [Takagi and Sugeno, 1985]. *Evolve* est un système d'inférence floue (SIF) d'ordre un, mais nous allons également voir les conclusions d'ordre zéro afin de mettre en évidence les différences. La figure 4.2 présente les architectures des systèmes d'inférence floue d'ordre zéro et un respectivement, sous la forme de réseaux de neurones à fonction de base radiale (RBF : *Radial Basis Function*) [Jang and Sun, 1993].

Conclusions d'ordre zéro Dans un SIF d'ordre zéro – dit de Mamdani – les coefficient d'appartenance aux c différentes classes $\theta_1^{(i)}, \dots, \theta_c^{(i)}$ des règles d'inférence sont indépendants de l'entrée du système. Les vecteurs de sortie $\hat{\mathbf{y}}^{(i)}$ des règles sont également indépendants de la donnée en entrée :

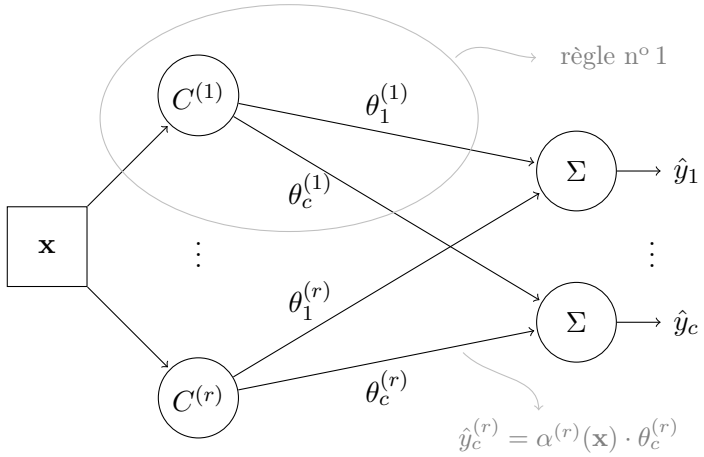
$$\hat{\mathbf{y}}^{(i)} = (\theta_1^{(i)}\,; \dots ; \theta_c^{(i)})^\top \tag{4.5}$$

La conclusion de la i-ème règle peut être reformulée comme suit :

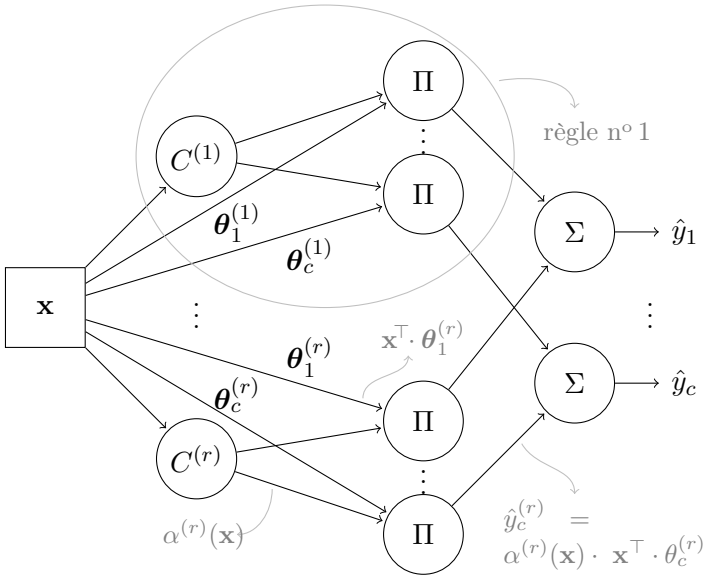
$$\hat{\mathbf{y}}^{(i)} = \boldsymbol{\theta}^{(i)\top} \tag{4.6}$$

Où $\boldsymbol{\theta}^{(i)} \in \mathbb{R}^c$ est le vecteur des coefficients d'appartenance, qui est indépendant de l'entrée du système.

La figure 4.2a présente un SIF d'ordre zéro sous la forme d'un réseau de neurones à fonction à base radiale (RBF : *Radial Basis Function*) [Jang and Sun, 1993].



(a) Système d'inférence floue d'ordre zéro.



(b) Système d'inférence floue d'ordre un.

FIGURE 4.2 – Système d'inférence floue (SIF) d'ordre zéro et d'ordre un sous la forme de réseaux de neurones à fonction de base radiale (RBF : *Radial Basis Function*).

$\mathbf{x}$ est le vecteur d'entrée, $C^{(i)}$ représente le i$^{\text{e}}$ prototype et $\alpha^{(r)}(\mathbf{x})$ son activation par la donnée $\mathbf{x}$, $\theta_k^{(i)}$ est le coefficient d'appartenance à la classe k pour la i$^{\text{e}}$ règle, $\boldsymbol{\theta}_k^{(i)}$ est le vecteur des coefficients de la fonction linéaire qui donne l'appartenance à la classe k pour la i$^{\text{e}}$ règle, $\hat{y}_1 \dots \hat{y}_c$ sont les degrés d'appartenance aux c classes.

Conclusions d'ordre un Dans un SIF d'ordre un – dit de Takagi-Sugeno – les vecteurs de sortie des règles d'inférence sont obtenus par des fonctions linéaires des entrées $l_k^{(i)}(\mathbf{x})$:

$$\hat{\mathbf{y}}^{(i)} = (l_1^{(i)}(\mathbf{x}) \,;\, \ldots \,;\, l_c^{(i)}(\mathbf{x}))^\top \tag{4.7}$$

Où $l_k^{(i)}(\mathbf{x})$ représente la fonction linéaire de la i-ème règle donnant le degré d'appartenance à la k-ème classe :

$$l_k^{(i)}(\mathbf{x}) = \mathbf{x}^\top \cdot \boldsymbol{\theta}_k^{(i)} = \theta_{0,k}^{(i)} + \theta_{1,k}^{(i)} \cdot x_1 + \cdots + \theta_{n,k}^{(i)} \cdot x_n \tag{4.8}$$

La conclusion de la i-ème règle peut être reformulé comme suit :

$$\hat{\mathbf{y}}^{(i)\top} = \mathbf{x}^\top \cdot \Theta^{(i)} \tag{4.9}$$

Où $\Theta^{(i)} \in \mathbb{R}^{n \times c}$ est la matrice regroupant les coefficients des fonctions linéaires pour chacune des c classes :

$$\Theta^{(i)} = (\boldsymbol{\theta}_1^{(i)} \,;\, \ldots \,;\, \boldsymbol{\theta}_c^{(i)}) = \begin{bmatrix} \theta_{1,1}^{(i)} & \cdots & \theta_{1,c}^{(i)} \\ \vdots & \ddots & \vdots \\ \theta_{n,1}^{(i)} & \cdots & \theta_{n,c}^{(i)} \end{bmatrix} \tag{4.10}$$

La figure 4.2 présente un SIF d'ordre un sous la forme d'un réseau de neurones à fonction à base radiale (RBF : *Radial Basis Function*) [Jang and Sun, 1993].

Les systèmes d'inférence floue d'ordre un permettent d'obtenir une meilleur précision que ceux d'ordre zéro, comme cela a été montré dans différentes études [Tung and Quek, 2009] [Ying et al., 1998].

4.2.3 Processus d'inférence

Le processus d'inférence est composé de trois étapes :

1. Le degré d'activation de chaque règle i est calculé pour la donnée $\mathbf{x}$ par l'équation 4.4, puis normalisé comme suit :

$$\alpha^{(i)}(\mathbf{x}) = \frac{\alpha^{(i)}(\mathbf{x})}{\sum_{k=1}^{r} \alpha^{(k)}(\mathbf{x})} \tag{4.11}$$

   Où r est le nombre de règles.

2. Les sorties $\hat{\mathbf{y}}^{(k)}$ des règles sont calculées en utilisant l'équation 4.6 et la sortie globale $\hat{\mathbf{y}}$ du système est calculée par inférence somme-produit :

$$\hat{\mathbf{y}} = \sum_{k=1}^{r} \alpha^{(k)}(\mathbf{x}) \cdot \hat{\mathbf{y}}^{(k)} \tag{4.12}$$

3. La classe prédite pour la donnée $\mathbf{x}$, parmi les c classes possibles, est celle qui correspond à la valeur de sortie la plus élevée :

$$\text{class}(\mathbf{x}) = \arg\max_{k=1}^{c}(\hat{y}_k) \tag{4.13}$$

4.3 Création de nouvelles règles

L'objectif de l'algorithme de création de règles est de détecter l'apparition de nouveaux motifs dans les données. Il est au minimum nécessaire de créer un prototype par classe pour garantir des performances de reconnaissance correctes. Certaines classes complexes, ou avec beaucoup de variance intra-classe, peuvent cependant nécessiter la création de plusieurs prototypes pour cette même classe, afin d'obtenir de bonnes performances de reconnaissance. Les systèmes d'inférence floue utilisent donc des algorithmes de regroupement *(clustering)* incrémental pour détecter le nombre de motifs dans le flux de données et créer les prototypes associés.

Le principe des algorithmes de regroupement est de rassembler les objets similaires dans des groupes, en minimisant la variance intra-groupe et en maximisant la variance inter-groupe, comme présenté en section 1.4.3. Dans le cadre de la création de règles dans les systèmes d'inférence floue évolutifs, ce regroupement doit être effectué en-ligne. Il existe de nombreux algorithmes de regroupement pour flux de données [Nguyen et al., 2015], avec différentes caractéristiques.

Dans *Evolve*, la création de nouveaux prototypes, et donc de nouvelles règles, est déclenchée par la méthode de regroupement incrémental *eClustering*, qui est basée sur une estimation récursive de la densité. Cependant, il existe aussi d'autres types de méthodes de regroupement incrémental [Silva et al., 2013] pouvant aussi être utilisées :

— les méthodes basées sur la densité ;
— les méthodes basées sur la distance ;
— les algorithmes basés sur des graphes ;
— les réseaux auto-organisés.

4.3.1 Regroupement basé sur la densité

Il existe différentes méthodes de regroupement incrémental basées sur une estimation de la densité, comme *eClustering* par exemple.

Dans *Evolve*, de nouvelles règles, avec les prototypes et les conclusions associés, sont créées lorsque cela est détecté comme nécessaire par l'algorithme de regroupement incrémental *eClustering* [Angelov and Filev, 2004].

eClustering *eClustering* est basé sur une estimation récursive de la densité.

Quand une nouvelle donnée $\mathbf{x}(t) \in \mathbb{R}^n$ arrive à l'instant t, son potentiel $P(\mathbf{x}(t))$ est calculé de la manière suivante :

$$P(\mathbf{x}(t)) = \frac{t-1}{(t-1)\cdot\alpha(t) + \gamma(t) - 2\cdot\zeta(t) + t - 1} \tag{4.14}$$

Où n est le nombre de caractéristiques des données et :

$$\alpha(t) = \sum_{j=1}^{n} x_j^2(t) \tag{4.15}$$

$$\gamma(t) = \gamma(t-1) + \alpha(t-1), \quad \gamma(1) = 0 \tag{4.16}$$

$$\zeta(t) = \sum_{j=1}^{n} x_j(t) \cdot \eta_j(t) \tag{4.17}$$

$$\eta_j(t) = \eta_j(t-1) + x_j(t-1), \quad \eta_j(1) = 0 \tag{4.18}$$

Lorsque le potentiel d'une donnée est supérieur au seuil de création $s_{creation}$, un nouveau prototype est créé à cet emplacement.

Dans *eClustering*, le seuil de création de nouveaux prototypes est égal au plus grand potentiel parmi les prototypes $C^{(i)}$ existants ($1 \leq i \leq r$, avec r le nombre de règles) :

$$s_{creation} = \max_{i=1..r} P(C^{(r)}) \tag{4.19}$$

Lorsqu'une donnée ne déclenche pas de création de prototype, les potentiels de tous les prototypes existants $C^{(i)}$ (de centre μ_i) sont mis à jour récursivement à l'instant t :

$$P(C^{(i)}) = \frac{(t-1) \cdot P(C^{(i)})}{t - 2 + P(C^{(i)}) + P(C^{(i)}) \cdot \sum_{j=1}^{n} \|\mu_i - x \cdot (t-1)\|_j^2} \tag{4.20}$$

4.3.2 Regroupement basé sur la distance

***Evolving Clustering Method* (ECM)** ECM [Kasabov and Song, 2002] est un algorithme de clustering en-ligne basé sur un seuil de distance. Il permet de partitionner dynamiquement un flux de données, en estimant automatiquement le nombre de groupes, et en mettant à jour leur centre et leur rayon. L'algorithme est basé sur un seuil de distance, qui est passé en paramètre.

Lors de l'arrivée d'une nouvelle donnée, les distances à tous les centres des groupes sont calculées. Si une distance à un centre est inférieure au rayon du groupe correspondant, autrement dit si une donnée arrive dans un groupe existant, alors l'algorithme ne fait rien. Si aucune distance n'est inférieure au rayon correspondant, le groupe le plus proche est modifié si le seuil de distance n'est pas dépassé. Si ce seuil est dépassé, alors un nouveau groupe est créé.

La distance utilisée par ECM lors de son utilisation avec EFuNN *(Evolving Fuzzy Neural Network)* est la distance Euclidienne. Il est cependant tout à fait possible d'utiliser cet algorithme avec une autre mesure de distance, comme la distance de Mahalanobis par exemple.

Le seuil de distance a une grande influence sur le nombre de groupes créés par l'algorithme. Un seuil élevé va produire de nombreuses catégories très précises, alors qu'un seuil bas va engendrer moins de catégories mais qui seront plus générales. La calibration de ce seuil peut être réalisée sur une base d'apprentissage, mais son estimation a priori dans le cas d'un algorithme de regroupement incrémental est complexe.

4.3.3 Réseaux auto-organisés

La majorité des réseaux auto-organisés (voir section 3.4.3) peuvent également être utilisés comme méthode de regroupement incrémental. Nous présentons ici des méthodes de quantification vectorielle et les réseaux ART.

Quantification vectorielle La quantification vectorielle *(Vector Quantization)* est à l'origine une technique de traitement du signal permettant de compresser des données [Kohonen, 1997]. L'idée est de réduire un espace vectoriel multidimensionnel vers un sous-espace en utilisant un dictionnaire. Le dictionnaire obtenu est un ensemble de prototypes qui forment une partition de l'espace. Par contre, le nombre de prototypes est fixé à priori. La quantification vectorielle est équivalente à l'algorithme des C-moyennes incrémentales [Chakraborty and Nagwani, 2011].

Les cartes auto-organisatrices, qui sont une extension de la quantification vectorielle, prennent également en compte le voisinage entre le neurone le plus proche et les autres.

Réseaux ART *(Adaptive Reasonance Theory)* Les réseaux ART sont composés d'une couche de neurones de comparaison et d'une couche de neurones de reconnaissance, d'un seuil de vigilance et d'un module de remise à zéro [Carpenter et al., 1991a]. La couche de comparaison prend en entrée le vecteur des caractéristiques d'une donnée et le transfert au neurone lui correspondant le mieux dans la couche de reconnaissance, c'est-à-dire celui donc les valeurs sont les plus proches du vecteur en entrée. Chaque neurone émet un signal de corrélation négatif, en fonction de son adéquation avec le vecteur d'entrée, vers tous les autres neurones de la couche de reconnaissance. Une fois le vecteur d'entrée classifié, le module de remise à zéro compare la confiance de cette classification avec le seuil de vigilance. Si la confiance est suffisante, le vecteur d'entrée est attribué au neurone correspondant et celui-ci est ajusté en conséquence. Si la confiance est inférieure au seuil de vigilance, le module de remise à zéro désactive le neurone correspondant et prend le neurone actif le plus proche du vecteur d'entrée. Le processus continue ainsi jusqu'à trouver un neurone avec une confiance supérieure au seuil de vigilance. Si aucun neurone n'a une confiance supérieure à ce seuil, alors un nouveau neurone est créé à partir du vecteur d'entrée.

Le module de remise à zéro a le rôle d'un détecteur de nouveautés. Le seuil de vigilance a donc une grande importance sur les catégories obtenues, comme le seuil de distance pour l'algorithme de regroupement incrémental ECM. Ce seuil est le plus souvent estimé empiriquement, sur une base d'apprentissage, et reste ensuite fixé. Dans le cas d'un fonctionnement en-ligne, l'estimation d'un seuil fixe est relativement complexe comme nous le détaillons ci-dessous.

Le principal avantage des réseaux ART est le très bon compromis plasticité/stabilité qu'ils permettent d'obtenir. Ils sont capables de s'adapter et d'intégrer de la nouveauté (plasticité), sans pour autant oublier ou détériorer la connaissance existante (stabilité).

4.3.4 Algorithmes basés sur des graphes

Il existe plusieurs algorithmes de regroupement basés sur des graphes comme Chameleon [Karypis et al., 1999], MOSAIC [Choo et al., 2007] et RepStream [Lühr and Lazarescu, 2008]. Nous détaillons ce dernier ci-dessous.

RepStream RepStream [Lühr and Lazarescu, 2008] est un algorithme de regroupement incrémental basé sur l'utilisation de deux graphes. Le premier graphe contient les centroïdes qui ont été retenus par l'algorithme jusque-là, et forme donc la partition obtenue de l'espace d'entrée. Le deuxième graphe contient les dernières données présentées à l'algorithme, et contient les différentes distances entre ces données. Ce deuxième graphe sert de mémoire à court terme pour prendre la décision de mise à jour du premier graphe à l'arrivée d'une nouvelle donnée. Cette mémoire permet de réduire nettement la sensibilité aux données aberrantes *(outliers)* en attendant que la création d'un nouveau groupe se confirme, plutôt que de le créer dès la première donnée différente.

4.3.5 Discussion

Le principal inconvénient des méthodes de regroupement incrémental est qu'elles nécessitent presque toutes un seuil pour la création de nouveaux groupes. Ce seuil est fortement dépendant du problème, et sa valeur optimale dépend de l'espace des caractéristiques, et des données d'utilisation. L'optimisation et la validation de ce seuil est difficilement réalisable lorsque l'apprentissage est réalisé en-ligne, surtout avec l'utilisateur dans la boucle. Une mauvaise valeur de ce seuil entraine soit un sous-apprentissage, avec le regroupement de plusieurs groupes dans le même, soit un sur-apprentissage, avec la division de groupes un plusieurs. Dans les deux cas, un nombre de groupes non adapté engendre un diminution des performances du système d'inférence floue.

Un autre problème récurrent des algorithmes de regroupement incrémental est la sensibilité au bruit et aux données aberrantes *(outliers)*. Des groupes sans fondement peuvent facilement être créés par des données bruitées qui sont suffisamment loin des centres existants.

Dans le cas des algorithmes incrémentaux, le principal moyen de réduire la sensibilité au bruit est d'avoir un algorithme avec une certaine mémoire à court terme, comme par exemple le second graphe de RepStream. Cette mémoire permet de modéliser la distribution des dernières données, et de ne déclencher la création d'un nouveau groupe que lorsqu'il est confirmé par plusieurs données.

4.4 Apprentissage des prémisses

Evolve est un système évolutif dans le sens où non seulement ses paramètres sont appris incrémentalement, mais sa structure évolue également : de nouvelles règles sont créées lorsque cela est nécessaire. Il manque cependant une caractéristique essentielle pour que *Evolve* soit un véritable système évolutif en-ligne : la capacité d'oubli.

Dans *Evolve* [Almaksour and Anquetil, 2011], à la fois les prémisses et les conclusions sont apprises incrémentalement. Nous proposons donc de nouveaux algorithmes intégrant de l'oubli, pour l'apprentissage des prémisses dans la section 5.3 et des conclusions dans la section 5.4.

4.4.1 Adaptation statistique des prototypes

Les prémisses sont modélisées par des prototypes dans l'espace d'entrée, qui sont incrémentalement adaptés pour modéliser l'ensemble des données.

Les prototypes sont chacun définis par un centre $\boldsymbol{\mu}_t^{(i)}$ et une matrice de covariance $\Sigma_t^{(i)}$, qui sont mis à jour de manière incrémentale à l'arrivée de la donnée $\mathbf{x}_t$ à l'instant t.

$$\boldsymbol{\mu}_t^{(i)} = \frac{(t-1) \cdot \boldsymbol{\mu}_{t-1}^{(i)} + \mathbf{x}_t}{t} \tag{4.21}$$

$$\Sigma_t^{(i)} = \frac{(t-1) \cdot \Sigma_{t-1}^{(i)} + (\mathbf{x}_t - \boldsymbol{\mu}_{t-1}^{(i)}) \cdot (\mathbf{x}_t - \boldsymbol{\mu}_t^{(i)})}{t} \tag{4.22}$$

Cette mise à jour incrémentale n'intégrant pas d'oubli, les prototypes vont peu à peu converger et se stabiliser dans l'espace d'entrée. Il ne peuvent alors plus suivre les changements de concepts.

4.4.2 Autres algorithmes d'adaptation

Une autre manière possible pour faire évoluer les prototypes est de tirer profit de l'algorithme de regroupement incrémental (voir section 4.3). Ces algorithmes permettent en effet de créer et de faire évoluer des prototypes à partir du flux de données. Il est donc tout à fait possible de réutiliser directement ces prototypes, qui seront mis-à-jour par l'algorithme de création de règles.

Les réseaux auto-organisés ART sont en particulier utilisés dans certain systèmes d'inférence floue [Reznáková et al., 2015].

Les prototypes utilisés par les différents algorithmes de regroupement incrémental restent la plupart du temps moins précis que les hyper-ellipses utilisées dans *Evolve*. De plus, ces algorithmes n'intègrent pas tous de l'oubli dans la mise à jour des prototypes. Il est alors plus intéressant de ne se servir de l'algorithme de regroupement incrémental seulement pour la création de nouvelles règles, et de faire évoluer les prototypes de manière séparée.

4.5 Apprentissage des conclusions

Les conclusions des systèmes d'inférence floue d'ordre un, et d'*Evolve* en particulier, sont modélisées par des fonctions linéaires. Ces fonctions permettent d'obtenir les degrés d'appartenance à toutes les classes pour chacune des règles. Il faut donc un algorithme d'apprentissage en-ligne pour optimiser les coefficients de ces fonctions linéaires. Dans la

majorité des systèmes d'inférence floue, ces coefficients sont optimisés par l'algorithme des moindres carrés récursifs *(recursive least squares)*.

4.5.1 Optimisation par l'algorithme des moindres carrés récursifs

Les coefficients $\theta_{j,k}^{(i)}$ des fonctions linéaires de chaque règle i sont regroupés dans une matrice $\Theta_t^{(i)}$ par règle :

$$\Theta^{(i)} = \begin{bmatrix} \theta_{1,1}^{(i)} & \dots & \theta_{1,c}^{(i)} \\ \vdots & & \vdots \\ \theta_{n,1}^{(i)} & \dots & \theta_{n,c}^{(i)} \end{bmatrix} \; ; \quad 1 \leq i \leq r \tag{4.23}$$

où r représente le nombre de règles, c le nombre de classes, et n la taille du vecteur de caractéristiques. Ces matrices $\Theta_t^{(i)}$ sont optimisées incrémentalement par l'algorithme des moindres carrés récursifs à l'arrivée de la donnée $\mathbf{x}$ (avec $\mathbf{y}$ le vecteur binaire d'appartenance au différentes classes) :

$$\Theta_t^{(i)} = \Theta_{t-1}^{(i)} + \alpha^{(i)}(\mathbf{x}) \cdot \mathrm{C}_t^{(i)} \cdot \mathbf{x}_t \cdot (\mathbf{y}_t^\top - \mathbf{x}_t^T \cdot \Theta_{t-1}^{(i)}) \; ; \quad \Theta_0^{(i)} = 0 \tag{4.24}$$

Où $\alpha^{(i)}(\mathbf{x})$ est le degré d'activation de la règle i par la données $\mathbf{x}$, et $\mathrm{C}^{(i)}$ est la matrice de covariance, également mise à jour récursivement :

$$\mathrm{C}_t^{(i)} = \mathrm{C}_{t-1}^{(i)} - \frac{\mathrm{C}_{t-1}^{(i)} \cdot \mathbf{x}_t \cdot \mathbf{x}_t^\top \cdot \mathrm{C}_{t-1}^{(i)}}{\frac{1}{\alpha^{(i)}(\mathbf{x})} + \mathbf{x}_t^\top \cdot \mathrm{C}_{t-1}^{(i)} \cdot \mathbf{x}_t} \; ; \quad C_0^{(i)} = 100 \cdot Id \tag{4.25}$$

Id représente la matrice identité et $\top$ l'opération de transposition.

Plus le nombre de données d'apprentissage augmente, plus les modifications apportées aux conclusions par l'algorithme des moindres carrés récursifs seront faibles. En effet, le poids accordé à chaque donnée étant équivalent, plus le nombre de données augmente, plus le poids de chaque donnée individuelle est faible. Cette propriété est illustrée par le fait que la matrice de covariance décroit avec le temps : $P_t < P_{t-1}$. Il est dit que le gain de l'algorithme tend vers zéro.

Cette propriété a pour conséquence de réduire la réactivité du système, la capacité d'apprentissage de nouveautés, avec le temps. Ainsi, au bout d'un certain temps, l'algorithme ne sera plus capable de s'adapter aux changements dans les données, ni d'apprendre de nouvelles classes (dans un temps raisonnable). Il est donc nécessaire de limiter le poids des anciennes données, autrement dit d'introduire de l'oubli dans l'algorithme des moindres carrés récursifs, pour maintenir la capacité d'apprentissage du système.

4.5.2 Intégration d'oubli dans les moindres carrés récursifs

L'introduction d'oubli dans l'algorithme des moindres carrés récursifs est un sujet qui a été beaucoup étudié dans la littérature dédiée au contrôle de systèmes physiques complexes [Lughofer and Angelov, 2011]. Pour cela, il faut travailler sur la matrice de covariance C_t qui représente la distribution des données dans l'algorithme des moindres carrés

récursifs. Le principe de l'oubli est d'introduire, en plus de la mise à jour d'observation (Equation 4.25), une mise à jour temporelle de cette matrice, également à l'instant t :

$$\bar{C}_t = \mathcal{F}(C_t) \tag{4.26}$$

$\mathcal{F}(\cdot)$ est la fonction d'oubli et cette mise à jour temporelle représente alors l'augmentation de l'incertitude entre deux observations consécutives. Par conséquent, cette fonction doit être telle que : $\mathcal{F}(C) > C$.

Cette formulation de l'oubli – dite bayésienne – permet d'unifier les différentes techniques d'oubli pour l'algorithme des moindres carrés récursifs [Kulhavy and Zarrop, 1993]. Chaque technique est alors caractérisée par sa fonction d'oubli $\mathcal{F}(\cdot)$.

La propriété la plus importante d'un algorithme d'estimation récursif avec oubli est que sa matrice de covariance reste bornée [Salgado et al., 1988], [Parkum et al., 1992]. En effet, si la matrice de covariance n'a pas de borne inférieure, le gain de l'algorithme tend vers zéro et le système perd ses capacités d'adaptation avec le temps. Sans borne supérieure, la matrice de covariance peut exploser sous l'effet de d'oubli, et engendrer des instabilités qui mènent à l'écroulement du système.

Cette section présente les principales techniques pour introduire de l'oubli dans l'algorithme des moindres carrés récursifs. La plus simple est l'utilisation d'un facteur d'oubli, qui créé une pondération exponentielle des données. Cette technique est par contre sujette au problème d'explosion de la matrice de covariance *(covariance wind-up problem)*. Plusieurs autres techniques ont été proposées pour tenter de pallier ce problème comme utiliser un facteur d'oubli variable ou de l'oubli directionnel.

Facteur d'oubli constant Le principe du facteur d'oubli exponentiel est d'introduire une pondération exponentiellement décroissante des anciennes données $\mathbf{x}_k$ dans le calcul de la matrice de covariance C_t :

$$C_t = (\sum_{k=1}^{t} \lambda^{t-k} \cdot \mathbf{x}_k \cdot \mathbf{x}_k^\top)^{-1} \tag{4.27}$$

$$= (\lambda^{t-1} \cdot \mathbf{x}_1^\top \cdot \mathbf{x}_1 + \cdots + \lambda \cdot \mathbf{x}_{t-1}^\top \cdot \mathbf{x}_{t-1} + \mathbf{x}_t^\top \cdot \mathbf{x}_t)^{-1} \tag{4.28}$$

Soit de manière récursive :

$$C_t^{-1} = \lambda \cdot C_{t-1}^{-1} + \mathbf{x}_t^\top \cdot \mathbf{x}_t \tag{4.29}$$

où $\lambda \in [0; 1]$ est le facteur d'oubli exponentiel (typiquement $\lambda \in [0.9; 0.99]$). Par conséquent, le poids des anciennes données est limité d'une part, et l'apprentissage des conclusions se concentre sur les dernières données d'autre part. C'est un moyen simple, en apparence, de maintenir les capacités d'apprentissage du système. Cette pondération exponentielle des données se traduit par la fonction d'oubli suivante :

$$\mathcal{F}(C) = \frac{C}{\lambda} \tag{4.30}$$

Cependant, le choix de ce facteur d'oubli est complexe. Si le facteur d'oubli est trop faible, la matrice de covariance va tendre vers zéro et l'algorithme va perdre ses capacités

d'adaptation. Si le facteur d'oubli est trop fort, la matrice de covariance va exploser et l'algorithme va devenir instable jusqu'à provoquer l'effondrement du système. Ce phénomène – appelé *covariance « wind-up »* – est dû au fait qu'il y a davantage d'information oubliée que d'information reçue.

Pour éviter ce problème, il est nécessaire d'ajuster le facteur d'oubli à la quantité d'information nouvellement disponible. C'est le principe du facteur d'oubli variable dans le temps.

Facteur d'oubli variable Le principe du facteur d'oubli variable dans le temps est d'ajuster la quantité d'information oubliée à la quantité d'information apprise. Pour cela, il faut disposer d'une mesure d'information ι_t que l'on va maintenir constante au cours du temps. Le facteur d'oubli est ensuite calculé $\lambda_t^{(i)} = \iota_t^{(i)}/\iota_{t-1}^{(i)}$ pour obtenir la fonction d'oubli :

$$\mathcal{F}_t^{(i)}(C) = \frac{C}{\lambda_t^{(i)}} = C \cdot \frac{\iota_{t-1}^{(i)}}{\iota_t^{(i)}} \tag{4.31}$$

[Fortescue et al., 1981] proposent ainsi d'utiliser l'erreur quadratique *a posteriori* (pondérée par l'activation $\alpha^{(i)}(\mathbf{x}_t)$ pour chaque règle i) comme métrique :

$$\iota_t^{(i)} = \alpha^{(i)}(\mathbf{x}_t) \cdot (\mathbf{y}_t - \mathbf{x}_t^\top \cdot \Theta_t^{(i)}) \tag{4.32}$$

Une autre mesure possible, reflétant la quantité d'information acquise par l'algorithme, est la trace de la matrice de covariance $C_t^{(i)}$:

$$\iota_t^{(i)} = \mathrm{trace}(C_t^{(i)}) \tag{4.33}$$

Un tel facteur d'oubli variable permet à l'algorithme de conserver un gain non nul au cours du temps et le système peut ainsi s'adapter à toute évolution des données. Cette approche permet aussi d'éviter l'explosion de la matrice de covariance, mais seulement si l'information est bien répartie selon les dimensions de l'espace d'entrée. Dans le cas contraire, certains éléments de la matrice de covariance peuvent tendre vers zéro pendant que d'autres explosent. Pour éviter cela, il faut non seulement faire varier le facteur d'oubli dans le temps mais également dans l'espace, c'est ce qu'on appelle l'oubli directionnel.

Oubli directionnel L'oubli directionnel *(directional forgetting)* a été proposé initialement par [Hägglund, 1985] et [Kulhavy, 1987] puis amélioré par [Bittanti et al., 1990] et [Cao and Schwartz, 2000]. C'est une technique qui fait varier l'oubli dans le temps mais aussi dans l'espace en introduisant de l'oubli seulement dans les directions où de l'information arrive.

La fonction d'oubli utilisée est donc paramétrée par la donnée courante :

$$\mathcal{F}_t^{(i)}(c) = c - (1 - \gamma_t^{(i)}) \cdot \alpha^{(i)}(\mathbf{x}_t) \cdot \mathbf{x}_t^\top \cdot \mathbf{x}_t \tag{4.34}$$

$$\gamma_t^{(i)} = \lambda_t^{(i)} - \frac{1 - \lambda_t^{(i)}}{\mathbf{x}_t^\top \cdot C_{t-1}^{(i)} \cdot \mathbf{x}_t} \tag{4.35}$$

L'avantage de cette technique est qu'elle maintient la matrice de covariance bornée, et garantit ainsi les performances du système. Dans les directions où beaucoup d'information arrive, on oublie beaucoup ; mais on oublie peu dans les directions qui sont peu excitées.

4.5.3 Autres algorithmes d'optimisation

D'autres algorithmes que les moindres carrés récursifs sont également envisageables pour l'optimisation des coefficients linéaires des conclusions. Les conclusions des systèmes d'inférence floue d'ordre un sont finalement des régressions binomiales. Il est donc possible d'utiliser les algorithmes utilisés pour les régressions binomiales en-ligne (voir section 3.2.2), comme par exemple le filtrage de Kalman, pour optimiser les coefficients des conclusions.

4.6 Conclusion

Evolve est un système d'inférence floue d'ordre un. Il est constitué d'un ensemble de règles associant des prototypes hyper-elliptiques orientés dans l'espace d'entrée à des conclusions obtenues par des fonctions linéaires. Les prototypes sont adaptés statistiquement pour modéliser les données, et les conclusions sont optimisées par l'algorithme des moindres carrés récursifs. De nouvelles règles doivent être créées par l'algorithme de regroupement incrémental *eClustering* lorsque cela est nécessaire.

Le classifieur *Evolve* est très adapté au contexte de la reconnaissance de commandes gestuelles pour plusieurs raisons. Il est capable de commencer à partir de rien et d'apprendre incrémentalement les nouvelles classes au fur et à mesure qu'elles sont ajoutées pendant son utilisation.

Les systèmes d'inférence floue permettent d'obtenir un compromis plasticité/stabilité très intéressant. Leur structure sous forme d'ensemble de règles permet de séparer les modélisations des différentes classes, et d'en ajouter facilement de nouvelles sans perturber la connaissance existante. L'apprentissage d'*Evolve* est rapide grâce aux capacités génératrices des prémisses, et il donne en même temps de bonnes performances à convergence grâce aux capacités discriminantes des conclusions.

L'algorithme d'apprentissage d'*Evolve* est incrémental en temps réel, mais n'est pas en-ligne dans la mesure où il n'intègre pas d'oubli. Le gain de l'apprentissage diminue donc avec l'augmentation du nombre de données d'apprentissage, et le modèle de connaissance finit par ne plus évoluer. *Evolve* n'est donc pas évolutif indéfiniment, et est donc incapable de suivre les changements de concepts. Ensuite, l'algorithme ECM est peu efficace pour déclencher la création de nouvelles règles lorsque le système travaille en dimension cinquante, comme c'est notre cas avec l'utilisation du jeu de caractéristiques HBF49 [Delaye and Anquetil, 2013].

Différents algorithmes alternatifs sont utilisables pour l'apprentissage d'*Evolve*. D'autres méthodes de regroupement incrémental, comme RepStream ou les réseaux ART, peuvent être utilisées pour déclencher la création de nouvelles règles. L'algorithme d'adaptation des prémisses nécessite l'intégration d'oubli, pour permettre aux prototypes de modéliser

l'évolution des données. Enfin, l'algorithme des moindres carrés récursifs n'est pas idéal pour l'optimisation des conclusions car l'intégration d'oubli y est difficile.

Nous présentons dans le prochain chapitre le classifieur évolutif *Evolve* ∞. Son algorithme d'apprentissage en-ligne intègre notamment de l'oubli, afin de maintenir ses capacités d'évolution indéfiniment. *Evolve* ∞ dispose également d'une option de rejet avancée, qui est utilisée comme algorithme de regroupement incrémental pour la création de nouvelles règles.

Deuxième partie

Contributions

Préambule

La deuxième partie présente les contributions de cette thèse, à savoir le classifieur évolutif *Evolve* ∞ et le superviseur actif en-ligne *IntuiSup*. Bien que développées en synergie, ces deux contributions sont complètement indépendantes. Le classifieur évolutif *Evolve* ∞ peut tout à fait être utilisé pour d'autres tâches d'apprentissage statistique. De même, le superviseur *IntuiSup* peut également servir à superviser l'apprentissage actif en-ligne d'autres classifieurs évolutifs. Le couplage du classifieur et du superviseur s'effectue grâce à la fonction de confiance du classifieur. Celui-ci doit donc être capable d'évaluer les limites de son modèle de connaissance. Le superviseur *IntuiSup* met alors à profit les informations fournies par la mesure de confiance du classifieur pour adapter l'apprentissage aux difficultés du système et à l'évolution de son environnement.

— **Le cinquième chapitre** présente les améliorations et spécificités du classifieur évolutif *Evolve* ∞ par rapport au classifieur incrémental *Evolve*. L'intégration d'oubli dans le processus d'apprentissage du système est notamment présenté, afin de permettre au système de rester évolutif indéfiniment. Cette capacité d'oubli le rend ainsi capable d'apprendre de nouvelles classes à n'importe quel moment de son utilisation, et de suivre les changements de concepts du flux de données. *Evolve* ∞ dispose également d'une capacité d'auto-évaluation de la qualité de son modèle, qui lui permet d'évaluer les limites de ses connaissances. Cette capacité d'évaluation permet d'obtenir la fonction de confiance qui est utilisée par *IntuiSup* pour superviser l'apprentissage lorsque l'utilisateur est dans la boucle. Elle permet également d'obtenir une option de rejet évolutive pour réduire son taux d'erreur dans les contextes où les mauvaises reconnaissances sont coûteuses.
Ce chapitre a donné lieu aux publications scientifiques suivante : [Boui 12], [Boui 13a], [Boui 13b] et [Boui 13c].

— **Le sixième chapitre** décrit le superviseur *IntuiSup* que nous avons créé pour superviser l'apprentissage actif en-ligne d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage. Il permet d'optimiser la coopération entre l'utilisateur et le système en minimisant les sollicitations tout en maximisant leur impact sur l'apprentissage. Ce superviseur est évolutif et permet de maintenir au cours du temps le compromis erreur/sollicitation choisi, en s'adaptant à l'évolution du classifieur et aux changements du flux de données. Il permet également de doper l'apprentissage en ajustant la répartition des sollicitations pour les concentrer aux

moments où elles sont les plus importantes : pendant l'apprentissage initial et durant les changements de concepts.
Ce chapitre a donné lieu aux publications scientifiques suivantes : [Boui 14a], [Boui 14b], [Boui 14c], [Boui 14d], [Boui 14e] et [Boui 15].

Chapitre 5

Evolve ∞ : un classifieur évolutif en-ligne

5.1 Introduction

Ce chapitre présente le classifieur évolutif *Evolve* ∞, qui est une évolution du classifieur *Evolve* [Almaksour and Anquetil, 2011]. Nous l'avons mis au point et optimisé pour l'apprentissage et la reconnaissance de commandes gestuelles, et de tracés manuscrits en général. C'est cependant un classifieur évolutif générique, qui peut aussi être utilisé dans n'importe quelle autre situation d'apprentissage en-ligne.

Ces travaux se basent sur *Evolve*, qui est un système d'inférence floue *(fuzzy snference system)* d'ordre un, aussi appelé système flou à base de règles *(fuzzy rule-based system)*. C'est un système composé d'un ensemble de règles, chacune composée d'une prémisse et d'une conclusion. Les prémisses forment la partie du système représentative des données, elles sont modélisées par des prototypes dans l'espace d'entrée (modèle génératif). Les conclusions permettent d'obtenir les frontières de décision, elles sont obtenues par des régressions linéaires (modèle discriminant).

L'algorithme d'apprentissage d'*Evolve* est incrémental, il est capable d'apprendre à partir de chaque nouvelle donnée, indépendamment les unes des autres. Cet apprentissage est réalisé en temps réel grâce à des algorithmes d'apprentissage de faible complexité. Comme *Evolve* est un classifieur évolutif, il supporte l'ajout de nouvelles classes « à-la-volée » pendant son utilisation. Cette capacité d'apprentissage pendant son utilisation en fait un classifieur très adapté pour les contextes où très peu de données d'apprentissage sont disponibles initialement, comme celui de la reconnaissance de commandes gestuelles personnalisées.

Les prémisses sont adaptées constamment, pour modéliser l'ensemble des données du flux. Les conclusions sont également optimisées incrémentalement, afin d'obtenir des frontières de décision qui minimisent le taux d'erreur global. Les algorithmes d'apprentissage des prémisses et des conclusions sont incrémentaux, et en temps réel. Ils n'intègrent cependant pas d'oubli, ce qui limite leur capacités d'adaptation dans le temps. Ils permettent d'obtenir un modèle de connaissance global, sur l'ensemble des données du flux. Ce modèle

construit continue de prendre en compte les premières données, même après un changement de concepts qui les rend obsolètes.

Une manière simpliste de maintenir les capacités d'évolution du système serait de l'apprendre sur une fenêtre glissante contenant les dernières données. Les nouvelles données seraient ajoutées à la fenêtre et apprises incrémentalement. De manière symétrique, les anciennes données serait désapprises décrémentalement. Le désapprentissage et l'oubli sont deux choses distinctes. Nous souhaitons ici permettre l'oubli des anciennes données, mais sans pour autant les désapprendre systématiquement, ce qui limiterait les performances du système.

Nous présentons dans ce chapitre le classifieur évolutif *Evolve* ∞, qui est basé sur *Evolve*, et dont l'algorithme d'apprentissage a été modifié pour intégrer de l'oubli. Cet oubli permet d'obtenir un apprentissage en-ligne, qui garantit le maintien dans le temps des capacités d'évolution du système. La contribution majeure est l'intégration d'oubli dans le processus d'apprentissage, afin d'obtenir un apprentissage en-ligne qui suive l'évolution du flux de données. L'intégration d'oubli permet de maintenir la capacité d'apprentissage du système au cours du temps, afin que le classifieur reste évolutif indéfiniment. Cet oubli est à la fois intégré dans le processus d'apprentissage des prémisses et dans celui des conclusions. L'oubli maintient le gain du processus d'apprentissage et permet d'empêcher que l'évolution du modèle de connaissance ralentisse jusqu'à s'arrêter. Le classifieur peut ainsi suivre les évolutions du style d'écriture de l'utilisateur avec le temps, lorsque celui-ci passe de novice à expert et déforme de plus en plus ses gestes, comme pour une signature.

La deuxième contribution principale présentée dans ce chapitre est la capacité de rejet évolutive des données dont dispose *Evolve* ∞. Elle combine une option de rejet de distance et une option de rejet de confusion, toutes les deux évolutives. L'option de rejet de distance permet de rejeter les données trop différentes données d'apprentissage afin de limiter les erreurs lors de la phase de reconnaissance. Elle est également utilisée lors de la phase d'apprentissage pour créer de nouvelles règles et étendre le modèle de connaissance du classifieur lorsque cela est nécessaire. L'option de rejet de confusion permet de limiter les erreurs commises par le système lorsqu'il hésite entre plusieurs classes lors de la phase de reconnaissance. Elle dispose d'une architecture avancée, avec des seuils de rejet multiples, pour de meilleures performances.

En plus de l'option de rejet, une mesure de confiance interne est également disponible, qui permet d'évaluer l'adéquation entre les données et le modèle de connaissance actuel du classifieur *Evolve* ∞. Cette mesure de confiance combine les notions de rejet de distance et de rejet de confusion, tout en étant basé sur la partie générative du modèle, ce qui lui permet d'être très réactive. Elle est utilisable à un stade très précoce de l'apprentissage, alors que l'option de rejet nécessite un certain nombre de données pour être efficace. Cette mesure de confiance est notamment utilisée par le superviseur actif en-ligne *IntuiSup*, que nous présentons dans le chapitre 6, pour superviser l'apprentissage avec l'utilisateur dans la boucle.

La création de nouvelles règles est réalisée par un nouvel algorithme de regroupement incrémental spécifique, qui tire parti de la capacité de rejet évolutive d'*Evolve* ∞. Cet algorithme permet de créer de nouvelles règles lorsque de nouvelles classes sont ajoutées, et

lorsqu'une classe évolue brusquement (changement de concepts abrupt). Il tire notamment parti de l'option de rejet de distance pour étendre le modèle de connaissance du système lorsque cela est nécessaire pendant la phase d'apprentissage du système.

Nous commençons par décrire l'architecture d'*Evolve* ∞, et ses différences par rapport à *Evolve* dans la section 5.2. Nous présentons les nouveaux algorithmes d'apprentissage intégrant de l'oubli, pour les prémisses dans la section 5.3, et pour les conclusions dans la section 5.4. Nous détaillons ensuite dans la section 5.5 la capacité de rejet évolutive d'*Evolve* ∞, et son utilisation par l'algorithme de création de règles dans la section 5.6.

5.2 Architecture d'*Evolve* ∞

Evolve ∞, comme *Evolve*, est un système d'inférence floue (SIF) d'ordre un, dit de Takagi-Sugeno [Takagi and Sugeno, 1985]. Cette section présente l'architecture d'*Evolve* ∞, qui est similaire à celle d'*Evolve* à l'exception de la fonction d'activation des prototypes utilisée.

Un système d'inférence floue (SIF) est composé d'un ensemble de règles d'inférence comme suit :

$$\textbf{Rule}^{(i)} : \textbf{SI } \mathbf{x} \text{ est proche de } C^{(i)} \tag{5.1}$$

$$\textbf{ALORS } \hat{\mathbf{y}}^{(i)} = (\hat{\mathbf{y}}_1^{(i)} ; \ldots ; \hat{\mathbf{y}}_c^{(i)})^\top \tag{5.2}$$

Où $\mathbf{x} \in \mathbb{R}^n$ est le vecteur des n caractéristiques, $C^{(i)}$ le prototype flou associé à la i-ème règle et $\hat{\mathbf{y}}^{(i)\top} \in \mathbb{R}^c$ le vecteur de résultat contenant les scores de c classes. $\top$ représente l'opération de transposition.

Les prémisses des règles sont les degrés d'appartenance floue aux prototypes des règles, qui sont les centroïdes des groupes *(clusters)* dans l'espace d'entrée. Les conclusions des règles sont les degrés d'appartenance floue à toutes les classes. Les sorties des différentes règles sont combinées pour produire le vecteur résultat global.

5.2.1 Structure des prémisses

Evolve ∞ utilise des prototypes de type hyper-elliptique orientée, qui sont chacun définis par un centre $\boldsymbol{\mu}^{(i)} \in \mathbb{R}^n$ et une matrice de covariance $\Sigma^{(i)} \in \mathbb{R}^{n \times n}$ où n est la taille du vecteur d'entrée (nombre de caractéristiques des données).

$$\Sigma^{(i)} = \begin{bmatrix} \sigma_1^2 & \ldots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \ldots & \sigma_n^2 \end{bmatrix} \tag{5.3}$$

À la différence d'*Evolve*, *Evolve* ∞ ne calcule pas l'activation des prototypes $\alpha^{(i)}(\mathbf{x})$ (le degré d'appartenance flou d'une donnée $\mathbf{x}$ au prototype $C^{(i)}$) avec une distribution normale multivariée mais avec la fonction suivante :

$$\alpha^{(i)}(\mathbf{x}) = \frac{1}{1 + \left(\mathbf{x} - \boldsymbol{\mu}^{(i)}\right)^\top (\Sigma^{(i)})^{-1} (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top\big)^{1/2}} \tag{5.4}$$

Cette fonction d'activation est plus progressive, et permet ainsi à toutes les règles de participer au processus de reconnaissance.

5.2.2 Structure des conclusions

Les conclusions des règles d'inférence donnent les degrés d'appartenance floue à toutes les classes. Ces degrés d'appartenance sont obtenus par des fonctions linéaires (ordre un).

$$\hat{\mathbf{y}}^{(i)\top} = (l_1^{(i)}(\mathbf{x})\,;\,\ldots\,;\,l_c^{(i)}(\mathbf{x})) \tag{5.5}$$

Où $l_k^{(i)}(\mathbf{x})$ représente la fonction linéaire de la i-ème règle (de coefficient $\theta_{0,k}^{(i)},\ldots,\theta_{n,k}^{(i)}$, regroupés dans le vecteur $\boldsymbol{\theta}_k^{(i)}$) donnant le degré d'appartenance de la donnée $\mathbf{x}$ à la k-ème classe :

$$l_k^{(i)}(\mathbf{x}) = \mathbf{x}^\top \cdot \boldsymbol{\theta}_k^{(i)} = \theta_{0,k}^{(i)} + \theta_{1,k}^{(i)} \cdot x_1 + \cdots + \theta_{n,k}^{(i)} \cdot x_n \tag{5.6}$$

La conclusion de la i-ème règle peut être reformulée comme suit :

$$\hat{\mathbf{y}}^{(i)\top} = \mathbf{x}^\top \cdot \Theta^{(i)} \tag{5.7}$$

Où $\Theta^{(i)} \in \mathbb{R}^{n\times c}$ est la matrice regroupant les coefficients des c fonctions linéaires :

$$\Theta^{(i)} = (\boldsymbol{\theta}_1^{(i)}\,;\,\ldots\,;\,\boldsymbol{\theta}_c^{(i)}) = \begin{bmatrix} \theta_{1,1}^{(i)} & \cdots & \theta_{1,c}^{(i)} \\ \vdots & \ddots & \vdots \\ \theta_{n,1}^{(i)} & \cdots & \theta_{n,c}^{(i)} \end{bmatrix} \tag{5.8}$$

5.2.3 Processus d'inférence

Le processus d'inférence est composé de trois étapes :

1. Les degrés d'activation $\alpha^{(i)}(\mathbf{x})$ de toutes les règles sont calculés par l'équation 5.4 pour la donnée $\mathbf{x}$, puis normalisés comme suit :

$$\bar{\alpha}^{(i)}(\mathbf{x}) = \frac{\alpha^{(i)}(\mathbf{x})}{\sum_{j=1}^{r} \alpha^{(j)}(\mathbf{x})} \tag{5.9}$$

   Où r est le nombre de règles.

2. Les sorties des règles $\hat{\mathbf{y}}^{(i)}$ sont calculées en utilisant l'équation 5.7 et la sortie globale du système $\hat{\mathbf{y}}$ est calculée par inférence somme-produit :

$$\hat{\mathbf{y}} = \sum_{k=1}^{r} \bar{\alpha}^{(k)}(\mathbf{x}) \cdot \hat{\mathbf{y}}^{(k)} \tag{5.10}$$

3. La classe prédite est celle qui correspond à la valeur de sortie la plus élevée :

$$\text{class}(\mathbf{x}) = \arg\max_{k=1}^{c}(\hat{y}_k) \tag{5.11}$$

5.3 Algorithme d'apprentissage des prémisses

Cette section présente le nouvel algorithme d'apprentissage des prémisses d'*Evolve* ∞, qui intègre de l'oubli afin de maintenir la capacité d'apprentissage du système indéfiniment. Cette contribution est inspirée de l'utilisation d'un facteur d'oubli exponentiel qui est parfois utilisé dans la littérature dédiée au contrôle de systèmes physiques complexes. C'est ce principe qui est peut être utilisé pour intégrer de l'oubli dans l'algorithme des moindres carrés récursifs, mais dont l'implémentation pose des problèmes (comme expliqué en section 4.5 page 76).

5.3.1 Adaptation statistique avec oubli

Pour rappel, les prototypes sont chacun définis dans l'espace d'entrée à l'instant t par un centre $\boldsymbol{\mu}_t^{(i)}$:

$$\boldsymbol{\mu}_t^{(i)} = (\mu_1^{(i)}; \ldots; \mu_n^{(i)})^\top \tag{5.12}$$

Et une matrice de covariance $\Sigma_t^{(i)}$:

$$\Sigma_t^{(i)} = \begin{bmatrix} \sigma_1^2 & \ldots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \ldots & \sigma_n^2 \end{bmatrix} \tag{5.13}$$

Où n est la dimension de l'espace d'entrée (nombre de caractéristiques), i le numéro de la règle ($1 \leq i \leq r$) et $\top$ représente l'opération de transposition.

À la fois le centre $\boldsymbol{\mu}_t^{(i)}$ et la matrice de covariance $\Sigma_t^{(i)}$ sont mis à jour de manière incrémentale. De l'oubli est intégré pour limiter l'importance des anciennes données et maintenir la capacité d'adaptation des prémisses. Pour cela, le poids accordé à l'ancienne valeur lors de la mise à jour statistique est limité à une valeur maximum. L'utilisation d'un poids maximum est équivalente à l'utilisation d'un facteur d'oubli exponentiel, et revient donc à optimiser les prémisses sur une fenêtre de données exponentiellement pondérée.

La valeur de ce poids maximum p_{max} est une variable dépendant du problème et doit être déterminée empiriquement pour ne pas dégrader les performances du système en rendant les prototypes instables. Elle dépend de différents paramètres comme le jeu de caractéristiques utilisé et de la difficulté intrinsèque du problème de classification. L'idée étant ici de maintenir les capacités d'évolution du système dans le temps, le poids maximum associé aux anciennes valeurs doit être suffisamment important pour maintenir une certaine stabilité des prémisses sans dégrader les performances du système. En pratique, nous plafonnons ce poids à celui de trente données, afin d'assurer une évolution suffisamment lente pour ne pas perturber le processus d'optimisation des conclusions.

Les centres et les matrices de covariance de prototypes sont mis à jour à l'instant t de la façon suivante :

$$\boldsymbol{\mu}_t^{(i)} = \frac{(p-1) \cdot \boldsymbol{\mu}_{t-1}^{(i)} + \mathbf{x}_t}{p} \tag{5.14}$$

$$\Sigma_t^{(i)} = \frac{(p-1)\cdot\Sigma_{t-1}^{(i)} + (\mathbf{x}_t - \boldsymbol{\mu}_{t-1}^{(i)})\cdot(\mathbf{x}_t - \boldsymbol{\mu}_t^{(i)})}{p} \tag{5.15}$$

Où p est le poids accordé à l'ancienne valeur :

$$p = min(nb_t^{(i)}, p_{max}) \tag{5.16}$$

Et $nb_t^{(i)}$ est le nombre de données ayant participé à l'apprentissage du i^{e} prototype jusqu'à l'instant t.

5.4 Algorithme d'apprentissage des conclusions

Le nouvel algorithme d'optimisation des conclusions d'*Evolve* ∞ utilise une fenêtre glissante qui conserve les dernières données d'apprentissage, afin de permettre l'oubli futur de la connaissance issue de ces données. Cette contribution est inspirée de l'algorithme de désapprentissage décrémental, qui est le symétrique de l'algorithme d'apprentissage incrémental qui est présenté ci-dessous. Toutefois, nous ne souhaitons pas désapprendre les anciennes données, ce qui limiterait trop les performances des conclusions, mais seulement permettre l'oubli des anciennes données. La connaissance issue des anciennes données n'étant pas forcément obsolète, nous ne souhaitons pas la désapprendre systématiquement. Par contre, il faut que cette connaissance puisse être remplacée par celle issue des nouvelles données. Nous souhaitons donc limiter le poids qui est attribué aux données passées dans le processus d'optimisation.

À l'arrivée d'une nouvelle donnée, l'apprentissage des conclusions d'*Evolve* ∞ se déroule en trois temps qui sont détaillés figure 5.1.

— Une première mise à jour incrémentale permet d'inclure la nouvelle donnée dans le processus d'optimisation.
— Une seconde mise à jour, appelée mise à jour décrémentale, permet d'intégrer de l'oubli dans le processus d'apprentissage afin qu'il reste réactif et efficace indéfiniment.
— Enfin, la fenêtre glissante de données est mise à jour, en ajoutant la nouvelle donnée et en supprimant la plus vieille si la fenêtre est pleine.

L'intérêt d'intégrer de l'oubli dans le processus d'apprentissage des conclusions est double. Premièrement, l'oubli est nécessaire pour limiter le poids des données passées dans le processus d'apprentissage, et ainsi maintenir la capacité du système à apprendre de nouvelles classes. Si chaque donnée avait le même poids relatif, inversement proportionnel au nombre de données, le gain de l'apprentissage tendrait vers zéro et le système se figerait peu à peu, perdant alors son caractère évolutif. Deuxièmement, la capacité d'oubli permet au système de suivre les évolutions du flux de données et de s'adapter aux changements de concepts [Widmer and Kubat, 1996] en oubliant les données obsolètes.

D'un autre côté, il ne faut pas trop oublier non plus. L'objectif est ici de maintenir le caractère évolutif du système et d'améliorer son comportement lorsque le flux de données n'est pas stationnaire. Par contre, il ne faut pas limiter les performances du système lorsque l'environnement est stationnaire, ni risquer de causer l'effondrement du système en oubliant

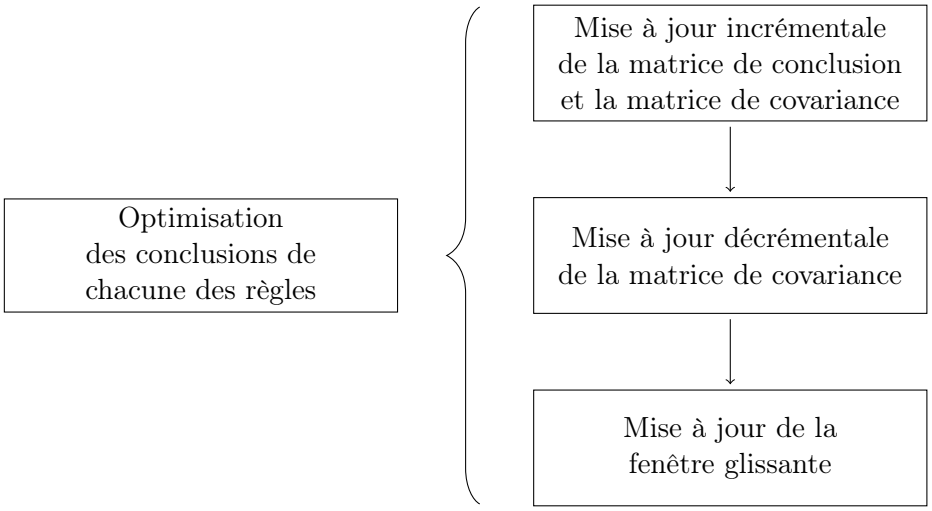


FIGURE 5.1 – Détail du processus d'apprentissage des conclusions d'*Evolve* ∞.

plus que le système n'apprend, ce qui est appelé « oubli catastrophique » *(catastrophic forgetting)*.

Nous expliquons tout d'abord comment sont initialisées les conclusions lors de la création de nouvelles règles. Nous présentons rapidement la mise à jour incrémentale des conclusions qui est faite de manière classique par l'algorithme des moindres carrés récursifs *(recursive least squares)*. Nous détaillons ensuite l'algorithme de mise à jour décrémentale que nous avons mis au point afin d'intégrer de l'oubli dans le processus d'apprentissage.

5.4.1 Mise à jour incrémentale

Comme il n'est pas envisageable de reconstruire les conclusions à l'arrivée de chaque nouvelle donnée, les conclusions sont mises à jour avec l'algorithme des moindres carrés récursifs *(recursive least squares)* comme originalement pour *Evolve* [Almaksour and Anquetil, 2011]. L'algorithme des moindres carrés récursifs [Haykin, 2001] est bien connu pour obtenir la solution au sens des moindres carrés incrémentalement. Il est ici utilisé pour mettre à jour et apprendre les conclusions incrémentalement.

Soit $\Theta_t^{(i)} \in \mathbb{R}^{n \times c}$ la matrice de conclusion de la i$^{\text{e}}$ règle à l'instant t. Cette matrice est mise à jour à l'arrivée de chaque nouvelle donnée (x_t, y_t) (avec $x_t \in \mathbb{R}^n$ le vecteur d'entrée des n caractéristiques, et $y_t^\top \in \mathbb{R}^c$ le vecteur de sortie binaire d'appartenance aux c classes) comme suit :

$$\Theta_t^{(i)} = \Theta_{(t-1)}^{(i)} + P_t^{(i)} \cdot x_t \cdot \alpha_t^{(i)}(\mathbf{x}) \cdot (y_t - x_t^T \cdot \Theta_{(t-1)}^{(i)}) \tag{5.17}$$

Où la matrice de corrélation $P_t^{(i)} \in \mathbb{R}^{n \times c}$ est également mise à jour :

$$P_t^{(i)} = P_{(t-1)}^{(i)} - \frac{P_{(t-1)}^{(i)} \cdot x_t \cdot x_t^T \cdot P_{(t-1)}^{(i)}}{\frac{1}{\alpha_t^{(i)}(\mathbf{x})} + x_t^T \cdot P_{(t-1)}^{(i)} \cdot x_t} \tag{5.18}$$

Avec $\alpha_t^{(i)} \in \mathbb{R}$ le degré d'activation normalisé de la i^{e} règle pour le vecteur x_t, et $\top$ représente l'opération de transposition.

Les conclusions des matrices sont initialisées à zéro, $\Theta_0^{(i)} = 0$, et les matrices de corrélation sont initialisées à la matrice identité pondérée $P_0^{(i)} = \Omega \cdot Id$, où Ω dépend du rapport signal sur bruit [Moustakides, 1997]. Classiquement $\Omega = 100$ pour l'apprentissage des conclusions d'un système d'inférence floue [Angelov and Filev, 2004].

5.4.2 Mise à jour décrémentale

Introduire de l'oubli dans l'algorithme des moindres carrés récursifs *(recursive least squares)* est un problème qui a beaucoup été étudié dans la littérature dédiée au contrôle [Lughofer and Angelov, 2011], mais qui reste sans solution universelle comme nous l'avons expliqué en section 3.2.2 (page 50). Le principe de l'oubli dans cet algorithme est d'empêcher la matrice de corrélation, que l'on peut voir comme la représentation du gain de l'apprentissage au cours du temps, de tendre vers zéro. Toute la difficulté réside dans le fait qu'il faut empêcher cette matrice de converger vers zéro, mais sans pour autant la faire diverger, ce qui cause l'effondrement des conclusions.

5.4.2.1 Techniques existantes

De nombreuses techniques d'oubli ont été proposées pour l'algorithme des moindres carrés récursifs, notamment dans la littérature dédiée au contrôle de systèmes physiques complexes. L'approche la plus courante est notamment l'utilisation d'un facteur d'oubli exponentiel [Kulhavy and Zarrop, 1993]. Ce facteur exponentiel revient virtuellement à utiliser l'algorithme des moindres carrés récursifs sur une fenêtre de données pondérées exponentiellement, ce qui correspond tout à fait à ce que l'on souhaite.

Cependant, cette technique marche mal lorsque le système n'est pas uniformément activé : la matrice de corrélation finit par diverger. Ce problème est connu comme le problème d'explosion de la matrice de corrélation *(covariance wind-up problem)* [Liavas and Regalia, 1999].

Ce problème apparaît lorsque certains éléments de la matrice de corrélation augmentent anormalement, ce qui fait que l'estimateur fait des pas trop grands, ratant la valeur optimale, et finit par diverger. Ce problème d'explosion est dû au fait que le facteur d'oubli produit un oubli uniforme, alors que l'excitation du système varie avec le temps, et n'est pas uniforme dans l'espace des caractéristiques. Plusieurs solutions *had-oc* ont été proposées pour maîtriser cette instabilité dans certains contextes, mais aucune n'est universelle [Fortescue et al., 1981] [Salgado et al., 1988] [Hägglund, 1983].

Au final, il n'existe que peu de stratégies d'oubli garantissant à la fois la non-convergence et la non-divergence de la matrice de corrélation. Nous présentons donc dans la prochaine section une nouvelle technique qui permet d'intégrer de l'oubli sans mettre en danger la stabilité du système, c'est-à-dire de garantir à la fois la non-convergence et la non-divergence de la matrice de covariance.

5.4.2.2 La nouvelle technique d'oubli directionnel différé DDF

Nous présentons ici une nouvelle technique d'oubli directionnel différé DDF *(Differed Directionnal Forgetting)*, non pas basée sur un facteur d'oubli exponentiel, mais sur les anciennes données qui sont stockées dans une fenêtre glissante. L'idée d'utiliser une fenêtre glissante sur un flux de données n'est pas nouvelle. Elle est notamment utilisée pour avoir des estimateurs réactifs qui s'adaptent aux changements de concepts. Nous utilisons cette fenêtre pour obtenir une nouvelle technique d'apprentissage décrémental pour l'algorithme des moindres carrés récursifs utilisé pour l'apprentissage des conclusions des systèmes d'inférence floue d'ordre un. Lorsqu'une nouvelle donnée a été apprise, elle est ajoutée à la fenêtre glissante, et lorsque celle-ci est pleine, la plus ancienne donnée est supprimée. Les données sont considérées anciennes et obsolètes lorsqu'elles sortent de la fenêtre glissante. La longueur de la fenêtre est un point sensible, sujet au compromis stabilité/réactivité, qui sera discuté ci-dessous.

Cette technique d'apprentissage décrémental est basée sur le principe du désapprentissage de la matrice de covariance. De la même manière que les conclusions sont mises à jour incrémentalement avec l'algorithme des moindres carrés récursifs, nous les mettons également à jour décrémentalement avec ce que nous appelons l'algorithme des moindres carrés dérécursifs, pour désapprendre les anciennes données. Par contre, seules les matrices de corrélation sont désapprises, pour intégrer de l'oubli et permettre à la connaissance future de remplacer la connaissance ancienne, mais les matrices de conclusion ne sont pas modifiées pour ne pas détériorer les performances du système. En désapprenant les anciennes données de la matrice de corrélation, on obtient un oubli directionnel, c'est-à-dire que l'oubli est effectué dans la direction des anciennes données.

En pratique l'oubli directionnel est obtenu par l'équation suivante, de mise à jour décrémentale de la matrice de covariance utilisée par les moindres carrés récursifs :

$$\overline{\mathrm{C}_t^{(i)}} = \mathrm{C}_t^{(i)} + \frac{\mathrm{C}_t^{(i)} \cdot \mathbf{x}_s \cdot \mathbf{x}_s^\top \cdot \mathrm{C}_t^{(i)}}{\frac{-1}{\alpha_s^{(i)}(\mathbf{x}_s)} + \mathbf{x}_s^\top \cdot \mathrm{C}_t^{(i)} \cdot \mathbf{x}_s} \tag{5.19}$$

Où $\mathrm{C}_t^{(i)}$ est la matrice de covariance de la i$^{\mathrm{e}}$ règle à l'instant t, et $\alpha_s^{(i)}(\mathbf{x}_s)$ son degré d'activation à l'instant s par la donnée $\mathbf{x}_s$ qui va sortir de la fenêtre glissante et qui va donc être oubliée. Cette équation est issue de la version dé-récursive des moindres carrés récursifs, que nous allons démontrer ci-dessous.

Cet oubli directionnel permet de garantir à la fois la non-convergence et la non-divergence de la matrice de corrélation, ce qui est fondamental pour garder le système fonctionnel indéfiniment. Les nouvelles classes sont apprises aussi rapidement après une longue période d'utilisation/apprentissage qu'au début de la vie du système. L'utilisation de la fenêtre glissante permet de s'adapter facilement aux changements de concepts en oubliant les anciennes données lorsqu'elles sortent de la fenêtre.

Algorithme des moindres carrés dé-récursifs *(de-recursive least squares)*
De la même manière que l'algorithme des moindres carrés récursifs permet d'incorporer

une nouvelle donnée à une solution existante, l'algorithme des moindres carrés dé-récursifs permet de retirer l'impact d'une donnée qui a été utilisée pour obtenir la solution existante.

Soit $\Theta^{(i)}_{s\to t} \in \mathbb{R}^{n\times c}$ la conclusion de la i$^{\text{e}}$ règle, optimisée sur les données de l'instant s $(\mathbf{x}_s, \mathbf{y}_s)$ à l'instant t $(\mathbf{x}_t, \mathbf{y}_t)$, avec $\mathbf{x}_k \in \mathbb{R}^n$ le vecteur d'entrée à l'instant k, et $\mathbf{y}_k \in \mathbb{R}^c$ le vecteur de sortie binaire associé.

Les solutions des moindres carrés peuvent être mises à jour décrémentalement, pour désapprendre l'ancienne donnée $(\mathbf{x}_s, \mathbf{y}_s)$ avec les formules suivantes :

$$\Theta^{(i)}_{(s+1)\to t} = \Theta^{(i)}_{s\to t} - \alpha^{(i)}_s(\mathbf{x}_s) \cdot \mathrm{C}^{(i)}_{(s+1)\to t} \cdot \mathbf{x}_s \cdot (\mathbf{y}_s^\top - \mathbf{x}_s^\top \cdot \Theta^{(i)}_{s\to t}) \tag{5.20}$$

Où $\alpha^{(i)}_s(\mathbf{x}_s)$ est le degré d'activation de la i$^{\text{e}}$ règle par la donnée $\mathbf{x}_s$. La matrice de corrélation $\mathrm{C}^{(i)}_{(s+1)\to t}$ est également mise à jour décrémentalement :

$$\mathrm{C}^{(i)}_{(s+1)\to t} = \mathrm{C}^{(i)}_{s\to t} + \frac{\mathrm{C}^{(i)}_{s\to t} \cdot \mathbf{x}_s \cdot \mathbf{x}_s^\top \cdot \mathrm{C}^{(i)}_{s\to t}}{\frac{-1}{\alpha^{(i)}_s(\mathbf{x}_s)} + \mathbf{x}_s^\top \cdot \mathrm{C}^{(i)}_{s\to t} \cdot \mathbf{x}_s} \tag{5.21}$$

Preuve:

La fonction de coût $J^{(i)}_{s\to t}$ de la i$^{\text{e}}$ règle (pour $1 \leq i \leq r$ et avec r le nombre de règles) à l'instant t est :

$$J^{(i)}_{s\to t} = \sum_{i=s}^{t} \alpha^{(i)}(\mathbf{x}_i) \cdot (\mathbf{y}_i - \mathbf{x}_i^T \cdot \Theta^{(i)}_t)^2 \tag{5.22}$$

Soit

$$J^{(i)}_{s\to t} = (\mathrm{Y}_{s\to t} - \mathrm{X}_{s\to t} \cdot \Theta^{(i)}_{s\to t})^\top \cdot \mathrm{A}^{(i)}_{s\to t} \cdot (\mathrm{Y}_{s\to t} - \mathrm{X}_{s\to t} \cdot \Theta^{(i)}) \tag{5.23}$$

Avec

$$\mathrm{X}_{s\to t} = [\mathbf{x}_s; \ldots; \mathbf{x}_t]^\top; \quad \mathbf{x}_k = [x_{k,1}; \ldots; x_{k,n}]^\top \tag{5.24}$$

$$\mathrm{Y}_{s\to t} = [\mathbf{y}_s^\top; \ldots; \mathbf{y}_t^\top]^T; \quad \mathbf{y}_k = [y_{k,1}; \ldots; y_{k,n}] \tag{5.25}$$

Où n est la dimension de l'espace d'entrée, $y_{k,l} = 1$ si $y_{k,l}$ appartient à la classe l et $y_{k,l} = 0$ sinon. Les matrices de pondération $\mathrm{A}^{(i)}_{s\to t}$ sont définies comme :

$$\mathrm{A}^{(i)}_{s\to t} = \begin{pmatrix} \alpha^{(i)}_s & & (0) \\ & \ddots & \\ (0) & & \alpha^{(i)}_t \end{pmatrix} \tag{5.26}$$

Où $\alpha^{(i)}_k$ est le degré d'activation de la i$^{\text{e}}$ règle par la donnée x_k, et Id la matrice identité. Les solutions des moindres carrés pondérés sont alors :

$$\Theta^{(i)}_{s\to t} = \mathrm{C}^{(i)}_{s\to t} \cdot \mathrm{X}^\top_{s\to t} \cdot \mathrm{A}^{(i)}_{s\to t} \cdot \mathrm{Y}_{s\to t} \tag{5.27}$$

Avec les matrices de corrélation :

$$R^{(i)}_{s\to t} = (\mathrm{C}^{(i)}_{s\to t})^{-1} = \mathrm{X}_{s\to t} \cdot \mathrm{A}^{(i)}_{s\to t} \cdot \mathrm{X}^\top_{s\to t} \tag{5.28}$$

Les matrices de corrélation peuvent être mise à jour décrémentalement comme suit :

$$\mathrm{R}^{(i)}_{(s+1)\to t} = \mathrm{R}^{(i)}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{x}_s^\top \tag{5.29}$$

Et en utilisant le lemme d'inversion matriciel (identité de Woodbury) :

$$\left(B + X \cdot C \cdot X^\top\right)^{-1} = \tag{5.30}$$

$$B^{-1} - B^{-1} \cdot X \left(C^{-1} + X^\top \cdot B^{-1} \cdot X\right)^{-1} X^\top \cdot B^{-1} \tag{5.31}$$

Où $B = \mathrm{R}^{(i)}_{s\to t}$, $X = \mathbf{x}_s$ et $C = -\alpha^{(i)}_s$.

Il est alors aisé d'obtenir l'équation 5.21 pour mettre à jour décrémentalement les matrices de corrélation.

À partir de l'équation 5.27 il est possible d'écrire :

$$\begin{aligned}
\mathrm{R}^{(i)}_{(s+1)\to t}\Theta^{(i)}_{(s+1)\to t} &= \mathrm{X}^\top_{(s+1)\to t} \cdot \mathrm{A}^{(i)}_{(s+1)\to t} \cdot \mathrm{Y}_{(s+1)\to t} && (5.32)\\
&= \mathrm{X}^\top_{s\to t} \cdot \mathrm{A}^{(i)}_{s\to t} \cdot \mathrm{Y}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{y}_s && (5.33)\\
&= \mathrm{R}^{(i)}_{s\to t} \cdot \Theta^{(i)}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{y}_s && (5.34)\\
&= (\mathrm{R}^{(i)}_{(s+1)\to t} + \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{x}_s^\top) \cdot \Theta^{(i)}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{y}_s && (5.35)\\
&= \mathrm{R}^{(i)}_{(s+1)\to t} \cdot \Theta^{(i)}_{s\to t} + \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{x}_s^\top \cdot \Theta^{(i)}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s \cdot \mathbf{y}_s && (5.36)\\
&= \mathrm{R}^{(i)}_{(s+1)\to t} \cdot \Theta^{(i)}_{s\to t} - \mathbf{x}_s \cdot \alpha^{(i)}_s (\mathbf{y}_s - \mathbf{x}_s^\top \cdot \Theta^{(i)}_{s\to t}) && (5.37)
\end{aligned}$$

Ce qui donne l'équation 5.20.

Désapprentissage et oubli L'algorithme des moindres carré dérécursifs *(de-recursive least squares)* permet de désapprendre complètement une donnée de manière décrémentale. Ce désapprentissage ce fait en deux temps : la matrice de corrélation est mise à jour pour enlever le poids de la donnée à désapprendre, puis la matrice de conclusion est mise à jour pour enlever l'impact de cette donnée. Ainsi, tous les effets que cette donnée a eu sur l'optimisation des conclusions sont enlevés, comme si cette donnée n'avait pas été utilisée pour l'apprentissage.

Dans notre cas, nous ne souhaitons pas désapprendre les anciennes données, mais seulement permettre leur oubli. La connaissance issue des anciennes données n'est pas forcément obsolète, nous ne souhaitons pas la désapprendre systématiquement. Par contre, il faut que cette connaissance puisse être remplacée par celle issue des nouvelles données. Nous souhaitons donc limiter le poids qui lui est accordée dans le processus d'optimisation.

Pour introduire de l'oubli dans le processus d'apprentissage des conclusions, nous utilisons l'équation 5.21 de mise à jour décrémentale de la matrice de corrélation, comme pour l'algorithme des moindres carrés dérécursifs, mais sans toucher à la matrice de conclusion (sans utiliser l'équation 5.20). De cette manière, le poids accordé aux anciennes données est limité, et les conclusions peuvent continuer d'évoluer indéfiniment, maintenant ainsi la capacité d'apprentissage du système.

Longueur de fenêtre et compromis stabilité/réactivité Un point important de cette approche est la longueur de la fenêtre glissante. En effet, il faut faire un compromis entre la stabilité et la réactivité du système qui sont directement liées à la longueur de la fenêtre. Lorsque l'environnement est stable (absence de changement de concepts), plus la fenêtre est longue, meilleure est la stabilité du système et son taux de reconnaissance (jusqu'à un taux maximum). Cependant, une fenêtre trop longue réduit la réactivité du système et détériore ses performances lorsque l'environnement change.

Nous nous focalisons ici sur le fait d'avoir une fenêtre suffisamment grande pour ne pas détériorer les performances en environnement stable. Une telle fenêtre est en effet suffisante pour limiter le poids des anciennes données et maintenir la capacité d'adaptation du système.

La longueur minimum de la fenêtre est une variable dépendant du problème qui doit être déterminée empiriquement. Elle dépend du nombre de classes, du jeu de caractéristiques utilisé et de la difficulté intrinsèque du problème de classification des symboles choisis par l'utilisateur. Nous lions la longueur de la fenêtre au nombre de classes pour éviter une diminution des performances lorsque de nouvelles classes sont ajoutées.

L'inconvénient de cette approche basée sur une fenêtre glissante est la nécessité de mémoriser les données de la fenêtre. Cependant, l'augmentation des besoins en mémoire reste relativement faible comparée aux besoins de l'algorithme initial.

Même si une fenêtre de taille variable pourrait être plus efficace, une fenêtre fixe nous permet de préserver le caractère évolutif du système sans pour autant en limiter les performances, comme nous le montrons expérimentalement au chapitre 7.

5.5 Capacité d'auto-évaluation et de rejet des données

Cette section présente la capacité d'auto-évaluation et de rejet, dont nous avons doté le système *Evolve* ∞, dans plusieurs buts. L'option de rejet de distance est utilisé par l'algorithme de création de règles, la mesure de confiance permet l'utilisation du superviseur actif en-ligne *IntuiSup*, et l'option de rejet de confusion permet de limiter les erreurs de manière classique. La figure 5.2 détaille l'architecture de cette option de rejet multiple.

Tout d'abord, nous avons créé une option de rejet de distance (voir section 2.4.1 page 42) afin de détecter les données qui ne correspondent à aucun des prototypes existants. Cette option de rejet de distance est notamment utilisée, à la place d'un algorithme de regroupement incrémental, pour créer de nouvelles règles lorsque cela est nécessaire.

Ensuite, nous présentons l'option de rejet de confusion évolutive (voir section 2.4.2 page 42) dont dispose le classifieur *Evolve* ∞ pour limiter les erreurs. Cette option de rejet de confusion a notamment une architecture multi-seuils pour une meilleure précision. Elle est apprise en synergie avec le classifieur par un nouvel algorithme d'apprentissage flou.

Enfin, nous avons mis au point une mesure de confiance interne, basée sur les prototypes, pour évaluer les difficultés du système dès son initialisation. Cette mesure permet de guider les utilisateurs lors de la définition de leur jeu de commandes gestuelles afin de ne pas choisir de symboles similaires ou difficiles à reconnaitre pour le classifieur. Cette mesure de confiance permet l'utilisation du superviseur actif en-ligne *IntuiSup*, que nous

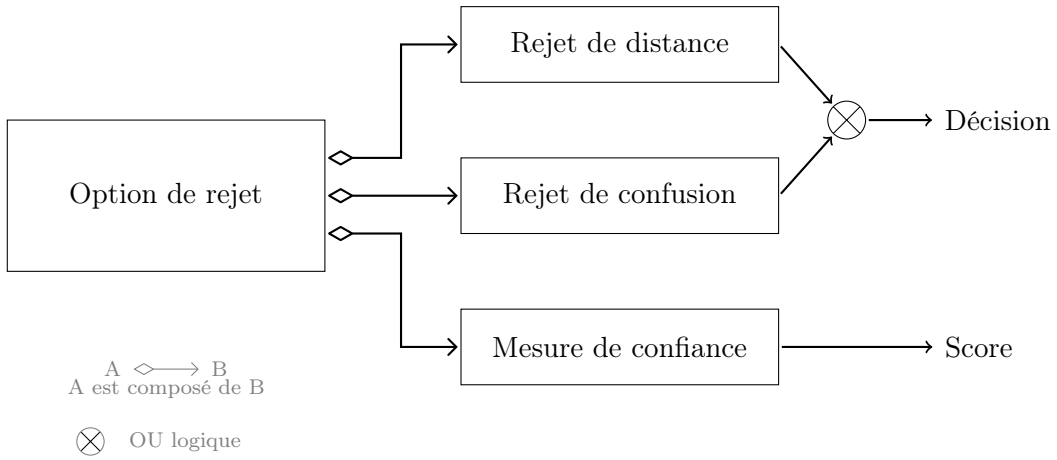


FIGURE 5.2 – Détail de l'architecture de l'option de rejet multiple et évolutive d'*Evolve* ∞.

présenterons dans le chapitre 6, pour superviser l'apprentissage lorsque l'utilisateur est dans la boucle.

5.5.1 Option de rejet de distance

Le rejet de distance permet de délimiter les connaissances du classifieur, et de rejeter les exemples qui ne correspondent pas aux données vues pendant l'apprentissage. Le rejet de distance est donc calculé sur les prototypes, qui forment la couche générative du modèle de connaissance du classifieur *Evolve* ∞.

En pratique, l'option de rejet de distance est évaluée par rapport à la distance aux prototypes. La distance de Mahalanobis est utilisée pour calculer la distance d'une donnée $\mathbf{x}$ à tous les prototypes $C^{(i)}$ (définis par leur centre $\boldsymbol{\mu}^{(i)}$ et leur matrice de covariance $\Sigma^{(i)}$).

$$distance(\mathbf{x}, C^{(i)}) = (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top \cdot (\Sigma^{(i)})^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top \tag{5.38}$$

Où $1 \leq i \leq r$, avec r le nombre de règles, et $\top$ représente l'opération de transposition.

Une donnée est rejetée lorsque :

$$distance(\mathbf{x}, C^{(i)}) > s_{max}; \quad \forall i \setminus 1 \leq i \leq r \tag{5.39}$$

Avec s_{max} le seuil de rejet correspondant à la plus petite distance inter-prototypes :

$$s_{max} = min(distance(C^{(i)}, C^{(j)})) \tag{5.40}$$

$$\forall (i,j) \setminus 1 \leq i \leq r,\ 1 \leq j \leq r,\ i \neq j \tag{5.41}$$

La principale contribution de cette option de rejet de distance est l'utilisation d'une mesure interne (la plus petite distance inter-prototypes) pour fixer le seuil de rejet. Cela permet de ne pas avoir de paramètre dépendant du problème et de l'espace de représentation utilisé à fixer à priori. L'estimation de ce type de paramètre est en effet difficilement faisable lorsque l'apprentissage est réalisé en-ligne et qu'il n'y a pas de base de validation.

Cette option de rejet de distance permet pendant la phase de reconnaissance de ne pas tenter de reconnaitre les symboles qui sont trop loin du modèle de connaissance du système, et qui correspondent soit à un symbole trop déformé, soit à une erreur de l'utilisateur qui a dessiné un symbole inexistant.

Cette option de rejet est aussi utilisée pendant la phase d'apprentissage pour créer de nouveaux prototypes, avec leur règle associée, lorsque l'utilisateur dessine un même symbole de plusieurs façons, ou lorsqu'il utilise plusieurs symboles pour la même commande. Ce nouvel algorithme de création de règle est détaillé en section 5.6.

5.5.2 Option de rejet de confusion évolutive

Cette section présente l'option de rejet de confusion dont nous avons doté notre classifieur, avec une architecture multi-seuils pour une meilleure précision. Cette architecture multi-seuils permet d'améliorer les performances de l'option de rejet [Fumera et al., 2000], en particulier sur le long terme. Pour apprendre efficacement ces seuils multiples, nous avons mis au point un algorithme d'apprentissage flou qui est présenté ci-dessous.

Utiliser le même seuil de rejet pour toutes les classes est loin d'être optimal ; utiliser un seuil différent par classe permet d'être plus précis. Les classes qui sont bien reconnues peuvent ainsi avoir un seuil de rejet plus bas que les classes qui sont proches et souvent confondues par le classifieur. Utiliser des seuils multiples, avec un seuil différent pour chaque classe, permet d'avoir une description plus fine du modèle de connaissance du classifieur et de ses limites, et donc une meilleure capacité de rejet. L'inconvénient de cette architecture plus complexe est qu'il faut plus de temps pour apprendre les différents seuils pour les différentes classes. Une architecture encore plus précise serait d'avoir un seuil différent pour chaque paire de classes. Cependant, une telle architecture aurait c^2 seuils (avec c le nombre de classes), ce qui nécessiterait beaucoup d'exemples pour l'apprentissage, et mettrait trop longtemps pour être utilisée en-ligne.

Utiliser un seuil par classe est un bon compromis, qui donne de bonnes performances mais ne nécessite pas trop d'exemples d'apprentissage. Notre architecture multi-seuils fonctionne comme suit. Nous calculons un score de confiance $conf$ comme la différence normalisée entre les scores $\hat{y}$ obtenus par les deux meilleures classes :

$$conf = \frac{\hat{y}_{first} - \hat{y}_{second}}{\hat{y}_{first}} \tag{5.42}$$

Où $\hat{y}_{first}$ et $\hat{y}_{second}$ sont respectivement le plus grand et le second plus grand score des différentes classes en sortie du système. Nous avons un vecteur de seuils (un seuil pour chaque classe) :

$$\mathbf{s} = (s_1; ...; s_c) \tag{5.43}$$

Où c est le nombre de classes. Une donnée est rejetée lorsque :

$$conf < s_k \tag{5.44}$$

Avec k l'indice de la classe ayant obtenu le plus grand score :

$$k = argmax_{k=1}^{c}(y_k) \tag{5.45}$$

Nous avons mis au point un algorithme d'apprentissage flou pour apprendre efficacement ce vecteur de seuils.

Algorithme d'apprentissage du vecteur multi-seuils Dans le contexte de l'apprentissage de commandes gestuelles, et plus généralement dans n'importe quelle situation d'apprentissage en-ligne, il est important d'avoir une option de rejet efficace rapidement. Pour accélérer le processus d'apprentissage, tous les seuils sont mis à jour par chaque donnée d'apprentissage. Pour cela, le processus d'adaptation des seuils est pondéré par les degrés d'appartenance floue.

Le vecteur de seuils $\mathbf{s_t}$ est obtenu à l'instant t à partir de la moyenne μ_t et de l'écart-type σ_t du score de confiance des données correctement reconnues de chaque classe.

$$\mathbf{s}_t^{dynamic} = \mu_t - \sigma_t \tag{5.46}$$

La moyenne μ_t et l'écart-type σ_t sont récursivement mis à jour, pour chaque classe, avec les formules suivantes :

$$\mu_t = \mu_{t-1} \cdot \frac{nb - \hat{y}_t}{nb} + \hat{y}_t \cdot \frac{\hat{y}_t}{nb} \tag{5.47}$$

$$\sigma_t = \sqrt{\frac{S_t}{t}} \tag{5.48}$$

$$S_t = S_{t-1} \cdot \frac{nb - \hat{y}_t}{nb} + (\hat{y}_t - \mu_t) \cdot (\hat{y}_t - \mu_{t-1}) \cdot \frac{\hat{y}_t}{nb} \tag{5.49}$$

Avec $\hat{y}_t$ est le score de la classe correspondante en sortie du système, $nb = min(n_t, nb\_max)$ le nombre de données passées utilisées pour le calcul des seuils. n_t est le vecteur des quantités floues de données appartenant à chacune des classes, et nb_max le nombre maximum de données prises en compte afin de maintenir le caractère évolutif du rejet.

Ces mises à jour sont pondérées par le vecteur des scores flous $\hat{\mathbf{y}}$. À la place de seulement mettre à jour le seuil correspondant à la classe ayant le plus grand score, tous les seuils sont mis à jour par chaque donnée, mais proportionnellement à son score. En d'autres termes, chaque donnée participe à l'apprentissage de tous les seuils de rejet, mais à différents niveaux, dépendants des scores des classes correspondantes pour cette donnée. Cet algorithme permet d'accélérer le processus d'apprentissage du vecteur de seuils, et son évolution, en tirant parti des proximités entre les classes.

L'utilisation d'une architecture de rejet multi-seuils permet d'obtenir un compromis erreur/rejet plus intéressant qu'avec l'utilisation d'un seuil unique, comme nous le montrons lors de la présentation des résultats dans la section 7.6.

5.5.3 Mesure de confiance interne

Le superviseur actif en-ligne de l'apprentissage nécessite une mesure de confiance pour évaluer les difficultés du classifieur. Pour cela, nous avons développé une mesure de confiance interne, basée sur les prototypes des règles qui forment la partie générative du modèle de

connaissance d'*Evolve* ∞. Cette mesure de confiance permet de mettre en valeur très tôt dans l'apprentissage du système, les données qui sont entre les modèles de deux classes.

De manière classique, le rejet de confusion est calculé à partir du vecteur de sortie du système, contenant les degrés d'appartenance à toutes les classes. Cependant, nous essayons d'évaluer la qualité du modèle de connaissance du classifieur à un stade très précoce de la vie du système. Par conséquent, les conclusions des règles sont encore imprécises et changeantes, et peu représentatives des confusions entre les modèles des différentes classes.

À la place d'utiliser les sorties des conclusions, qui peuvent être vues comme la partie discriminante du système, nous avons basé notre mesure de confiance sur les prémisses du système, qui peuvent être vues comme la partir générative du système. De cette manière, nous pouvons détecter une confusion lorsqu'un symbole active plusieurs prototypes modélisant des classes différentes à des degrés similaires, ce qui veut dire que cette donnée est entre les différents modèles de classes du système.

De plus, cette mesure de confiance interne peut être utilisée pour aider les utilisateurs lors de la définition de leur jeu de commandes gestuelles. L'utilisation de commandes gestuelles donne lieu à une situation d'apprentissage croisé où l'utilisateur doit mémoriser le jeu de commande, et le système les modéliser. D'un côté, permettre aux utilisateurs de personnaliser leur jeu de commandes gestuelles est essentiel pour faciliter leur mémorisation. De l'autre côté, permettre aux utilisateurs de choisir leur propres symboles peut donner lieu à des symboles étranges, ou trop proches dans l'espace de représentation des caractéristiques, qui sont complexes à reconnaitre pour le classifieur.

Pour faciliter cette situation d'apprentissage croisé, nous avons mis au point un mécanisme de détection des confusions pour aider les utilisateurs lors de l'étape initiale de définition des commandes gestuelles. Le but est de mettre en lumière les symboles similaires que le classifieur va avoir tendance à confondre ou mal reconnaitre. Cela permet à l'utilisateur de changer les symboles problématiques, ou de dessiner davantage d'exemples pour améliorer la modélisation de ces classes difficiles.

En pratique, la mesure de confiance interne est calculée selon les principes du rejet de confusion, mais sur les activations des prototypes. La distance de Mahalanobis est utilisée pour calculer la distance d'une donnée $\mathbf{x}$ à tous les prototypes $C^{(i)}$ (définis par leur centre $\boldsymbol{\mu}^{(i)}$ et leur matrice de covariance $\Sigma^{(i)}$).

$$distance(C^{(i)}, \mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top \cdot (\Sigma^{(i)})^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu}^{(i)})^\top \tag{5.50}$$

À partir de cette distance, nous calculons une mesure de similarité semblable à l'activation des prototypes :

$$sim(C^{(i)}, \mathbf{x}) = \frac{1}{1 + distance(C^{(i)}, \mathbf{x})} \tag{5.51}$$

Avec cette mesure de similarité, nous calculons la mesure de confiance comme :

$$conf = sim_{first} - sim_{second} \tag{5.52}$$

Où sim_{first} et sim_{second} sont respectivement la plus grande et la seconde plus grande valeur de similarité. Cette mesure peut ensuite être utilisée pour évaluer la confusion du système à un stade très précoce de l'apprentissage.

Cette mesure de confiance est différente des options de rejet présentées ci-dessus, et fusionne les notions de rejet de distance et de confusion au sein de la même mesure. Les données qui ne correspondent à aucun prototype (rejet de distance) obtiennent un faible score de confiance car s_{first} et s_{second} sont faibles. Les données qui semblent correspondre à plus d'un prototype (rejet de confusion) obtiennent un faible score de confiance car s_{first} et s_{second} sont proches. Ensuite, cette mesure est basée sur la partie génératrice du modèle de connaissance du classifieur, ce qui lui permet d'être efficace à un stade très précoce de l'apprentissage du système.

Cette mesure de confiance permet notamment au superviseur actif en-ligne *IntuiSup* de prendre la décision d'échantillonnage, comme détaillé en section 6.3. Elle permet également de guider l'utilisateur lors de la phase initiale de définition des gestes. Ces propriétés sont validées expérimentalement dans la section 7.6.

5.6 Algorithme de création de règles

La création de nouvelles règles est habituellement réalisée par un algorithme de regroupement *(clustering)* incrémental. Réaliser un partitionnement des données est une tâche complexe lorsque les données arrivent en flux. La majorité des algorithmes existants se basent sur deux structures de données : une première contenant le partitionnement actuel qui évolue avec les données, et une deuxième contenant des informations sur le flux de données pour prendre la décision de création de nouveaux groupes *(clusters)*.

L'algorithme que nous avons mis au point est beaucoup plus simple : il cherche seulement à compléter le partitionnement existant. Lorsque le partitionnement existant est insuffisant, un nouveau groupe est créé pour le compléter et uniformiser le découpage de l'espace.

L'algorithme utilisé pour créer de nouvelles règles, avec les prototypes et conclusions associés, est en deux étapes. Une première étape de regroupement incrémental naïf permet d'assurer une séparation minimum des données. Une deuxième étape, basée sur l'utilisation du rejet de distance, permet d'avoir une séparation plus fine. Contrairement à celui d'*Evolve*, cet algorithme permet la création de nouvelles règles, même lorsque le système travaille avec un espace de représentation de dimension cinquante, comme c'est le cas avec le jeu de caractéristiques HBF49 [Delaye and Anquetil, 2013] que nous utilisons.

5.6.1 Regroupement incrémental naïf

L'algorithme de regroupement incrémental naïf est très simple, il consiste à créer au minimum une règle par classe. Même si toutes les règles participent à la reconnaissance de toutes les classes, il est indispensable d'avoir au moins une règle par classe pour garantir de bonnes performances de reconnaissance. Le prototype associé peut ainsi suivre l'évolution des données de cette classe, sans devoir faire de compromis avec les autres classes. Il parait en effet peu logique d'avoir un seul prototype modélisant deux symboles différents.

L'objectif de cet algorithme de regroupement incrémental naïf est de garantir une séparation minimale, qu'il y ait au moins une règle par classe. Cette séparation minimale

des données dans l'espace d'entrée peut ensuite être affinée par un algorithme de regroupement incrémental avancé. Dans tous les cas, indépendamment du second algorithme utilisé, ce regroupement incrémental naïf est obligatoire pour garantir le bon fonctionnement de l'apprentissage incrémental.

En pratique, lorsqu'une nouvelle classe est ajoutée, la première donnée de cette classe sert de centre au nouveau prototype, et la matrice de covariance est initialisée à sa valeur par défaut.

Cet algorithme de regroupement incrémental naïf permet déjà, en dépit de sa simplicité, d'obtenir des résultats très satisfaisants tant qu'il n'y a pas plusieurs types de symboles différents par classe.

5.6.2 Utilisation du rejet de distance

La seconde étape de regroupement est réalisée à l'aide de l'option de rejet de distance. C'est une méthode très simple, mais efficace car elle permet de créer de nouveaux prototypes là où il en manque. Le principal avantage de l'utilisation de l'option de rejet de distance évolutive d'*Evolve* ∞ est que son seuil de rejet est basé sur une mesure interne. Ce seuil est fixé de manière automatique et en accord avec les données et l'espace de représentation utilisé comme nous l'avons expliqué en section 5.5.1. Le rejet de distance permet de délimiter le modèle de connaissance du classifieur. L'utiliser comme algorithme de regroupement incrémental permet d'étendre le modèle du classifieur là où c'est nécessaire.

L'utilisation de la capacité de rejet du classifieur permet d'obtenir un partitionnement relativement uniforme de l'espace d'entrée. De nouveaux prototypes sont créés dans les espaces vides entre les prototypes existants. Cette technique créée ainsi un maillage de l'espace d'entrée en étendant le partitionnement existant, avec approximativement la même densité.

Enfin, le fait que le seuil de rejet utilisé pour le rejet de distance soit la plus petite distance entre les prototypes existants garantit un espacement minimum. Le modèle du classifieur est donc étendu, mais la densité de prototypes n'augmentera pas plus que nécessaire. Par conséquent, cette technique nous préserve du problème de sur-partitionnement.

5.7 Conclusion

Nous avons présenté dans ce chapitre le classifieur évolutif en-ligne *Evolve* ∞ que nous avons mis au point à partir du classifieur *Evolve*. Nous avons décrit plusieurs contributions que nous avons apportées à ce système, dont un algorithme d'apprentissage intégrant de l'oubli pour maintenir son caractère évolutif, et une capacité de rejet multiple et évolutive.

Evolve est un système d'inférence floue *(fuzzy inference system)* d'ordre un. C'est un système à deux étages : les prémisses et les conclusions des règles d'inférence. Les prémisses forment la partie générative du système, elles sont modélisées par des prototypes dans l'espace d'entrée. Les conclusions forment la partie discriminante du système, elles sont obtenues par des régressions linéaires.

L'algorithme d'apprentissage d'*Evolve* est incrémental, et fonctionne en temps réel grâce à sa faible complexité. Il traite les données une par une pour construire un modèle de connaissance global, sur l'ensemble des données du flux. Son architecture sous forme d'ensemble de règles lui permet d'intégrer de nouvelles connaissances et de nouvelles classes pendant son utilisation, sans pour autant perturber la connaissance existante, et ainsi d'obtenir un très bon compromis plasticité/stabilité. L'algorithme d'apprentissage *Evolve* n'intègre pas d'oubli, ce qui fait qu'il n'est pas capable de suivre des changements de concepts s'ils interviennent après une longue période d'apprentissage. Le gain de son algorithme d'apprentissage diminue avec le temps, l'inertie de son modèle de connaissance augmente et le système finit par ne plus évoluer.

Evolve ∞ intègre notamment de l'oubli dans son algorithme d'apprentissage pour maintenir la capacité d'évolution du système au cours du temps. Cet oubli est à la fois intégré dans le processus d'apprentissage des prémisses et dans celui des conclusions. L'oubli maintient le gain du processus d'apprentissage et permet d'empêcher la convergence du modèle de connaissance du système. Le classifieur peut ainsi suivre les évolutions du style d'écriture de l'utilisateur avec le temps, lorsque celui-ci passe de novice à expert et déforme de plus en plus ses gestes, à la manière d'une signature.

Evolve ∞ dispose également d'une capacité de rejet multiple et évolutive. L'option de rejet de distance permet de rejeter les données qui ne correspondent pas aux données d'apprentissage. Elle est notamment utilisée par l'algorithme de création de règles pour étendre le modèle de connaissance du classifieur lorsque cela est nécessaire. L'option de rejet de confusion permet de limiter les erreurs commises par le système lorsqu'il hésite entre plusieurs classes. Elle dispose d'une architecture avancée, avec des seuils de rejet multiples, pour de meilleures performances. Un algorithme d'apprentissage flou permet d'apprendre ces différents seuils et de les faire évoluer en synergie avec l'apprentissage du classifieur. Une mesure de confiance interne est également disponible, et permet d'évaluer la confiance du système lors du processus de reconnaissance, même au tout début de l'apprentissage du système. Elle permet notamment au superviseur actif en-ligne *IntuiSup*, qui est présenté chapitre 6, de prendre la décision d'interagir avec l'utilisateur lorsque celui-ci est dans la boucle d'apprentissage. Cette mesure peut être utilisée à un stade très précoce de l'apprentissage, ce qui est particulièrement utile dans la période d'apprentissage initiale du système.

Dans *Evolve* ∞, la création de nouvelles règles est réalisée par un algorithme de regroupement incrémental spécifique. Cet algorithme permet de créer de nouvelles règles lorsque de nouvelles classes sont ajoutées, et lorsqu'une classe évolue brusquement (changements de concepts abrupts). Il tire notamment parti de l'option de rejet de distance pour étendre le modèle de connaissance du système lorsque cela est nécessaire.

Evolve ∞ est un classifieur évolutif générique, qui peut être utilisé dans n'importe quelle situation d'apprentissage en-ligne. La capacité d'*Evolve* ∞ à apprendre pendant son utilisation en fait un classifieur très adapté pour les contextes où très peu de données d'apprentissage sont disponibles initialement, comme celui de la reconnaissance de commandes gestuelles personnalisées. Lorsque l'utilisateur est dans la boucle d'apprentissage, il est nécessaire d'avoir un système de supervision actif en-ligne comme celui que nous allons

présenter dans le prochain chapitre.

Chapitre 6

IntuiSup : un superviseur actif en-ligne

6.1 Introduction

Ce chapitre présente le superviseur *IntuiSup (Intuitive Supervisor)* qui permet de superviser l'apprentissage d'un classifieur de commandes gestuelles. Cet environnement a été mis au point pour superviser l'apprentissage du classifieur *Evolve* ∞, mais peut aussi bien superviser l'apprentissage de n'importe quel autre classifieur évolutif, comme ceux vus au chapitre 3. Le seul prérequis est que le classifieur fournisse un score de confiance lors de la reconnaissance des données pour pouvoir réaliser la sélection des données d'apprentissage.

L'apprentissage et la reconnaissance de commandes gestuelles est une situation d'apprentissage croisé : l'utilisateur doit mémoriser les commandes et le classifieur doit les apprendre à les reconnaitre. Afin de pouvoir apprendre et améliorer le classifieur pendant son utilisation, il est nécessaire d'interagir avec l'utilisateur pour pouvoir consolider l'étiquetage des données. Les données peuvent en effet être étiquetées avec l'étiquette reconnue dans certains cas, mais il est parfois nécessaire que l'utilisateur confirme ou corrige cet étiquetage pour apprendre efficacement.

D'un autre côté, solliciter constamment l'utilisateur est ennuyeux, et réduit considérablement la fluidité d'utilisation des commandes gestuelles. Il faut faire un compromis entre le nombre d'interactions avec l'utilisateur et le nombre d'erreurs de reconnaissance. Nous avons donc mis au point un système de supervision active en-ligne pour optimiser la coopération entre l'utilisateur et le classifieur dans cette situation d'apprentissage croisé.

Le système de supervision actif en-ligne *IntuiSup* est composé de deux parties comme présenté figure 6.1 :

— la méthode d'étiquetage implicite ;
— la méthode d'étiquetage explicite ;

avec deux variantes pour l'étiquetage explicite :

— la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* ;
— la méthode d'échantillonnage avec « dopage » B-ESU *(Boosted-ESU)* ;

La superviseur *IntuiSup* combine un étiquetage implicite de la majorité des données bien

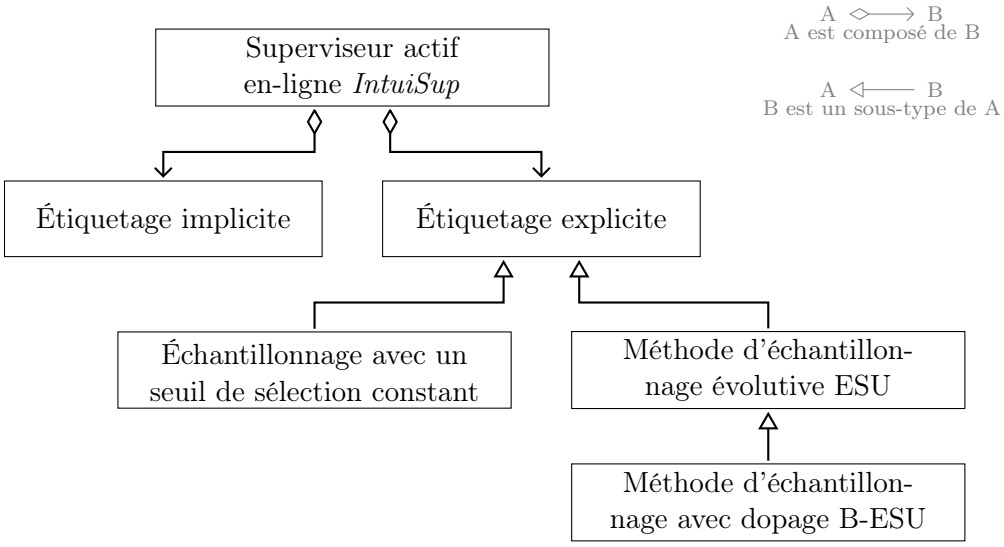


FIGURE 6.1 – Détail de l'architecture du superviseur actif en-ligne *IntuiSup*.

reconnues, avec un étiquetage explicite des données complexes. Cet étiquetage explicite est réalisé par la méthode d'échantillonnage évolutive ESU, qui peut-être complétée par la méthode de « dopage » *(boosting)* de l'apprentissage pour former B-ESU.

Le système de supervision actif en-ligne *IntuiSup* permet de superviser l'apprentissage en-ligne d'un classifieur évolutif lorsque l'utilisateur est dans la boucle. Une majorité des données bien reconnues est étiquetée implicitement pour renforcer le modèle du classifieur sans solliciter l'utilisateur. Pour permettre au classifieur d'apprendre de ses erreurs, le système de supervision interagit également avec l'utilisateur pour permettre l'apprentissage, avec les bonnes étiquettes, des données mal reconnues. L'échantillonnage des données qui sont étiquetées explicitement est basé une mesure de confiance du classifieur. Cela permet de sélectionner les données qui sont éloignées du modèle de connaissance actuel du classifieur, et d'apprendre à partir des données complexes et mal modélisées pour l'instant.

Cette méthode d'échantillonnage qui sélectionne les données qui ne sont pas en adéquation avec le modèle du classifieur, est similaire à une option de rejet. D'une manière similaire, il est ici nécessaire de faire un compromis entre le nombre d'erreurs et de sollicitations de l'utilisateur. Nous avons donc réalisé un sondage parmi les participants à une expérimentation de commandes gestuelles qui nous a permis de choisir le compromis E2RR : une erreur pour deux rejets/sollicitations *(an Error for 2 Rejections Rate)*, comme expliqué en section 2.4.3 (page 43). Puisque le système est dans une situation d'apprentissage en-ligne, le compromis erreur/sollicitation va changer avec l'évolution du système. La méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* est basée sur une modélisation statistique de l'évolution du système, pour faire évoluer l'échantillonnage et ainsi maintenir le compromis erreur/sollicitation souhaité au cours du temps.

Cette méthode d'échantillonnage est basée sur la mesure de confiance interne qui est fournie par *Evolve* ∞. L'intérêt d'utiliser cette mesure de confiance interne, plutôt que la

capacité de rejet d'*Evolve* ∞, est qu'elle est très réactive et efficace à un stade très précoce de l'apprentissage. La capacité de rejet d'*Evolve* ∞ est en effet basée sur une architecture multi-seuils qui nécessite un certain nombre d'exemples d'apprentissage avant d'être pleinement efficace. Enfin, le compromis erreur/sollicitation auquel fait face le superviseur actif *IntuiSup* est différent du compromis erreur/rejet auquel est confronté le classifieur *Evolve* ∞. Dans le cadre de l'apprentissage actif, les interactions ont un impact sur l'apprentissage du classifieur, et donc sur le futur taux d'erreur. Il est alors intéressant de sortir du compromis idéal erreur/rejet pendant une courte période afin d'obtenir un compromis futur plus intéressant.

La méthode d'échantillonnage évolutive ESU est complétée par une méthode de « dopage » de l'apprentissage pour obtenir B-ESU *(Boosted-ESU)*, qui permet d'optimiser l'impact des interactions avec l'utilisateur sur l'apprentissage. L'idée de cette méthode est de redistribuer les interactions au cours du temps pour les concentrer aux moments où elles seront les plus bénéfiques pour le classifieur. Cette méthode augmente automatiquement le taux d'échantillonnage au début de l'utilisation du système, et pendant les changements de concepts, pour augmenter la quantité de données étiquetées et accélérer le processus d'apprentissage. Cette amélioration plus rapide du classifieur ainsi obtenue permet ensuite de réduire le taux d'échantillonnage, car le classifieur commet moins d'erreurs. Le taux d'échantillonnage obtenu au final est plus bas, et les interactions avec l'utilisateur sont réduites pour la suite de l'utilisation du système.

6.2 Stratégie de supervision active en-ligne

Cette section présente la stratégie de supervision active en-ligne qui est mise en place par le superviseur *IntuiSup* pour superviser l'apprentissage d'un classifieur lorsque l'utilisateur est dans la boucle. Cette stratégie tire profit des actions de l'utilisateur pour étiqueter implicitement la majorité des données bien reconnues et renforcer le modèle de connaissance du système. À cela est combinée une méthode d'échantillonnage évolutive pour déclencher des interactions avec l'utilisateur et étiqueter explicitement certaines données. Cela permet de pouvoir apprendre à partir des données qui sont complexes et difficiles à reconnaitre, et à partir desquelles il est très bénéfique d'apprendre.

Dans le contexte de la reconnaissance de commandes gestuelles, l'utilisateur initialise le système avec très peu de données par classe (trois exemples par classe dans nos expériences). Pour améliorer la reconnaissance des commandes gestuelles, le classifieur utilisé est évolutif et apprend pendant son utilisation. En même temps que le classifieur apprend, l'utilisateur apprend également : il doit mémoriser le jeu de symboles et retenir quel symbole est associé à quelle commande [Li 13]. Dans cette situation d'apprentissage croisé, plusieurs cas peuvent se produire :

Cas A Le bon geste est dessiné, et est bien reconnu : la commande souhaitée est exécutée ;

Cas B L'utilisateur commet une erreur et se trompe de symbole ;

Cas C Le classifieur commet une erreur et se trompe de commande ;

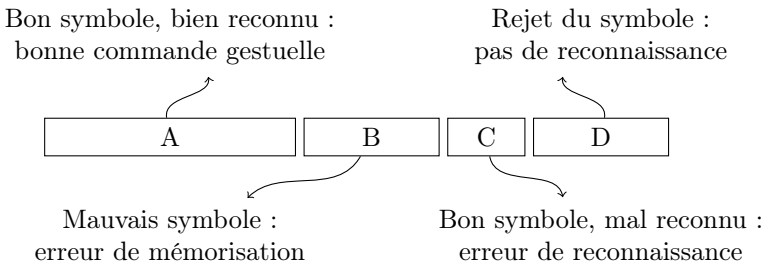


FIGURE 6.2 – Partitionnement des données suite à l'apprentissage croisé de l'utilisateur et du classifieur.

Cas D Le système rejette le symbole et demande à l'utilisateur de valider ou corriger la commande proposée.

Lorsque qu'une erreur est commise, soit par l'utilisateur soit par le classifieur (cas B ou C), la commande exécutée n'est pas celle qui est désirée. L'utilisateur doit alors annuler la (mauvaise) commande qui a été effectuée et essayer de nouveau celle qui est souhaitée. À partir de cela, il est possible de répartir les données dans quatre catégories comme illustré figure 6.2.

L'algorithme d'apprentissage du classifieur est un algorithme supervisé, il est donc nécessaire d'étiqueter les données d'utilisation pour pouvoir apprendre en-ligne. La stratégie de supervision présentée ici est duale, elle combine supervision implicite, sans solliciter l'utilisateur, et supervision explicite, où l'utilisateur est sollicité pour obtenir les étiquettes des données (vérité terrain).

6.2.1 Supervision implicite : sans interaction avec l'utilisateur

La supervision implicite de l'apprentissage consiste à étiqueter les données d'utilisation implicitement, en utilisant les étiquettes reconnues par le classifieur lorsque celles-ci sont implicitement validées par l'utilisateur.

Étiqueter les données avec les étiquettes reconnues par le classifieur est facile, et gratuit en matière d'interactions avec l'utilisateur. Cependant, cet étiquetage contient des erreurs à chaque fois qu'une commande est mal reconnue par le classifieur. Pour éviter de détériorer le modèle de connaissance du système en apprenant sur des données mal étiquetées, nous tirons parti de l'action suivante de l'utilisateur pour apprendre seulement lorsqu'il valide implicitement la commande reconnue. En effet, lorsque l'utilisateur dessine un symbole, celui-ci est reconnu par le classifieur et la commande reconnue est exécutée. Deux cas sont alors possibles :

— L'utilisateur continue ses actions, ce qui indique que la commande exécutée lui convient, il valide implicitement la commande reconnue (cas A de la figure 6.2).
— L'utilisateur annule la commande, soit parce qu'il a dessiné un mauvais symbole (erreur de mémorisation, cas B), soit parce qu'elle ne correspond pas au symbole qu'il a dessiné (erreur de reconnaissance, cas C), ou alors juste parce qu'il a changé d'avis.

La supervision implicite consiste à utiliser l'étiquette reconnue par le classifieur, mais seulement lorsqu'elle a été implicitement validée par l'utilisateur (en continuant ses actions sans annuler la commande effectuée). Le classifieur apprend ainsi à partir des données qu'il a correctement reconnues (cas A) et renforce son modèle de connaissance sans pour autant risquer de le détériorer en cas d'erreur de mémorisation (cas B) ou de reconnaissance (cas C).

Cette stratégie permet d'être sûr de ne pas détériorer le modèle de connaissance du classifieur, mais limite l'apprentissage aux données bien reconnues. Pour pouvoir améliorer efficacement le classifieur, il est indispensable de pouvoir apprendre à partir des données qui sont complexes et mal reconnues. Pour cela, il est nécessaire d'interagir avec l'utilisateur pour obtenir les bonnes étiquettes des données mal reconnues : c'est la supervision explicite.

6.2.2 Supervision explicite : étiquetage par l'utilisateur

Apprendre à partir des données qui sont mal reconnues nécessite d'interagir avec l'utilisateur pour obtenir l'étiquette du symbole qu'il a dessiné (vérité terrain). Il est évident que solliciter l'utilisateur après chaque commande rendrait l'utilisation de commandes gestuelles très laborieuse. Les interactions avec l'utilisateur ont un coût, elles doivent donc être les plus limitées possible. Il faut donc sélectionner soigneusement les données qui seront étiquetées par l'utilisateur : c'est un problème d'apprentissage actif. De plus, l'apprentissage des commandes gestuelles étant continu, la décision d'échantillonnage doit être prise donnée par donnée.

Pour cela, nous utilisons la méthode d'échantillonnage par incertitude maximale qui consiste à étiqueter les données pour lesquelles le classifieur est le moins confiant lors du processus de reconnaissance. Cette méthode a le double avantage de sélectionner des données qui ne sont pas bien modélisées par le système, et donc intéressantes pour son apprentissage, et de réduire les erreurs de reconnaissance, qui ont aussi un coût pour l'utilisateur. La supervision explicite de l'apprentissage permet ainsi au classifieur d'apprendre à partir des données qui sont complexes à reconnaitre (cas D de la figure 6.2), et pour lesquelles il aurait probablement commis une erreur de reconnaissance.

Au final, la stratégie de supervision en-ligne active d'*IntuiSup* permet d'entrainer le classifieur sur les données des catégories A (validées implicitement) et D (validées explicitement) de la figure 6.2. Les données des catégories C et D ne sont pas utilisées, car leur étiquetage n'a pas été validé, ni implicitement ni explicitement. Ce choix enlève quelques données qui ne sont pas utilisées pour l'apprentissage du classifieur, mais permet d'être sûr de ne pas dégrader le modèle de connaissance du classifieur avec des données mal étiquetées. Cette hypothèse est vérifiée dans la section 8.3 où nous montrons expérimentalement que l'apprentissage semi-supervisé à partir de ces données n'est pas intéressant.

Par conséquent, la méthode d'échantillonnage du superviseur a une grande influence sur le processus d'apprentissage en-ligne du classifieur. Plus la quantité de données sélectionnées est importante, plus le nombre de données disponibles pour l'apprentissage est important ; or apprendre à partir des données complexes est très bénéfique pour le classifieur.

L'efficacité de la stratégie de supervision active en-ligne d'*IntuiSup* est montrée au chapitre 8. La prochaine section présente la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* utilisée pour sélectionner les données qui sont explicitement étiquetées par l'utilisateur.

6.3 Méthode d'échantillonnage évolutive ESU

Cette section présente la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* que nous avons développé pour réaliser la sélection des données qui sont explicitement étiquetées par l'utilisateur. Cette méthode d'échantillonnage est basée sur une mesure de confiance interne fournie par le classifieur lors du processus de reconnaissance. D'une manière similaire à la capacité de rejet évolutive d'*Evolve* ∞, la méthode d'échantillonnage d'*IntuiSup* va évoluer en synergie avec le classifieur supervisé. Cette évolution permet de maintenir le compromis erreur/interaction souhaité malgré l'évolution du classifieur au cours du temps. L'ajout de « dopage » permet ensuite de sortir ponctuellement de ce compromis afin de l'améliorer à plus long terme comme expliqué en section 6.4.

Le contexte des commandes gestuelles implique une situation d'apprentissage croisé : l'utilisateur doit mémoriser les symboles, et le classifieur doit les modéliser pour pouvoir les reconnaitre. Dans cette situation, nous essayons d'optimiser la coopération entre l'utilisateur et le système pour minimiser les interactions avec l'utilisateur, tout en permettant l'apprentissage efficace du classifieur pour limiter les erreurs de reconnaissance. Pour cela, il est nécessaire de faire évoluer la méthode d'échantillonnage pour l'adapter à cet environnement dynamique et maintenir le compromis erreur/sollicitation voulu.

Pour optimiser l'apprentissage du système, et donc ses performances, il est possible d'ajuster la méthode d'échantillonnage pour augmenter la quantité de données étiquetées explicitement. Cependant, il faut également garder à l'esprit l'impact que cet étiquetage a sur les interactions avec l'utilisateur, et leur coût pour ce dernier. Il est nécessaire de trouver un équilibre entre le nombre d'erreurs de reconnaissance et la quantité de sollicitations de l'utilisateur. La stratégie la plus simple consiste à choisir un seuil de sélection sur la mesure de confiance qui optimise le compromis erreur/sollicitation de manière classique [Gorski, 1997] [Hansen et al., 1997]. Les données complexes avec les scores de confiance les plus bas sont alors sélectionnées, car elles sont intéressantes pour l'apprentissage du classifieur.

Pour un système hors-ligne, les données sélectionnées pour être étiquetées sont réparties de manière homogène au cours du temps. Par contre, notre système est en-ligne et s'améliore avec le temps, le nombre de données sélectionnées a alors tendance à décroître avec le temps. Dans l'étude que nous avons menée, les utilisateurs ont jugé les erreurs deux fois plus gênantes que les sollicitations/rejets, ce qui nous a mené au compromis d'une erreur pour deux rejets/sollicitations : E2RR *(an Error for 2 Rejections Rate)*. Il est donc nécessaire de faire évoluer la méthode d'échantillonnage du superviseur pour maintenir le compromis E2RR, malgré l'évolution du classifieur au cours du temps (amélioration dans le cas général, ou détérioration en cas de changement de concepts).

Nous avons développé une méthode d'échantillonnage évolutive qui permet de maintenir le compromis erreur/sollicitation souhaité au cours du temps, alors que le système apprend

et que l'environnement change. Pour faire évoluer la méthode d'échantillonnage en synergie avec l'évolution du système, nous avons mis au point un algorithme en-ligne qui met à jour le seuil de sélection pendant l'apprentissage du système : la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)*.

Dans cette section, nous étudions et comparons trois algorithmes pour faire évoluer la méthode d'échantillonnage du système.

— l'algorithme AMTL *(Automatic Multiple Threshold Learning)*,
— un algorithme basé sur le filtrage de Kalman,
— Le nouvel algorithme ESU *(Evolving Sampling by Uncertainty)*.

Les deux premiers sont des algorithmes de l'état de l'art qui permettent d'estimer et de faire évoluer un seuil de rejet. Le seuil d'échantillonnage est ensuite fixé proportionnellement à ce seuil de rejet idéal donné par l'algorithme. Nous les avons adaptés à notre situation d'échantillonnage en-ligne, mais leurs performances sont limitées pour notre problème (comme nous le verrons en section 8.4).

Nous avons développé un troisième algorithme pour répondre spécifiquement aux contraintes de notre problème : la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)*. Contrairement aux deux premiers algorithmes qui mettent à jour le seuil de sélection de manière corrective, l'algorithme ESU met à jour ce seuil de manière préventive, ce qui est bien plus efficace que de seulement le corriger en cas d'erreur ou de sollicitation/rejet inutile. Les performances de la méthode d'échantillonnage ESU sont présentées en section 8.4.

6.3.1 L'algorithme AMTL *(Automatic Multiple Thresholds Learning)*

L'algorithme AMTL *(Automatic Multiple Thresholds Learning)* est un algorithme générique pour l'apprentissage de seuils de rejet multiples [Mouchere and Anquetil, 2006a] [Mouchere and Anquetil, 2006b]. C'est un algorithme glouton basé sur des heuristiques empiriques qui permet d'optimiser des seuils de rejet multiples, à partir d'une base d'exemples à rejeter et de contre-exemples à accepter, pour obtenir le compromis erreur/rejet voulu.

L'algorithme AMTL commence avec des seuils de rejet très élevés, qui rejettent toutes les données. Ces seuils sont ensuite descendus pour accepter de plus en plus de données, dont inévitablement des contre-exemples. L'algorithme s'arrête une fois le compromis erreur/rejet voulu atteint sur la base d'apprentissage. AMTL n'est pas un algorithme en-ligne, il nécessite un ensemble d'apprentissage pour optimiser les seuils de rejet. Il peut être rendu incrémental en utilisant une fenêtre glissante de données, même si cela a un coût en matière de mémoire nécessaire.

Le principal avantage et intérêt de l'algorithme AMTL est qu'il optimise directement le ou les seuils pour atteindre le compromis erreur/rejet voulu. Il faut juste choisir le compromis souhaité, en l'occurrence le compromis E2RR donnée par notre sondage, et l'algorithme donne le seuil correspondant.

L'inconvénient de cette approche est que les seuils de rejet ne sont modifiés qu'en cas de mauvais rejet ou de mauvaise acceptation. Un mauvais rejet est une donnée correctement reconnue qui est quand même rejetée ; et une mauvaise acceptation est une donnée mal

reconnue qui est quand même acceptée. Cette mise à jour des seuils est seulement corrective, ce qui manque de réactivité dans notre contexte où il faut apprendre à partir de très peu de données.

6.3.2 Algorithme basé sur le filtrage de Kalman

Cet algorithme est basé sur les principes du filtre de Kalman pour estimer la séparation entre les exemples à rejeter et les contre-exemples à accepter [Meinhold and Singpurwalla, 1983]. Le filtrage de Kalman est un algorithme d'estimation linéaire quadratique qui permet d'obtenir un estimateur statistiquement optimal d'une variable inconnue à partir de mesures bruitées.

Dans notre cas, nous présentons au filtre de Kalman les mauvais rejets et les mauvaises acceptations pour estimer la séparation entre les exemples et les contre-exemples. L'algorithme commence avec une valeur de séparation initiale et la corrige à chaque fois qu'une donnée est soit rejetée inutilement, soit acceptée à tort. Le seuil de rejet est alors fixé proportionnellement à cette séparation entre les exemples et contre-exemples pour obtenir le compromis erreur/rejet souhaité.

Le filtrage de Kalman est un processus en deux temps. L'étape de prédiction consiste à prédire la valeur de la variable x_t, qui représente la séparation entre exemples et contre-exemples, et son incertitude p_t, à partir des valeurs passées x_{t-1} et p_{t-1}.

$$x_t = x_{t-1} \tag{6.1}$$
$$p_t = p_{t-1} + q \tag{6.2}$$

Où q est un paramètre représentant le bruit du processus. L'étape de mise à jour consiste à corriger les valeurs prédites à partir de l'observation courante y_t pour obtenir une prédiction plus précise.

$$k_t = p_t/(p_t + r) \tag{6.3}$$
$$x_t = x_t + k_t \cdot (y_t - x_t) \tag{6.4}$$
$$p_t = (1 - k_t) \cdot p_{t-1} \tag{6.5}$$

Où r est un paramètre représentant le bruit de mesure. À partir de cette séparation, nous fixons un facteur θ pour obtenir la valeur du seuil de sélection t_t permettant d'avoir le compromis erreur/sollicitation voulu.

$$t_t = \theta \cdot x_t \tag{6.6}$$

Cet algorithme basé sur le filtrage de Kalman est une manière très efficace d'estimer et de suivre l'évolution de la séparation entre les exemples et contre-exemples à partir des mesures très bruitées que nous avons ici (en utilisant les mauvais rejets et les mauvaises acceptations). L'inconvénient de cette approche est là encore que le seuil de sélection n'est mis à jour que lorsque le processus de rejet échoue, c'est à dire lorsqu'une donnée est rejetée inutilement ou acceptée à tort. En d'autres termes, comme pour l'algorithme AMTL, le seuil de sélection est mis à jour de manière corrective et non pas préventive.

6.3.3 Algorithme d'échantillonnage évolutif ESU

Nous avons conçu un nouvel algorithme pour faire évoluer la méthode d'échantillonnage et suivre l'évolution du système en-ligne. L'algorithme ESU *(Evolving Sampling by Uncertainty)* adapte le seuil de sélection de manière continue et préventive, et non pas seulement corrective comme l'algorithme AMTL et comme celui basé sur le filtrage de Kalman. Nous avons mis au point un estimateur ζ_t de l'évolution de la confiance du système, que nous utilisons pour faire évoluer le seuil de sélection. L'estimation de la progression de la confiance moyenne est faite en suivant l'évolution récente de la moyenne et de l'écart-type des valeurs de confiance.

Le seuil est mis à jour de la manière suivante. La moyenne μ_t et l'écart-type σ_t sont calculés à l'instant t de manière glissante :

$$\mu_t = \mu_{t-1} \cdot \frac{nb-1}{nb} + y_t \cdot \frac{1}{nb} \tag{6.7}$$

$$\sigma_t = \sqrt{\frac{S_t}{t}} \tag{6.8}$$

$$S_t = S_{t-1} \cdot \frac{nb-1}{nb} + (y_t - \mu_t) \cdot (y_t - \mu_{t-1}) \cdot \frac{1}{nb} \tag{6.9}$$

Avec

$$nb = \min(t, nb_{max}), \quad nb_{max} = 100 \tag{6.10}$$

afin d'obtenir une estimation sur les nb_{max} dernières valeurs. Ce mode de calcul est équivalent à l'utilisation d'une fenêtre de données exponentiellement pondérées. L'estimateur de l'évolution de la confiance du système est calculé à partir de μ_t et de σ_t :

$$\zeta_t = \mu_t - \sigma_t \tag{6.11}$$

Ensuite, nous fixons un facteur θ pour obtenir le seuil t_t qui donne le compromis erreur/sollicitation souhaité.

$$t_t = \theta \cdot \zeta_t = \theta \cdot (\mu_t - \sigma_t) \tag{6.12}$$

L'intérêt principal de cet algorithme basé sur une estimation continue de l'évolution de la confiance moyenne du système est qu'il est préventif, et non correctif. Il met à jour le seuil de sélection pour chaque donnée d'apprentissage, même si c'est une erreur rejetée ou une bonne reconnaissance acceptée. Au contraire, les deux algorithmes de l'état de l'art, AMTL et le filtrage de Kalman, ne mettent à jour le seuil de sélection que pour les données qui ne sont pas correctement positionnées par rapport au seuil de rejet/sélection (mauvais rejets et mauvaises acceptations). L'efficacité de l'algorithme ESU pour faire évoluer le seuil de sélection est montrée dans la section 8.4.

La méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* permet d'estimer efficacement le seuil de sélection pour maintenir le compromis erreur/sollicitation voulu au cours du temps. Cela permet de faire évoluer la méthode d'échantillonnage du système avec l'amélioration du classifieur et les changements de son environnement, ce qui est nécessaire pour tirer pleinement profit de la stratégie de supervision active en-ligne d'*IntuiSup*.

En plus de cette méthode d'échantillonnage évolutive ESU, nous avons créé une méthode de « dopage » qui accélère l'apprentissage du système, à la fois lors de l'apprentissage initial et en cas de changement de concepts. La méthode d'échantillonnage avec dopage B-ESU *(Boosted-ESU)*, qui permet de sortir ponctuellement du compromis erreur/sollicitation idéal afin de l'améliorer à plus long terme, est présentée dans la prochaine section.

6.4 Méthode de « dopage » de l'apprentissage B-ESU

Cette section présente la technique de « dopage » de l'apprentissage B-ESU *(Boosted-ESU)* que nous avons conçue pour optimiser la coopération entre le système et l'utilisateur et améliorer la vitesse d'apprentissage du système. Cette technique permet d'accélérer la convergence initiale du système et l'adaptation aux changements de concepts, en augmentant la quantité de données d'apprentissage aux bons moments, puis en réduisant les interactions avec l'utilisateur lorsque le processus d'apprentissage a convergé. L'idée de cette technique est de ne pas sélectionner de manière régulière les données pour l'apprentissage, mais d'en sélectionner davantage dans les périodes clefs : pendant l'apprentissage initial et pendant les changements de concepts.

Le compromis erreur/sollicitation est relativement complexe dans ce contexte d'apprentissage croisé de commandes gestuelles. Sélectionner une donnée qui aurait été mal reconnue représente non seulement une erreur évitée, mais représente également une donnée d'apprentissage supplémentaire. Une donnée supplémentaire pour l'apprentissage est l'occasion d'améliorer le classifieur, qui fera alors moins d'erreur dans le futur, et qui nécessitera donc moins de sollicitations.

Pour un même nombre d'interactions avec l'utilisateur, le classifieur commet moins d'erreurs si les données sélectionnées sont concentrées au début de l'utilisation/apprentissage du système, car cela permet d'accélérer la vitesse d'apprentissage du système. Nous avons donc développé une technique d'accélération de l'apprentissage qui augmente le taux de sélection durant la phase d'apprentissage initiale et quand un changement de concepts se produit. Ce taux de sélection plus élevé permet d'augmenter la quantité de données qui est étiquetée explicitement par l'utilisateur, et d'accélérer l'amélioration du système, et finalement de réduire le taux de sélection à une valeur plus faible qu'avant, car le système commet alors moins d'erreurs.

Nous montrons expérimentalement dans la section 8.5 que l'utilisateur est au total sollicité un nombre de fois égal à moyen terme, et même un nombre de fois plus faible à long terme.

6.4.1 Principe du « dopage »

L'idée du « dopage » de l'apprentissage est d'adapter dynamiquement le taux de sélection des données par le superviseur au taux d'apprentissage actuel du classifieur. Nous voulons avoir un fort taux de sélection au début de l'utilisation du système, lorsqu'il est en plein apprentissage, pour accélérer sa convergence. Ensuite, lorsque le système a convergé,

et que ses performances sont meilleures, le taux de sélection peut être réduit car moins d'erreurs sont commises, et les sollicitations de l'utilisateur sont alors réduites.

Nous avons choisi de lier le dopage de l'apprentissage, c'est-à-dire l'augmentation du seuil de sélection du superviseur, à l'évolution du taux d'erreur du classifieur. Cette méthode permet d'adapter automatiquement et dynamiquement l'augmentation du taux de sélection à l'évolution du système, à la fois pendant l'apprentissage initial et lors des changements de concepts.

Lorsqu'il y a peu de classes pendant l'apprentissage initial, ou que les différentes classes sont faciles à reconnaitre pour le classifieur, le taux d'erreur décroit rapidement et le dopage disparait rapidement. Au contraire, lorsque les classes sont nombreuses, ou complexes à reconnaitre, le dopage reste jusqu'à ce que l'apprentissage du système converge.

Au final, lorsque l'apprentissage a convergé, le compromis erreur/sollicitation obtenu avec cette méthode de dopage de l'apprentissage actif est plus intéressant. Moins d'erreur et moins de sollicitations seront faits par le classifieur pendant l'utilisation future du système.

De plus, si de nouvelles classes sont ajoutées, ou si un changement de concepts arrive, le taux d'erreur augmente et le dopage de l'apprentissage réapparait jusqu'à ce que la nouveauté soit apprise.

6.4.2 Dopage de l'échantillonnage

En pratique, l'accélération de l'apprentissage crée une augmentation temporaire du seuil de sélection s qui est obtenu par l'ajout d'un terme de « dopage ».

$$s = s_{dynamic} + s_{boost} \tag{6.13}$$

Le terme de dopage du seuil de sélection s_{boost} est lié à la valeur absolue du gradient du taux de reconnaissance du système.

$$|\frac{\partial\ error\_rate}{\partial\ time}| \tag{6.14}$$

Le gradient du taux de reconnaissance du système est approximé sur une fenêtre glissante comme la différence entre les taux d'erreur sur les deux moitiés de la fenêtre.

$$\delta = |\frac{nb_{errors}^{(first\_half)}}{nb_{samples}^{(first\_half)}} - \frac{nb_{errors}^{(second\_half)}}{nb_{samples}^{(second\_half)}}| \tag{6.15}$$

Finalement, le terme de dopage est limité à la valeur s_{boost_max} pour ne pas créer trop de sollicitations utilisateur, afin de ne pas trop ennuyer l'utilisateur.

$$s_{boost} = min(\delta, s_{boost\_max}) \tag{6.16}$$

Pour nos expériences, nous utilisons une fenêtre glissante de taille égale à dix fois le nombre de classes, ce qui permet une estimation relativement stable du comportement du système. Le processus de dopage de l'échantillonnage est plafonné par le taux de sollicitation maximum accepté par les utilisateurs. L'estimation précise de ce taux de sollicitation

maximum nécessite un sondage auprès des utilisateurs, et est pour l'instant fixé empiriquement à 20% des données du flux.

Le but de cette méthode d'échantillonnage avec dopage B-ESU *(Boosted-ESU)* est de permettre au classifieur de converger plus rapidement, à la fois lors de la phase d'apprentissage initiale et lors des changements de concepts. Les expérimentations présentées dans la section 8.5 confirme le bien fondé de cette approche, puisqu'elle permet une réduction du taux d'erreur final de plus de 15%.

6.5 Conclusion

Apprendre un classifieur pour reconnaitre des commandes gestuelles pendant leur utilisation est une situation d'apprentissage croisé. C'est un problème d'apprentissage en-ligne pour le classifieur, et il faut donc interagir avec l'utilisateur pour pouvoir étiqueter les données et améliorer efficacement le modèle de connaissance du classifieur. D'un autre côté, solliciter constamment l'utilisateur est ennuyeux, et réduit considérablement la fluidité d'utilisation des commandes gestuelles. Il faut faire un compromis entre le nombre d'interactions avec l'utilisateur et le nombre d'erreurs de reconnaissance.

Nous avons présenté la stratégie de supervision active en-ligne *IntuiSup* qui permet l'apprentissage actif en-ligne d'un classifieur évolutif avec l'utilisateur dans la boucle. Nous avons vu qu'une majorité des données peut être étiquetée implicitement pour renforcer le modèle du classifieur sans solliciter l'utilisateur. Comme il est fondamental de pouvoir apprendre à partir des données mal reconnues, avec les bonnes étiquettes, pour améliorer le classifieur, nous interagissons avec l'utilisateur pour superviser activement l'apprentissage en-ligne. En particulier, nous utilisons une mesure de confiance du classifieur pour prendre la décision d'échantillonnage de la stratégie d'apprentissage actif, et apprendre à partir des données qui ne correspondent pas au modèle de connaissance actuel du classifieur.

Le superviseur actif en-ligne *IntuiSup* est composé de deux parties :

— la méthode d'étiquetage implicite ;

— la méthode d'étiquetage explicite ;

avec deux méthodes d'échantillonnage pour l'étiquetage explicite :

— la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* ;

— la méthode d'échantillonnage avec « dopage » B-ESU *(Boosted-ESU)*.

Il faut faire un compromis entre le nombre d'erreurs et de sollicitations. Nous avons donc réalisé un sondage parmi les participants à une expérimentation de commande gestuelles qui nous a permis de choisir le compromis « idéal » E2RR : une erreur pour deux sollicitations. Comme le système est dans une situation d'apprentissage en-ligne, le compromis erreur/sollicitation va changer avec l'évolution du système. Nous avons présenté la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)*, basée sur une modélisation statistique de l'évolution du système, qui permet de faire évoluer la méthode d'échantillonnage afin de maintenir le compromis erreur/sollicitation souhaité.

Cette méthode d'échantillonnage évolutive est complétée par la méthode de « dopage » de l'apprentissage pour permettre d'accélérer la vitesse d'apprentissage du classifieur. La méthode d'échantillonnage B-ESU *(Boosted-ESU)* augmente automatiquement le taux de

sélection des données à faire étiqueter par l'utilisateur, au début de l'utilisation du système, ainsi que pendant les changements de concepts, pour augmenter la quantité de données étiquetées et accélérer le processus d'apprentissage. Cette amélioration plus rapide du classifieur ainsi obtenue permet ensuite de réduire le taux de sélection, car le classifieur commet moins d'erreurs, et ainsi de réduire les sollicitations de l'utilisateur. L'utilisateur sera au total sollicité un nombre de fois égal à moyen terme, et même un nombre de fois plus faible à long terme.

Le superviseur actif en-ligne *IntuiSup* pour l'apprentissage d'un classifieur évolutif avec l'utilisateur dans la boucle est générique. Il peut être utilisé avec n'importe quel classifieur évolutif. La seule contrainte est que le classifieur fournisse une mesure d'adéquation des données avec son modèle de connaissance, lors du processus de reconnaissance, afin de prendre la décision d'échantillonnage.

L'efficacité du superviseur actif en-ligne *IntuiSup* est validée expérimentalement dans le chapitre 8.

Troisième partie

Validation expérimentale

Préambule

La troisième partie donne les résultats expérimentaux obtenus avec le classifieur *Evolve* ∞ et le superviseur *IntuiSup*. Différents scénarios de test permettent d'évaluer et de valider expérimentalement leurs capacités et leurs performances. Leur sensibilité au contexte applicatif, leur paramétrage et leur positionnement par rapport à l'état de l'art sont également évalués dans différentes situations d'apprentissage.

— **Le septième chapitre** présente les performances obtenues avec *Evolve* ∞ pour la reconnaissance de gestes manuscrits, et plus généralement pour différents problèmes de classification de l'état de l'art. *Evolve* ∞ est ainsi comparé et positionné, en terme de rapidité d'apprentissage et de taux de reconnaissance à convergence, par rapport à différents systèmes de l'état de l'art. Ses capacités d'apprentissage et d'évolution sont ensuite misent en avant, au travers de l'ajout de nouvelles classes en cours d'utilisation. L'apprentissage et la reconnaissance de flux de données comportant des changements de concepts met en évidence l'intérêt de la capacité d'oubli d'*Evolve* ∞. L'efficacité de sa capacité d'auto-évaluation et de rejet est également mise en avant lors de l'apprentissage à partir de très peu de données.

— **Le huitième chapitre** montre l'efficacité du superviseur évolutif *IntuiSup* pour réaliser l'apprentissage actif en-ligne d'un classifieur évolutif. Son application à l'apprentissage de commandes gestuelles confirme sa capacité à optimiser les interactions avec utilisateur pour superviser l'apprentissage lorsque l'utilisateur est dans la boucle, en tirant parti de la capacité d'auto-évaluation du classifieur. Ses capacités d'évolution sont ensuite évaluées pour s'adapter à l'évolution du classifieur et aux changements du flux de données, et pour maintenir le compromis erreur/rejet choisi au cours du temps. Enfin, l'intérêt de la répartition et du dopage *(boosting)* des interactions est mis en évidence pour accélérer l'apprentissage initial du classifieur et son adaptation aux changements de concepts.

Chapitre 7

Évaluation du classifieur évolutif en-ligne *Evolve* ∞

7.1 Introduction

L'utilisation de commandes gestuelles personnalisées nécessite un classifieur évolutif pour plusieurs raisons. Tout d'abord, comme les classes sont personnalisées par l'utilisateur final, les données d'initialisation sont limitées. Il est donc souhaitable que le système apprenne pendant son utilisation pour continuer à s'améliorer et atteindre les meilleures performances possibles. Ensuite, il est nécessaire que l'utilisateur puisse ajouter de nouvelles commandes à n'importe quel moment de son utilisation du système. Il est alors nécessaire d'intégrer ces nouvelles classes dans le système sans pour autant dégrader la reconnaissance des anciennes classes. Enfin, le style d'écriture de l'utilisateur va évoluer au fur et à mesure qu'il va se familiariser avec le système de commandes gestuelles. Ses symboles vont devenir de plus en plus fluides et rapides, et probablement se déformer de plus en plus, comme des signatures. Il faut alors que le système s'adapte à cette évolution des données et conserve de bonnes performances.

Nous avons présenté dans le chapitre 5 le classifieur évolutif *Evolve* ∞, qui est un système d'inférence floue d'ordre un, dit de Takagi-Sugeno. C'est un système qui est composé d'un ensemble de règles d'inférence. Il apprend rapidement à partir de peu de données, grâce aux capacités génératrices des prémisses des règles. Les capacités discriminantes des conclusions d'ordre un lui permettent d'obtenir également de bonnes performances après suffisamment d'exemples d'apprentissage. C'est un système qui supporte très bien l'ajout de classes en cours d'utilisation grâce à son architecture sous forme d'ensemble de règles. La création de nouvelles règles permet d'intégrer de la nouveauté au modèle du classifieur, sans pour autant perturber la connaissance déjà existante. *Evolve* ∞ dispose d'un algorithme d'apprentissage de faible complexité, qui lui permet d'être appris en-ligne. Il est capable d'évoluer indéfiniment pour s'adapter au flux de données, et en particulier de suivre les changements de concepts.

Dans ce chapitre, nous comparons *Evolve* ∞ à *Evolve* et évaluons les deux contributions principales que comporte *Evolve* ∞, à savoir sa capacité d'oubli et son algorithme

de regroupement *(clustering)* incrémental. Nous commençons par comparer *Evolve* ∞ à d'autres classifieurs en terme de performances sur des problèmes de classification de l'état de l'art. L'intérêt de la capacité d'oubli d'*Evolve* ∞ est testé sur différents scénarios, et l'inertie d'*Evolve* ∞ est comparée à celle d'*Evolve*. Le comportement du système lors de l'ajout de nouvelles classes à moyen et long terme est étudié. Le temps d'apprentissage de ces nouvelles classes est mesuré et permet de montrer que le gain de l'apprentissage est maintenu indéfiniment pour *Evolve* ∞. L'effet de l'oubli est également mis en évidence lorsque le flux de données présente des changements de concepts, qu'ils soient brusques ou progressifs.

Ce chapitre présente l'évaluation expérimentale du classifieur évolutif *Evolve* ∞. La section 7.2 présente le protocole d'évaluation utilisé. La section 7.3 justifie le choix d'*Evolve* ∞ pour l'apprentissage à partir de peu de données, en montrant sa vitesse d'apprentissage. La section 7.4 montre l'intérêt de l'intégration d'oubli dans le processus d'apprentissage pour maintenir la capacité d'évolution du système à long terme. La section 7.5 met en évidence l'intérêt de l'oubli pour l'apprentissage sur des flux avec des changements de concepts.

7.2 Méthodologie

Pour évaluer les performances du classifieur *Evolve* ∞, nous avons utilisé un protocole de test spécifique – appelé « préquentiel » – que nous allons décrire ci-dessous.

Cette section présente également les bases de données que nous avons utilisées pour nos expérimentations, avec notamment ILGDB [Renau-Ferrer et al., 2012] et Ironoff-digits [Viard-Gaudin et al., 1999]. Nous détaillons en particulier la base de données ILGDB, qui contient des commandes gestuelles personnalisées.

Nous présentons également le jeu de caractéristiques HBF49 [Delaye and Anquetil, 2013] que nous avons choisit pour sa polyvalence.

7.2.1 Protocole d'évaluation

Pour évaluer notre classifieur d'une manière réaliste et représentative de l'utilisation d'un système évolutif, nous avons utilisé le protocole d'évaluation incrémental « prédictif séquentiel », ou « préquentiel » [Gama et al., 2009]. Un système incrémental commence par essayer de reconnaitre une donnée, puis l'utilise ensuite pour apprendre une fois qu'il a obtenu l'étiquette correspondante. Le protocole de test préquentiel consiste à évaluer le système de la même manière : chaque donnée est d'abord utilisée comme donnée de test, puis ensuite comme donnée d'apprentissage.

De cette manière, il est possible de tester puis d'entrainer un système sur les mêmes données, sans pour autant biaiser son évaluation puisque les données sont encore inconnues lorsqu'elles sont utilisées pour tester le système. Le taux de reconnaissance obtenu est ainsi représentatif de celui qui est obtenu lors de l'utilisation réelle d'un système évolutif appris en-ligne.

Nous l'avons utilisé avec un facteur d'oubli exponentiel afin de faire converger le taux de reconnaissance obtenu vers celui qui serait obtenu lors d'une évaluation hors-ligne classique,

Base de données	Classes	Scripteurs	Exemples	Carac.	Nature	WI	WD
ILGDB	**588**	38	6 629	49	mono	non	oui
Ironoff-digits	10	**700**	4 086	49	multi	oui	non
LaViola	**48**	11	11 602	49	multi	oui	oui
NicIcon	14	34	**26 163**	49	multi	oui	oui
Pen-Digits	10	44	10 992	**19**	multi	oui	oui
Japanese-Vowels	9	-	9 961	12	**autre**	-	-

TABLE 7.1 – Détails des propriétés des différentes bases de données utilisées pour la validation expérimentale. Les colonnes classes, scripteurs, exemples et caractéristiques (carac.) indiquent respectivement les nombres de classes, scripteurs, exemples et caractéristiques. La nature des bases est décrite par les codes suivants : mono pour les symboles mono-strokes, multi pour les symboles multi-strokes, et autre pour les autres types de données. Les colonnes WD et WI indiquent la possibilité d'utiliser la base pour des expérimentations mono-scripteurs *(writer dependent)* et multi-scripteurs *(writer independent)*.

sur une base de données de test séparée *(holdout error rate)* [Gama et al., 2009]. Le taux de reconnaissance est mis à jour incrémentalement après chaque reconnaissance, en appliquant le facteur d'oubli. En pratique nous avons utilisé un facteur d'oubli exponentiel de $1 - 1/c$, avec c le nombre de classes, soit 0.95 pour ILGDB. Ce facteur d'oubli élevé permet d'obtenir une « moyenne glissante » sur l'ensemble des dernières données de chacune des classes environ. On obtient ainsi un taux de reconnaissance qui varie avec le flux de données, et qui est représentatif du taux de reconnaissance qui serait perçu par l'utilisateur.

7.2.2 Bases de données

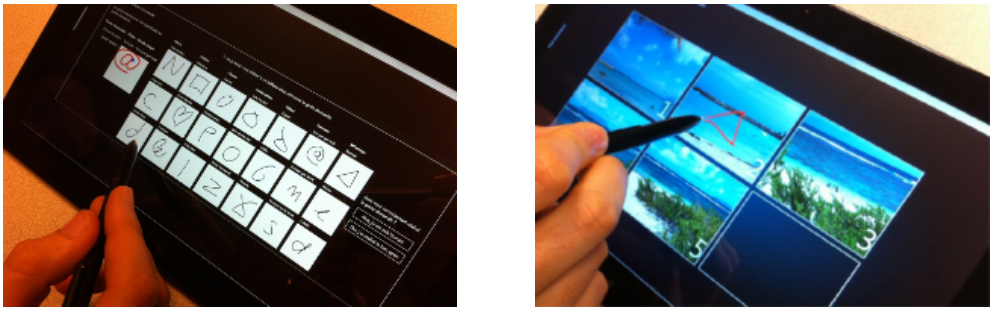
Pour nos expérimentations, nous avons utilisé les bases de données suivantes, dont les propriétés sont résumées dans le tableau 7.1 :

— ILGDB [Renau-Ferrer et al., 2012] ;
— Ironoff-digits [Viard-Gaudin et al., 1999] ;
— LaViola [LaViola and Zeleznik, 2007] ;
— NicIcon [Niels et al., 2008] ;
— Pen-Digits [Frank and Asuncion, 2010] ;
— Japanese-Vowels [Frank and Asuncion, 2010].

La plupart de nos expérimentations utilisent ILGDB, qui est une base de données de commandes gestuelles personnalisées que nous allons détailler ci-dessous. Les bases de données Ironoff-digits, LaViola et NicIcon permettent d'évaluer *Evolve* ∞ à plus long terme qu'avec ILGDB, tout en utilisant toujours le jeu de caractéristiques HBF49. Ironoff-digits a en particulier un grand nombre de scripteurs différents, LaViola un grand nombre de classes, et NicIcon un grand nombre de symboles. La figure 7.1 montre des exemples de symboles de ces différentes bases de données. Les bases Pen-Digits et Japanese-Vowels permettent quant à elles de valider le comportement de notre système avec d'autres jeux de caractéristiques que HBF49.

(a) ILGDB (b) Ironoff-digits (c) NicIcon (d) LaViola

FIGURE 7.1 – Exemples de symboles des différentes bases de données utilisées pour la validation expérimentale.



(a) Interface de définition du jeu de commandes gestuelles personnalisées.

(b) Application de manipulation de photos à l'aide de commande gestuelles

FIGURE 7.2 – Environnement d'acquisition immersif d'ILGDB : application de manipulation de photos à l'aide de commandes gestuelles personnalisées.

7.2.2.1 ILGDB : *Intuidoc Loustic Gesture Data Base*

Nous avons évalué notre approche en utilisant principalement la base de données ILGDB [1] *(IntuiDoc Loustic Gesture Data Base)* [Renau-Ferrer et al., 2012] avec les caractéristiques HBF49 fournies. Cette base de données contient des commandes gestuelles qui ont été récoltées dans un environnement immersif. La figure 7.2a montre l'interface de définition du jeu de commande gestuelles personnalisées, et la figure 7.2b montre le logiciel de manipulation de photos qui a servi à récolter les données, avec des commandes telles que copier, coller, zoomer, mettre en noir et blanc, etc.

ILGDB contient 6 629 symboles manuscrits mono-strokes, répartis dans 21 classes, qui ont été dessinés par 38 scripteurs. Les utilisateurs sont répartis dans trois groupes :

- les scripteurs du groupe 1 devaient choisir un jeu de gestes personnalisés sans contraintes (voir figure 7.3a) ;
- les scripteurs du groupe 2 devaient choisir un jeu de racines personnalisées auxquelles étaient ajoutées des terminaisons imposées (voir figure 7.3b) ;
- les scripteurs du groupe 3 devaient apprendre un jeu de gestes prédéfinis (voir figure 7.3c).

1. Disponible gratuitement à l'adresse `http://www.irisa.fr/intuidoc/ILGDB.html`

Le flux de symboles de chaque utilisateur est réparti en cinq phases. La phase 0 contient 3 symboles par classe, et correspond à la phase d'initialisation du système. Les phases 1 et 3 simulent l'utilisation de commandes gestuelles, en contenant un total de 34 symboles mais avec différentes fréquences suivant les classes. Les phases 2 et 4 contiennent 1 symbole par classe, et permettent d'avoir une vision générale des performances du système.

Cette base de données est très intéressante, et unique, pour trois raisons. Premièrement, les symboles sont ordonnés chronologiquement selon leur ordre de dessin. Cet ordonnancement permet de voir l'évolution du style d'écriture des utilisateurs avec le temps, au fur et à mesure qu'il passe de novice à expert. La figure 7.3d montre un exemple de déformation progressive d'une commande gestuelle au fur et à mesure de son utilisation. Cette propriété est unique parmi les bases de données de symboles manuscrits, et est absolument nécessaire pour évaluer de manière réaliste un système évolutif appris en-ligne.

Deuxièmement, pour la majorité de la base de données, les différents symboles ont été choisis librement par les utilisateurs eux-mêmes. La figure 7.3 montre des exemples de gestes choisis par les scripteurs d'ILGDB. Les différentes classes sont par conséquent très diversifiées, ce qui est représentatif de l'utilisation réelle d'un système de commandes gestuelles personnalisables.

Troisièmement, les fréquences des différentes classes varient de 5 à 17 exemples par symbole et par utilisateur. Le flux de données de chaque utilisateur est divisé en 5 phases, et pour deux d'entre elles, différentes classes sont plus ou moins représentées, certaines sont même absentes.

Ces trois raisons font que la base de données ILGDB est unique et réaliste. En particulier, elle est très représentative des difficultés rencontrées lors de la reconnaissance d'un flux de données comme celui obtenu lors de l'utilisation de commandes gestuelles. Des exemples de symboles inventés par les scripteurs de cette base de données sont présentés figure 7.3.

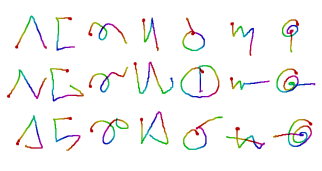
ILGDB est donc une base de données unique pour l'évaluation d'un système de reconnaissance évolutif sur flux de petite échelle. Son seul inconvénient est le petit nombre de données par scripteur (moins de 180), et par classe pour chaque scripteur (de 5 à 17 exemples). Cela est dû au temps nécessaire à l'acquisition des symboles dans un environnement immersif : ces 180 commandes gestuelles correspondent à plus de 45 minutes d'utilisation de l'application de manipulation de photos. Pour pallier cet inconvénient, nous avons également utilisé d'autres bases de données qui contiennent plus de données.

7.2.2.2 Basse de données Ironoff

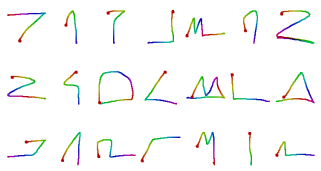
Ironoff [Viard-Gaudin et al., 1999] est composée de chiffres, de lettres, de divers symboles et de mots manuscrits qui ont été écrits par 700 scripteurs. Pour pouvoir évaluer les performances de notre classifieur à long terme, nous avons utilisé la sous partie Ironoff-digits, qui est composée de 4086 chiffres répartis dans 10 classes.

L'avantage de cette base de données est qu'elle contient beaucoup plus de données par classe que ILGDB. Par contre, Ironoff n'est utilisable qu'en mode multi-utilisateur, car le nombre de données par scripteur est très faible. Les données ne sont donc pas ordonnées chronologiquement, comme la plupart des bases de données de référence de la littérature.

(a) ILGDB groupe 1 : symboles librement inventés par les utilisateurs.



(b) ILGDB groupe 2 : symboles à racines libres et terminaisons imposées.



(c) ILGDB groupe 3 : symboles imposés aux utilisateurs.

(d) Exemple d'évolution du style d'écriture d'un symbole avec le temps. Plus l'utilisateur s'habitue au système, plus il trace son symbole de manière rapide et cursive.

FIGURE 7.3 – Exemples de symboles de la base de données de commandes gestuelles personnalisées ILGDB [Renau-Ferrer et al., 2012].

7.2.2.3 Basse de données LaViola

LaViola [LaViola and Zeleznik, 2007] est composée de 11 602 symboles répartis dans 48 classes. Elle contient des chiffres, des lettres et des symboles mathématiques écrits par 11 scripteurs.

L'avantage de cette base de données est qu'elle contient beaucoup plus de classes que ILGDB. Elle permet de valider la capacité d'*Evolve* ∞ à supporter un plus grand nombre de classes.

7.2.2.4 Basse de données NicIcon

NicIcon [Niels et al., 2008] est une base de données de pictogrammes dédiés à la gestion de situation de crise. Elle est composée de 26 163 symboles répartis dans 14 classes qui ont été écrits par 34 scripteurs.

Si les symboles qui la composent sont facilement identifiables par l'œil et le cerveau humain, ils constituent un défi pour les systèmes d'apprentissage automatique. La manière de tracer les symboles varie suivant les scripteurs. Ainsi pour une même classe le nombre de strokes change et les sens de tracé ne sont pas homogènes.

7.2.2.5 Bases de données de référence de l'UCI

Nous avons aussi testé *Evolve* ∞ sur des bases de données de référence de l'*University of California Irvine* (UCI) [Frank and Asuncion, 2010] [2] pour évaluer ses performances avec d'autres jeux de caractéristiques et en dehors du champ de la reconnaissance de symboles manuscrits. Nous avons choisi des jeux de données qui suivent deux critères : ils sont multi-classes, car notre système est destiné aux problèmes de classification avec plusieurs classes, et ils sont assez grands pour pouvoir tester les performances de notre système à long terme. Nous avons donc sélectionné les jeux de données suivants :

Pen-Digits Ce jeu de données contient environ 11 000 chiffres manuscrits, répartis dans 10 classes et décrits par 19 caractéristiques, qui ont été écrits par 44 scripteurs sur une interface tactile.

Japanese-Vowels Ce jeu de données est composé de près de 10 000 exemples de voyelles japonaises, réparties dans 9 classes correspondant aux 9 locuteurs, et décrites par 12 caractéristiques.

7.2.3 Jeu de caractéristiques HBF49

HBF49 *(Heterogeneous Baseline Feature-set)* [Delaye and Anquetil, 2013] est un jeu de 49 caractéristiques pour la reconnaissance de symboles manuscrits. Il a été conçu pour couvrir différents aspects des symboles manuscrits afin de pouvoir reconnaitre des tracés de différents types : en-ligne, hors-ligne, mono-stroke et multi-stroke, et dans de nombreux contextes : chiffres et lettres manuscrits, symboles mathématiques, formes géométriques, schéma de mobiliers, etc.

Ce jeu de caractéristiques a été testé sur des bases de données très variées : NicIcon, Ironoff-digits, CVCsymb [Romeu et al., 2006], LaViola, HHReco [Hse and others, 2004] et ImiSketchS [3]. À chaque fois, HBF49 permet d'obtenir des performances comparables à celles publiées dans la littérature, en utilisant seulement ces 49 caractéristiques et des classifieurs standards.

Le principal avantage de ce jeu de caractéristiques pour nous est qu'il forme un espace de représentation de relativement faible dimension. Plus la dimensionnalité d'un modèle est élevé, plus le nombre de données nécessaires pour l'optimiser est important. Dans le cadre de l'apprentissage en-ligne d'un classifieur évolutif à partir de peu de données, il est fondamental d'avoir un modèle de relativement faible dimensionnalité afin de pouvoir l'optimiser rapidement.

Dans le contexte de la reconnaissance de commandes gestuelles personnalisées, les symboles des différentes classes ne sont pas connus à l'avance. Il n'est donc pas possible d'optimiser un jeu de caractéristiques spécifique pour leur reconnaissance. La polyvalence et l'efficacité d'HBF49 font que ce jeu de caractéristiques est très adapté à la reconnaissance de commandes gestuelles personnalisées.

2. *UCI machine learning repository* : `http://archive.ics.uci.edu/ml/`
3. `http://www.irisa.fr/intuidoc/IMIsketchSDB.html`

Ce jeu de caractéristiques universel permet aussi d'obtenir des points de référence pour la reconnaissance de symboles manuscrits, et permet de comparer facilement différents classifieurs en faisant abstraction de l'espace de représentation utilisé.

7.3 Performances générales

Cette section présente les performances générales d'*Evolve* ∞, en particulier sa vitesse d'apprentissage à partir de peu de données.

7.3.1 Vitesse d'apprentissage en-ligne

Evolve ∞ est un classifieur évolutif à l'apprentissage rapide grâce aux capacités génératrices des prémisses des règles d'inférence dont il est composé.

La figure 7.4 présente la vitesse d'apprentissage d'*Evolve* ∞ sur la base de données de commandes gestuelles personnalisées ILGDB. Il est intéressant de constater que *Evolve* ∞ et *Evolve* ont des performances quasiment identiques. Ceci signifie que l'intégration d'oubli ne dégrade en aucun cas la vitesse d'apprentissage du système.

La vitesse d'apprentissage d'*Evolve* ∞ est similaire à celle du classifieur du proche voisin *(nearest neighbor)*. Pendant la phase d'initialisation (phase 0), le classifieur du plus proche voisin obtient même des performances légèrement meilleures. Au bout de la phase d'initialisation, qui contient trois exemples par classe, *Evolve* ∞ a le même taux de reconnaissance que le classifieur du plus proche voisin.

À partir de la phase 1, *Evolve* ∞ commence à être plus performant. À partir de la phase 2, il obtient un taux de reconnaissance supérieur de 10% à celui du classifieur du plus proche voisin. Il faut environ 130 exemples à *Evolve* ∞ pour atteindre un taux de reconnaissance 90%, soit environ six exemples par classe.

Le classifieur Bayésien naïf apprend nettement moins vite pendant l'initialisation car il nécessite au moins trois exemples par classe pour estimer ses probabilités conditionnelles. Il rejoint le classifieur du plus proche voisin à la fin de la phase 3 et le dépasse pendant la phase 4. Le classifieur Bayésien naïf reste cependant à un taux de reconnaissance inférieur à celui d'*Evolve* ∞.

7.4 Inertie du système

Cette section s'attache à évaluer l'intérêt de l'oubli pour maintenir le gain de l'apprentissage et limiter l'inertie du système. Nous allons commencer par mettre en évidence l'augmentation de l'inertie du modèle de connaissance du système. Ensuite, nous allons comparer l'inertie d'*Evolve* ∞ à celle d'*Evolve* et montrer qu'elle reste faible et constante, même à long terme. Cette capacité d'apprentissage infini est permise par l'intégration d'oubli dans le processus d'apprentissage, comme présenté en section 5.4 notamment.

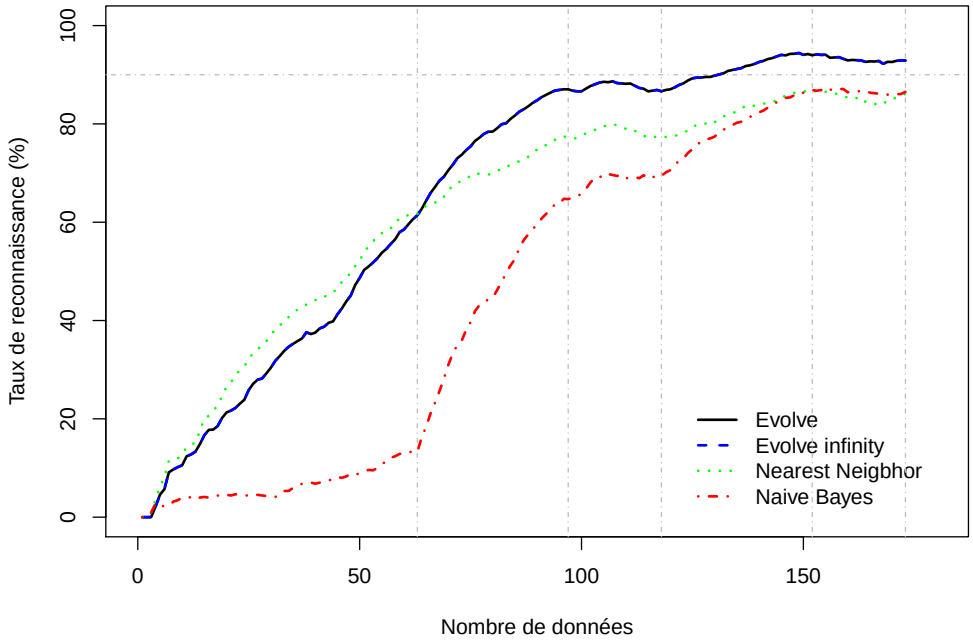


FIGURE 7.4 – Base de données ILGDB – Comparaison de la vitesse d'apprentissage d'*Evolve* ∞ à celle d'*Evolve*, d'un classifieur du plus proche voisin et d'un classifieur Bayésien naïf.

7.4.1 Mise en évidence du problème

Pour évaluer l'inertie du modèle de connaissance du système, nous mesurons le nombre d'exemples nécessaires pour apprendre de nouvelles classes. Nous avons entrainé le classifieur *Evolve* sur 7 classes de la base de données Ironoff-digits, pendant plus ou moins longtemps, puis nous avons introduit les 3 dernières classes. Les résultats sont moyennés sur 30 ordres de présentation des données différents en faisant varier les classes introduites en cours d'apprentissage.

La figure 7.5 montre l'évolution du taux de reconnaissance lors de l'ajout de ces nouvelles classes. En regardant les résultats d'*Evolve*, dont l'apprentissage n'intègre pas d'oubli, il est clair que sa réactivité décroit avec le temps. Il est flagrant que plus les nouvelles classes sont introduites tard, plus le temps nécessaire à leur apprentissage est important. Après l'apprentissage de 2 100 données, il faut 400 exemples supplémentaires pour faire remonter le taux de reconnaissance au-dessus de 90% après l'ajout des 3 nouvelles classes. L'inertie de son modèle de connaissance augmente et sa capacité d'apprentissage diminue au fur et à mesure. Au bout d'un certain temps, *Evolve* n'apprend plus, le classifieur n'est pas réellement évolutif.

Dans *Evolve*, le poids donné à chaque donnée lors de l'apprentissage est inversement proportionnel au nombre de données apprises jusque-là. Le gain de l'apprentissage diminue donc avec le temps, et le modèle de connaissance d'*Evolve* tend à se figer. Par conséquent, l'inertie du classifieur *Evolve* augmente avec le temps, et il perd rapidement ses capacités

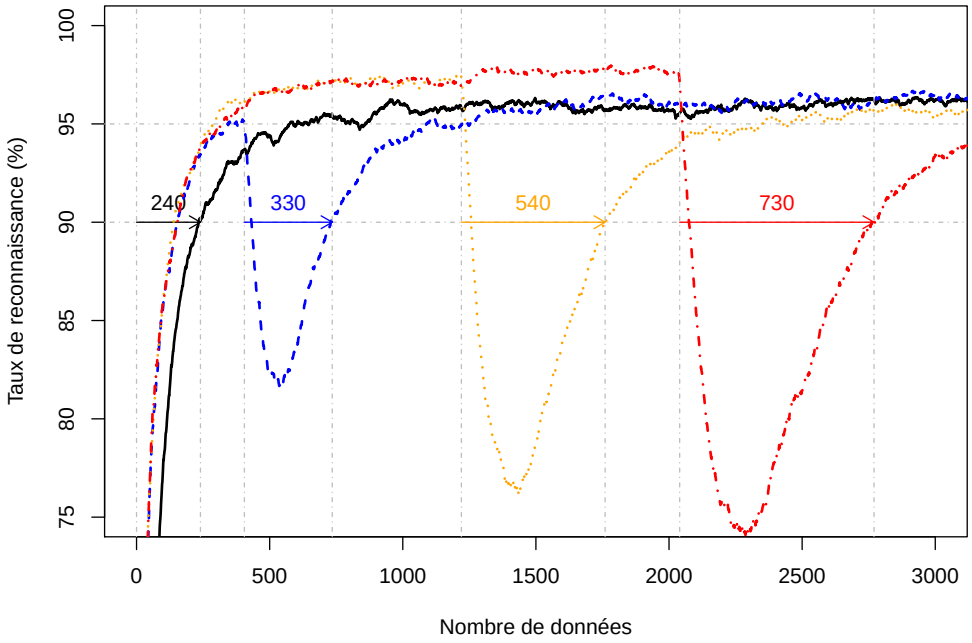


FIGURE 7.5 – Base de données Ironoff-digits – Évolution de l'inertie du système *Evolve* avec le temps. Les différentes courbes correspondent à différentes durée d'apprentissage avant l'introduction des nouvelles classes. Courbe noire : introduction de toutes les classes dès le début ; courbe bleue : introduction après 10% des exemples ; courbe orange : introduction après 30% des exemples ; courbe rouge : introduction après 50% des exemples.

d'évolution. Ce scenario de test montre clairement la nécessité d'introduire de l'oubli dans l'apprentissage d'un système pour qu'il reste évolutif indéfiniment.

7.4.2 Insuffisance des techniques existantes

La manière la plus simple d'intégrer de l'oubli dans l'algorithme des moindres carrés récursifs est l'utilisation d'un facteur d'oubli exponentiel (comme présentée en section 4.5 page 76). Le problème du facteur d'oubli exponentiel est qu'il fait diverger certains éléments de la matrice de covariance de l'algorithme des moindres carrés récursifs, ce qui entraine un oubli catastrophique. La figure 7.6 met en évidence le phénomène d'oubli catastrophique, qui cause l'effondrement du système. Cet effondrement se produit à plus ou moins long terme suivant le facteur d'oubli utilisé, mais se produit inéluctablement. Il est dû à la divergence de certains éléments de la matrice de covariance utilisée par l'algorithme des moindres carrés récursifs. Ce phénomène est la conséquence de l'utilisation du facteur d'oubli exponentiel.

Il est nécessaire d'intégrer de l'oubli dans le processus d'apprentissage afin de maintenir les capacités d'évolution du système, mais sans pour autant provoquer son effondrement. La méthode d'oubli doit à la fois garantir la non-convergence de la matrice de covariance pour

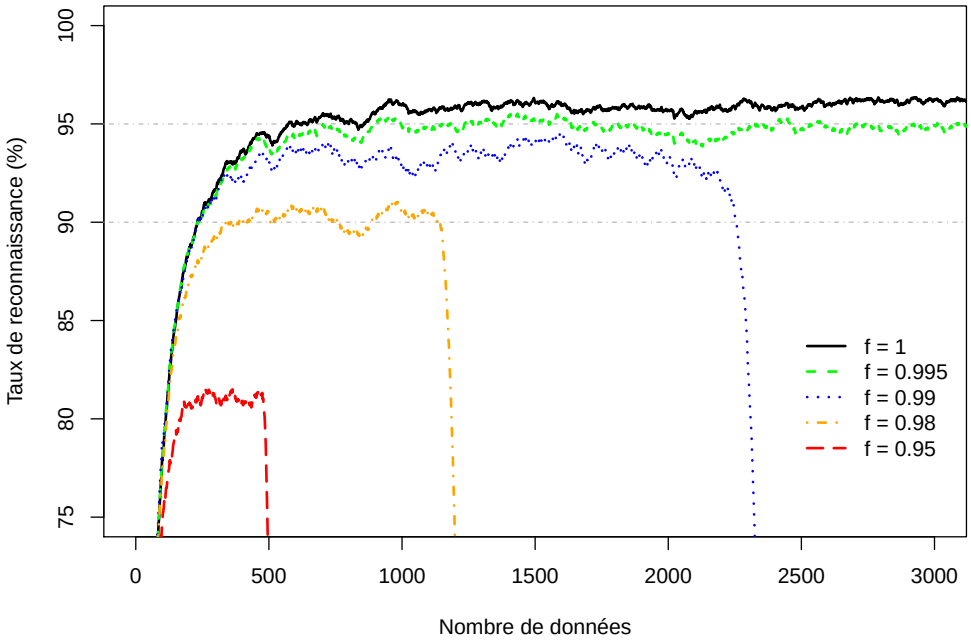


FIGURE 7.6 – Base de données Ironoff-digits – Phénomène d'oubli catastrophique lors de l'utilisation d'un facteur d'oubli exponentiel dans l'algorithme des moindres carrés recursifs. Les performances du système *Evolve* s'effondrent, à plus ou moins long terme suivant le facteur d'oubli utilisé. Plus l'oubli est important, plus l'effondrement des conclusions se produit rapidement.

maintenir le caractère évolutif du classifieur, mais également garantir sa non-divergence pour éviter le phénomène d'oubli catastrophique. L'utilisation d'un facteur d'oubli ne permet pas de garantir ces propriétés de non-convergence ni de non-divergence, et n'est donc pas adaptée. Il est nécessaire d'oublier la connaissance qui a été apprise au bout d'un certain temps. Nous allons montrer l'efficacité du nouvel algorithme d'oubli qui est utilisé dans *Evolve* ∞ (voir section 5.4 page 92).

7.4.3 Validation d'*Evolve* ∞

Nous avons testé l'évolutivité et la réactivité d'*Evolve* ∞, et nous les avons comparées avec celles d'*Evolve*. Pour cela, nous avons utilisé le même protocole en entrainant le système avec 7 classes, pendant des périodes de temps plus ou moins longues (400, 1 200 et 2 000 exemples d'apprentissage), puis nous avons introduit 3 nouvelles classes. Nous avons alors mesuré le temps nécessaire pour mettre à jour le modèle de connaissance et apprendre les nouvelles classes. La figure 7.7 présente les résultats qui sont moyennés sur 30 expériences, avec différentes classes et des ordres des données aléatoires.

Au contraire d'*Evolve*, l'intégration d'oubli dans l'algorithme d'apprentissage permet à *Evolve* ∞ de limiter l'inertie de son modèle de connaissance. Le classifieur a besoin du même nombre d'exemples d'apprentissage pour intégrer de la nouveauté, peu importe

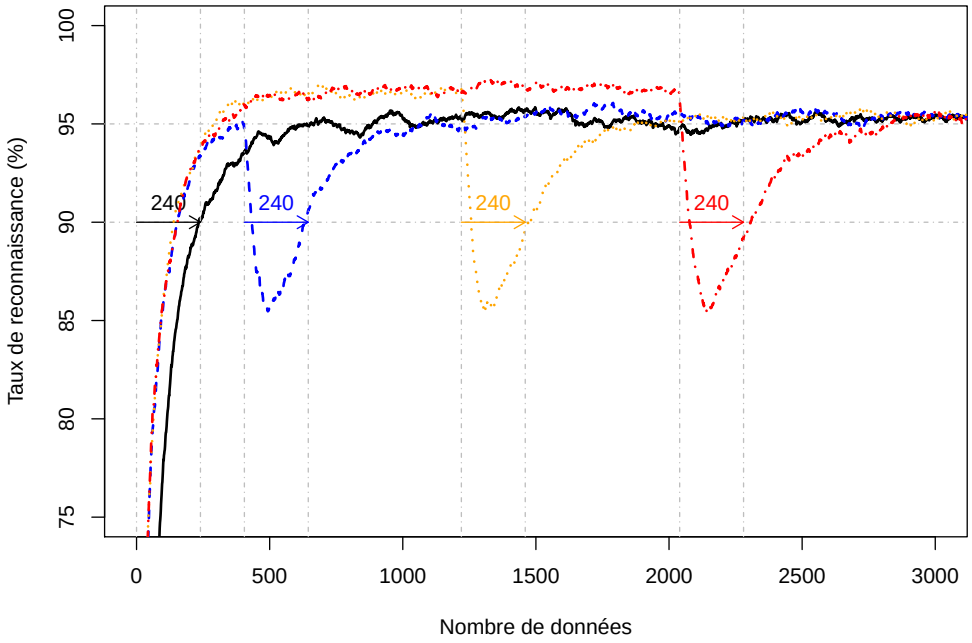


FIGURE 7.7 – Base de données Ironoff-digits – Évolution de l'inertie du système *Evolve* ∞ avec le temps. L'apprentissage de nouvelles classes après différentes durées d'apprentissage nécessite toujours le même nombre d'exemples.

quand elle est introduite. La réactivité d'*Evolve* ∞ est indépendante de son âge, le système conserve ses capacités d'évolution indéfiniment.

La figure 7.8 présente les résultats obtenus avec différentes techniques d'oubli, lors de l'ajout de 3 nouvelles classes après 1200 exemples d'apprentissage. Les trois techniques d'oubli sont l'utilisation d'un facteur d'oubli *(Evolve Forgetting Factor)*, le désapprentissage des anciennes données *(Evolve Unlearning)* et l'oubli directionnel différé (DDF) utilisé dans *Evolve* ∞ *(Evolve Infinity)*. Toutes ces techniques permettent d'obtenir un système réactif.

L'utilisation du facteur d'oubli exponentiel n'est cependant pas envisageable car elle compromet la stabilité du système à long terme. Si le désapprentissage *(Evolve Unlearning)* et l'oubli directionnel différé (*Evolve* ∞) donnent des performances similaires au moment de l'ajout des nouvelles classes, le désapprentissage limite les performances du système lorsque l'environnement est stationnaire. Le désapprentissage des anciennes données, à la place de leur oubli, provoque une augmentation relative du taux d'erreur plus de de 60% (de 3% à 5%).

L'algorithme d'oubli directionnel différé, qui est utilisé par *Evolve* ∞, permet au système d'être réactif lorsque l'environnement est dynamique, tout en conservant de bonnes performances lorsque l'environnement est stationnaire. *Evolve* ∞ obtient ainsi d'aussi bonnes performances qu'*Evolve* dans les périodes où l'utilisation d'oubli n'est pas nécessaire. Par contre, *Evolve* ∞ conserve une très bonne réactivité lorsque l'environnement change, même après une utilisation prolongée : il est évolutif indéfiniment.

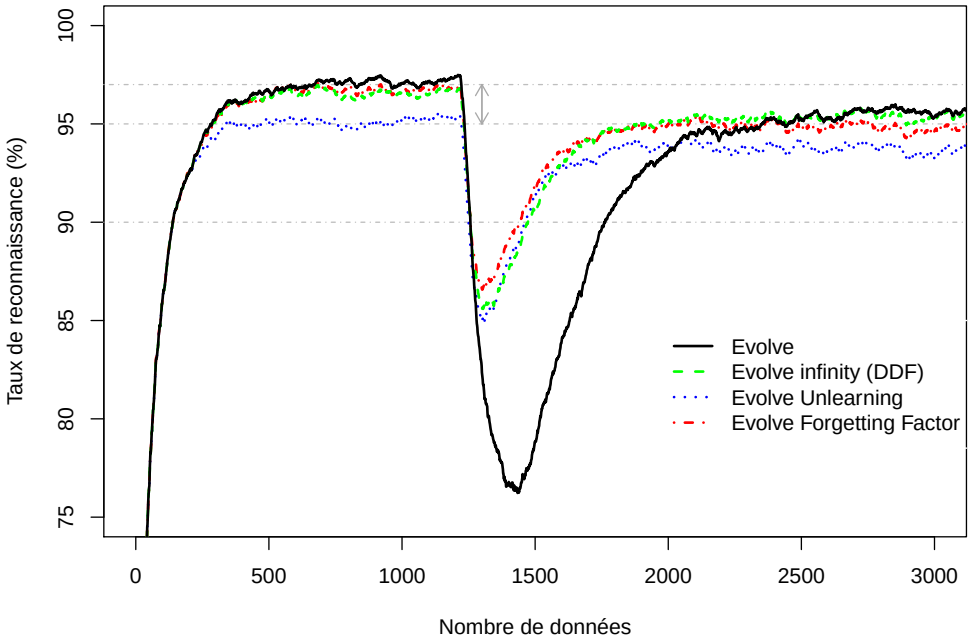
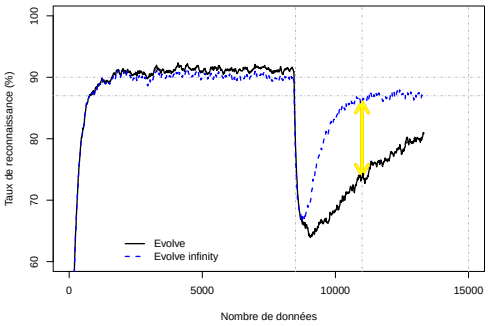
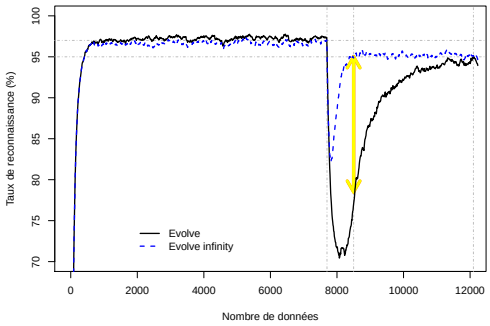


FIGURE 7.8 – Base de données Ironoff-digits – Comparaison de l'inertie du système, avec différents mécanismes d'oubli, lors de l'apprentissage de nouvelles classes en cours d'utilisation. Les trois techniques, le désapprentissage (Unlearning), l'utilisation d'un facteur d'oubli (Forgetting Factor) et l'oubli directionnel différé d'*Evolve* ∞, permettent de réduire efficacement l'inertie par rapport à *Evolve*. Le désapprentissage (Unlearning) dégrade les performances lorsque l'environnement est stationnaire.

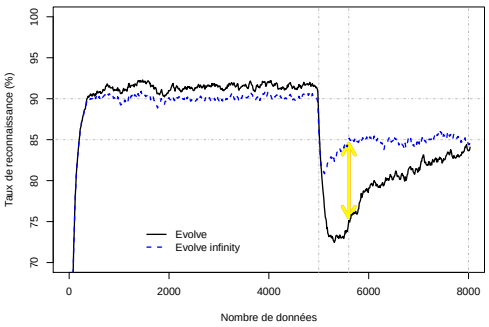
La figure 7.9 met en évidence la différence de réactivité entre *Evolve* et *Evolve* ∞ sur les bases de données LaViola, NicIcon, Japanese-Vowels et Pen-Digits. Dans cet environnement dynamique, l'utilisation d'oubli permet de limiter l'inertie du modèle de connaissance et de maintenir le gain de l'apprentissage. *Evolve* ∞ est ainsi capable d'apprendre les nouvelles classes rapidement, contrairement à *Evolve*. La capacité d'oubli permet à *Evolve* ∞ de converger plus rapidement et d'atteindre ses performances finales plus tôt. Il faut seulement 500 à 800 exemples pour que *Evolve* ∞ intègre la nouveauté liée aux nouvelles classes. Au contraire, il faut de 2 400 à plus de 4 000 exemples à *Evolve* pour réussir à modéliser les nouvelles classes correctement.



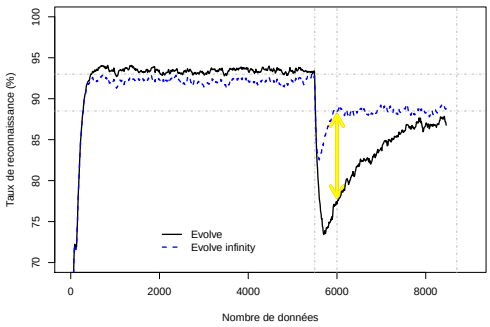
(a) Base de données LaViola.



(b) Base de données NicIcon.



(c) Base de données Japanese-Vowels (UCI).



(d) Base de données Pen-Digits (UCI).

FIGURE 7.9 – Inertie du système lors de l'ajout de 30% des classes après l'apprentissage de 50% des données de la base de données. L'intégration d'oubli dans le processus d'apprentissage d'*Evolve* ∞ lui permet d'être réactif même après une longue période d'apprentissage. La différence de performance lorsque l'environnement est stationnaire reste minime.

7.5 Adaptation aux changements de concepts

Cette section présente les résultats expérimentaux obtenus avec *Evolve* ∞ lors de l'apparition de changements de concepts. Nous allons voir le comportement d'*Evolve* ∞ lors de l'évolution progressive des concepts appris. Nous allons en particulier montrer le peu d'impact de la taille de la fenêtre utilisée par l'algorithme d'oubli directionnel différé, contrairement au cas du désapprentissage. Enfin, la nécessité de l'utilisation d'oubli sera mise en évidence lors de la présence de changements de concepts brusques.

Pour évaluer la capacité d'*Evolve* ∞ à suivre les changements de concepts, nous avons utilisé la bases de données ILGDB, avec le jeu de caractéristiques HBF49 (voir section 7.2). Nous avons ensuite mis les données des différents scripteurs à la suite. De cette manière, nous obtenons un flux de données de plus grande taille, et avec des changements de concepts à chaque changement de scripteur.

En utilisant les scripteurs du groupe 3, dont le jeu de symboles était imposé, les changements de concepts entre les scripteurs sont légers, car ils sont seulement dûs aux différences de style d'écriture. En utilisant les scripteurs du groupe 1, qui ont choisi librement leur jeu de symboles, les changements de concepts sont alors très abrupts, car les symboles associés à chaque classe changent complètement lors des changements de scripteur.

7.5.1 Changements de concepts progressifs

Nous avons testé *Evolve* ∞ sur un scénario simulant des changements de concepts progressifs, en utilisant les 11 scripteurs du groupe 3 d'ILGDB, dont les jeux de symboles sont identiques. Nous avons mis ces 11 scripteurs à la suite, ce qui crée des légers changements de concepts lors du passage du style d'écriture d'un scripteur à celui du suivant. La figure 7.10 présente les résultats moyens sur 30 ordres des scripteurs différents.

En présentant les 11 scripteurs du groupe 3 à la suite, les scripteurs et leur style d'écriture changent, mais le jeu de symboles reste constant. Ce scenario de test est ainsi légèrement changeant, mais ne nécessite pas forcement l'utilisation d'oubli. Le système de référence *Evolve* obtient de bons résultats, même sans oubli. Les changements de concepts ne suivent pas une évolution logique, mais alternent entre les styles des différents scripteurs, dont certains se ressemblent. Sans oubli, le système apprend de tous les scripteurs et construit un modèle multi-scripteurs qui lui permet d'obtenir de bonnes performances dans ce scénario où l'évolution est aléatoire.

Evolve ∞ obtient quant à lui un taux de reconnaissance très proche de celui d'*Evolve*. L'intégration d'oubli dans le processus d'apprentissage permet de maintenir un système quasiment mono-scripteur. *Evolve* ∞ est alors limité par le petit nombre de données d'apprentissage disponible pour chaque scripteur, mais obtient néanmoins de bon résultats.

L'oubli limite ici le poids des anciennes données, ce qui permet de conserver la connaissance préalablement acquise jusqu'à ce qu'elle soit remplacée. Cela permet au système de suivre les changements de concepts, mais ce n'est pas plus intéressant que de construire un modèle multi-scripteurs dans ce scénario où les changements de concepts ne suivent pas une progression logique.

7.5.2 Impact de la longueur de la fenêtre

Le choix de la longueur de la fenêtre glissante utilisée par l'algorithme d'oubli directionnel différé DDF, qui est utilisé pour l'apprentissage décrémental des conclusions, est sujet au compromis précision/réactivité. De la même manière, le choix du facteur d'oubli utilisé pour la mise à jour des prémisses implique une fenêtre de données implicite, et est sujet au même compromis précision/réactivité.

Plus la fenêtre est longue, meilleure est la précision du système car plus le système a de données à sa disposition pour optimiser son modèle de connaissance. En revanche, ce plus grand nombre de données crée de l'inertie qui réduit la réactivité du système en cas d'ajout de classes ou de changements de concepts. Inversement, plus la fenêtre est courte, meilleure est la réactivité du système car les nouvelles données ont un poids plus important dans le processus d'optimisation du modèle de connaissance. Par contre, une petite fenêtre peut

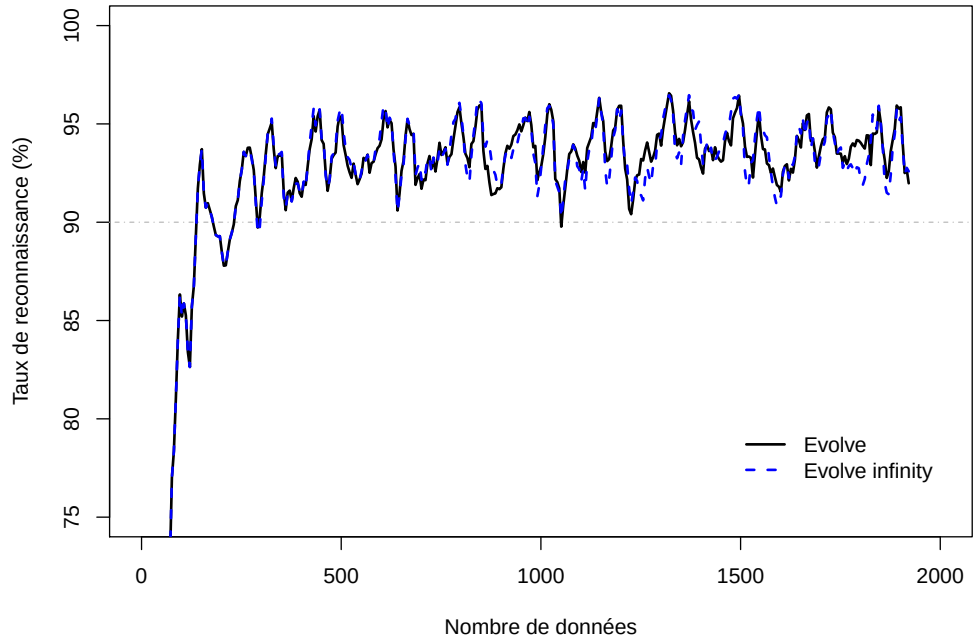


FIGURE 7.10 – Base de données ILGDB groupe 3 (symboles imposés) – Adaptation aux changements de concepts légers créés par les changements de style d'écriture entre les différents scripteurs. L'utilisation d'oubli dans *Evolve* ∞ fait que son modèle est mono-utilisateur, mais cela ne l'empêche pas d'obtenir d'aussi bonnes performances qu'*Evolve* qui construit un modèle multi-utilisateur.

limiter les performances du classifieur lorsque l'environnement est stationnaire, et qu'il dispose d'un nombre insuffisant d'exemples pour construire un modèle de connaissance suffisamment précis.

Dans le contexte applicatif des commandes gestuelles, l'environnement est intrinsèquement dynamique, mais il ne faut pas pour autant limiter la précision du modèle de connaissance que construit le classifieur. Il faut donc limiter le poids des anciennes données, pour améliorer les performances en environnement dynamique, mais sans dégrader les performances en environnement stationnaire. C'est pourquoi nous choisissons une longueur de fenêtre suffisamment grande pour ne pas limiter la précision du modèle de connaissance du classifieur. Même une taille de fenêtre plus grande que nécessaire est quand même suffisante pour maintenir le caractère évolutif du classifieur. Le principal inconvénient d'une fenêtre trop grande est la nécessité de mémoriser plus d'exemples qu'il n'est obligatoire. En pratique nous utilisons une taille de fenêtre choisie empiriquement de dix fois le nombre de classes.

La figure 7.11 montre l'impact de la longueur de la fenêtre de l'algorithme d'oubli d'*Evolve* ∞ sur ses performances en environnement stationnaire.

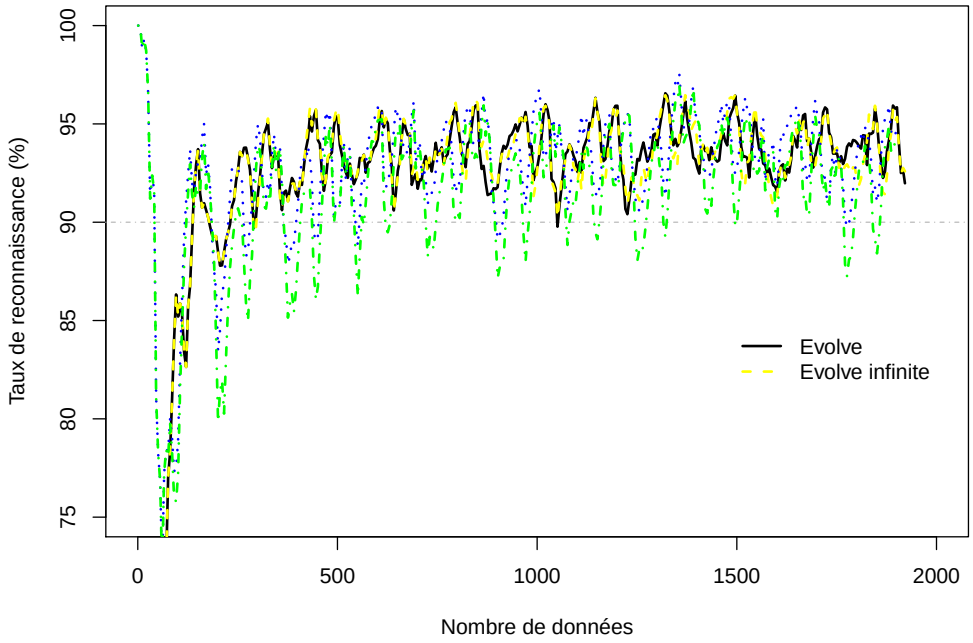


FIGURE 7.11 – Base de données ILGDB groupe 3 (symboles imposés) – Impact de la longueur de la fenêtre de l'algorithme d'oubli d'*Evolve* ∞ sur les performances, et comparaison à l'impact de la taille de la fenêtre en cas de désapprentissage *(Evolve Unlearning)*). La taille de fenêtre est indiquée en nombre d'exemples par classe.

7.5.3 Changements de concepts brusques

Nous avons également testé *Evolve* ∞ dans un scénario simulant des changements de concepts abrupts. Pour cela, nous avons utilisé les 21 scripteurs du groupe 1 d'ILGDB dont les jeux de symboles sont différents. En mettant ces 21 scripteurs à la suite, nous avons simulé des changements de concepts brusques car les symboles associés à chaque classe changent complètement à chaque changement d'utilisateur. La figure 7.12 présente les résultats moyens sur 30 ordres de présentation des scripteurs différents.

Ce scénario de test est très difficile car les symboles associés à toutes les classes changent complètement d'un scripteur à l'autre. Par conséquent, le taux de reconnaissance s'effondre à chaque changement de scripteur. Nous nous intéressons ici à la capacité du système à évoluer pour s'adapter aux nouvelles données, et à remonter son taux de reconnaissance entre les changements de concepts. L'utilisation d'oubli est donc indispensable dans ce scénario fortement dynamique.

Les résultats obtenus pour ce scénario sont très contrastés. Sans capacité d'oubli, *Evolve* n'est pas capable d'adapter son modèle aux changements du flux de données et finit avec un taux de reconnaissance maximum qui ne dépasse plus 50% (courbe rouge). *Evolve* ∞ parvient à maintenir de bonnes performances maximales (courbe verte), malgré la difficulté du scénario, en faisant remonter son taux de reconnaissance au dessus de 90% à la

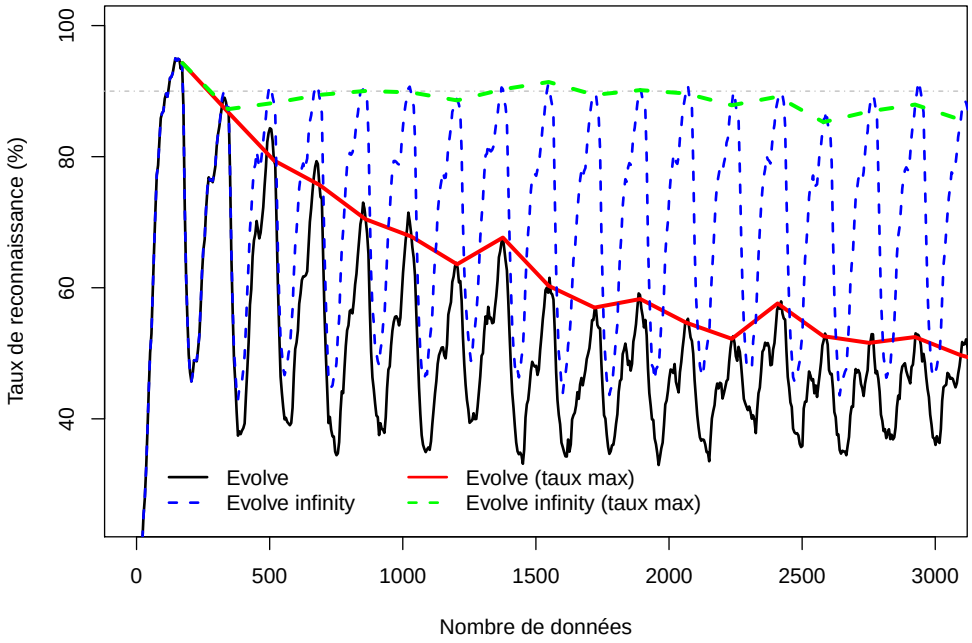


FIGURE 7.12 – Base de données ILGDB groupe 1 (symboles différents) – Adaptation aux changements de concepts abrupts créés par les changements de jeu de symboles entre les différents scripteurs. L'utilisation d'oubli est ici fondamentale pour oublier les données lorsqu'elles deviennent obsolètes. *Evolve* ∞ conserve de bonnes performances maximales (courbe verte) alors que celles d'*Evolve* s'effondrent (courbe rouge), car il n'est pas possible de construire un modèle multi-utilisateur avec des jeux de symboles différents.

fin des données de presque tous les scripteurs. L'intégration d'oubli dans le processus d'apprentissage permet à *Evolve* ∞ d'oublier les données obsolètes après les changements de concepts, pour reconstruire son modèle de connaissance sur le nouveau jeu de symboles du nouveau scripteur. *Evolve* ∞ conserve ainsi un modèle mono-utilisateur, contrairement à *Evolve* qui ne peut que construire un modèle multi-utilisateur qui regroupe alors des jeux de symboles différents. Maintenir un modèle mono-utilisateur, et donc l'utilisation d'oubli, est indispensable dans ce scénario fortement dynamique.

7.6 Option de rejet

Cette section présente l'évaluation expérimentale des capacités d'auto-évaluation et de rejet d'*Evolve* ∞. Nous présentons l'application de la mesure de confiance interne pour assister l'utilisateur dans le choix de son jeu de commandes gestuelles.

7.6.1 Mesure de confiance interne

Cette section présente une application de la mesure de confiance interne (présentée en section 5.5.3 page 101) pour assister les utilisateurs lors de la personnalisation de leur jeu

FIGURE 7.13 – Étape de personnalisation des commandes gestuelles.

de commandes gestuelles. Cette mesure de confiance est basée sur les prototypes des règles d'inférence afin de permettre d'évaluer la qualité du modèle de connaissance d'*Evolve* ∞ à un stade très précoce de l'apprentissage.

7.6.1.1 Protocole expérimental

Nous avons évalué cette application de la mesure de confiance dans un contexte réel d'utilisation de commandes gestuelles, pour obtenir des résultats représentatifs du système, mais aussi des utilisateurs. Pour cela nous avons créé un logiciel de manipulation de photos, permettant de les visualiser et de les modifier en utilisant des commandes gestuelles. Ce logiciel propose 18 commandes basiques, regroupées dans 6 catégories, pour manipuler les photos (comme « suivant », « zoom », « copier », etc.). Nous avons mis au point un protocole qui simule une utilisation réelle de commandes gestuelles pour évaluer l'impact de l'assistance aux utilisateurs lors de l'étape de définition des commandes.

Le protocole de test est divisé en quatre phases : une phase d'initialisation, une première phase de test (*test1*), une phase d'utilisation et une deuxième phase de test (*test2*). Pendant la phase d'initialisation, les utilisateurs doivent personnaliser les commandes gestuelles en choisissant les symboles associés à chaque commande. Pendant chacune des phases suivantes, il était demandé aux utilisateurs d'effectuer diverses actions sur les photos en utilisant leurs commandes gestuelles.

Durant chaque test, trois taux on été mesurés : le taux de reconnaissance du classifieur (*reco*), le taux de mémorisation de l'utilisateur (*memo*), ainsi que le taux d'apprentissage croisé (*cross*). Le taux d'apprentissage croisé représente le taux de commandes à la fois bien effectuées et bien exécutées, c'est-à-dire lorsque l'utilisateur a dessiné le bon symbole et que le système l'a bien reconnu.

Deux groupes d'utilisateurs ont été faits : un premier groupe (*Group1*) qui a été laissé complètement libre pendant la phase de définition de ses commandes, et un deuxième (*Group2*) qui a bénéficié d'une assistance lors de la personnalisation (voir figure 7.13). Cette assistance consiste à informer l'utilisateur de la mesure de confiance du classifieur sur les différents symboles choisis. L'utilisateur peut ainsi changer les gestes complexes, ou trop similaires à des commandes qu'il aura déjà définies, pour améliorer leur reconnaissance par le classifieur.

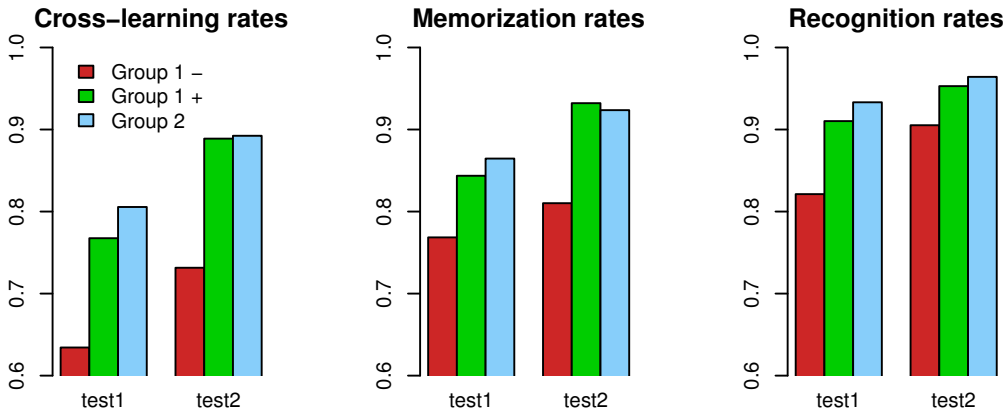


FIGURE 7.14 – Différence de taux d'apprentissage croisé (*cross*), de taux de reconnaissance du classifieur (*reco*), et de taux de mémorisation de l'utilisateur (*memo*), obtenus lors de la première (*test1*) et de la deuxième (*test2*) phase de test, en fonction des groupes d'utilisateurs (*Group1–*, *Group1+* et *Group2*). Le premier groupe (*Group1*) a choisi librement son jeu de commandes, le sous-groupe *Group1–* correspond aux utilisateurs ayant des symboles complexes dans leur jeu de symboles, et le sous-groupe *Group1+* correspond aux autres utilisateurs du groupe *Group1*. Le deuxième groupe (*Group2*) a bénéficié d'assistance lors de la définition de son jeu de commandes gestuelles personnalisées.

L'hypothèse testée lors de cette campagne d'expérimentation est qu'il est possible d'améliorer significativement les performances d'apprentissage croisé en assistant les utilisateurs lors de la définition de leur jeu de commandes. Cette assistance consiste à informer automatiquement l'utilisateur des difficultés du classifieur *Evolve* ∞ en se servant de sa mesure de confiance interne.

7.6.1.2 Amélioration de l'apprentissage croisé

Cette campagne d'expérimentation a été réalisée avec 55 utilisateurs, de 21 à 54 ans, tous habitués à l'informatique mais novices dans l'utilisation de commandes gestuelles. Le premier groupe (*Group1*) contient 39 utilisateurs et le deuxième (*Group2*) en contient 16.

Le premier groupe (*Group1*) a été séparé a posteriori en deux sous-groupes, en fonction du nombre de symboles ayant un faible score de confiance dans leur jeu de commandes. Le premier sous-groupe (*Group1+*) contient 27 utilisateurs qui ont utilisé moins de trois symboles complexes dans leur jeu de commandes. Le deuxième sous-groupe (*Group1–*) contient 12 utilisateurs qui ont utilisé au moins trois symboles complexes à reconnaitre pour le classifieur.

La figure 7.14 montre les différence de taux d'apprentissage croisé (*cross*), de taux de reconnaissance du classifieur (*reco*), et de taux de mémorisation de l'utilisateur (*memo*), obtenus lors de la première (*test1*) et de la deuxième (*test2*) phase de test, en fonction des groupes d'utilisateurs (*Group1–*, *Group1+* et *Group2*). D'une manière générale, les performances obtenues lors de la première phase de test (*test1*) sont relativement limitées puisque le classifieur vient juste d'être initialisé à partir de trois exemples par classe, et que mémoriser 18 commandes gestuelles n'est pas une tâche simple pour tous les utilisateurs. Par

Comparaison	Mesure	Test statistique	Résultat
G1+ > G1-	*cross - test1*	$\chi^2 = 5.5687$, df = 1, p = 0.01828	Significatif
G1+ > G1-	*cross - test2*	$\chi^2 = 7.8593$, df = 1, p = 0.00505	Très sign.
G1+ > G1-	*reco - test2*	$\chi^2 = 3.8429$, df = 1, p = 0.04996	Peu sign.
G2 > G1-	*cross - test1*	$\chi^2 = 8.4392$, df = 1, p = 0.00367	Très sign.
G2 > G1-	*cross - test2*	$\chi^2 = 6.9080$, df = 1, p = 0.008581	Très sign.
G2 > G1-	*reco - test2*	$\chi^2 = 4.8478$, df = 1, p = 0.02768	Significatif

TABLE 7.2 – Tests statistiques de différence entre les groupes pour le taux d'apprentissage croisé (*cross*), et le de taux de reconnaissance du classifieur (*reco*), obtenus lors de la première (*test1*) et de la deuxième (*test2*) phase de test, en fonction des groupes d'utilisateurs (*Group1–*, *Group1+* et *Group2*). Le premier groupe (*Group1*) a choisi librement son jeu de commandes, le sous-groupe *Group1–* correspond aux utilisateurs ayant des symboles complexes dans leur jeu de symboles, et le sous-groupe *Group1+* correspond aux autres utilisateurs du groupe *Group1*. Le deuxième groupe (*Group2*) a bénéficié d'assistance lors de la définition de son jeu de commandes gestuelles personnalisées.

contre, tous les taux mesurés pour chacun des trois groupes augmentent nettement entre la première et la deuxième phase de test, ce qui montre que le classifieur et les utilisateurs apprennent et progressent pendant la phase d'utilisation des commandes gestuelles.

Le sous-groupe *Group1–*, qui correspond aux utilisateurs qui ont des symboles complexes dans leur jeu de symboles, obtient de moins bonnes performances que les utilisateurs qui n'en ont pas (sous-groupe *Group2*). Le deuxième groupe (*Group2*), qui a bénéficié d'assistance lors de la définition de son jeu de commandes gestuelles personnalisées, obtient des performances similaires aux utilisateurs qui n'ont pas utilisé de symboles complexes.

Cela montre que le système d'assistance à la personnalisation des commandes gestuelles, qui a été mis en place à partir de la mesure de confiance d'*Evolve* ∞, garantit pour tous les utilisateurs une définition optimale du jeu de symboles, sans pour autant nuire à leur capacité de mémorisation des 18 commandes. En effet, imposer aux utilisateurs 18 commandes gestuelles « idéales » pour le classifieurs fait chuter de façon très importante leur capacité de mémorisation. Il est complexe pour un utilisateur de retenir 18 commandes sans qu'il ait pu les définir lui même [Li 13]. Le résultat obtenu à travers ce test est donc très important pour garantir l'acceptabilité et l'utilisabilité du système dans sa phase d'amorçage.

Pour analyser plus finement ces résultats et les consolider, nous avons effectué des mesures de significativité. Les résultats des tests statistiques sont présentés table 7.2. On observe tout d'abord que le groupe Group1+ et le deuxième groupe (Group2) obtiennent des résultats significativement meilleurs que le groupe G1- au niveau de leur taux d'apprentissage croisé. Ensuite, pour ce qui est du taux de reconnaissance du classifieur, le groupe *Group1+* et le deuxième groupe (*Group2*) obtiennent de meilleures performances que le groupe *Group1–* lors de la deuxième phase de test (*test2*). La variance des taux de reconnaissance est trop élevée lors de la première phase de test (*test1*) pour pouvoir en tirer des conclusions.

En résumé, pour garantir l'utilisabilité et surtout l'acceptabilité du système à court terme, il est important de pouvoir assister les utilisateurs dans leur personnalisation des commandes gestuelles. En effet on a constaté que 12 utilisateurs sur 39 (plus de 30%) ont

défini, si on ne les guide pas, des gestes qui sont complexes ou ambigües. Ces gestes sont par conséquent difficiles à reconnaitre par le système et en même temps difficile à mémoriser par les utilisateurs. Le système d'aide à la définition des gestes, basés sur la mesure de confiance interne d'*Evolve* ∞, permet de guider efficacement les utilisateurs dans leur choix de gestes, et ainsi d'optimiser l'apprentissage croisé : c'est-à-dire non seulement le taux de reconnaissance obtenu à court terme par le classifieur mais aussi le taux de mémorisation des utilisateurs. Ce facteur est prépondérant pour garantir une bonne acceptabilité du système pour l'ensemble des utilisateurs.

Au final, les utilisateurs qui ont peu de symboles complexes dans leur jeu de commandes obtiennent de meilleures performances. En particulier, la mesure de confiance interne d'*Evolve* ∞ est efficace pour informer l'utilisateur des difficultés de reconnaissance du classifieur, et ce dès la phase d'initialisation qui est un stade très précoce de l'apprentissage.

7.6.2 Option de rejet multi-seuils

Evolve ∞ dispose d'une option de rejet de confusion basée sur une architecture multi-seuils. L'utilisation d'un seuil différent pour chaque classe permet d'obtenir un meilleur compromis erreur/rejet que l'utilisation d'un seuil unique.

Pour évaluer cette architecture de rejet multi-seuils, nous avons entrainé le système sur les données concaténées de trois scripteurs du groupe 3 d'ILGDB (symboles imposés). Ce protocole est présenté en détails dans la section 8.2.1. Les seuils de rejet sont appris avec l'algorithme présenté en section 5.5.2 (page 100). L'option de rejet est ensuite évaluée sur la base de données de test pour mesurer le compromis erreur/rejet obtenu.

La figure 7.15 présente les compromis erreur/rejet obtenus avec les architectures mono-seuil et multi-seuils. La courbe erreur/rejet est obtenue en faisant varier le seuil de rejet unique pour changer la proportion de données rejetées.

Il est clair que l'utilisation d'une architecture multi-seuils permet d'obtenir un compromis erreur/rejet plus intéressant que celui obtenu avec un seuil de rejet unique. Avec un seuil par classe, le taux d'erreur final est de 3.31%, à la place de 3.96% avec un seuil unique (pour un taux de rejet identique de 6.33%).

7.7 Conclusion

Nous avons présenté dans ce chapitre l'évaluation expérimentale du classifieur évolutif *Evolve* ∞, qui est un système d'inférence floue d'ordre un, dit de Takagi-Sugeno. C'est un système qui apprend rapidement à partir de peu de données, grâce aux capacités génératrices des prémisses des règles. Les capacités discriminantes des conclusions d'ordre un lui permettent d'obtenir également de bonnes performances après suffisamment de données d'apprentissage. C'est un système qui supporte très bien l'ajout de classes en cours d'utilisation grâce à son architecture sous forme d'ensemble de règles. La création de nouvelles règles permet d'intégrer de la nouveauté au modèle du classifieur, sans pour autant perturber la connaissance déjà existante. *Evolve* ∞ dispose d'un algorithme d'apprentissage de faible complexité, ce qui lui permet d'être appris en-ligne. Il est capable d'évoluer in-

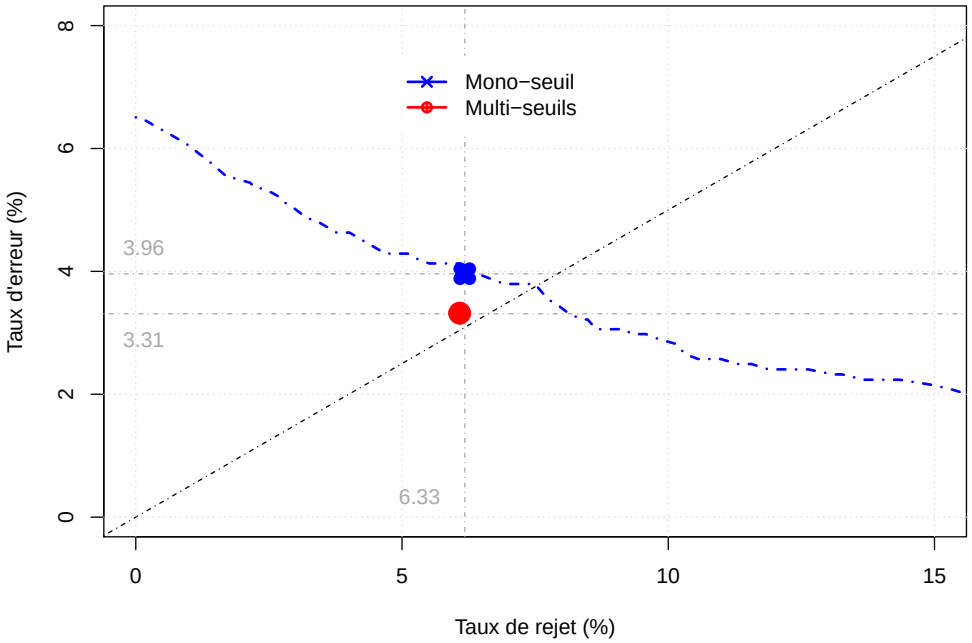


FIGURE 7.15 – Base de données ILGDB groupe 3 – Performances de l'option de rejet de confusion multi-seuils d'*Evolve* ∞. La courbe erreur/rejet est obtenue en faisant varier le seuil de rejet constant. L'utilisation d'une architecture multi-seuils permet d'obtenir un compromis erreur/rejet plus intéressant.

définiment pour s'adapter au flux de données et en particulier suivre les changements de concepts.

L'intérêt de la capacité d'oubli d'*Evolve* ∞ a été testé sur différents scénarios, et son inertie a été comparée à celle d'*Evolve*. Le comportement d'*Evolve* ∞ lors de l'ajout de nouvelles classes à moyen et long terme a montré le maintien du gain de l'apprentissage au cours du temps. Le temps d'apprentissage de ces nouvelles classes est constant pour *Evolve* ∞, peu importe quand ces nouvelles classes sont introduites. Sans oubli par contre, le modèle de connaissance d'*Evolve* converge et n'apprend plus au bout d'un certain temps. Les effets de l'oubli ont également été mis en évidence lorsque le flux de données présente des changements de concepts. *Evolve* ∞ est capable de s'adapter à tout changement dans la distribution des données, même brusques et radicaux, contrairement à *Evolve* dont les performances s'effondrent.

Différentes approches ont été comparées pour introduire de l'oubli dans le processus d'apprentissage du système. Le nouvel algorithme d'oubli directionnel différé (DDF) d'*Evolve* ∞ permet de limiter efficacement l'inertie du modèle de connaissance du système. Contrairement à l'utilisation d'un facteur d'oubli exponentiel, l'algorithme d'oubli DDF ne compromet pas la stabilité du système, même à long terme. Il ne limite pas non plus ses performances lorsque l'environnement est stationnaire, contrairement au désapprentissage du système sur une fenêtre glissante.

Le classifieur évolutif *Evolve* ∞ permet l'utilisation de commandes gestuelles person-

nalisées grâce à plusieurs caractéristiques. Tout d'abord, le système apprend pendant son utilisation pour continuer à s'améliorer et atteindre les meilleures performances possibles. Cela permet à l'utilisateur de personnaliser les classes, en ne fournissant que peu de données pour l'initialisation du système, tout en obtenant de bonnes performances après quelque temps d'utilisation. Ensuite, *Evolve* ∞ permet l'intégration de nouvelles classes à n'importe quel moment de l'apprentissage du système, ce qui permet à l'utilisateur d'ajouter de nouvelles commandes à n'importe quel moment. Enfin, le système s'adapte à toute évolution des données et conserve de bonnes performances malgré le changement du style d'écriture de l'utilisateur, ou le changement des symboles utilisés.

Chapitre 8

Évaluation du superviseur actif en-ligne *IntuiSup*

8.1 Introduction

L'utilisation de commandes gestuelles personnalisées est une situation d'apprentissage croisé, où l'utilisateur doit mémoriser le jeu de commandes et le classifieur doit apprendre à les reconnaitre. Le seul moyen d'obtenir les étiquettes des données sans erreur est d'interagir avec l'utilisateur. Or interagir avec l'utilisateur a un coût, cela réduit considérablement la fluidité d'utilisation des commandes gestuelles. Toutes les données ne peuvent donc pas être étiquetées et il faut soigneusement choisir celles qui le seront pour être utilisées pour l'apprentissage du classifieur. Le rôle du superviseur est de sélectionner ces données et de gérer la fréquence des interactions avec l'utilisateur. Comme les données arrivent sous forme d'un flux, la décision d'utiliser ou non chaque donnée pour l'apprentissage doit être réalisée « à la volée », donnée par donnée, sans connaissance de celles qui vont arriver dans le futur. L'apprentissage du classifieur nécessite donc un mécanisme de supervision actif en-ligne.

Le superviseur *IntuiSup* permet de superviser l'apprentissage d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage. Il permet de minimiser les sollicitations de l'utilisateur tout en maximisant l'apprentissage du classifieur. La sélection des données est réalisée en-ligne, donnée par donnée, en utilisant la fonction de confiance du classifieur pour sélectionner les données sur lesquelles le classifieur a le moins de certitude. Cet échantillonnage, similaire à une option de rejet des données, est évolutif. Il s'adapte à l'évolution du système pour maintenir le rapport erreur/sollicitation proche de celui désiré. Le superviseur utilise une méthode d'échantillonnage avec dopage *(boosting)* pour accélérer l'apprentissage en augmentant les interactions avec l'utilisateur aux moments importants : lors de l'apprentissage initial et lors des changements de concepts.

Ce chapitre présente les différents tests que nous avons effectués pour valider expérimentalement le superviseur *IntuiSup*, et sa capacité à superviser l'apprentissage actif en-ligne d'un classifieur avec l'utilisateur dans la boucle. Nous montrons l'efficacité des différentes contributions (présentées dans le chapitre 6). La superviseur actif en-ligne *IntuiSup*

comprend deux méthodes :

— la méthode d'étiquetage implicite ;

— la méthode d'étiquetage explicite ;

avec deux variantes pour l'étiquetage explicite :

— la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* ;

— la méthode d'échantillonnage avec « dopage » B-ESU *(Boosted-ESU)*.

Les différents tests expérimentaux du superviseur *IntuiSup* sont effectués sur la base de données de commandes gestuelles personnalisées ILGDB [Renau-Ferrer et al., 2012]. Elle a été récoltée dans un environnement immersif où les utilisateurs utilisaient une application de manipulation de photos pendant plus de 45 minutes chacun. Le superviseur *IntuiSup* est en effet conçu pour superviser l'apprentissage d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage et que les données évoluent avec le temps. La base de données ILGDB permet de réaliser des expérimentations qui sont les plus proches possibles d'une utilisation réelle, sans faire de campagne de tests réels.

Nous commençons par présenter le protocole et la base de données utilisée pour l'évaluation de notre approche en section 8.2. Puis nous validons la stratégie de supervision active en-ligne combinant étiquetage implicite et explicite pour l'apprentissage d'un classifieur évolutif en section 8.3. Nous montrons ensuite l'efficacité de l'option de rejet évolutive ESU pour faire évoluer la capacité de rejet et maintenir le compromis erreur/rejet au cours du temps en section 8.4. Enfin, nous présentons l'intérêt de la méthode d'échantillonnage avec dopage *(boosting)* B-ESU pour accélérer la vitesse d'apprentissage du classifieur, à la fois pendant l'apprentissage initial et les changements de concepts en section 8.5.

8.2 Méthodologie

Pour évaluer le superviseur *IntuiSup* pour l'apprentissage actif en-ligne de commandes gestuelles, nous avons utilisé la base de données ILGDB [Renau-Ferrer et al., 2012] avec le jeu de caractéristiques HBF49 [Delaye and Anquetil, 2013], qui sont décrits en section 7.2. Nous présentons dans cette section le protocole d'évaluation particulier que nous avons mis au point pour évaluer le superviseur *IntuiSup*.

La base de données ILGDB [Renau-Ferrer et al., 2012] contient des commandes gestuelles personnalisées qui ont été collectées dans un environnement immersif. Cette base de données est très intéressante pour trois raisons comme nous l'avons décrit en section 7.2.2.1. Tout d'abord, les symboles sont ordonnés chronologiquement ce qui permet de voir l'évolution du style d'écriture des utilisateurs avec le temps. Ensuite, les fréquences des différentes classes ne sont pas homogènes et varient au cours du temps. Enfin, pour toute une partie de la base de données, les différents symboles utilisés ont été librement choisis par les utilisateurs, ce qui donne une base de données très diversifiée. ILGDB est donc une base de données unique et particulièrement intéressante pour l'évaluation d'un système de supervision actif en-ligne.

HBF49 [Delaye and Anquetil, 2013] est un jeu de caractéristiques universel pour la reconnaissance de symboles manuscrits comme nous l'avons présenté en section 7.2.3. Il a été conçu pour couvrir différents aspects des symboles manuscrits afin de pouvoir reconnaitre

des tracés de différents types et dans de nombreux contextes. Ce jeu de caractéristiques permet d'obtenir des performances comparables à celles publiées dans la littérature, en utilisant seulement ces 49 caractéristiques et des classifieurs standards. La polyvalence et l'efficacité d'HBF49 font que ce jeu de caractéristiques est très adapté à la reconnaissance de commandes gestuelles personnalisables.

8.2.1 Protocole de test

Le superviseur *IntuiSup* est conçu pour superviser l'apprentissage d'un classifieur évolutif lorsque l'utilisateur est dans la boucle d'apprentissage. L'évaluation d'un superviseur sans utilisateur réel est délicate, puisqu'il n'est pas possible de simuler l'effet de l'évolution du système sur le comportement de l'utilisateur. En revanche, nous utilisons la base de données ILGDB pour simuler l'effet de l'évolution de l'utilisateur sur le système. Cela permet de réaliser des expérimentations qui sont les plus proches possibles d'une utilisation réelle, sans faire des campagnes de saisie réelles pour chaque test.

Pour nos expérimentations, nous avons simulé une utilisation un peu plus longue que celle des utilisateurs d'ILGDB (qui correspond à 45 minutes d'utilisation continue, et qui sont réparties en cinq phases de 0 à 4). Pour cela, nous avons concaténé les données de trois utilisateurs (notés X, Y et Z) ayant utilisé le même jeu de symboles (groupe 3 d'ILGDB). Ce protocole nous permet également de simuler deux changements de concepts lors des changements de scripteurs, qui ont chacun leur propre style d'écriture. Le système est initialisé avec la phase 0 du premier utilisateur (X0) contenant trois symboles par classe. Les phases 1, 2 et 3 des trois utilisateurs sont ensuite utilisées pour simuler l'utilisation du système de commandes gestuelles (~ 270 symboles). La concaténation des données des trois scripteurs est réalisée comme suit : X1, X2, X3, Y1, Y2, Y3, Z1, Z2 puis Z3 (les phases Y0 et Z0 ne sont pas utilisées). Les phases 4 des trois utilisateurs (X4, Y4 et Z4) sont utilisées comme base de test (63 symboles) pour évaluer les performances du système entre chacune des différentes phases d'utilisation. Les tests sont effectués et moyennés sur 30 triplets d'utilisateurs (X, Y et Z) différents.

8.2.2 Méthode d'évaluation

Notre méthode d'échantillonnage pour l'étiquetage explicite des données, qui est basée sur une sélection des données par le principe de l'incertitude maximale, est similaire à une option de rejet des données. Lorsqu'une donnée est sélectionnée par la méthode d'échantillonnage, elle n'est pas reconnue, comme si elle était rejetée, et doit alors être étiquetée par l'utilisateur.

Pour évaluer les performances d'une option de rejet d'un classifieur hors-ligne, la mesure la plus courante est l'aire sous la courbe erreur/rejet [Mouchere and Anquetil, 2006b]. Lorsque l'apprentissage est réalisé hors-ligne, une base de validation est utilisée pour tracer la courbe erreur/rejet en faisant varier le seuil de rejet (de sélection). Il faut alors choisir le compromis erreur/rejet – appelé point de fonctionnement – souhaité pour la phase d'utilisation. Cependant, cette approche n'est pas pertinente pour un système en-ligne, où il n'y a plus de base de validation, et où les phases d'apprentissage et d'utilisation sont

mélangées. Il n'est alors plus possible de tracer la courbe erreur/rejet pour choisir un point de fonctionnement avant l'utilisation du système.

Dans le cas d'une méthode d'échantillonnage en-ligne, le point de fonctionnement (compromis erreur/rejet choisi) a un impact sur le nombre de données sélectionnées, et donc sur l'apprentissage du classifieur. Il n'est donc pas possible d'entrainer le classifieur et ensuite de choisir un point de fonctionnement a posteriori. Pour un système en-ligne, il est nécessaire de choisir un point de fonctionnement a priori, avant le début de l'utilisation du système. Ensuite, le classifieur est entrainé avec le point de fonctionnement sélectionné, et le compromis erreur/rejet évolue avec l'apprentissage du classifieur. On parle alors de la trajectoire suivie par le point de fonctionnement au fur et à mesure de l'évolution du système.

Pour mettre en évidence ce phénomène, nous avons utilisé la base de test (composé des phases X4, Y4 et Z4, voir section 8.2.1) pour tracer des courbes erreur/rejet à différents stades de l'apprentissage du système. Pour tracer ces courbes, il est nécessaire de faire varier le seuil de rejet (sélection), et pour chaque seuil, de compter le nombre d'erreurs et de rejets (sélections) sur la base de test. Dans le cas d'une utilisation réelle, le tracé de courbes erreur/rejet n'est pas possible, puisque son tracé prendrait trop de temps et qu'aucune base de test n'est alors disponible.

La figure 8.1 montre la courbe erreur/rejet obtenue sur la base de test après l'initialisation du système sur les données de la phase X0. À ce moment-là, il faut choisir le point de fonctionnement initial, qui est ensuite utilisé pendant l'apprentissage en-ligne. Le point de fonctionnement (compromis erreur/rejet) évolue avec le temps. La figure 8.2 montre alors la trajectoire du point de fonctionnement, obtenue à partir du point de fonctionnement initial choisi. Les courbes erreur/rejet suivantes (courbes X1 à Z3, en gris sur la figure 8.2) sont alors « fictives », car il n'est pas possible de les calculer pendant l'utilisation réelle du système. De plus, le compromis final ne peut être obtenu que suite au choix de celui utilisé pour l'apprentissage. Pour choisir un autre compromis sur la courbe finale, il faut changer le point de fonctionnement initial, ce qui change alors l'apprentissage en-ligne et donne une courbe finale différente.

Pour pouvoir évaluer les performances d'un système en-ligne avec une option de rejet, nous proposons une nouvelle approche : la trajectoire du point de fonctionnement. Cette trajectoire permet de voir l'évolution du compromis erreur/sollicitation et d'évaluer si le compromis souhaité est maintenu au cours du temps. De plus, le point de fonctionnement final permet d'évaluer l'état final du système, et de comparer différentes configurations en terme de compromis erreur/sollicitation. Nous évaluons donc les performances des options de rejet évolutives en utilisant le compromis final erreur/rejet et la trajectoire du point de fonctionnement pour arriver jusque-là.

8.3 Stratégie de supervision active en-ligne

Nous cherchons ici à évaluer les différentes méthodes d'étiquetage, notamment implicite et explicite, pour la supervision active en-ligne de l'apprentissage.

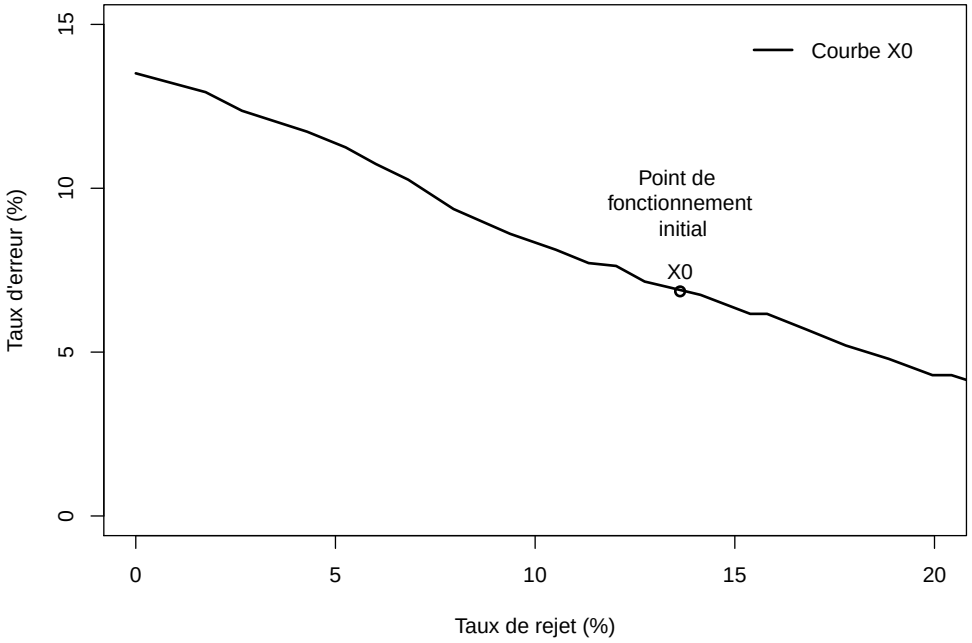


FIGURE 8.1 – Courbe erreur/rejet obtenue sur la base de test après l'initialisation du système, et point de fonctionnement erreur/sollicitation choisi a priori.

8.3.1 Combinaison des méthodes d'étiquetage

Nous souhaitons évaluer l'efficacité d'*IntuiSup* pour améliorer la qualité du modèle de connaissance du classifieur. Pour cela nous avons évalué le classifieur sur la base de test (X4+Y4+Z4, cf 8.2.1) sans utiliser l'option de rejet. Le taux d'erreur total correspond au nombre de données mal reconnues, sans rejeter de données.

Le tableau 8.1 présente les taux d'erreurs totaux, pour cinq types de supervision (voir section 6.2 page 109). Le premier correspond à l'initialisation du système (apprentissage sur X0) et à l'absence d'apprentissage en-ligne ensuite (les phases X1 à Z3 ne sont alors pas utilisées).

La deuxième est l'auto-supervision qui consiste à étiqueter les données d'apprentissage avec l'étiquette reconnue par le classifieur lui-même, c'est la méthode d'apprentissage semi-supervisée la plus facile à mettre en œuvre. Cet étiquetage peut par contre contenir des étiquettes erronées lorsque les données sont mal reconnues. Comme prévu, apprendre de manière auto-supervisée détériore les performances du classifieur : le taux d'erreur total (sans utiliser d'option de rejet) augmente de 11.53% à 13.28%. Il vaut donc mieux ne pas apprendre que d'apprendre sur des données dont l'étiquetage contient des erreurs qui vont dégrader le modèle de connaissance du classifieur.

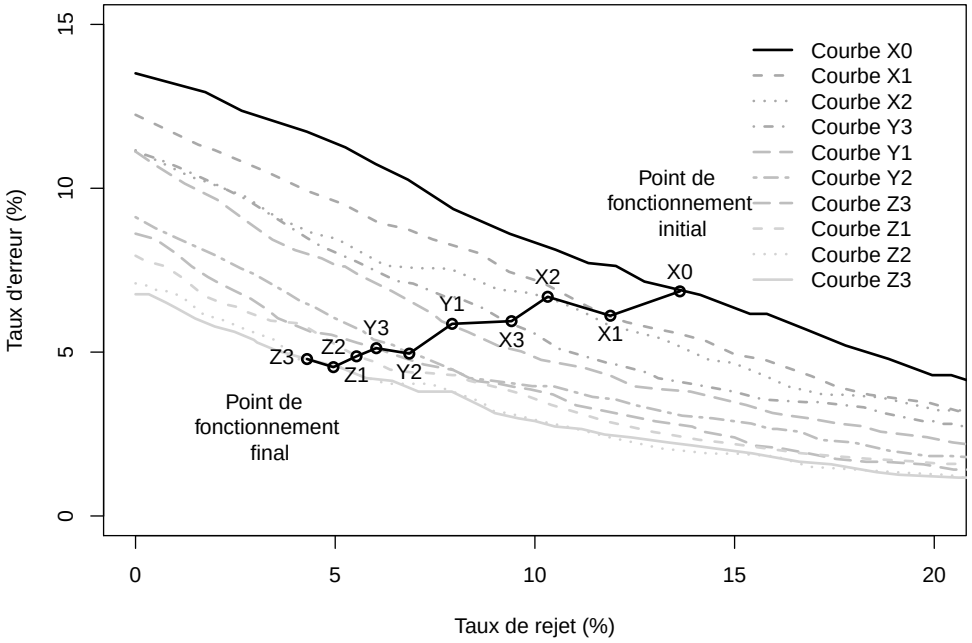


FIGURE 8.2 – Évolution et trajectoire du point de fonctionnement erreur/sollicitation avec l'apprentissage du système et l'évolution des données. La courbe Ui correspond à la courbe erreur/rejet fictive qui est obtenue sur la base de test après l'apprentissage de la phase i de l'utilisateur U.

La méthode d'étiquetage implicite consiste à apprendre à partir des données d'utilisation avec l'étiquette reconnue mais seulement lorsque celle-ci est validée implicitement par l'utilisateur en poursuivant ses actions (sans annuler la commande effectuée), c'est le cas A de la figure 6.2 (page 110). La stratégie de supervision implicite permet d'améliorer les performances du classifieur (réduction du taux d'erreur total de 11.53% à 10.15%), mais pas autant que la stratégie de supervision explicite.

La méthode d'étiquetage explicite consiste à faire explicitement étiqueter par l'utilisateur les données pour lesquelles le classifieur est le plus incertain lors de la reconnaissance, c'est le cas D (voir figure 6.2 page 110). Les cas B et C correspondent aux erreurs de mémorisation et de reconnaissance, qui ne sont donc ni validées implicitement ni explicitement. Cette méthode d'étiquetage permet d'obtenir un taux d'erreur total de 7.27%.

Il est essentiel que le classifieur puisse apprendre à partir de ses erreurs pour pouvoir s'améliorer efficacement. Finalement, la stratégie de supervision *IntuiSup*, qui combine supervision implicite et explicite, permet d'améliorer efficacement les performances du classifieur (taux d'erreur total de 6.52%).

Stratégie de supervision	Taux d'erreur total (%)
Pas d'apprentissage en-ligne	11.53
Auto-supervision (A+B+C+D)	13.28
Supervision implicite (A)	10.15
Supervision explicite (D)	7.27
Stratégie *IntuiSup* (A+D)	6.52

TABLE 8.1 – Comparaison des différentes stratégies de supervision de l'apprentissage en-ligne. Le taux d'erreur total prend en compte les données mal reconnues et rejetées. A représente les données bien reconnues, B les erreurs de mémorisation, C les erreurs de reconnaissance et D les données rejetées/sélectionnées (cf. figure 6.2 page 110). L'absence d'apprentissage en-ligne consiste à n'utiliser que les données d'initialisation, et à ne pas apprendre ensuite. L'auto-supervision consiste à étiqueter les données d'apprentissage avec les étiquettes proposées par le classifieur. La supervision implicite consiste étiqueter implicitement les données bien reconnues lorsque l'utilisateur poursuit ses actions. La supervision explicite consiste à interagir avec l'utilisateur pour étiqueter les données. La stratégie *IntuiSup* combine la supervision implicite et la supervision explicite.

8.3.2 Utilisation des erreurs de mémorisation et de reconnaissance

La stratégie de supervision d'*IntuiSup* combine les méthodes d'étiquetage implicite et explicite (catégories A et D de la figure 6.2 page 110), mais n'utilise pas les erreurs de mémorisation de l'utilisateur (catégorie B), ni les erreurs de reconnaissance du classifieur (catégorie C).

Le tableau 8.2 présente les taux d'erreur totaux (sans utiliser d'option de rejet) et les taux d'interaction avec l'utilisateur obtenus pour l'éventuelle exploitation des données non utilisées. La stratégie de référence, sans utiliser les données correspondants aux erreurs de mémorisation et de reconnaissance (catégories B et C), est la stratégie de supervision *IntuiSup* ci-dessus, qui permet d'obtenir un taux d'erreur total de 6.52%.

Utiliser les erreurs de mémorisation et de reconnaissance nécessite une stratégie d'étiquetage pour ces données. L'auto-étiquetage est la méthode d'apprentissage semi-supervisée la plus facile à mettre en œuvre : elle consiste à étiqueter les données d'apprentissage avec les étiquettes proposées par le classifieur. Cependant l'auto-étiquetage des données ne sert ici à rien, les quelques erreurs d'étiquetage suffisent à annuler le bénéfice des données d'apprentissage supplémentaires (augmentation du taux d'erreur de 6.52% à 6.77%).

Solliciter l'utilisateur pour étiqueter explicitement ces données permet de réduire le taux d'erreur du classifieur (de 6.77% à 5.51%), mais au prix d'une forte augmentation des sollicitations de l'utilisateur. Le taux d'interaction augmente de 7.64% à 16.4%. Cela représenterait une augmentation de 4.80 sollicitations à 10.32 sur la base de test de 63 symboles (115% d'augmentation relative). Cette légère amélioration des performances est donc très coûteuse, et inintéressante du point de vue de l'utilisateur. Finalement, nous n'utilisons pas les données correspondant aux erreurs de mémorisation et de reconnaissance, qui représentent $\sim 9\%$ des données totales, dans le processus d'apprentissage en-ligne du classifieur.

Le superviseur actif en-ligne *IntuiSup* utilise seulement les données qui ont été soit implicitement validées (catégorie A de la figure 6.2 page 110), soit explicitement étiquetées

Stratégie *IntuiSup* (A+D) et	Taux d'erreur total final (%)	Taux d'interaction utilisateur (%)
Pas d'apprentissage (B+C)	6.52	7.39
Auto-supervision (B+C)	6.77	7.64
Supervision explicite (B+C)	5.51	16.4

TABLE 8.2 – Performance et coût d'une stratégie de supervision sur les données restantes (B et C), en plus de la stratégie *IntuiSup* sur les données A et D. A représente les données bien reconnues, B les erreurs de mémorisation, C les erreurs de reconnaissance et D les données rejetées/sélectionnées (cf. figure 6.2 page 110). L'auto-supervision consiste à étiqueter les données d'apprentissage avec l'étiquette reconnues pas le classifieur. La supervision explicite consiste à interagir avec l'utilisateur pour étiqueter les données.

par l'utilisateur (catégorie D).

En résumé, le superviseur actif en-ligne *IntuiSup* permet de superviser l'apprentissage d'un classifieur évolutif avec l'utilisateur dans la boucle. Il combine une méthode implicite qui permet d'étiqueter la majorité des données bien reconnues, et une méthode explicite qui permet d'étiqueter les données complexes à reconnaitre. De cette manière, il optimise à la fois les performances du classifieur tout en minimisant les interactions avec l'utilisateur.

Dans cette section, la méthode d'étiquetage explicite utilisait uniquement un seuil de sélection constant. Nous évaluons maintenant une autre méthode d'échantillonnage des données, que celle du seuil constant, pour l'étiquetage explicite par l'utilisateur.

8.4 Méthode d'échantillonnage ESU pour l'étiquetage explicite

La méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* permet de faire évoluer la quantité de données sélectionnées en synergie avec l'évolution du classifieur.

8.4.1 Comparaison des méthodes d'évolution du seuil d'échantillonnage

Même si le seuil de rejet reste constant, le système de reconnaissance évolue et s'améliore, ce qui fait que le compromis erreur/rejet change avec le temps. Au fur et à mesure que les performances du système s'améliorent, la confiance moyenne du système s'améliore également. Par conséquent, le taux de rejet du système décroit plus rapidement que le taux d'erreur, et le compromis erreur/rejet est modifié. La proportion des données sélectionnées pour l'apprentissage devient trop faible par rapport au taux d'erreur restant.

Cette section présente les résultats obtenus avec différents algorithmes pour faire évoluer l'option de rejet qui est utilisée par la stratégie de supervision *IntuiSup*. Comme la méthode d'échantillonnage évolue, les données utilisées par la méthode de supervision explicite pour l'apprentissage du classifieur changent suivant l'algorithme utilisé. La figure 8.3 compare les trajectoires des points de fonctionnement obtenus avec la méthode d'échantillonnage

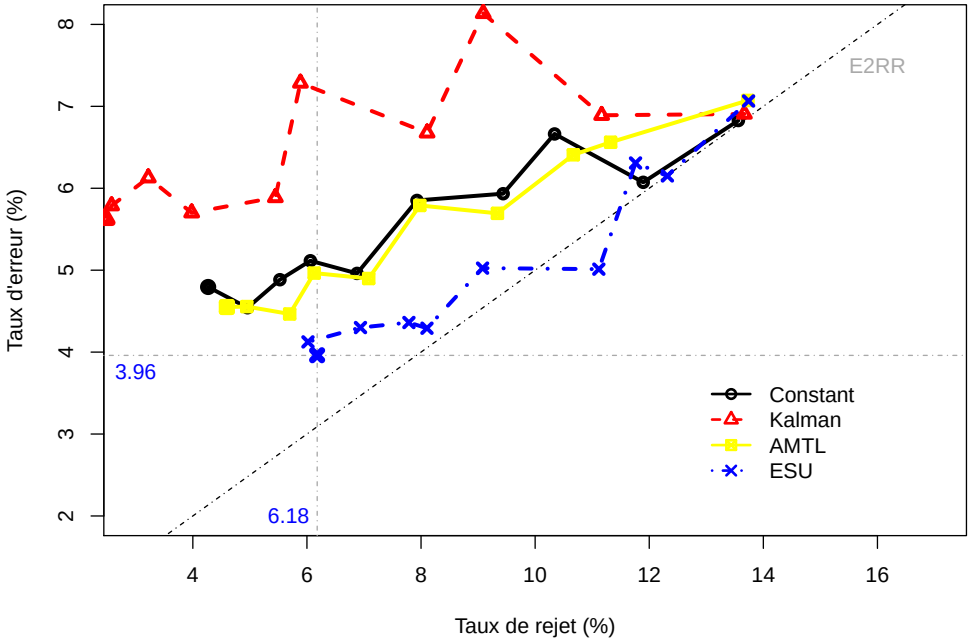


FIGURE 8.3 – Comparaison des trajectoires du point de fonctionnement obtenus avec les différents algorithmes d'adaptation du rejet, par rapport au compromis E2RR souhaité.

ESU et avec les deux algorithmes de l'état de l'art (AMTL et filtrage de Kalman) ainsi qu'un seuil de rejet constant comme référence.

L'objectif est de maintenir le compromis E2RR (1 Error for 2 Rejections Rate) souhaité de deux rejets pour une erreur, qui convient le mieux aux utilisateurs (voir section 2.4.3 page 43). Le compromis E2RR est représenté par la diagonale sur la figure 8.3.

L'algorithme AMTL *(Adaptive Multiple Threshold Learning)* donne des performances similaires au seuil de rejet constant, et n'est pas capable de maintenir deux fois plus de rejets que d'erreurs. L'algorithme basé sur le filtrage de Kalman n'arrive pas à maintenir le compromis erreur/rejet, et s'en éloigne au point de générer beaucoup d'erreurs pour très peu de rejets.

Par contre, l'algorithme ESU *(Evolving Sampling by Uncertainty)* permet de faire évoluer le seuil de sélection de manière à suivre l'évolution du système. La trajectoire de fonctionnement obtenue avec la méthode d'échantillonnage évolutive ESU suit globalement la diagonale visée, et donne un compromis final de 3.96% d'erreurs pour 6.18% de rejets.

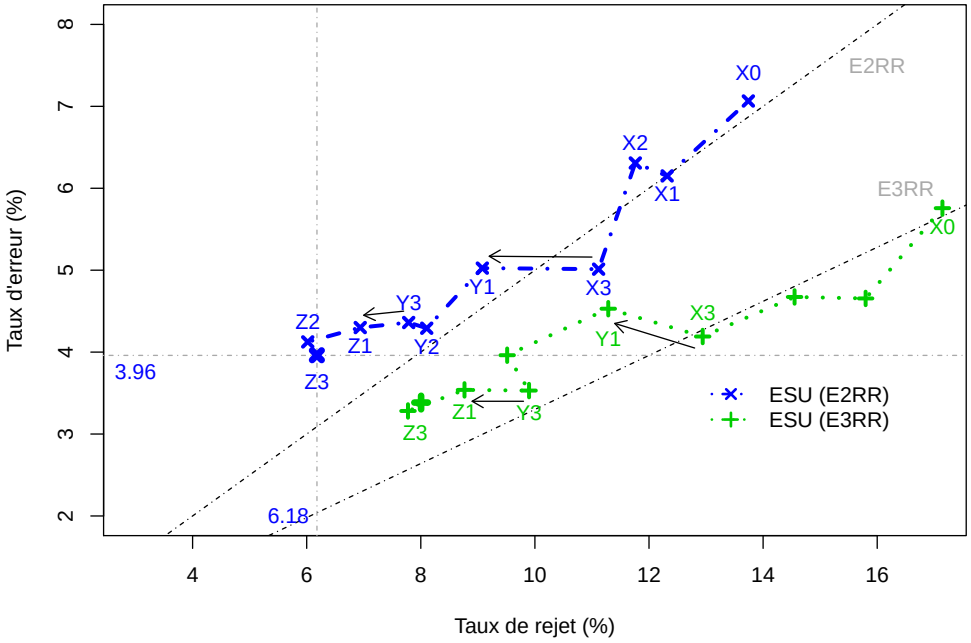


FIGURE 8.4 – Comparaison des trajectoires des points de fonctionnement obtenus avec la méthode d'échantillonnage évolutive ESU, pour différents compromis erreur/rejet : E2RR (an Error for 2 Rejections Rate) et E3RR (an Error for 3 Rejections Rate). Les flèches mettent en évidence les effets des changements de concepts dus aux changements de scripteurs.

8.4.2 Suivi de différents compromis erreur/sollicitation

L'option de rejet évolutive ESU permet de suivre n'importe quel autre compromis souhaité que E2RR *(an Error for 2 Rejection Rate)*. Il est juste nécessaire d'ajuster le point de fonctionnement de départ lors de l'initialisation du système, et l'algorithme ESU maintient ensuite ce compromis erreur/sollicitation constant.

La figure 8.4 montre ainsi la capacité du système à suivre aussi bien le compromis E2RR que le compromis E3RR *(an Error for 3 Rejection Rate)* d'une erreur pour trois rejets.

La figure 8.4 met également en évidence l'effet des changements de concepts dus aux changements de scripteurs (représentés par des flèches). Le taux d'erreur augmente nettement par rapport au taux de reconnaissance après les phases Y1 et Z1, qui sont les premières phases des deuxième et troisième scripteurs.

8.5 Méthode d'échantillonnage avec dopage B-ESU

La méthode d'échantillonnage avec dopage *(boosting)* B-ESU *(Boosted-ESU)* est un raffinement de la méthode ESU. La méthode B-ESU modifie la répartition des sollicitations de l'utilisateur pour les concentrer dans les moments clefs de l'apprentissage.

8.5.1 Amélioration du point de fonctionnement final

La figure 8.5 montre les effets de la méthode de *boosting* de l'apprentissage en-ligne B-ESU *(Boosted-ESU)* sur la trajectoire du point de fonctionnement (obtenue avec la stratégie de supervision *IntuiSup*).

La méthode d'échantillonnage avec dopage *(boosting)* B-ESU augmente le taux de sélection au début de l'utilisation/apprentissage du système, et le réduit ensuite au fur et à mesure que le système apprend. Au final, le point de fonctionnement obtenu en utilisant la méthode d'échantillonnage avec dopage est plus intéressant. Le taux d'erreur est 16% plus faible (de 3.96% à 3.31%) que celui obtenu sans dopage, pour le même taux de rejet (6.33%). Augmenter le nombre de données disponibles pour l'apprentissage du système au début permet d'augmenter la vitesse d'apprentissage du système, et d'obtenir un système plus performant au final.

Par comparaison, la figure 8.5 montre également la trajectoire obtenue sans dopage, mais pour une compromis erreur/rejet différent (E3RR) pour obtenir un taux d'erreur final similaire (3.31%). Augmenter le seuil de rejet (sans dopage) pour obtenir ce taux d'erreur final (3.31%), génère une augmentation du taux de rejet de 27% : de 6.33% à 8.01%.

En résumé, la méthode d'échantillonnage avec dopage B-ESU permet d'améliorer les performances du système, en terme de vitesse d'apprentissage et en terme de compromis erreur/rejet final. La méthode d'échantillonnage avec dopage B-ESU(E2RR) implique une nombre de sollicitations de l'utilisateur intermédiaire entre les deux politiques d'échantillonnage évolutif ESU(E2RR) et ESU(E3RR) qui sont sans dopage. Le point de fonctionnement (compromis erreur/rejet) final obtenu avec dopage est plus intéressant que ceux obtenus par les deux méthodes d'échantillonnage sans dopage. Par rapport à la méthode ESU(E2RR), B-ESU(E2RR) permet une réduction relative de 16.4% du taux de rejet (de 3.96% à 3.31%). Par rapport à la méthode ESU(E3RR), B-ESU(E2RR) permet une réduction relative de 21.0% du taux d'erreur (de 8.01% à 6.33%).

8.5.2 Amélioration du comportement lors des changements de concepts

On peut aussi remarquer sur la figure 8.5 que le taux de rejet augmente très légèrement aux points de fonctionnement pour les phases Y1 et Z1 (correspondant aux changements de scripteurs) lorsque la méthode d'échantillonnage avec dopage est utilisée, alors que le taux de rejet décroit sinon. Ces deux points correspondent en fait aux changements de concepts dus aux changements d'utilisateur. Lorsque l'utilisateur change, le style d'écriture change également, ce qui génère une légère augmentation du taux d'erreurs. La méthode d'échantillonnage avec dopage B-ESU augmente alors le taux de sélection (et donc de sollicitations de l'utilisateur) pour accélérer l'apprentissage et l'adaptation aux changements

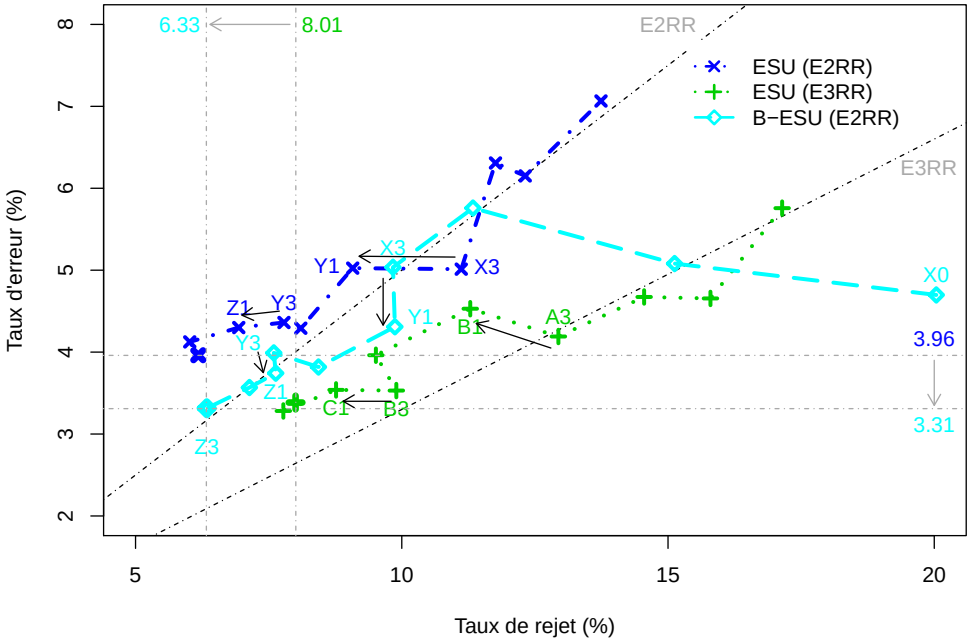


FIGURE 8.5 – Impact de la méthode d'échantillonnage avec dopage *(boosting)* B-ESU de l'apprentissage en-ligne sur la trajectoire du point de fonctionnement erreur/rejet final. Les flèches mettent en évidence les effets des changements de concepts dus aux changements de scripteurs.

de concepts.

Le tableau 8.3 présente le nombre moyen de sollicitations de l'utilisateur sur la base de test entre chaque phase d'utilisation/apprentissage. Le nombre total moyen de sollicitations sur la globalité de l'expérience varie de 31.0 à 34.5 avec la méthode d'échantillonnage avec dopage (11.3% d'augmentation relative). Cependant, cette augmentation est ponctuelle et temporaire. Le nombre de sollicitations est supérieur lors de l'apprentissage initial et des changements de concepts, mais inférieur une fois que le système a appris la nouveauté, et atteint son taux de reconnaissance maximal. Répartir différemment les interactions avec l'utilisateur permet de réduire le taux d'erreur de 16% (de 3.96 à 3.31) à la fin de l'expérience, et donc pendant toute l'utilisation future du système.

Comparé à une proportion de rejets plus importante (compromis E3RR), utiliser la méthode d'échantillonnage avec dopage avec une proportion générale de rejets plus faible (compromis E2RR) permet de réduire le taux d'interaction avec l'utilisateur de 5.5% (de 36.5 à 34.5) pendant l'expérience. Plus important encore, le système obtenu à la fin de l'expérience, et qui continuerait ensuite d'être utilisé dans le cadre d'une utilisation réelle, commet moins d'erreurs et sollicite moins l'utilisateur.

La méthode d'échantillonnage avec dopage *(boosting)* B-ESU permet d'accélérer l'évo-

Test après phase	X0	X1	X2	X3	Y1	Y2	Y3	Z1	Z2	Z3	Total
ESU (E2RR)	4.6	4.1	3.9	3.7	3.0	2.7	**2.6**	**2.3**	2.0	**2.1**	31.0
B-ESU (E2RR)	**6.7**	**5.0**	3.8	3.3	3.3	2.8	**2.5**	**2.6**	2.4	**2.1**	34.5
ESU (E3RR)	**5.7**	**5.3**	4.8	4.3	3.8	3.2	3.3	2.9	2.6	2.7	36.5

TABLE 8.3 – Comparaison du nombre moyen de sollicitations utilisateurs sur les 21 gestes de la base de test (par utilisateur). La méthode B-ESU commence avec un taux de sélection élevée pour accélérer l'apprentissage initial, mais qui diminue rapidement. Le taux de sélection augmente à nouveau lors de la phase Z1 pour accélérer l'évolution suite au changement de scripteur, puis diminue ensuite. Le taux de sélection final est identique à celui sans dopage, mais va continuer de diminuer car le système commet moins d'erreurs que sans dopage.

lution du système aux moments clefs de son apprentissage. La méthode B-ESU augmente le nombre de données sélectionnées pour être étiquetées explicitement par l'utilisateur et être utilisées pour l'apprentissage du classifieur. Ce dopage permet d'accélérer à la fois l'apprentissage initial du système, et son évolution lors des changements de concepts. Cette amélioration plus rapide du système fait qu'il commet ensuite moins d'erreurs et nécessite donc moins d'interactions de supervision pour l'utilisateur.

8.6 Conclusion

Ce chapitre a présenté l'évaluation expérimentale du système *IntuiSup* pour superviser l'apprentissage actif en-ligne d'un classifieur évolutif avec l'utilisateur dans la boucle. Il permet de maximiser l'apprentissage du classifieur tout en minimisant les sollicitations de l'utilisateur. La sélection des données qui sont utilisées pour l'apprentissage est réalisée en-ligne, donnée par donnée. Le superviseur *IntuiSup* utilise la fonction de confiance du classifieur pour sélectionner les données pour lesquelles le classifieur est le plus incertain. Cet échantillonnage, similaire à une option de rejet des données, est évolutif et s'adapte à l'évolution du système pour maintenir le rapport erreurs/sollicitations proche de celui désiré. Le superviseur utilise une méthode d'échantillonnage avec dopage pour accélérer l'apprentissage en augmentant les interactions utilisateur aux moments importants : lors de l'apprentissage initial et lors des changements de concepts.

L'efficacité du superviseur *IntuiSup* a été évalué en utilisant la base de données de commandes gestuelles personnalisées ILGDB. C'est une base de données ordonnée chronologiquement où le style d'écriture des utilisateurs évolue avec le temps. Les symboles des différentes classes ont été choisis par les utilisateurs et la fréquence des classes varie au cours du temps. Cette base de données est très représentative des difficultés rencontrées lors de la reconnaissance de commandes gestuelles, et plus largement, elle représente un défi pour les systèmes d'apprentissage et de reconnaissance.

La stratégie de supervision d'*IntuiSup* permet de superviser de manière active l'apprentissage d'un classifieur évolutif avec l'utilisateur dans la boucle. Elle combine supervision implicite et supervision explicite pour permettre l'apprentissage efficace du classifieur, avec un taux de reconnaissance de 93.48%, en ne sollicitant l'utilisateur que pour 7.64% des données. L'échantillonnage des données qui sont étiquetées explicitement par l'utilisateur est réalisé à l'aide de la fonction de confiance du classifieur pour sélectionner les données pour lesquelles le système est le plus incertain. Cela permet d'apprendre à partir des données qui sont complexes et qui ont le plus de chance d'améliorer le modèle de connaissance du classifieur.

L'option de rejet évolutive ESU permet de faire évoluer la sélection des données utilisées pour l'apprentissage du classifieur. La méthode d'apprentissage actif d'un classifieur évolutif doit évoluer avec le système pour s'adapter à son amélioration avec le temps. L'option de rejet ESU permet de maintenir le compromis erreur/rejet choisi pendant l'apprentissage en-ligne du classifieur comme nous l'avons montré avec les compromis E2RR et E3RR.

La méthode B-ESU d'échantillonnage avec dopage *(boosting)* de l'apprentissage en-ligne permet de répartir les sollicitations de l'utilisateur différemment au cours du temps, en les concentrant aux moments où leur impact est maximum. La méthode d'échantillonnage avec dopage B-ESU permet ainsi d'accélérer la vitesse d'apprentissage initiale, et celle d'évolution lors des changements de concepts. Cet apprentissage plus rapide permet d'obtenir un meilleur taux de reconnaissance en fin d'expérience, et ainsi de réduire le taux de rejet de 15% pour toute l'utilisation future du système.

Ce chapitre a présenté les différents tests que nous avons effectués pour valider expérimentalement le superviseur évolutif *IntuiSup* pour l'apprentissage actif en-ligne d'un

classifieur avec l'utilisateur dans la boucle. Nous avons évalué l'efficacité des différentes contributions (présentées dans le chapitre 6) qui composent le superviseur *IntuiSup* :

— la méthode d'étiquetage implicite ;
— la méthode d'étiquetage explicite ;

avec deux variantes pour l'étiquetage explicite :

— la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* ;
— la méthode d'échantillonnage avec « dopage » B-ESU *(Boosted-ESU)*.

Bien que développé avec le classifieur *Evolve* ∞, le superviseur *IntuiSup* peut très bien fonctionner avec n'importe quel autre classifieur évolutif. La seule nécessité est que le classifieur fournisse une fonction de confiance lors de la reconnaissance.

Le superviseur *IntuiSup* permet notamment l'utilisation de commandes gestuelles personnalisées. C'est une situation d'apprentissage croisé, où l'utilisateur et le système apprennent en parallèle, à partir de leurs interactions mutuelles. Le superviseur *IntuiSup* permet d'étiqueter implicitement la majorité des données, et de sélectionner les quelques données qui seront étiquetées explicitement par l'utilisateur. Ces interactions avec l'utilisateur permettent d'optimiser l'apprentissage du classifieur, sans pour autant réduire la fluidité d'utilisation des commandes gestuelles. L'échantillonnage des données utilisées pour l'apprentissage évolue pour maintenir le taux d'interaction avec l'utilisateur et l'évolution du système. Le superviseur *IntuiSup* permet également d'accélérer l'apprentissage des nouvelles commandes gestuelles en fournissant davantage de données étiquetées pour l'apprentissage du classifieur aux moments opportuns.

Conclusion

Conclusion générale

Synthèse

L'utilisation de commandes gestuelles est une nouvelle méthode d'interaction sur interface tactile. Une bonne méthode pour faciliter la mémorisation de ces commandes gestuelles est de laisser l'utilisateur les personnaliser. Ces travaux sont appliqués à la reconnaissance de commandes gestuelles personnalisées.

Ce contexte applicatif induit une situation d'apprentissage croisé, où l'utilisateur doit mémoriser le jeu de symboles et le système doit apprendre à reconnaitre les différents symboles. Cela implique un certain nombre de contraintes, à la fois sur le système de reconnaissance de symboles et sur le système de supervision de son apprentissage. Il faut par exemple que le classifieur puisse apprendre à partir de peu de données, continuer à apprendre pendant son utilisation et suivre toute évolution des données indéfiniment. Le superviseur doit quant à lui optimiser la coopération entre l'utilisateur et le système de reconnaissance pour minimiser les interactions tout en maximisant l'apprentissage.

Cette thèse a présenté des travaux de recherche orientés selon deux axes : celui des classifieurs évolutifs et celui de leur supervision active en-ligne.

La première problématique explorée dans cette thèse concerne les classifieurs évolutifs sur flux de petite échelle. Ces flux ont peu de données et celles-ci sont susceptibles de changer très rapidement. Il faut alors que le classifieur soit très réactif, capable d'apprendre à partir de très peu de données, y compris de nouvelles classes pendant son utilisation. Il est également nécessaire que le système soit dynamique, pour suivre toute évolution des données, et qu'il reste dynamique indéfiniment.

La deuxième problématique soulevée dans cette thèse concerne la supervision active en-ligne, d'un classifieur évolutif, lorsque l'utilisateur final est dans la boucle d'apprentissage. Dans ce cas, l'étiquetage des données par l'utilisateur a un coût, et il est alors nécessaire de ne sélectionner que les données les plus intéressantes pour l'apprentissage. Il faut optimiser le processus de supervision pour minimiser les sollicitations de l'utilisateur tout en maximisant l'apprentissage du système.

En réponse à cette double problématique, deux contributions principales ont été présentées : le classifieur *Evolve* ∞ et le superviseur *IntuiSup*.

Le classifieur *Evolve* ∞ est un système d'apprentissage et de reconnaissance évolu-

tif, dont les capacités d'apprentissage sont maintenues indéfiniment. C'est une évolution du système d'inférence floue *Evolve*, intégrant notamment de l'oubli dans le processus d'apprentissage pour maintenir le gain de l'apprentissage au cours du temps et suivre les changements de concepts (changements dans la distribution des données).

Son organisation sous forme d'ensemble de règles lui permet de pouvoir incorporer de nouvelles connaissances et de nouvelles classes, sans pour autant détériorer la connaissance existante. Les capacités génératrices des prémisses des règles d'inférence permettent d'apprendre à partir de peu données, et les capacités discriminantes des conclusions permettent d'obtenir de bonnes performances après suffisamment d'exemples d'apprentissage. Le système peut facilement être appris sur un flux de données grâce à un algorithme d'apprentissage de faible complexité.

Evolve ∞ intègre en particulier de l'oubli dans son processus d'apprentissage. Cette capacité d'oubli permet de maintenir le gain de l'apprentissage avec le temps et ainsi de conserver un système qui reste évolutif et réactif indéfiniment. L'oubli est également nécessaire pour que le système puisse suivre et s'adapter aux changements de concepts, en remplaçant la connaissance issue des anciennes données lorsqu'elles deviennent obsolètes.

Evolve ∞ dispose également d'une capacité de rejet des données multiple et évolutive, composée d'une option de rejet de distance et d'une option de rejet de confusion à seuils multiples. L'option de rejet de distance permet de rejeter les données qui sont trop différentes des exemples appris jusque là. L'option de rejet de confusion permet de limiter les erreurs commises par le système lorsque des données ont de fortes probabilités d'appartenir à plusieurs classes. Cette option de rejet dispose d'une architecture avancée, avec des seuils de rejet multiples pour de meilleures performances, et un algorithme d'apprentissage flou pour faire évoluer les différents seuils en accord avec l'évolution du système. *Evolve* ∞ dispose également d'une mesure de confiance interne, qui permet d'évaluer la qualité et les limites du modèle de connaissance du système à n'importe quel moment de son apprentissage. Cette mesure est en particulier utilisée par *IntuiSup* pour superviser l'apprentissage avec l'utilisateur dans la boucle.

Nous avons validé expérimentalement l'efficacité de la capacité d'oubli d'*Evolve* ∞ sur la base de données de commandes gestuelles personnalisées ILGDB, et sur plusieurs bases de données de référence (Ironoff-digits, LaViola, NicIcon, Pen-Digits et Japanese-Vowels). L'oubli permet de limiter l'inertie du système et de maintenir sa capacité d'apprentissage de nouvelles classes avec le temps. L'algorithme d'oubli directionnel différé (DDF) que nous avons présenté permet notamment de garantir le maintien de la capacité d'apprentissage du système sans compromettre sa stabilité.

Le superviseur *IntuiSup* est un système de supervision actif en-ligne pour l'apprentissage d'un classifieur évolutif lorsque l'utilisateur est dans la boucle. Ce superviseur associe plusieurs méthodes de sélection des données à étiqueter. Cette sélection est composée de deux parties : la méthode d'étiquetage implicite et la méthode d'étiquetage explicite. Nous avons présenté deux variantes de cette dernière, la méthode d'échantillonnage évolutive ESU *(Evolving Sampling by Uncertainty)* et la méthode d'échantillonnage avec « dopage » B-ESU *(Boosted-ESU)*.

La stratégie de supervision d'*IntuiSup* permet de superviser l'apprentissage de commandes gestuelles personnalisées pendant leur utilisation. Elle étiquette la majorité des données implicitement, et sollicite l'utilisateur pour obtenir l'étiquette de certaines données soigneusement sélectionnées. Cet échantillonnage est réalisé en utilisant la mesure de confiance du classifieur pour sélectionner les exemples les plus intéressants pour l'apprentissage.

La méthode d'échantillonnage évolutive ESU permet de maintenir le compromis erreur/rejet désiré au cours du temps. Nos travaux utilisent principalement le compromis E2RR *(an Error for 2 Rejection Rate)*, mais n'importe quel autre compromis peut être choisi et utilisé par l'algorithme ESU. La méthode d'échantillonnage est ainsi adaptée pour évoluer en accord avec le classifieur évolutif et maintenir le compromis erreur/rejet souhaité.

La méthode de dopage *(boosting)* de l'apprentissage B-ESU permet d'accélérer la vitesse d'apprentissage du classifieur. Cette méthode augmente automatiquement le taux de sélection des données au début de l'utilisation du système, et lors des changements de concepts, pour augmenter la quantité de données étiquetées et accélérer l'apprentissage. Cette amélioration plus rapide des performances du classifieur permet ensuite de réduire le taux de rejet à une valeur plus faible, car le classifieur commet moins d'erreurs, et de limiter les interactions utilisateur. La méthode de dopage B-ESU permet de modifier la répartition des sollicitations utilisateur au cours du temps pour les concentrer pendant les périodes d'apprentissage.

Ce superviseur actif en-ligne est générique, il peut être utilisé pour entrainer n'importe quel classifieur évolutif. La seule contrainte est que le classifieur utilisé fournisse une mesure de confiance lors du processus de reconnaissance.

Les performances du superviseur *IntuiSup* ont été évaluées sur la base de données de commandes gestuelles personnalisée ILGDB. L'efficacité de la stratégie de supervision d'*IntuiSup* a été validée expérimentalement. La méthode d'échantillonnage évolutive ESU a montré sa capacité à suivre différents compromis erreur/sollicitation. La méthode d'échantillonnage avec dopage B-ESU a permis d'améliorer davantage les performances, en concentrant les interactions aux moments clefs de l'apprentissage.

Perspectives

Ces travaux ouvrent deux axes de perspectives principaux que nous détaillons maintenant, pour le classifieur *Evolve* ∞ tout d'abord, et pour le superviseur *IntuiSup* ensuite.

Le premier axe de perspectives rassemble les pistes qui pourraient être explorées pour améliorer le classifieur *Evolve* ∞, notamment du coté de la gestion de l'ensemble de règles d'inférence.

Il n'y a actuellement pas d'algorithme de gestion de l'ensemble de règles dans *Evolve* ∞. Les règles sont créées par l'algorithme de regroupement incrémental, mais aucun algorithme ne remet en cause l'existence des règles une fois qu'elles ont été créées. Cette gestion n'est pas indispensable car l'incorporation d'oubli dans le processus d'apprentissage permet la

transformation et la réutilisation des règles qui sont obsolètes. Il serait cependant envisageable de supprimer totalement les anciennes règles qui ne sont plus utilisées, lorsque les commandes correspondantes sont supprimées par exemple, ou lorsqu'un changement de concept a déclenché la création d'une nouvelle règle pour cette classe. Il faut cependant s'assurer que les conclusions des règles à supprimer ne sont pas trop impliquées dans la reconnaissance d'autres classes avant de les enlever. De même, lorsque les prototypes de deux règles correspondant à la même classe se rapprochent, il serait envisageable de les fusionner si les conclusions sont suffisamment similaires.

La création de plusieurs règles par classe est fondamentale pour permettre de bonnes performances de reconnaissance des classes regroupant plusieurs symboles différents sous la même étiquette. Par contre, la création de trop nombreuses règles complexifie inutilement le modèle de connaissance du classifieur, et peut ralentir son apprentissage. L'algorithme de création de règles pourrait également être amélioré, notamment en intégrant une mémoire à court terme. Cette mémoire permettrait de réduire la sensibilité au bruit en ne créant de nouvelles règles que lorsque le besoin est confirmé par plusieurs données. La création de nouvelles règles n'est pas déclenchée lors de l'arrivée de données aberrantes isolées. Différents algorithmes de l'état de l'art pourraient être utilisés, comme RepStream [Lühr and Lazarescu, 2008] par exemple. Il reste cependant nécessaire d'évaluer leur efficacité dans le cadre d'un système d'inférence floue, et avec différents jeux de caractéristiques de différentes dimensions.

Enfin, une autre piste de recherche pour améliorer les performances de reconnaissance d'*Evolve* ∞ est d'explorer d'autres algorithmes pour l'optimisation des paramètres des fonctions linéaires des conclusions. Il pourrait par exemple être envisageable de se servir de l'algorithme d'optimisation des vastes marges pour apprendre les conclusions [Juang et al., 2007] [Cheng and Juang, 2011].

Le second axe de perspectives concerne le système de supervision *IntuiSup*, par exemple pour essayer de tirer davantage parti du classifieur utilisé.

Le superviseur *IntuiSup* utilise actuellement la fonction de confiance fournie par le classifieur, qui permet une supervision efficace dès le début de l'apprentissage. Cependant, après une certaine période d'apprentissage du classifieur *Evolve* ∞, la capacité de rejet multiple devient plus précise que la fonction de confiance interne. Il serait donc possible d'étudier l'utilisation de l'option de rejet multiple à la place de la fonction de confiance interne. Cette option de rejet est évolutive et s'adapte à l'apprentissage du système. Elle maintient donc le compromis erreur/rejet constant et pourrait être utilisée pour la supervision de l'apprentissage. Par contre, elle ne s'adapte pas à l'évolution de l'environnement et ne permet pas d'accélérer l'apprentissage initial ni l'adaptation aux changements de concepts. Il faudrait alors créer un algorithme de dopage *(boosting)* de l'apprentissage avec cette option de rejet avancée.

Le superviseur *IntuiSup* a été testé sur la base de données de commandes gestuelles personnalisée ILGDB. Cette base contient des symboles très variés, puisqu'ils ont été choisis par les utilisateurs eux-même, et permet de voir l'évolution du style d'écriture des utilisateur avec le temps. Elle a été récoltée dans un environnement immersif où les utilisateurs

utilisaient une application de manipulation de photos pendant plus de 45 minutes. Nous avons également simulé une utilisation plus longue et des changements de concepts pour valider le système à plus long terme.

Il pourrait également être intéressant de le tester en situation réelle avec l'utilisateur dans la boucle. Il est en effet impossible de modéliser précisément les effets que peuvent avoir les erreurs et les sollicitations du système sur l'utilisateur. Celui-ci peut en effet plus ou moins déformer ses gestes, ou au contraire s'appliquer davantage, en fonction du comportement et des réponses du système. Cette situation d'apprentissage croisé ne peut pas être simulée, il est nécessaire d'avoir un utilisateur réel dans la boucle.

Une campagne de tests à long terme permettrait également de mettre en avant l'intérêt de la capacité d'oubli d'*Evolve* ∞ pour le suivi des changements de concepts légers lorsque ceux-ci ont une progression logique. Il faudrait pour cela effectuer une campagne sur plusieurs mois, avec des utilisateurs novices et dans un contexte applicatif réel, ce qui est complexe et coûteux à mettre en place.

En dehors des deux principaux axes de perspectives précédemment évoqués, ces travaux ouvrent plusieurs champs d'exploration intéressants. Des capacités d'évolution supplémentaires pourraient être étudiées en explorant la sélection dynamique de caractéristiques. Ensuite, il serait possible d'envisager l'utilisation du classifieur *Evolve* ∞, voire même du superviseur *IntuiSup*, pour l'apprentissage en-ligne sur flux de grande échelle, même si les contraintes sont différentes. Enfin, nous avons étudié ici les effets de l'évolution de l'utilisateur sur le système, mais il serait pertinent d'étudier aussi les effets de l'évolution du système sur l'utilisateur.

Il serait également intéressant d'explorer les mécanismes de sélection dynamique de caractéristiques, afin de pouvoir faire évoluer l'espace de représentation au cours du temps. Cela permettrait de choisir automatiquement les caractéristiques les plus adaptées pour différencier les différentes classes dans les cas très atypiques. De nouvelles caractéristiques pourraient ainsi être ajoutées lors de l'ajout de nouvelles classes pour améliorer leur modélisation et leur discrimination. Un classifieur évolutif, couplé avec un espace de représentation également évolutif, permettrait ainsi d'obtenir un système encore plus flexible et polyvalent.

Il serait envisageable d'utiliser le classifieur *Evolve* ∞, et pourquoi pas le superviseur *IntuiSup*, pour l'apprentissage sur flux de grande échelle.

L'algorithme d'apprentissage du classifieur évolutif *Evolve* ∞ est peu complexe, ce qui fait qu'il est également utilisable sur des flux rapides et de grandes dimensions. Comme il donne de bonnes performances avec peu de données, il devrait a priori donner également de bon résultats avec beaucoup de données. Le point critique sera l'adaptation de la capacité d'oubli qui utilise actuellement une fenêtre de données.

Le superviseur actif en-ligne *IntuiSup* est conçue pour superviser l'apprentissage d'un système lorsque l'étiquetage des données est coûteux, ce qui est également le cas des flux de grande échelle. Ses capacités d'évolution lui permettent de s'adapter à l'évolution du sys-

tème qu'il supervise et à son environnement, ce qui laisse présager de bonnes performances sur des flux de grandes échelles.

Nous avons étudié dans ces travaux les effets de l'évolution de l'utilisateur sur le système de reconnaissance, mais symétriquement, l'évolution du système a un effet sur le comportement de l'utilisateur.

Le fait que l'utilisateur interagisse plus ou moins avec l'utilisateur va influer sur la perception des difficultés du classifieur par l'utilisateur. Il est alors possible que l'utilisateur adapte son style d'écriture en fonction des difficultés du classifieur, en s'appliquant davantage lors du tracé des symboles complexes par exemple.

Les interactions avec l'utilisateur ont également un effet sur sa concentration et son attention envers le système. Il est en effet connu que lorsqu'un système n'interagit pas assez avec l'opérateur contrôlant son fonctionnement, la vigilance de son opérateur baisse. C'est le syndrome de « l'utilisateur hors de la boucle » *(out-of-the-loop problem)* [Kaber and Endsley, 1997]. L'utilisateur laisse alors passer davantage d'erreurs sans corriger le système « à-la-volée ». Il annule ensuite les commandes erronées, ce qui réduit la quantité de données qui est étiquetée implicitement. Il pourrait ainsi être intéressant de toujours conserver un taux d'interaction minimum avec l'utilisateur.

Ces travaux permettent l'utilisation de nouvelles méthodes d'interaction évolutives, comme les commandes gestuelles personnalisées sur interface tactile. C'est maintenant la Machine qui s'adapte à l'Humain, pour permettre des interactions plus intuitives et plus naturelles. Avec le développement des caméras en relief et des capteurs de profondeur, de nouvelles interfaces gestuelles en trois dimensions pourront voir le jour en suivant la même philosophie.

Les technologies de l'information continuent de se développer, et de plus en plus de données différentes sont disponibles, et avec elles, autant de nouvelles applications potentielles. Ces applications pourront tirer profit des systèmes d'apprentissage automatiques enligne toujours plus performants, pour s'adapter à leurs utilisateurs. Les interfaces Homme-Machine se dirigent aujoud'hui vers des interactions évolutives, qu'elles soient gestuelles, vocales ou écrites, et convergent vers des procédés d'interaction qui pourraient enfin être qualifiés d'intuitifs. De nouvelles interfaces vont se développer dont l'idée même n'est pas encore concevable. Et pourtant, l'écriture manuscrite est toujours un moyen de communication omniprésent, et le restera probablement encore longtemps.

Bibliographie

Publications de l'auteur

[Boui 12] M. Bouillon, E. Anquetil, and A. Almaksour. "Decremental Learning of Evolving Fuzzy Inference Systems Using a Sliding Window". In : *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA)*, pp. 598–601, Boca Raton, USA, 2012.

[Boui 13a] M. Bouillon, E. Anquetil, and A. Almaksour. "Decremental Learning of Evolving Fuzzy Inference Systems, Application to Handwritten Gesture Recognition". In : *Proceedings of the 9th International Conference on Machine Learning and Data Mining (MLDM)*, pp. 115–129, New-York, USA, 2013.

[Boui 13b] M. Bouillon, E. Anquetil, and A. Almaksour. "Étude des techniques d'oubli dans les moindres carrés récursifs pour l'apprentissage incrémental de systèmes d'inférence floue évolutifs : application à la reconnaissance de formes". In : *Actes de la 13e Conférence Francophone sur l'Extraction et la Gestion des Connaissances*, pp. 15–24, Toulouse, France, 2013.

[Boui 13c] M. Bouillon, P. Li, E. Anquetil, and G. Richard. "Using Confusion Reject to Improve (User and) System (Cross) Learning of Gesture Commands". In : *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1017–1021, Washington DC, USA, 2013.

[Boui 14a] M. Bouillon and E. Anquetil. "Man-Machine Cooperation for the On-Line Training of an Evolving Classifier". In : *Proceedings of the IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pp. 1–7, Linz, Austria, 2014.

[Boui 14b] M. Bouillon and E. Anquetil. "Optimisation de la coopération utilisateur/système pour l'apprentissage en-ligne d'un classifieur évolutif". In : *Actes du dix-neuvième congrès national sur la Reconnaissance de Formes et l'Intelligence Artificielle (RFIA)*, Rouen, France, 2014.

[Boui 14c] M. Bouillon and E. Anquetil. "Stratégies de supervision pour l'apprentissage en-ligne d'un classifieur évolutif de commande gestuelles". In : *Actes du Colloque International Francophone sur l'Écrit et le Document (CIFED)*, pp. 293–308, Nancy, France, 2014.

[Boui 14d] M. Bouillon and E. Anquetil. "Supervision Strategies for the Online Learning of an Evolving Classifier for Gesture Commands". In : *Proceedings of the*

22nd International Conference on Pattern Recognition (ICPR), pp. 2029–2034, Stockholm, Sweden, 2014.

[Boui 14e] M. Bouillon, E. Anquetil, P. Li, and G. Richard. "User Interaction Optimization for an Evolving Classifier of Handwritten Gesture Commands". In : *Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pp. 720–725, 2014.

[Boui 15] M. Bouillon and E. Anquetil. "Handwriting Analysis with Online Fuzzy Models". In : *Proceedings of the 17th Conference of the International Graphonomics Society (IGS)*, pp. 71–74, Pointe-à-Pitre, Guadeloupe, 2015. Best student paper award.

[Li 13] P. Li, M. Bouillon, E. Anquetil, and G. Richard. "User and System Cross-Learning of Gesture Commands on Pen-Based Devices". In : *Proceeding of the 14th International Conference on Human-Computer Interaction (INTERACT)*, pp. 337–355, 2013.

Références de l'état de l'art

[Abe and Mamitsuka, 1998] Abe, N. and Mamitsuka, H. (1998). Query learning strategies using boosting and bagging. In *Machine Learning : Proceedings of the Fifteenth International Conference (ICML'98)*, page 1. Morgan Kaufmann Pub.

[Aggarwal et al., 2004] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2004). On demand classification of data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–508. ACM.

[Aggarwal et al., 2006] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2006). A framework for on-demand classification of evolving data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 18(5) :577–589.

[Aha et al., 1991] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning*, 6(1) :37–66.

[Ahn et al., 2009] Ahn, J.-H., Lee, J., Jo, J., Choi, Y.-H., and Lee, Y. (2009). Online Character Recognition Using Elastic Curvature Matching. In *Seventh International Conference on Advances in Pattern Recognition (ICAPR)*, pages 395–397.

[Al-Naymat et al., 2012] Al-Naymat, G., Chawla, S., and Taheri, J. (2012). SparseDTW : A Novel Approach to Speed up Dynamic Time Warping. *arXiv :1201.2969 [cs]*. arXiv : 1201.2969.

[Almaksour and Anquetil, 2011] Almaksour, A. and Anquetil, E. (2011). Improving premise structure in evolving Takagi-Sugeno neuro-fuzzy classifiers. *Evolving Systems*, 2(1) :25–33.

[Almaksour and Anquetil, 2013] Almaksour, A. and Anquetil, E. (2013). ILClass : Error-driven antecedent learning for evolving Takagi-Sugeno classification systems. *Applied Soft Computing*.

[Almaksour et al., 2011] Almaksour, A., Anquetil, E., Plamondon, R., and O'Reilly, C. (2011). Synthetic handwritten gesture generation using sigma-lognormal model for evolving handwriting classifiers. In *15th Biennial Conference of the International Graphonomics Society*.

[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 Lughofer, 2008] Angelov, P. and Lughofer, E. (2008). Data-driven evolving fuzzy systems using eTS and FLEXFIS : comparative analysis. *International Journal of General Systems*, 37(1) :45–67.

[Angelov et al., 2008] Angelov, P., Lughofer, E., and Zhou, X. (2008). Evolving fuzzy classifiers using different model architectures. *Fuzzy Sets and Systems*, 159(23) :3160–3182.

[Angelov and Zhou, 2008] Angelov, P. and Zhou, X. (2008). Evolving Fuzzy-Rule-Based Classifiers From Data Streams. *IEEE Transactions on Fuzzy Systems*, 16(6) :1462–1475.

[Beringer and Hullermeier, 2007] Beringer, J. and Hullermeier, E. (2007). Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6) :627–650.

[Bifet and Gavalda, 2007] Bifet, A. and Gavalda, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. In *SDM*, volume 7, page 2007. SIAM.

[Bifet et al., 2010] Bifet, A., Holmes, G., and Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In *Machine learning and knowledge discovery in databases*, pages 135–150. Springer.

[Bifet et al., 2009] Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., and Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148. ACM.

[Bishop, 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[Bittanti et al., 1990] Bittanti, S., Bolzern, P., and Campi, M. (1990). Convergence and exponential convergence of identification algorithms with directional forgetting factor. *Automatica*, 26(5) :929–932.

[Bordes and Bottou, 2005] Bordes, A. and Bottou, L. (2005). The Huller : A Simple and Efficient Online SVM. In Gama, J., Camacho, R., Brazdil, P. B., Jorge, A. M., and Torgo, L., editors, *Machine Learning : ECML 2005*, number 3720 in Lecture Notes in Computer Science, pages 505–512. Springer Berlin Heidelberg.

[Bordes et al., 2005] Bordes, A., Ertekin, S., Weston, J., and Bottou, L. (2005). Fast Kernel Classifiers with Online and Active Learning. *Journal of Machine Learning Research*, 6 :1579–1619.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

[Bouchachia, 2011] Bouchachia, A. (2011). Incremental learning with multi-level adaptation. *Neurocomputing*, 74(11) :1785–1799.

[Bouchachia et al., 2010] Bouchachia, A., Prossegger, M., and Duman, H. (2010). Semi-supervised incremental learning. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–6. IEEE.

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

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

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

[Cao and Schwartz, 2000] Cao, L. and Schwartz, H. (2000). A directional forgetting algorithm based on the decomposition of the information matrix. *Automatica*, 36(11) :1725–1731.

[Carlson et al., 2010] Carlson, A., Betteridge, J., Wang, R. C., Hruschka Jr, E. R., and Mitchell, T. M. (2010). Coupled semi-supervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110. ACM.

[Carpenter and Grossberg, 1988] Carpenter, G. and Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3) :77–88.

[Carpenter et al., 1992] Carpenter, G., Grossberg, S., Markuzon, N., Reynolds, J. H., Rosen, A. B., and others (1992). Fuzzy ARTMAP : A neural network architecture for incremental supervised learning of analog multidimensional maps. *Neural Networks, IEEE Transactions on*, 3(5) :698–713.

[Carpenter et al., 1991a] Carpenter, G., Grossberg, S., Rosen, D., and others (1991a). ART 2-A : An adaptive resonance algorithm for rapid category learning and recognition. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 151–156. IEEE.

[Carpenter et al., 1991b] Carpenter, G. A., Grossberg, S., and Reynolds, J. H. (1991b). ARTMAP : Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural networks*, 4(5) :565–588.

[Carpenter et al., 1991c] Carpenter, G. A., Grossberg, S., and Rosen, D. B. (1991c). Fuzzy ART : Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural networks*, 4(6) :759–771.

[Cauwenberghs and Poggio, 2001] Cauwenberghs, G. and Poggio, T. (2001). Incremental and Decremental Support Vector Machine Learning. volume 13, pages 409–415.

[Chakraborty and Nagwani, 2011] Chakraborty, S. and Nagwani, N. K. (2011). Analysis and study of incremental k-means clustering algorithm. In *High Performance Architecture and Grid Computing*, pages 338–341. Springer.

[Chen et al., 2014] Chen, Z., Anquetil, E., Mouchère, H., and Viard-Gaudin, C. (2014). A graph modeling strategy for multi-touch gesture recognition.

[Chen et al., 2015] Chen, Z., Anquetil, E., Mouchère, H., and Viard-Gaudin, C. (2015). Recognize multi-touch gestures by graph modeling and matching.

[Cheng and Juang, 2011] Cheng, W.-Y. and Juang, C.-F. (2011). An incremental support vector machine-trained TS-type fuzzy system for online classification problems. *Fuzzy Sets and Systems*, 163(1) :24–44.

[Chiu, 1994] Chiu, S. L. (1994). Fuzzy model identification based on cluster estimation. *Journal of intelligent and Fuzzy systems*, 2(3) :267–278.

[Choo et al., 2007] Choo, J., Jiamthapthaksin, R., Chen, C.-s., Celepcikay, O. U., Giusti, C., and Eick, C. F. (2007). MOSAIC : A proximity graph approach for agglomerative clustering. In *Data Warehousing and Knowledge Discovery*, pages 231–240. Springer.

[Chow, 1970] Chow, C. (1970). On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1) :41 – 46.

[Choy et al., 2006] Choy, M. C., Srinivasan, D., and Cheu, R. (2006). Neural Networks for Continuous Online Learning and Control. *IEEE Transactions on Neural Networks*, 17(6) :1511–1531.

[Chu and Zaniolo, 2004] Chu, F. and Zaniolo, C. (2004). Fast and light boosting for adaptive mining of data streams. In *Advances in Knowledge Discovery and Data Mining*, pages 282–292. Springer.

[Cohn, 1994] Cohn, D. A. (1994). Neural network exploration using optimal experiment design.

[Cornuéjols and Miclet, 2010] Cornuéjols, A. and Miclet, L. (2010). Apprentissage artificiel, concepts et algorithms, Eyrolles. Technical report, ISBN 2-212-11020-0.

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

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1) :21–27.

[Cox, 1958] Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242.

[Crawford, 1989] Crawford, S. L. (1989). Extensions to the CART algorithm. *International Journal of Man-Machine Studies*, 31(2) :197–217.

[Dagan and Engelson, 1995] Dagan, I. and Engelson, S. P. (1995). Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157.

[Deckert and Stefanowski, 2014] Deckert, M. and Stefanowski, J. (2014). RILL : Algorithm for Learning Rules from Streaming Data with Concept Drift. In *Foundations of Intelligent Systems*, pages 20–29. Springer.

[del Campo-Avila et al., 2006] del Campo-Avila, J., Ramos-Jiménez, G., Gama, J., and Morales-Bueno, R. (2006). Improving prediction accuracy of an incremental algorithm driven by error margins. *Knowledge Discovery from Data Streams*, 57.

[Delaye and Anquetil, 2013] Delaye, A. and Anquetil, E. (2013). HBF49 feature set : A first unified baseline for online symbol recognition. *Pattern Recognition*, 46(1) :117–130.

[Deng and Kasabov, 2003] Deng, D. and Kasabov, N. (2003). On-line pattern analysis by evolving self-organizing maps. *Neurocomputing*, 51 :87–103.

[Domeniconi and Gunopulos, 2001] Domeniconi, C. and Gunopulos, D. (2001). Incremental support vector machine construction. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 589–592. IEEE.

[Domingos and Hulten, 2000] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM.

[Domingos and Hulten, 2001] Domingos, P. and Hulten, G. (2001). Catching up with the Data : Research Issues in Mining Data Streams. In *DMKD*.

[Domingos and Pazzani, 1997] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, 29(2-3) :103–130.

[Dong et al., 2005] Dong, J.-x., Krzyżak, A., and Suen, C. Y. (2005). Fast SVM training algorithm with decomposition on very large data sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(4) :603–618.

[Dubuisson and Masson, 1993] Dubuisson, B. and Masson, M. (1993). A statistical decision rule with incomplete knowledge about classes. *Pattern recognition*, 26(1) :155–165.

[Elwell and Polikar, 2011] Elwell, R. and Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *Neural Networks, IEEE Transactions on*, 22(10) :1517–1531.

[Fan et al., 2008] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR : A Library for Large Linear Classification. *J. Mach. Learn. Res.*, 9 :1871–1874.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (1996). Advances in Knowledge Discovery and Data Mining.

[Feng et al., 2010] Feng, J., Zhang, C., and Hao, P. (2010). Online learning with self-organizing maps for anomaly detection in crowd scenes. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3599–3602. IEEE.

[Fern and Givan, 2003] Fern, A. and Givan, R. (2003). Online ensemble learning : An empirical study. *Machine Learning*, 53(1-2) :71–109.

[Ferrer-Troyano et al., 2006] Ferrer-Troyano, F., Aguilar-Ruiz, J. S., and Riquelme, J. C. (2006). Data streams classification by incremental rule learning with parameterized generalization. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 657–661. ACM.

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

[Frank and Asuncion, 2010] Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

[French, 1999] French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4) :128–135.

[Freund et al., 1996] Freund, Y., Schapire, R. E., and others (1996). Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156.

[Frélicot et al., 1995] Frélicot, C., Masson, M. H., and Dubuisson, B. (1995). Reject options in fuzzy pattern classification rules. In *Proc. of 3rd European Congress on Intelligent Techniques and Soft Computing*, volume 75. Citeseer.

[Fujii et al., 1998] Fujii, A., Tokunaga, T., Inui, K., and Tanaka, H. (1998). Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4) :573–597.

[Fumera and Roli, 2002] Fumera, G. and Roli, F. (2002). Support vector machines with embedded reject option. In *Pattern Recognition with Support Vector Machines*, pages 68–82. Springer.

[Fumera et al., 2000] Fumera, G., Roli, F., and Giacinto, G. (2000). Reject option with multiple thresholds. *Pattern Recognition*, 33(12) :2099–2101.

[Furao et al., 2007] Furao, S., Ogura, T., and Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, 20(8) :893–903.

[Gabrys and Bargiela, 2000] Gabrys, B. and Bargiela, A. (2000). General fuzzy min-max neural network for clustering and classification. *Neural Networks, IEEE Transactions on*, 11(3) :769 –783.

[Gama, 2012] Gama, J. (2012). A survey on learning from data streams : current and future trends. *Progress in Artificial Intelligence*, 1(1) :45–55.

[Gama et al., 2011] Gama, J., Kosina, P., and others (2011). Learning decision rules from data streams. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1255. Citeseer.

[Gama et al., 2005] Gama, J., Medas, P., and Rodrigues, P. (2005). Learning decision trees from dynamic data streams. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 573–577. ACM.

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

[Gama et al., 2009] Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338. ACM.

[Gama et al., 2014] Gama, J., Žliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4) :44.

[Gent et al., 2012] Gent, I. P., Miguel, I., and Moore, N. C. (2012). An empirical study of learning and forgetting constraints. *AI Communications*, 25(2) :191–208.

[Globerson and Roweis, 2005] Globerson, A. and Roweis, S. T. (2005). Metric learning by collapsing classes. In *Advances in neural information processing systems*, pages 451–458.

[Golubitsky and Watt, 2010] Golubitsky, O. and Watt, S. M. (2010). Distance-based classification of handwritten symbols. *International Journal on Document Analysis and Recognition (IJDAR)*, 13(2) :133–146.

[Gorski, 1997] Gorski, N. (1997). Optimizing error-reject trade off in recognition systems. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 1092–1096. IEEE.

[Grira et al., 2005] Grira, N., Crucianu, M., and Boujemaa, N. (2005). Active semi-supervised fuzzy clustering for image database categorization. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 9–16. ACM.

[Guo and Greiner, 2007] Guo, Y. and Greiner, R. (2007). Optimistic Active-Learning Using Mutual Information. In *IJCAI*, volume 7, pages 823–829.

[Hand and Yu, 2001] Hand, D. J. and Yu, K. (2001). Idiot's Bayes—Not So Stupid After All ? *International Statistical Review*, 69(3) :385–398.

[Hansen et al., 1997] Hansen, L. K., Liisberg, C., and Salamon, P. (1997). The error-reject tradeoff. *Open Systems & Information Dynamics*, 4(2) :159–184.

[Hashemi and Yang, 2009] Hashemi, S. and Yang, Y. (2009). Flexible decision tree for data stream classification in the presence of concept change, noise and missing values. *Data Mining and Knowledge Discovery*, 19(1) :95–131.

[Haykin, 2001] Haykin, S. O. (2001). *Adaptive Filter Theory (4th Edition)*. Prentice Hall, 4 edition.

[Ho et al., 2010] Ho, W. L., Tung, W. L., and Quek, C. (2010). An evolving Mamdani-Takagi-Sugeno based neural-fuzzy inference system with improved interpretability-accuracy. In *2010 IEEE International Conference on Fuzzy Systems (FUZZ)*, pages 1–8.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366.

[Hse and others, 2004] Hse, H. and others (2004). Sketched symbol recognition using zernike moments. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 1, pages 367–370. IEEE.

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

[Hyafil and Rivest, 1976] Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1) :15–17.

[Hägglund, 1983] Hägglund, T. (1983). *New estimation techniques for adaptive control*. Institutionen för reglerteknik, Lunds tekniska högskola. Lund University.

[Hägglund, 1985] Hägglund, T. (1985). Recursive Estimation of Slowly Time Varying Parameters. pages 1137–1142, York, UK.

[Iglesias et al., 2013] Iglesias, J., Ledezma, A., and Sanchis, A. (2013). Ensemble method based on individual evolving classifiers. In *2013 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 56–61.

[Jackowski, 2013] Jackowski, K. (2013). Fixed-size ensemble classifier system evolutionarily adapted to a recurring context with an unlimited pool of classifiers. *Pattern Analysis and Applications*, 17(4) :709–724.

[Jang and Sun, 1993] Jang, J.-S. and Sun, C.-T. (1993). Functional equivalence between radial basis function networks and fuzzy inference systems. *Neural Networks, IEEE Transactions on*, 4(1) :156–159.

[John and Langley, 1995] John, G. H. and Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Juang et al., 2007] Juang, C.-F., Chiu, S.-H., and Chang, S.-W. (2007). A Self-Organizing TS-Type Fuzzy Network With Support Vector Learning and its Application to Classification Problems. *IEEE Transactions on Fuzzy Systems*, 15(5) :998–1008.

[Kaber and Endsley, 1997] Kaber, D. B. and Endsley, M. R. (1997). Out-of-the-loop performance problems and the use of intermediate levels of automation for improved control system functioning and safety. *Process Safety Progress*, 16(3) :126–131.

[Kalman, 1960] Kalman, R. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering*, (82 (Series D)) :35–45.

[Karypis et al., 1999] Karypis, G., Han, E.-H., and Kumar, V. (1999). Chameleon : Hierarchical clustering using dynamic modeling. *Computer*, 32(8) :68–75.

[Kasabov, 1998] Kasabov, N. K. (1998). Evolving fuzzy neural networks : theory and applications for on-line adaptive prediction, decision making and control. *Aust. J. Intell. Inf. Process. Syst.(Australia)*, 5(3) :154–160.

[Kasabov and Song, 2002] Kasabov, N. K. and Song, Q. (2002). DENFIS : dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *Fuzzy Systems, IEEE Transactions on*, 10(2) :144–154.

[Klinkenberg and Joachims, 2000] Klinkenberg, R. and Joachims, T. (2000). Detecting Concept Drift with Support Vector Machines. In *ICML*, pages 487–494. Citeseer.

[Kohonen, 1990] Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9) :1464–1480.

[Kohonen, 1997] Kohonen, T. (1997). *Learning vector quantization.* Springer.

[Kolter and Maloof, 2003] Kolter, J. Z. and Maloof, M. (2003). Dynamic weighted majority : A new ensemble method for tracking concept drift. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 123–130. IEEE.

[Kosina and Gama, 2012] Kosina, P. and Gama, J. (2012). Handling time changing data with adaptive very fast decision rules. In *Proceedings of the 2012 European conference on Machine Learning and Knowledge Discovery in Databases*, pages 827–842. Springer-Verlag.

[Kosko, 1994] Kosko, B. (1994). Fuzzy systems as universal approximators. *Computers, IEEE Transactions on*, 43(11) :1329–1333.

[Kulhavy, 1987] Kulhavy, R. (1987). Restricted exponential forgetting in real-time identification. *Automatica*, 23(5) :589–600.

[Kulhavy and Zarrop, 1993] Kulhavy, R. and Zarrop, M. B. (1993). On a general concept of forgetting. *International Journal of Control*, 58(4) :905–924.

[Kuncheva, 2004] Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In *Proceedings of the 5th International Workshop on Multiple Classifier Systems*, pages 1–15. Springer.

[Langley et al., 1992] Langley, P., Iba, W., and Thompson, K. (1992). An analysis of Bayesian classifiers. In *AAAI*, volume 90, pages 223–228.

[Laskov et al., 2006] Laskov, P., Gehl, C., Krüger, S., and Müller, K.-R. (2006). Incremental Support Vector Learning : Analysis, Implementation and Applications. *Journal of Machine Learning Research*, 7 :1909–1936.

[Last, 2002] Last, M. (2002). Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2) :129–147.

[LaViola and Zeleznik, 2007] LaViola, J. J. and Zeleznik, R. C. (2007). A Practical Approach for Writer-Dependent Symbol Recognition Using a Writer-Independent Symbol Recognizer. *Pattern Analysis and Machine Intelligence, IEEE Transaction on*, 29(11) :1917–1926.

[Law and Zaniolo, 2005] Law, Y.-N. and Zaniolo, C. (2005). An adaptive nearest neighbor classification algorithm for data streams. In *Knowledge Discovery in Databases : PKDD 2005*, pages 108–120. Springer.

[Lethelier et al., 1995] Lethelier, E., Leroux, M., and Gilloux, M. (1995). An automatic reading system for handwritten numeral amounts on French checks. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 92–97. IEEE.

[Li et al., 2012] Li, P. Y., Renau-Ferrer, N., Anquetil, E., and Jamet, E. (2012). Semi-customizable Gestural Commands Approach and Its Evaluation. In *2012 International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 473–478.

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

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

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

[Lughofer and Buchtala, 2013] Lughofer, E. and Buchtala, O. (2013). Reliable All-Pairs Evolving Fuzzy Classifiers. *IEEE Transactions on Fuzzy Systems*, 21(4) :625–641.

[Lughofer, 2008] Lughofer, E. D. (2008). FLEXFIS : A Robust Incremental Learning Approach for Evolving Takagi-Sugeno Fuzzy Models. *Fuzzy Systems, IEEE Transactions on*, 16(6) :1393–1410.

[Lühr and Lazarescu, 2008] Lühr, S. and Lazarescu, M. (2008). Connectivity based stream clustering using localised density exemplars. In *Advances in Knowledge Discovery and Data Mining*, pages 662–672. Springer.

[Maloof and others, 2003] Maloof, M. and others (2003). Incremental rule learning with partial instance memory for changing concepts. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 2764–2769. IEEE.

[Maloof and Michalski, 2000] Maloof, M. A. and Michalski, R. S. (2000). Selecting examples for partial memory learning. *Machine Learning*, 41(1) :27–52.

[Mamdani, 1977] Mamdani, E. (1977). Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis. *Computers, IEEE Transactions on*, C-26(12) :1182 –1191.

[Markou and Singh, 2003a] Markou, M. and Singh, S. (2003a). Novelty detection : a review—part 1 : statistical approaches. *Signal processing*, 83(12) :2481–2497.

[Markou and Singh, 2003b] Markou, M. and Singh, S. (2003b). Novelty detection : a review—part 2 : : neural network based approaches. *Signal processing*, 83(12) :2499–2521.

[McCallum and Nigamy, 1998] McCallum, A. K. and Nigamy, K. (1998). Employing EM and pool-based active learning for text classification. In *Proc. International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer.

[Meinhold and Singpurwalla, 1983] Meinhold, R. J. and Singpurwalla, N. D. (1983). Understanding the Kalman filter. *The American Statistician*, 37(2) :123–127.

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

[Mitoma et al., 2005] Mitoma, H., Uchida, S., and Sakoe, H. (2005). Online character recognition based on elastic matching and quadratic discrimination. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 36–40. IEEE.

[Mitra et al., 2000] Mitra, P., Murthy, C. A., and Pal, S. K. (2000). Data condensation in large databases by incremental learning with support vector machines. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 708–711. IEEE.

[Morasso et al., 1993] Morasso, P., Barberis, L., Pagliano, S., and Vergano, D. (1993). Recognition experiments of cursive dynamic handwriting with self-organizing networks. *Pattern Recognition*, 26(3) :451–460.

[Moreno-Seco et al., 2002] Moreno-Seco, F., Micó, L., and Oncina, J. (2002). Extending LAESA fast nearest neighbour algorithm to find the k nearest neighbours. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 718–724. Springer.

[Moskovitch et al., 2007] Moskovitch, R., Nissim, N., Stopel, D., Feher, C., Englert, R., and Elovici, Y. (2007). Improving the detection of unknown computer worms activity using active learning. In *KI 2007 : Advances in Artificial Intelligence*, pages 489–493. Springer.

[Mouchere and Anquetil, 2006a] Mouchere, H. and Anquetil, E. (2006a). Generalization capacity of handwritten outlier symbols rejection with neural network. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

[Mouchere and Anquetil, 2006b] Mouchere, H. and Anquetil, E. (2006b). A Unified Strategy to Deal with Different Natures of Reject. In *18th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 792 –795.

[Mouchère et al., 2007a] Mouchère, H., Anquetil, E., Miclet, L., and Bayoudh, S. (2007a). Synthetic On-line Handwriting Generation by Distortions and Analogy. pages 10–13.

[Mouchère et al., 2007b] Mouchère, H., Anquetil, E., and Ragot, N. (2007b). Writer style adaptation in online handwriting recognizers by a fuzzy mechanism approach : The adapt method. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(01) :99–116.

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

[Muhlbaier et al., 2009] Muhlbaier, M. D., Topalis, A., and Polikar, R. (2009). Learn. NC : Combining Ensemble of Classifiers With Dynamically Weighted Consult-and-Vote for Efficient Incremental Learning of New Classes. *Neural Networks, IEEE Transactions on*, 20(1) :152–168.

[Muslea et al., 2000] Muslea, I., Minton, S., and Knoblock, C. A. (2000). Selective sampling with redundant views. In *AAAI/IAAI*, pages 621–626.

[Ng and Jordan, 2002] Ng, A. Y. and Jordan, M. I. (2002). On Discriminative vs. Generative Classifiers : A comparison of logistic regression and naive Bayes. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press.

[Nguyen et al., 2015] Nguyen, H.-L., Woon, Y.-K., and Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowledge and Information Systems*, pages 1–35.

[Niels et al., 2008] Niels, R. M. J., Willems, D. J. M., and Vuurpijl, L. G. (2008). The NicIcon database of handwritten icons.

[Nissim et al., 2012] Nissim, N., Moskovitch, R., Rokach, L., and Elovici, Y. (2012). Detecting unknown computer worm activity via support vector machines and active learning. *Pattern Analysis and Applications*, 15(4) :459–475.

[Nissim et al., 2014] Nissim, N., Moskovitch, R., Rokach, L., and Elovici, Y. (2014). Novel active learning methods for enhanced PC malware detection in windows OS. *Expert Systems With Applications*, 41(13) :5843–5857.

[Osuna et al., 1997] Osuna, E., Freund, R., and Girosi, F. (1997). An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE.

[Oza, 2005] Oza, N. C. (2005). Online bagging and boosting. In *Systems, man and cybernetics, 2005 IEEE international conference on*, volume 3, pages 2340–2345. IEEE.

[Parkum et al., 1992] Parkum, J., Poulsen, N. K., and Holst, J. (1992). Recursive forgetting algorithms. *International Journal of Control*, 55(1) :109–128.

[Pedrycz, 1998] Pedrycz, W. (1998). Conditional fuzzy clustering in the design of radial basis function neural networks. *Neural Networks, IEEE Transactions on*, 9(4) :601–612.

[Pitrelli et al., 2006] Pitrelli, J. F., Subrahmonia, J., and Perrone, M. P. (2006). Confidence modeling for handwriting recognition : algorithms and applications. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(1) :35–46.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : programs for machine learning*. Elsevier.

[Rai et al., 2009] Rai, P., Daumé III, H., and Venkatasubramanian, S. (2009). Streamed Learning : One-Pass SVMs. *arXiv :0908.0572 [cs, stat]*. arXiv : 0908.0572.

[Ramos-Jiménez et al., 2006] Ramos-Jiménez, G., Campo-Ávila, J. d., and Morales-Bueno, R. (2006). Incremental Algorithm Driven by Error Margins. In Todorovski, L., Lavrač, N., and Jantke, K. P., editors, *Discovery Science*, Lecture Notes in Computer Science, pages 358–362. Springer Berlin Heidelberg. DOI : 10.1007/11893318_42.

[Renau-Ferrer et al., 2012] Renau-Ferrer, N., Li, P., Delaye, A., and Anquetil, E. (2012). The ILGDB database of realistic pen-based gestural commands. In *Proceeding of the 21st International Conference on Pattern Recognition*, pages 3741–3744.

[Reznáková et al., 2015] Reznáková, M., Tencer, L., and Cheriet, M. (2015). SO-ARTIST. *Appl. Soft Comput.*, 31(C) :132–152.

[Robnik-Šikonja and Kononenko, 2003] Robnik-Šikonja, M. and Kononenko, I. (2003). Theoretical and empirical analysis of ReliefF and RReliefF. *Machine learning*, 53(1-2) :23–69.

[Romeu et al., 2006] Romeu, J. M., Lamiroy, B., Sanchez, G., and Llados, J. (2006). Automatic adjacency grammar generation from user drawn sketches. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 1026–1029. IEEE.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386.

[Roy and McCallum, 2001] Roy, N. and McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, pages 441–448.

[Ruping, 2001] Ruping, S. (2001). Incremental Learning with Support Vector Machines. *Data Mining, IEEE International Conference on*, 0 :641.

[Saad, 1998] Saad, D. (1998). *On-line learning in neural networks*, volume 17. Cambridge University Press.

[Sakoe and Chiba, 1978] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1) :43–49.

[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.

[Salperwyck et al., 2015] Salperwyck, C., Lemaire, V., and Hue, C. (2015). Incremental Weighted Naive Bays Classifiers for Data Stream. *Data Science, Learning by Latent Structures, and Knowledge Discovery*, pages 179–190.

[Salzberg, 1991] Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine learning*, 6(3) :251–276.

[Sankaranarayanan et al., 2007] Sankaranarayanan, J., Samet, H., and Varshney, A. (2007). A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31(2) :157–174.

[Sebe et al., 2002] Sebe, N., Lew, M., Cohen, I., Garg, A., and Huang, T. (2002). Emotion recognition using a Cauchy Naive Bayes classifier. In *16th International Conference on Pattern Recognition, 2002. Proceedings*, volume 1, pages 17–20 vol.1.

[Seidl et al., 2009] Seidl, T., Assent, I., Kranen, P., Krieger, R., and Herrmann, J. (2009). Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th international conference on extending database technology : advances in database technology*, pages 311–322. ACM.

[Settles, 2010] Settles, B. (2010). Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

[Settles and Craven, 2008] Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics.

[Settles et al., 2008] Settles, B., Craven, M., and Ray, S. (2008). Multiple-instance active learning. In *Advances in neural information processing systems*, pages 1289–1296.

[Silva et al., 2013] Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C., and Gama, J. (2013). Data stream clustering : A survey. *ACM Computing Surveys (CSUR)*, 46(1) :13.

[Steil, 2004] Steil, J. (2004). Backpropagation-decorrelation : online recurrent learning with O(N) complexity. In *2004 IEEE International Joint Conference on Neural Networks, 2004. Proceedings*, volume 2, pages 843–848 vol.2.

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

[Street and Kim, 2001] Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM.

[Syed et al., 1999a] Syed, N. A., Huan, S., Kah, L., and Sung, K. (1999a). *Incremental Learning with Support Vector Machines.*

[Syed et al., 1999b] Syed, N. A., Liu, H., and Sung, K. K. (1999b). Handling concept drifts in incremental learning with support vector machines. 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. *Systems, Man, and Cybernetics, IEEE Transactions on*, 15(1) :116–132.

[Tappert, 1982] Tappert, C. (1982). Cursive Script Recognition by Elastic Matching. *IBM Journal of Research and Development*, 26(6) :765–771.

[Tax and Duin, 2004] Tax, D. M. and Duin, R. P. (2004). Support vector data description. *Machine learning*, 54(1) :45–66.

[Tay and Khalid, 1997] Tay, Y. H. and Khalid, M. (1997). Comparison of fuzzy ARTMAP and MLP neural networks for hand-written character recognition. In *Proceedings of the IFAC Symposium on AI in Real-Time Control (AIRTC 1997)*, pages 363–371. Citeseer.

[Tencer et al., 2015] Tencer, L., Reznakova, M., and Cheriet, M. (2015). TITS-FM : Transductive incremental Takagi-Sugeno fuzzy models. *Applied Soft Computing*, 26 :531–544.

[Tomanek and Hahn, 2009] Tomanek, K. and Hahn, U. (2009). Semi-supervised active learning for sequence labeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP : Volume 2-Volume 2*, pages 1039–1047. Association for Computational Linguistics.

[Tsang et al., 2005] Tsang, I. W., Kwok, J. T., and Cheung, P.-M. (2005). Core vector machines : Fast SVM training on very large data sets. In *Journal of Machine Learning Research*, pages 363–392.

[Tung and Quek, 2009] Tung, W. L. and Quek, C. (2009). A mamdani-takagi-sugeno based linguistic neural-fuzzy inference system for improved interpretability-accuracy representation. In *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on*, pages 367–372. IEEE.

[Tur et al., 2005] Tur, G., Hakkani-Tür, D., and Schapire, R. E. (2005). Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2) :171–186.

[Uchida and Sakoe, 2005] Uchida, S. and Sakoe, H. (2005). A survey of elastic matching techniques for handwritten character recognition. *IEICE transactions on information and systems*, 88(8) :1781–1790.

[Usunier et al., 2010] Usunier, N., Bordes, A., and Bottou, L. (2010). Guarantees for approximate incremental SVMs. In *International Conference on Artificial Intelligence and Statistics*, pages 884–891.

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

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[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. ICDAR '99, pages 455–.

[Vidal, 1994] Vidal, E. (1994). New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters*, 15(1) :1–7.

[Vuori et al., 2001] Vuori, V., Laaksonen, J., Oja, E., and Kangas, J. (2001). Experiments with adaptation strategies for a prototype-based recognition system for isolated handwritten characters. *International Journal on Document Analysis and Recognition*, 3(3) :150–159.

[Walker and Duncan, 1967] Walker, S. H. and Duncan, D. B. (1967). Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1-2) :167–179.

[Wang et al., 2003] Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining Concept-drifting Data Streams Using Ensemble Classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235, New York, NY, USA. ACM.

[Warmuth et al., 2003] Warmuth, M. K., Liao, J., Rätsch, G., Mathieson, M., Putta, S., and Lemmen, C. (2003). Active learning with support vector machines in the drug discovery process. *Journal of Chemical Information and Computer Sciences*, 43(2) :667–673.

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

[Widmer, 1994] Widmer, G. (1994). Combining robustness and flexibility in learning drifting concepts. In *ECAI*, pages 468–472. PITMAN.

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

[Wilpon et al., 1990] Wilpon, J. G., Rabiner, L., Lee, C.-H., and Goldman, E. R. (1990). Automatic recognition of keywords in unconstrained speech using hidden Markov models. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(11) :1870–1878.

[Wobbrock et al., 2009] Wobbrock, J. O., Morris, M. R., and Wilson, A. D. (2009). User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on*

Human Factors in Computing Systems, CHI '09, pages 1083–1092, New York, NY, USA. ACM.

[Wobbrock et al., 2007] Wobbrock, J. O., Wilson, A. D., and Li, Y. (2007). Gestures without libraries, toolkits or training : a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 159–168, New York, NY, USA. ACM.

[Yang et al., 2011] Yang, J., Xu, J., Li, M., Zhang, D., and Wang, C. (2011). A real-time command system based on hand gesture recognition. In *2011 Seventh International Conference on Natural Computation (ICNC)*, volume 3, pages 1588–1592.

[Yang and Webb, 2002] Yang, Y. and Webb, G. I. (2002). A comparative study of discretization methods for naive-bayes classifiers. In *Proceedings of PKAW*, volume 2002. Citeseer.

[Yap et al., 2008] Yap, K. S., Lim, C., and Abidin, I. (2008). A Hybrid ART-GRNN Online Learning Neural Network With a -Insensitive Loss Function. *IEEE Transactions on Neural Networks*, 19(9) :1641–1646.

[Ying et al., 1998] Ying, H., Ding, Y., Li, S., and Shao, S. (1998). Typical takagi-sugeno and mamdani fuzzy systems as universal approximators : necessary conditions and comparison. In *Fuzzy Systems Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, volume 1, pages 824–828. IEEE.

[Yuan et al., 2012] Yuan, G.-X., Ho, C.-H., and Lin, C.-J. (2012). Recent Advances of Large-Scale Linear Classification. *Proceedings of the IEEE*, 100(9) :2584–2603.

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

[Zhang, 2000] Zhang, G. (2000). Neural networks for classification : a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews*, 30(4) :451–462.

[Zhaoxin et al.,] Zhaoxin, C., Anquetil, E., Mouchére, H., and Viard-Gaudinb, C. Recognize multi-touch gestures by graph modeling and matching. *Drawing, Handwriting Processing Analysis : New Advances and Challenges*, page 51.

[Zhu, 2005] Zhu, X. (2005). Semi-Supervised Learning Literature Survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison.

[Zhu et al., 2003] Zhu, X., Lafferty, J., and Ghahramani, Z. (2003). Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, pages 58–65.

[Žliobaite, 2010] Žliobaite, I. (2010). Learning under concept drift : an overview. *arXiv preprint arXiv :1010.4784*.

Table des matières

Table des matières

Liste des figures

Liste des tableaux

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Apprentissage actif en-ligne d'un classifieur évolutif, application à la reconnaissance de commandes gestuelles

Nom Prénom de l'auteur : BOUILLON MANUEL

Membres du jury :
- Madame VISANI Muriel
- Monsieur RAMEL Jean-Yves
- Monsieur ANQUETIL Eric
- Monsieur LEMAIRE Vincent
- Monsieur PLAMONDON Réjean

Président du jury : [Réjean PLAMONDON]

Date de la soutenance : 18 Mars 2016

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état
~~Thèse pouvant être reproduite après corrections suggérées~~

Fait à Rennes, le 18 Mars 2016

Signature du président de jury

[Réjean PLAMONDON]

Le Directeur,

M'hamed DRISSI

Résumé

L'utilisation de commandes gestuelles est une nouvelle méthode d'interaction sur interface tactile. Une bonne méthode pour faciliter la mémorisation de ces commandes gestuelles est de laisser l'utilisateur les personnaliser. Ce contexte applicatif induit une situation d'apprentissage croisé, où l'utilisateur doit mémoriser le jeu de symboles et le système doit apprendre à reconnaitre les différents symboles.

Cela implique un certain nombre de contraintes, à la fois sur le système de reconnaissance de symboles et sur le système de supervision de son apprentissage. Il faut par exemple que le classifieur puisse apprendre à partir de peu de données, continuer à apprendre pendant son utilisation et suivre toute évolution des données indéfiniment. Le superviseur doit quant à lui optimiser la coopération entre l'utilisateur et le système de reconnaissance pour minimiser les interactions tout en maximisant l'apprentissage.

Cette thèse présente d'une part, le système d'apprentissage évolutif *Evolve* ∞, capable d'apprendre rapidement à partir de peu de données et de suivre les changements de concepts. D'autre part, elle introduit le superviseur actif en-ligne *IntuiSup* qui permet d'optimiser la coopération entre le système et l'utilisateur, lors de l'utilisation de commandes gestuelles personnalisées notamment.

Evolve ∞ est un système d'inférence floue, capable d'apprendre rapidement grâce aux capacités génératrices des prémisses des règles, tout en permettant d'obtenir une précision élevée grâce aux capacités discriminantes des conclusions d'ordre un. L'intégration d'oubli dans le processus d'apprentissage permet de maintenir le gain de l'apprentissage indéfiniment, permettant ainsi l'ajout de classes à n'importe quel moment de l'utilisation du système et garantissant son évolutivité « à vie ».

Le superviseur actif en-ligne *IntuiSup* permet d'optimiser les interactions avec l'utilisateur pour entrainer un système d'apprentissage lorsque l'utilisateur est dans la boucle. Il permet de faire évoluer la proportion de données que l'utilisateur doit étiqueter en fonction de la difficulté du problème et de l'évolution de l'environnement (changements de concepts). L'utilisation d'une méthode de « dopage » de l'apprentissage permet d'optimiser la répartition de ces interactions avec l'utilisateur pour maximiser leur impact sur l'apprentissage.

Abstract

Using gesture commands is a new way of interacting with touch sensitive interfaces. In order to facilitate user memorization of several commands, it is essential to let the user customize the gestures. This applicative context gives rise to a cross-learning situation, where the user has to memorize the set of commands and the system has to learn and recognize the different gestures.

This situation implies several requirements, from the recognizer and from the system that supervizes its learning process. For instance, the recognizer has to be able to learn from few data samples, to keep learning during its use and to follow indefinitely any change of the data flow. The supervisor has to optimize the cooperation between the recognizer and the system to minimize user interactions while maximizing recognizer learning.

This thesis presents on the one hand the evolving recognition system *Evolve* ∞, that is capable of fast learning from few data samples, and that follows concept drifts. On the other hand, this thesis also presents the online active supervisor *IntuiSup*, that optimizes user-system cooperation when the user is in the training loop, as during customized gesture command use for instance.

The evolving classifier *Evolve* ∞ is a fuzzy inference system that is fast learning thanks to the generative capacity of rule premises, and at the same time giving high precision thanks to the discriminative capacity of first order rule conclusion. The use of forgetting in the learning process allows to maintain the learning gain indefinitely, enabling class adding at any stage of system learning, and guaranteeing lifelong evolving capacity.

The online active supervisor *IntuiSup* optimizes user interactions to train a classifier when the user is in the training loop. The proportion of data that is labeled by the user evolves to adapt to problem difficulty and to follow environment evolution (concept drifts). The use of a boosting method optimizes the timing of user interactions to maximize their impact on classifier learning process.

INSA RENNES

N° d'ordre : 16 ISAR 04 / D16 - 04

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes – CS 14315 – F-35043 Rennes Cedex
Tel : 02 23 23 82 00 – Fax : 02 23 23 83 96