Étude des mécanismes d'adaptation et de rejet pour l'optimisation de classifieurs : Application à la reconnaissance de l'écriture manuscrite en-ligne

Harold Mouchère

N° d'ordre: D 07 - 27

THÈSE

Présentée devant

l'Institut National des Sciences Appliquées de Rennes

pour obtenir le grade de :

Docteur de l'Institut National des Sciences Appliquées de Rennes
Mention Informatique

par

Harold Mouchère

Équipe d'accueil : IMADOC - IRISA
École Doctorale : Matisse
Titre de la thèse :

Étude des mécanismes d'adaptation et de rejet pour l'optimisation de classifieurs : Application à la reconnaissance de l'écriture manuscrite en-ligne.

soutenue le 5 décembre 2007 devant la commission d'examen

M. :	Maurice	Milgram	Président
MM. :	Laurent	Heutte	Rapporteurs
	Lambert	Schomaker	
MM. :	Hubert	Cardot	Examinateurs
	Laurent	Miclet	
	Éric	Anquetil	
	Guy	Lorette	

«-*Les machines font de leur mieux.*»

Isaac Asimov, nouvelle « Hérédité »

Remerciements

Je remercie tout d'abord tous les membres de mon jury de thèse qui ont pris de leurs temps pour lire et juger mon travail ainsi que pour leur déplacement le jour de la soutenance. Merci à Maurice Milgram qui m'a fait l'honneur de présider mon jury de thèse, à Laurent Heutte et Lambert Schomaker, rapporteurs, pour leur relecture attentive et leurs commentaires constructifs, à Hubert Cardot et Laurent Miclet pour l'intérêt qu'ils ont porté à mon travail.

Je ne remercierai jamais assez Guy Lorette, mon directeur de thèse, et Eric Anquetil, pour leur encadrement, leurs idées, leur soutien... Guy, en bon « grand-père scientifique » m'a fait profiter de ses connaissances du domaine de recherche et de son expérience dans grand monde de la recherche. Eric a été un très bon formateur, toujours à l'écoute de nouvelles idées, toujours plein d'enthousiasme, toujours prêt à me re-motiver. Encore merci à vous deux, cette thèse, ainsi que mes communications écrites et orales, n'auraient pas été possibles sans vous.

Je voudrais aussi remercier ici Sabri Bayoudh et Laurent Miclet pour notre collaboration scientifique courte mais intense et fructueuse.

Merci à toute l'équipe IMADOC pour son accueil, sa bonne humeur, ses pauses café, les séminaires au vert...

Merci beaucoup, Solen, François et Sébastien, mes colocataires de bureau, pour avoir supporté mes blagues et mes coups de blues pendant les phases de rédaction, mais surtout pour votre amitié.

Je remercie mes proches, famille et amis, qui ont contribué de près ou de loin à ces trois ans.

Mais surtout, MERCI à Estelle, pour son soutien et sa patience face à mon manque de disponibilité.

Table des matières

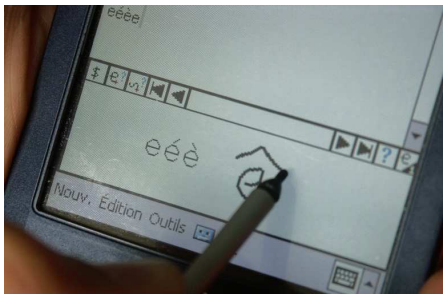
Table des figures

Liste des algorithmes

Introduction

L'émergence de l'informatique nomade a rendu indispensable l'utilisation des interfaces orientées stylo : l'utilisateur écrit et dessine directement des textes, des croquis, des commandes, etc sur l'écran tactile du système. Ce type d'interface a de nombreux avantages dont celui d'être accessible dans de nombreuses situations (nécessité de déplacement, bruit, station debout...) dans lesquelles les utilisateurs peuvent se trouver (contrôle de chantier, médecin à l'hôpital...). Les interfaces classiques avec un clavier et/ou une souris ne sont pas possibles dans ces conditions d'utilisation. De plus les interfaces stylo permettent de réduire la taille du périphérique puisque le clavier n'est plus nécessaire. Cette thèse se place dans ce contexte général d'utilisation de périphériques mobiles de petite taille et donc ayant peu de ressources disponibles en terme de puissance de calcul et de mémoire : téléphone intelligent (*smartphone*), assistant personnel (PDA)... La figure 1 présente deux types de périphériques mobiles avec chacun une interface différente pour la reconnaissance de l'écriture.

(a) Un smartphone



(b) Un PDA

FIG. 1: Exemples de systèmes nomades avec leur système de reconnaissance de l'écriture manuscrite.

Dans ces exemples, il s'agit de systèmes de reconnaissance de caractères. En effet, chaque caractère est reconnu séparément des autres caractères. De plus, dans ce type de contexte applicatif, la reconnaissance se fait à la volée, c'est-à-dire au fur et à mesure de la saisie des caractères par l'utilisateur.

Pour permettre une mise en œuvre conviviale et efficace de ces nouvelles modalités

de communication, il est nécessaire de concevoir des moteurs de reconnaissance robustes et performants pour interpréter l'écriture manuscrite et les tracés graphiques. L'équipe IMADOC dispose déjà de systèmes performants pour la reconnaissance d'écriture manuscrite. L'objectif de ces travaux est donc d'améliorer les performances des systèmes de reconnaissance existants. Pour cela nous proposons de diminuer le nombre d'erreurs de reconnaissance, qu'ils commettent encore, en étudiant deux mécanismes complémentaires, le *rejet* des erreurs potentielles et l'*adaptation* à l'écriture de l'utilisateur :

- *le rejet* permet d'éviter de proposer à l'utilisateur des réponses absurdes ;
- *l'adaptation* permet de spécialiser automatiquement le système générique de reconnaissance au style d'écriture du scripteur courant.

Le premier axe de recherche, **le rejet**, permet de décider si la réponse du système de reconnaissance peut être considérée comme pertinente ou non. En plus du cas d'utilisation normale du système de reconnaissance, nous pouvons distinguer trois cas où le système ne peut pas donner de réponse pertinente. Ces situations sont résumées par la figure 2 en les appliquant à un système de reconnaissance de lettres minuscules.

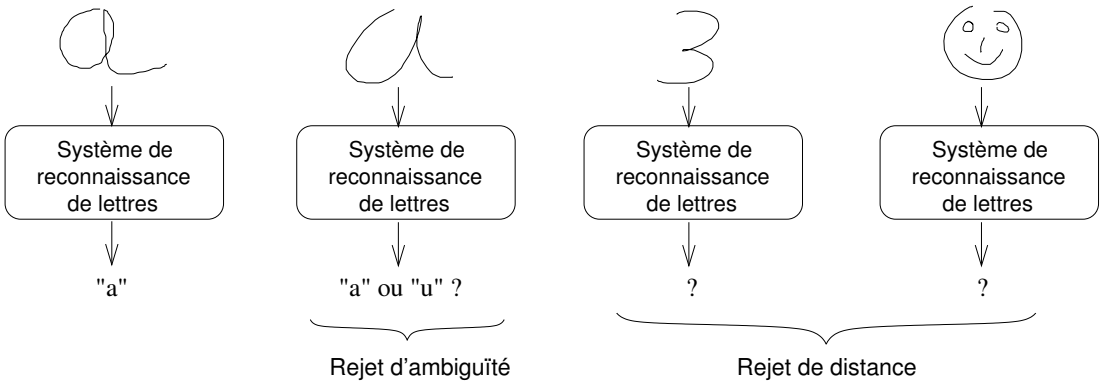


FIG. 2: Cas d'utilisation du rejet.

Le système ne peut pas répondre correctement si la forme à reconnaître est ambiguë (dans cet exemple, même un être humain ne peut décider sans contexte), si le système utilisé n'est pas le bon (un système de reconnaissance de lettres ne peut pas reconnaître les chiffres) ou si la forme ne correspond à rien du tout (gribouillage ou symbole inexistant). Dans tout ces cas nous proposons de rejeter la forme, c'est-à-dire de signifier à l'utilisateur l'impossibilité de reconnaître cette forme. Nous pouvons alors distinguer deux types de rejet :

- *le rejet d'ambiguïté* : la forme est associée par le système à deux classes distinctes de façon équivalente, le système ne peut pas prendre de décision sûre et a beaucoup de chances de se tromper, il faut donc rejeter la forme ;
- *le rejet de distance* : la forme à reconnaître ne correspond pas du tout à une forme que le système a appris à reconnaître, la réponse du classifieur ne peut donc pas être pertinente et il faut donc la rejeter.

Le principe du second axe de recherche, **l'adaptation**, est de spécialiser automatiquement un système de reconnaissance de caractères conçu pour reconnaître l'écriture de n'importe quel utilisateur en un système spécialisé dans la reconnaissance de l'écriture d'une seule personne, l'utilisateur principal du périphérique mobile. Cette adaptation se fait à la volée, c'est-à-dire au fur et à mesure que l'utilisateur utilise le système. Comme le montre la figure 3, à chaque reconnaissance validée par l'utilisateur, l'adaptation utilise ce nouvel exemple de caractère pour optimiser le système de reconnaissance.

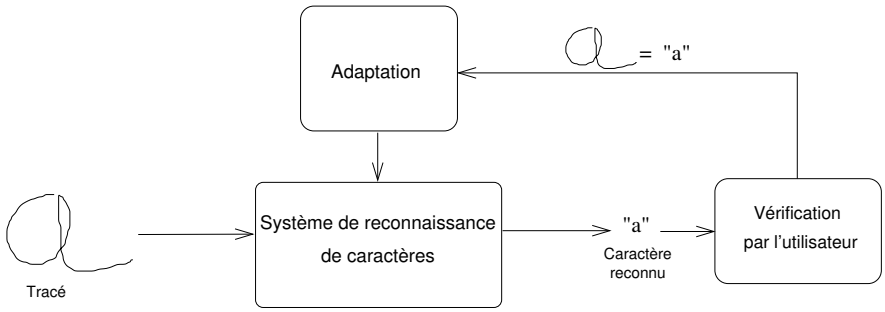


FIG. 3: Schématisation du processus d'adaptation.

Par rapport à l'apprentissage initial du système générique qui utilise une grosse quantité d'exemples saisis par des centaines de scripteurs différents, l'adaptation n'a quant à elle que très peu de données disponibles. En effet nous souhaitons que le processus soit transparent pour l'utilisateur, il n'est donc pas possible de lui demander de saisir plusieurs dizaines de fois chaque caractère. De plus, il faut que l'adaptation soit rapide pour augmenter le confort d'utilisation rapidement, on ne peut donc pas attendre que l'utilisateur ait saisi naturellement un nombre important de caractères. Pour remédier à ce problème, nous proposons en plus d'une stratégie d'adaptation efficace, de synthétiser artificiellement des caractères ressemblant à ceux de l'utilisateur. Ainsi pour chaque caractère entré par le scripteur, des dizaines d'autres respectant le même style d'écriture peuvent être générés et utilisés pour l'adaptation.

Tout au long du développement de ces deux axes de recherche, nous garderons à l'esprit le contexte applicatif visé et donc les contraintes qui y sont liées, même si certaines considérations dépassent ces limites. La première contrainte déjà évoquée est la réalisation de systèmes légers pouvant être embarqués sur des périphériques mobiles à faibles ressources. La seconde contrainte, plus subjective mais liée, est de vouloir conserver une certaine compréhension du fonctionnement des systèmes obtenus (systèmes de type *boîtes blanches*). Cette dernière contrainte justifie certains choix de systèmes de reconnaissance ou de stratégies tout au long de ce manuscrit mais aussi la présence d'expérimentations plus *visuelles* dans le but d'illustrer des concepts pour mieux les comprendre.

Ce manuscrit est constitué d'une partie introductive puis est divisé en deux parties principales : *rejeter pour éviter les erreurs* et *s'adapter pour corriger les erreurs*.

Le premier chapitre présente quelques principes généraux sur la reconnaissance de formes en se focalisant sur notre contexte applicatif, l'écriture en-ligne. Dans ce chapitre nous décrivons ce qu'est l'encre électronique et comment nous l'utilisons dans ce manuscrit. Nous mettons en valeur les aspects associés aux classifieurs qu'il nous semble important de préciser pour bien appréhender la suite du manuscrit, notamment la différence entre modélisation intrinsèque et modélisation discriminante. De plus nous rappelons, par rapport à ces notions, les principes de cinq classifieurs que nous utilisons.

La première partie débute en introduisant la notion de rejet à travers plusieurs exemples d'utilisation de cette notion mettant en valeur les différents aspects du problème.

Le second chapitre présente un état de l'art des différentes approches du rejet avec un point de vue synthétique dans le but de formaliser ces différentes approches. Dans cet état de l'art nous avons proposé une approche globale et générique de la notion de rejet en détaillant cinq points clés transversaux : la définition des deux natures de rejet, l'influence des connaissances sur ce qui doit être rejeté, les principales architectures pour les options de rejet, l'importance du choix des classifieurs utilisés et l'évaluation du rejet. À la fin de ce chapitre nous proposons une extension des méthodes d'évaluation du rejet pour se doter d'outils performants et génériques de comparaison des différentes approches.

Dans le troisième chapitre, nous proposons une extension générique d'une option de rejet utilisant une architecture à base de seuils qui répond parfaitement aux contraintes liées à notre applicatif et à notre souci de compréhension du fonctionnement du système. Cette approche utilise la notion de fonctions de confiance pour évaluer la pertinence de la réponse du classifieur. Elle permet une approche générique du rejet. À partir de ce formalisme, nous proposons deux algorithmes génériques d'apprentissage du rejet et nous les comparons à des algorithmes déjà existants dans l'état de l'art.

Le quatrième chapitre regroupe les différentes expérimentations effectuées sur le rejet. Dans ces expérimentations nous comparons les différentes architectures existantes avec notre approche générique pour le rejet d'ambiguïté et le rejet de distance. Une base artificielle est d'abord utilisée pour mieux comprendre le fonctionnement des différentes options de rejet et de leurs algorithmes d'apprentissage. Puis un problème réel de rejet appliqué à la reconnaissance de l'écriture manuscrite en-ligne est utilisé pour comparer ces différentes approches. Nous concluons ces résultats par un bilan résumant les différents choix à faire suivant le contexte d'utilisation du rejet.

La seconde partie concerne l'adaptation à la volée au scripteur.

Dans le cinquième chapitre nous proposons d'utiliser la synthèse de caractères en-ligne pour palier au problème du manque d'exemples d'apprentissage lors de l'adaptation à la volée. Pour cela nous comparons les approches de synthèses issues du domaine de la reconnaissance de caractères hors-ligne avec de nouvelles approches dédiées à l'écriture en-ligne. Nous proposons aussi d'utiliser le principe intuitif de l'analogie dont l'utilisation a été possible grâce à une collaboration avec Sabri Bayoudh et Laurent Miclet. Nous montrons comment nos stratégies de synthèse de caractères permettent d'obtenir des classifieurs mono-scripteur appris à partir de très peu d'exemples du scripteur, aussi performants que s'ils avaient été appris avec trois fois plus de données d'origine.

Le sixième chapitre présente notre stratégie d'adaptation à la volée d'un Système d'Inférence Floue. Cette approche est basée sur notre méthode ADAPT permettant de modifier de plusieurs manières les prototypes flous représentant les caractères génériques. La stratégie que nous proposons fait le lien avec les chapitres précédents dans le but d'augmenter la rapidité d'adaptation. En effet, elle utilise la synthèse de caractères en-ligne pour enrichir la diversité des données d'adaptation et le rejet pour qualifier la reconnaissance.

Le septième chapitre présente nos expérimentations sur l'adaptation en-ligne au scripteur. Nous montrons l'apport de chaque étape de notre stratégie d'adaptation par des tests simulant une utilisation réelle d'utilisation du système. Ces résultats ont ensuite été confirmés par l'embarquement du système de reconnaissance et de la stratégie d'adaptation sur un périphérique mobile.

La conclusion générale termine ce manuscrit en présentant quelques perspectives concernant le rejet ainsi que l'adaptation.

Chapitre 1

Principes généraux sur la reconnaissance de l'écriture en-ligne

Contrairement aux documents papier qui sont numérisés sous forme d'images, les documents saisis en-ligne (et plus spécifiquement les caractères et gestes) sont stockés sous forme d'encre électronique. Les documents en-ligne peuvent être saisis en utilisant plusieurs types de périphériques :

- la souris d'un ordinateur de bureau ;
- une tablette graphique sans retour visuel ;
- un écran tactile sur lequel on dessine avec le doigt ou un crayon passif (guichet automatique, PDA, smartphone...) ;
- un écran sensitif utilisé avec un stylo spécifique (TabletPC...) ;
- un stylo enregistrant sa position absolue (comme le stylo Anoto utilisant une caméra sur un papier spécifique) ;
- ...

Ces différents périphériques font que l'encre électronique enregistrée pourra être de nature et de qualité différentes. Les domaines applicatifs sont tout aussi vastes, de l'écriture de textes à la saisie de schémas en passant par le remplissage de formulaires ou les gestes d'édition de documents.

Mais la principale difficulté rencontrée lors de la reconnaissance de l'écriture manuscrite est la variabilité des styles d'écriture. En effet, la forme des caractères manuscrits varie énormément d'un scripteur à l'autre et même pour un scripteur donné suivant le contexte du caractère (sa position dans le mot, les lettres voisines...). Cette variabilité est même source d'ambiguïté entre les caractères puisque un même tracé peut avoir différentes significations suivant le contexte ou selon le scripteur.

Ces propriétés font que la reconnaissance de l'écriture manuscrite est un domaine applicatif pour la reconnaissance de formes très riche en difficultés et en défis.

Dans ces travaux nous nous limitons à la reconnaissance de caractères isolés (lettres ou chiffres). En effet cette problématique, déjà très étudiée, est la brique de base de

beaucoup de systèmes plus complexes permettant la reconnaissance de mots, de phrases, de textes. Une petite amélioration de l'efficacité de la reconnaissance des caractères peut permettre de diminuer grandement la complexité des étapes suivantes. Le fait que nous nous trouvions dans un contexte de reconnaissance embarquée ajoute des contraintes sur le système de reconnaissance dues aux ressources limitées des périphériques mobiles. Ces contraintes limitent les stratégies de reconnaissance des formes que nous pouvons utiliser.

Dans la suite de ce chapitre, nous présentons quelques notions de reconnaissance des formes en mettant en valeur les points importants pour la compréhension de la suite de ce manuscrit. Nous prenons comme exemple applicatif la reconnaissance de caractères en-ligne. Ensuite nous décrivons les classifieurs que nous utiliserons dans nos expérimentations.

1.1 Formalisation de la reconnaissance de formes

Le problème de reconnaissance des formes peut être formalisé[1] par une fonction de classement qui à une entrée e décrite dans l'espace E associe une sortie s d'un espace S :

$$\begin{aligned} & f \ : \ E \rightarrow S \\ & \forall e \in E, \exists s \in S \quad f(e) = s \ . \end{aligned} \tag{1.1}$$

Les entrées du système représentent la forme à reconnaître et les sorties les *classes* auxquelles elles appartiennent. La fonction de décision f est concrétisée par un *classifieur*.

Dans la plupart des cas, les classifieurs sont basés sur l'hypothèse simplificatrice que les formes à reconnaître d'une même classe ne sont pas réparties au hasard dans E mais regroupées en une ou plusieurs zones de taille et de forme inconnues. Un classifieur doit donc définir dans l'espace des formes les régions associées à chaque classe de S grâce à certaines *connaissances*. Ces régions sont séparées deux à deux par des *frontières de décision*. La forme et les propriétés de ces régions dépendent de la nature de l'espace des entrées E mais aussi de la *modélisation* des connaissances choisie pour réaliser le classifieur.

Les connaissances d'un classifieur sont de deux natures, celles apportées *a priori* par l'expert du domaine et celles acquises automatiquement. Dans certains cas, les connaissances de l'expert peuvent suffire à définir le classifieur. Mais dans les problèmes complexes, une phase d'*apprentissage* automatique à partir d'une *base d'apprentissage* est nécessaire. Ensuite les performances du classifieur sont évaluées sur une *base de test* pour vérifier qu'il est capable de généraliser les connaissances apprises à des formes différentes de celles utilisées pour l'apprentissage. C'est la *phase de généralisation*.

Nous présentons donc maintenant les différents types d'espaces d'entrée utilisés en reconnaissance de l'écriture manuscrite en-ligne, les différentes natures de sorties possibles et les deux modélisations principales pour représenter les connaissances du classi-

[1] Pour une formalisation plus complète et précise nous renvoyons le lecteur vers les références [29, 35].

fieur. La section 1.2 présentera les différents classifieurs auxquels nous ferons références dans ce manuscrit.

1.1.1 Les entrées du classifieur

D'une façon générale, l'entrée d'un classifieur est une forme qui correspond à une observation de l'environnement extérieur au système. Cette forme est acquise au moyen d'un capteur et présentée sous la forme d'un signal (temporel ou statique). Dans notre contexte, ce signal brut est appelé *encre électronique*. Il peut être présenté directement au classifieur ou bien peut subir au préalable un ensemble de modifications allant du simple pré-traitement jusqu'à l'extraction de caractéristiques.

1.1.1.1 L'encre électronique

L'encre électronique décrit un tracé manuscrit acquis via un périphérique enregistrant différents paramètres au court du temps. Le contenu de ce signal temporel dépend donc du matériel utilisé. D'une façon générale, l'encre électronique est définie par une fonction paramétrique $p(t)$ discrète donnant la position du crayon et diverses informations complémentaires. Il s'agit donc d'un vecteur d'au moins deux dimensions pouvant représenter à l'instant t :

- les coordonnées du crayon, notées $x(t)$ et $y(t)$, dans un repère absolue ;
- la pression du crayon sur la surface, nulle (ou négative) si le crayon ne touche pas la surface ;
- l'inclinaison du crayon par rapport à la surface (spécifiée par deux angles) ;
- un mode du tracé (certains crayons possèdent un bouton ou une gomme) ;
- ...

Suivant le type d'application, différentes informations peuvent être sélectionnées. D'un autre coté, de nouvelles informations peuvent être calculées à partir de celles-ci comme la vitesse instantanée ou la courbure du tracé.

Le temps écoulé entre deux points est généralement constant et détermine le taux d'échantillonnage du signal qui dépend du type de périphérique utilisé. Plus ce taux est élevé plus l'encre électronique est fidèle au tracé réellement effectué par l'utilisateur.

Ce signal brut peut être utilisé directement comme entrée du classifieur. Mais ce mode de fonctionnement peut poser plusieurs problèmes. Il impose notamment de pouvoir traiter des signaux dans lesquels la quantité d'information utile peut varier (en fonction de la taille de la forme, de la durée d'acquisition, de la précision du capteur). C'est pourquoi ce signal brut subit très souvent une série de pré-traitements. L'objectif de ceux-ci est d'obtenir une description de la forme qui soit la plus stable possible, c'est-à-dire qu'entre différentes acquisitions de la même forme, les signaux obtenus doivent être les plus semblables possibles. Ces traitements dépendent fortement du cadre applicatif.

L'encre électronique peut subir des traitements cherchant à résumer l'information, à normaliser le signal ou supprimer des défauts de l'acquisition :

- la segmentation isole les entités indépendantes dans le signal (trouver les caractères dans un mot par exemple) ;
- le ré-échantillonnage temporel ou spatial permet de palier des problèmes d'acquisition imparfaite (point manquant, intervalle de temps non régulier) ou de réduire la quantité d'information (*i.e.* le nombre de points) ;
- les transformations géométriques normalisent certaines variabilités des styles d'écriture (taille, inclinaison, rotation...) ;
- le filtrage permet de supprimer les points aberrants dus à des problèmes d'acquisition ;
- ...

Dans ce manuscrit nous utilisons du signal d'origine seulement les points du tracé qui ont une pression strictement positive, c'est-à-dire les points où le crayon touche la surface sensible. Pour certains caractères écrits en plusieurs traits, il y a donc une discontinuité dans le signal. Chaque trait est séparé par un point où la pression est nulle et est appelé *trace*. Un tracé est donc composé d'une ou plusieurs traces. Ensuite seuls les coordonnées des points sont utilisés pour définir l'encre électronique, nous avons donc :

$$p(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} . \tag{1.2}$$

Mais ce signal n'est pas utilisé directement en entrée du classifieur, en effet un certain nombre de caractéristiques sont extraites de l'encre électronique.

1.1.1.2 Les caractéristiques utilisées

Pour réduire la complexité du signal d'origine, une possibilité très souvent utilisée consiste à extraire du signal brut un certain nombre de caractéristiques ou attributs décrivant la forme. Celles-ci sont très souvent numériques (quantité de points, centre de gravité, position des points remarquables, ...) mais peuvent aussi être symboliques. L'objectif est d'obtenir une description à la fois réduite et la plus pertinente possible pour pouvoir différencier les classes et donc faciliter le travail du classifieur. L'utilisation de caractéristiques en entrée du classifieur simplifie donc la tâche de classification mais nécessite une sélection préalable des caractéristiques pour bien les choisir et un traitement supplémentaire de l'encre pour les extraire pendant l'utilisation.

Dans cette étude, nous ne nous intéressons pas directement au problème de l'extraction de ces caractéristiques. Il s'agit en effet d'un domaine de recherche à part entière.

Nous utilisons donc un ensemble de caractéristiques dédiées à l'écriture manuscrite comme celles définies dans [3, 4]. Ces caractéristiques sont basées sur plusieurs propriétés stables de l'écriture latine : les zones descendantes, les boucles... Ces caractéristiques ont permis de porter le système de reconnaissance de caractères Résif sur smartphone [5]. Parmi la centaine de caractéristiques ainsi définie, nous utiliserons une sélection de 21 caractéristiques qui d'après de précédentes expérimentations, ont montré être un bon compromis entre complexité et performance des classifieurs.

Dans la suite nous ne considérerons que le cas général de la représentation des entrées par un vecteur $e = [e^1, ..., e^n]^T$ de l'espace E à n dimensions. Chaque dimension représente une caractéristique particulière ayant son propre espace de définition et prenant des valeurs numériques. C'est sur cet espace E qu'il s'agit maintenant de définir la modélisation du classifieur.

1.1.2 Modélisation intrinsèque et modélisation discriminante

Une des principales difficultés en reconnaissance de formes est de choisir et de concevoir le classifieur et la modélisation sur laquelle il repose.

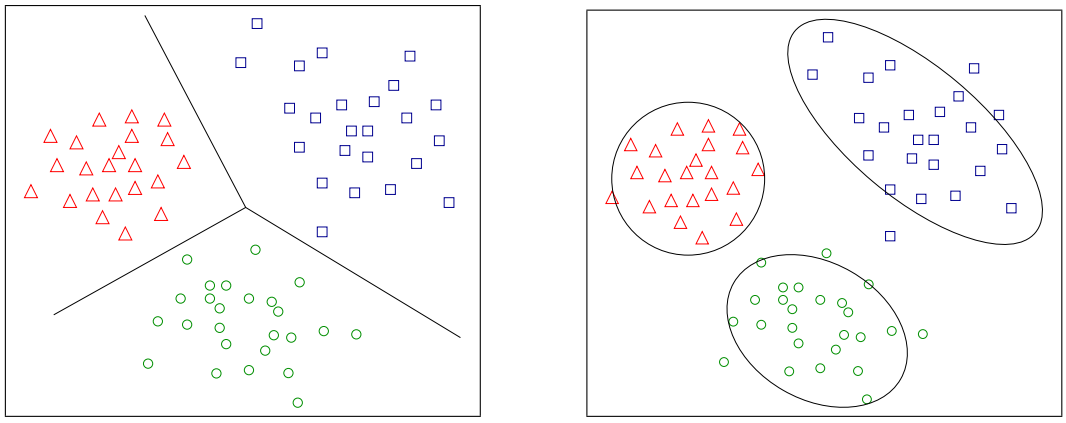
Nous définissons la *modélisation* d'un classifieur comme le choix des connaissances utilisées dans le classifieur et surtout la façon dont elles sont organisées.

Les *connaissances* d'un classifieur sont les éléments du système de reconnaissance sur lesquels reposent la modélisation et qui permettent la classification. Elles sont soit issues de connaissances *a priori* injectées par un expert, soit issues d'une extraction automatique à partir de données.

En général deux types de modélisation peuvent être distingués selon leur objectif :

- *modélisation discriminante* : description implicite des classes par la définition des frontières de décision pour les discriminer ;
- *modélisation intrinsèque*[2] : description explicite des classes par la définition de leurs propriétés intrinsèques.

La figure 1.1 schématise la différence entre ces deux modélisations pour un exemple à trois classes définies dans deux dimensions.



(a) (b)

FIG. 1.1 – Deux types de modélisation : discriminante (a) et intrinsèque (b).

Dans la modélisation discriminante, le but est de chercher, dans l'espace E de représentation des formes, les frontières séparant le mieux les classes (une hypersurface dans le cas général). Les classes sont donc comparées les unes aux autres afin de rechercher

[2] intrinsèque : caractère qui appartient à l'objet lui-même, et non à ses relations avec les autres.

ce qui les différencie pour déterminer les frontières de décision. Pour la classification, il suffit de trouver de quel côté de la frontière se trouve la forme à classer. Cette modélisation très proche de l'objectif de classification est souvent très performante car seules les connaissances utiles pour la classification sont considérées.

Dans la modélisation intrinsèque, l'objectif n'est pas de trouver directement les frontières séparant les classes mais plutôt de les décrire par des propriétés les caractérisant dans l'espace de représentation. On parle aussi de *modélisation générative* puisqu'à partir de cette représentation, il est possible de générer des formes dans E fidèlement à la répartition de chaque classe. Cette caractérisation prend souvent la forme d'un *prototype* servant de modèle pour chaque classe. Ce sont ces prototypes qui servent de connaissance au classifieur. Cette modélisation est souvent utilisée pour mesurer la similarité entre une forme en entrée et les différentes classes modélisées. Cette modélisation rend un peu plus interprétable le système grâce aux prototypes. Pour effectuer la classification, il suffit de mettre en concurrence les modèles de chaque classe par le biais d'une fonction de décision plus ou moins simple.

1.1.3 Les sorties du classifieur

La sortie du classifieur doit contenir assez d'information pour désigner la classe de la forme reconnue. L'ensemble des classes possibles est généralement connu *a priori* par expertise du problème et noté $\{c_1, ..., c_k, ..., c_C\}$ pour C classes. Nous ne traiterons pas du cas où cet ensemble n'est pas initialement connu.

Quelque soit la modélisation choisie pour le classifieur, plusieurs natures d'espaces de sortie S peuvent être utilisées en fonction de l'information supplémentaire qu'elles apportent en plus de la classification. Quatre types de sorties sont généralement distingués :

- le type classe : une seule classe est fournie en sortie, celle jugée la plus adaptée à l'entrée présentée ;
- le type ensemble : un ensemble de classes possibles est rendu sans priorité, peu utilisé car difficilement exploitable en pratique ;
- le type rang : extension du type précédent, un ordre de préférence est donné pour toutes les classes (ou parfois un sous-ensemble) ;
- le type mesure : extension du type précédent, les classes sont ordonnées suivant une quantité (mesure de confiance, score, probabilité...).

Dans la suite, nous considérons le cas général où la sortie du classifieur est un vecteur $[s_1, ..., s_k, ..., s_C]^T \in S$ où chaque élément s_k est un indicateur sur l'adéquation de la forme présentée en entrée par rapport à la classe c_k. Parmi les nombreuses possibilités de mesure d'adéquation, nous allons dégager trois types de scores qui correspondent à trois qualités qui ne sont pas mutuellement exclusives :

- les scores probabilistes : les sorties du classifieur donnent directement la probabilité $P(k|X)$ de la classe k sachant la forme X, ou bien ces probabilités sont calculées à partir des $P(X|k)$ grâce à la relation de Bayes :

$$P(k|X) = \frac{P(X|k)P(k)}{\sum_j P(X|j)P(j)} = \frac{P(X|k)P(k)}{P(X)} \ , \tag{1.3}$$

dans ce cas, la somme des s_k est toujours 1 ;
- les scores discriminants : le score informe juste sur la décision, *i.e.* de quel côté de la frontière est la forme (par exemple score positif ou négatif pour un SVM) ;
- les scores intrinsèques : chaque score s_k donne un degré d'appartenance de la forme à la classe correspondante c_k, les scores sont indépendants entre eux, donc plusieurs peuvent être fortement activés ou au contraire aucun.

1.2 Quelques classifieurs et leur apprentissage

Dans cette section nous présentons quelques classifieurs qui seront utilisés dans la suite de ce manuscrit. Le but n'est pas de faire une description précise des algorithmes (le lecteur intéressé pourra se référer à la littérature par exemple [14, 29, 35, 11]) mais plutôt d'énoncer leurs propriétés en fonction des éléments précédemment présentés et de préciser les choix faits pour leur apprentissage et leur utilisation lors de nos expérimentations.

1.2.1 Les k plus proches voisins

L'approche par les k plus proches voisins (k-PPV) possède au moins deux avantages : elle est très simple et elle ne nécessite pas d'apprentissage. L'ensemble des connaissances du système est constitué simplement de la base d'apprentissage elle-même. La règle de décision consiste à attribuer à une entrée X la classe représentée majoritairement dans son voisinage. Ce voisinage est défini par les k plus proches individus de X parmi ceux se trouvant dans la base d'apprentissage. Les seuls paramètres à déterminer sont donc la valeur de k ainsi que la métrique utilisée pour trouver les plus proches voisins.

Dans plusieurs cadres applicatifs, cette approche a montré des résultats intéressants. En effet, la probabilité d'erreur avec un k-PPV converge vers le risque bayésien quand la quantité de données d'apprentissage augmente, quelque soit k. Cependant elle possède plusieurs inconvénients. Son efficacité dépend directement de la pertinence et de la qualité de la base d'apprentissage (densité des exemples dans des différentes régions de E, présence d'erreurs d'étiquetage...). Autre inconvénient, la base d'apprentissage doit être stockée, ce qui nécessite en général une place mémoire importante ou l'utilisation de techniques de réduction de cette base. Enfin la recherche des plus proches voisins est coûteuse en temps de calcul.

Dans sa version la plus simple, un k-PPV rend en sortie seulement la classe majoritaire dans le voisinage de l'entrée. La sortie de ce classifieur est alors purement discriminante. Il est aussi possible d'utiliser la liste des classes présentes parmi ces k voisins en les triant suivant leur fréquence d'apparition. Certaines techniques permettent de convertir cette liste en score approximant une probabilité. Par exemple en fixant s_i à la fréquence d'apparition de la classe c_i parmi les k voisins (la distance entre l'entrée et les k voisins peut aussi être utilisée).

Dans ce manuscrit, nous utilisons les k-PPV comme classifieurs de référence dans les expérimentations pour évaluer la synthèse de caractères. En effet les k-PPV peuvent être appris avec très peu de données ce qui est une propriété très intéressante dans ce cadre. Nous utilisons la version la plus simple avec la distance euclidienne, $k = 1$ et sans réduction de la base d'apprentissage.

1.2.2 Les réseaux de neurones de type perceptron

Les classifieurs de type réseaux de neurones viennent de recherches orthogonales entre les sciences informatiques et la biologie. Plus précisément, ceux-ci ont été définis à partir d'un modèle de neurone artificiel simulant le mode de fonctionnement du neurone biologique. Du point de vue de la classification, ils reposent sur une modélisation discriminante. Un neurone permet de définir une fonction discriminante linéaire g dans l'espace de représentation E des formes. Cette fonction réalise une combinaison linéaire du vecteur d'entrée de la forme e présentée en entrée :

$$g(e) = Sig(w^t e + w_0) \ , \tag{1.4}$$

où w est un vecteur de poids de la combinaison linéaire, w_0 est le biais et Sig est la fonction sigmoïde. Ainsi, $g(e) = 0$ définit un hyperplan permettant de séparer E en deux régions de décision. Ce sont les poids w et w_0 qui portent les connaissances permettant d'associer à chaque région la bonne classe. Pour les problèmes à plusieurs classes, généralement autant de fonctions discriminantes que de classes sont établies. L'extension à des réseaux de type perceptron multi-couches (MLP, pour *Multi-Layer Perceptron*) permet d'obtenir des frontières de décision de complexité quelconques. Chacune des couches est activée par l'équation 1.4 en prenant en entrée l'activation de la couche précédente. Tous les neurones d'une couches sont connecté à tous les neurones de la couche précédentes et seulement celle-ci (il n'y a notamment pas de boucles). L'apprentissage des poids est généralement fait par des méthodes comme la rétro-propagation du gradient de l'erreur [11].

Les MLP sont très largement utilisés en classification et en reconnaissance de formes, notamment pour leurs bonnes performances et leur simplicité. Ils ont aussi leur points faibles. Ainsi l'apprentissage par l'algorithme de rétro-propagation du gradient de l'erreur risque de converger vers un minimum local donnant une solution sous-optimale. La détermination de l'architecture et des paramètres du réseau est aussi un inconvénient majeur. En effet, avant d'entreprendre l'apprentissage, il faut déterminer le nombre de couches cachées du classifieur ainsi que le nombre de neurones de chacune des couches. Aussi, sans connaissances *a priori*, il faut rechercher l'architecture optimale par essais successifs. Un autre inconvénient majeur réside dans le manque d'interprétabilité (*a posteriori*) des connaissances réparties dans les liaisons inter-neurones complexes ainsi que la grande quantité nécessaire de données pour leur apprentissage.

Les scores de sortie des MLP sont par nature discriminants. Mais les MLP étant des approximateurs universels, ils sont capables d'apprendre une distribution de probabilité. Ainsi, si dans une région de E plusieurs classes apparaissent, les scores vont

se partager entre chacune de ces classes dans les mêmes proportions. Cette propriété ne permet qu'une approximation des probabilités *a posteriori* dont la qualité dépend de la technique d'apprentissage, de l'architecture utilisée et de la représentativité de la base d'apprentissage. Dans tous les cas, même si les capacités de généralisation des MLP sont bonnes, l'interprétation des scores de sortie (discriminante ou probabiliste) n'est valide que dans les régions de E où des données d'apprentissage sont en quantité suffisante.

Dans ce manuscrit, nous utilisons les MLP dans nos expérimentations sur le rejet car il s'agit d'un classifieur très étudié dans ce domaine [67]. Ses capacités de discrimination font de lui un classifieur efficace pour le rejet d'ambiguïté mais pas pour le rejet de distance.

L'apprentissage que nous utilisons se fait en utilisant la technique classique de descente du gradient de l'erreur. Nous séparons la base d'apprentissage en deux pour obtenir une base de validation permettant de détecter la fin de l'apprentissage et éviter un effet de sur-apprentissage. De plus, dans la plupart des expérimentations, plusieurs architectures sont testées en modifiant le nombre de neurones cachés (une seule couche cachée) et la meilleure architecture testée sur la base de validation est automatiquement choisie.

1.2.3 Les systèmes à vastes marges

Les systèmes à vastes marges (SVM pour *Support Vectors Machines*) [30] sont des techniques de classification qui ont de très bonnes performances dans de nombreux domaines. Ces systèmes reposent sur une modélisation discriminante qui s'appuie sur la détermination de *supports* (ou *vecteurs supports*) qui sont des formes remarquables de la base d'apprentissage. Deux notions clés sont à la base de leur fonctionnement. La première qui rend les SVM très efficaces en généralisation repose sur le principe de marge de classification. Dans le cas d'un problème à deux classes linéairement séparables, la discrimination optimale est obtenue par un hyperplan qui maximise sa distance (la marge) avec les individus des deux classes. Pour obtenir cette marge, un SVM recherche les formes de la base d'apprentissage (les supports) qui sont les plus proches de cet hyperplan, c'est à dire les plus difficiles à classer. Dans le cas de classes non linéairement séparables, un système de marges élastiques est mis en place pour tolérer des erreurs de classification et donc tolérer des vecteurs supports à l'intérieur de la marge. Ce compromis entre erreurs et taille de la marge est alors réglé par un paramètre d'apprentissage C.

La deuxième notion importante est la possibilité d'obtenir des fonctions de décision non linéaires. Pour cela, l'idée consiste à effectuer une transformation non linéaire ϕ de l'espace E dans un espace avec un plus grand nombre de dimensions (éventuellement infini) qui permet de se ramener au cas linéaire. En pratique, cette transformation se fait par le biais de fonctions noyaux. Ces fonctions noyaux, notées $K(supp_k, X)$, combinant un vecteur support $supp_k$ avec l'entrée X peuvent avoir plusieurs formes (polynômes,

fonctions à base radiale, produit scalaire[3]...) satisfaisant un certain nombre de critères. Le score de sortie $s(X)$ d'un SVM (un seul score pour deux classes : positif pour désigner une classe et négatif pour l'autre) est alors interprété comme une combinaison linéaire des noyaux appliqués à chaque vecteur support :

$$s(X) = \sum_k y_k \alpha_k K(supp_k, X) \ , \tag{1.5}$$

avec $y_k \in \{-1, 1\}$ désignant l'étiquette du vecteur support $supp_k$ et α_k les multiplicateurs de Lagrange.

D'une façon générale, les SVM ont aussi quelques inconvénients. Le principal réside dans le choix de la fonction noyau adaptée au problème considéré. De plus le classifieur est conçu à l'origine pour traiter les problèmes à deux classes[4]. L'extension aux problèmes multi-classes est souvent faite en considérant un ensemble de problèmes binaires de type « une classe contre toutes les autres » ou « une classe contre une autre ». Enfin les algorithmes d'apprentissage ne sont pas encore très performants pour appréhender des problèmes multi-classes avec des bases d'apprentissage importantes, ce qui conduit à des durées d'apprentissage très longues et des systèmes avec beaucoup de paramètres (vecteurs supports et poids de combinaison).

Les scores de sortie des SVM dans le cas d'un problème multi-classes correspondent aux scores obtenus pour chaque sous-problème binaire. Si on ne considère que le simple cas de la combinaison de SVM « une classe contre toutes », le score de chaque sous-classifieur peut être directement utilisé pour définir le score de la classe correspondante. La décision se fait alors en considérant le score maximum. Il s'agit donc de scores purement discriminants. Il est possible de convertir ces sorties discriminantes en probabilités en ajoutant un post-traitement aux scores par l'utilisation de sigmoïdes ou de fonctions *soft-max* [75]. Ces approches convertissent les scores à valeurs quelconques en des valeurs entre 0 et 1 dont la somme est 1. La forme de la fonction de conversion est définie par un paramètre appris automatiquement à partir des données d'apprentissage pour minimiser la log-vraisemblance négative.

Les SVM sont des classifieurs beaucoup utilisés dans la littérature grâce à leur forte performance, c'est pourquoi nous utiliserons les SVM comme classifieur de référence dans nos expérimentations sur la synthèse de caractères. Les capacité de discrimination de ces classifieurs sont souvent exploités dans les systèmes avec une option de rejet, c'est pourquoi nous étudierons leur efficacité pour les deux types de rejet.

Dans ce manuscrit, nous utilisons des SVM multi-classes du type « une classe contre toutes les autres » utilisant des noyaux Gaussiens avec une déviation standard de σ. Nous utilisons l'implémentation de la bibliothèque Torch [25]. Dans certaines expérimentations nous utilisons la normalisation des sorties pour les convertir en probabilités par une fonction de conversion *soft-max* comme décrite dans [75]. Lorsqu'ils ne sont pas

[3]Le noyau de type produit scalaire euclidien correspondant à ne pas modifier E.

[4]Il existe maintenant des SVM multiclasses (voir *SVM Multiclasses, Théorie et Applications.* Y. Guermeur (2007). *HDR, Université Nancy 1*).

précisés, les paramètres d'apprentissages C et σ sont optimisés en utilisant une base de validation.

Nous utiliserons les RBFN comme classifieur de référence mais surtout dans nos expérimentations sur le rejet. En effet la modélisation intrinsèque des connaissances via les prototypes leur procure de bonne capacité pour le rejet de distance.

1.2.4 Les réseaux de neurones à fonctions à base radiale

Les réseaux de neurones à fonctions à base radiale (RBFN pour *Radial Basis Function Networks*) s'appuient aussi sur la détermination de supports dans l'espace de représentation E. Cependant, à la différence des SVM, ceux-ci peuvent aussi correspondre à des formes fictives, nous les appellerons donc *prototypes*. Ils sont associés à une zone d'influence définie par une distance (euclidienne, Mahalanobis...) et une fonction à base radiale (Gaussienne, cloche...). La fonction discriminante g d'un RBFN à une sortie est définie à partir de la distance de la forme en entrée à chacun des prototypes et de la combinaison linéaire des fonctions à base radiale correspondantes :

$$g(X) = w_0 + \sum_{i=1}^{N} w_i \phi(d(X, supp_i)) \ , \tag{1.6}$$

où $d(X, supp_i)$ est la distance entre l'entrée X et le support $supp_i$, $\{w_0, ..., w_N\}$ sont les poids de la combinaison et ϕ la fonction à base radiale.

L'apprentissage de ce type de modèle peut se faire en une ou deux étapes. Dans le premier cas, une méthode de type gradient est utilisée pour ajuster l'ensemble des paramètres en minimisant une fonction objectif basée sur un critère comme les moindres carrés. Dans le deuxième cas, une première étape consiste à déterminer les paramètres liés aux fonctions à base radiale (position des prototypes et zones d'influence). Pour déterminer les centres, des méthodes de classification non supervisée sont souvent utilisées. Les poids de la couche de sortie peuvent, dans une seconde étape, être appris par différentes méthodes comme la pseudo-inverse ou une descente de gradient. Dans le cas d'un apprentissage en deux étapes, les RBFN possèdent alors plusieurs avantages par rapport aux MLP. Par exemple l'apprentissage séparé des fonctions à base radiale et de leur combinaison permet un apprentissage rapide, simple et évite les problèmes de minima locaux.

La structure des RBFN et des SVM utilisant des noyaux à base radiale est identique (si les bases radiales utilisées sont aussi identiques dans les deux classifieurs). Cependant la manière dont sont déterminés les supports produit une modélisation et un comportement différents. En effet l'apprentissage des supports des SVM s'appuie sur les éléments difficiles à classer alors que les prototypes des RBFN représentent la répartition des exemples dans E. De plus la gestion des problèmes multi-classes est plus simple dans les RBFN. Nous verrons dans la section suivante que les RBFN sont très semblables sous certaines conditions aux Systèmes d'Inférence Floue.

La modélisation des RBFN est à la fois discriminante et intrinsèque. En effet la couche de fonctions à base radiale correspond à une description intrinsèque des données

d'apprentissage et la couche de combinaison en sortie cherche ensuite à discriminer les différentes classes.

Dans ce manuscrit, nous utilisons des RBFN avec un apprentissage en deux étapes. La fonction à base radiale est du type fonction de Cauchy :

$$\phi(d) = \frac{1}{1+d} \ , \tag{1.7}$$

et la distance utilisée est la distance de Mahalanobis :

$$d_{Q_i}(X, \mu_i) = (X - \mu_i)^T Q_i^{-1} (X - \mu_i) \ , \tag{1.8}$$

avec μ_i le centre du prototype i et Q_i sa matrice de covariance. L'apprentissage des prototypes se fait par l'algorithme des C-moyennes possibilistes [60, 61] effectué sur chaque classe séparément. Cette approche permet de renforcer le côté intrinsèque de la modélisation [94]. Les poids de la couche de sortie sont calculés sans biais ($w_0 = 0$) avec la méthode de la pseudo-inverse [11] permettant de trouver l'optimum global en résolvant un système d'équations linéaires. Le seul paramètre de cet apprentissage est le nombre de prototypes appris pour chaque classe. Si dans les expérimentations ce nombre n'est pas indiqué, il est alors optimisé en utilisant une base de validation.

1.2.5 Les systèmes d'inférence floue

Les systèmes à base de règles sont très souvent utilisés en intelligence artificielle notamment pour la mise en œuvre de systèmes experts. Mais ils sont aussi couramment employés en reconnaissance et en classification, notamment quand le mécanisme de décision doit être interprétable pour l'utilisateur.

Une règle de décision en reconnaissance de formes se présente sous la forme d'une formule en logique des propositions :

« **SI** la forme X satisfait la condition $Cond_i$ **ALORS** la classe est c_j »

La prémisse de la règle correspond à une condition que doit satisfaire la forme pour que la conclusion soit valide. Il s'agit d'une proposition logique qui peut être simple ou complexe et qui porte en général sur la valeur de certains attributs de la forme. Cette prémisse est mise en relation avec la conclusion (aussi une proposition logique) par une implication. En définissant plusieurs de ces règles, il est possible de décrire les concepts associés aux différentes classes. Des mécanismes de raisonnement comme la déduction peuvent ensuite être mis en œuvre pour prendre des décisions.

Les *systèmes d'inférence floue* (SIF) étendent les principes des systèmes à base de règles classiques en modélisant les imperfections liées aux connaissances manipulées grâce aux outils de la théorie des sous-ensembles flous. Les mécanismes de raisonnement en résultant sont ainsi plus robustes et plus proches de la réalité. Le lecteur intéressé pourra se reporter aux nombreux ouvrages relatifs aux ensemble flous et aux différents types de systèmes d'inférence floue [13, 14].

Les SIF que nous utilisons dans ce manuscrit sont du type Takagi-Sugeno [103] d'ordre 0 constitués de N règles. Chaque règle R_r est composée d'une prémisse et d'une conclusion. La prémisse correspond à une modélisation intrinsèque d'une classe ou d'une partie d'une classe par un prototype flou P_r défini dans E. La conclusion associe à chaque prototype son degré d'appartenance s_c^r à chaque classe c. Dans un problème à C classes, les règles s'écrivent donc :

$$\text{« } \textbf{SI } X \text{ est } P_r \textbf{ ALORS } s_1^r = a_1^r \textbf{ ... et } s_c^r = a_c^r \textbf{ ... et } s_C^r = a_C^r \text{ »},$$

avec X la forme à reconnaître dans E et les a_c^r sont des valeurs constantes (ordre 0). Les prototypes flous P_r sont définis par le degré d'appartenance $\beta_r(\vec{X})$ d'une forme X à l'ensemble flou correspondant. On parle aussi du *degré d'activation* du prototype. Cette fonction d'appartenance correspond à une fonction à base radiale hyper-ellipsoïdale de centre μ_r et dont la forme est donnée par la matrice de covariance Q_r. Le degré d'appartenance utilise une fonction de Cauchy utilisant la distance de Mahalanobis $d_{Q_r}(X, \mu_r)$:

$$\beta_r(X) = \frac{1}{1 + d_{Q_r}(X, \mu_r)} \ . \tag{1.9}$$

Pour déterminer la classe d'une forme inconnue X, l'activation β_r de chacun des N prototypes flous est d'abord calculée. Ensuite chaque règle est appliquée et combinée par l'inférence floue de type *somme-produit* pour calculer un score s_c pour chaque classe :

$$s_c = \frac{\sum_{r=1}^{N} \beta_r s_c^r}{\sum_{r=1}^{N} \beta_r}. \tag{1.10}$$

Cette équation montre comment les différents prototypes participent à la reconnaissance de toutes les classes : plus un prototype est activé et plus il appartient à la classe, plus il participe au score global de la classe.

L'apprentissage des SIF peut se faire en utilisant des techniques de descente de gradient comme définies dans [76]. Dans ce manuscrit, les prototypes flous sont appris en appliquant l'algorithme de classification non supervisé des C-Moyennes possibilistes [60, 61] sur chaque classe séparément. Ainsi les prototypes permettent une description intrinsèque de chaque classe [94]. Ensuite, les conclusions a_c^r de chaque règle sont calculées grâce à la méthode de la pseudo-inverse [11]. Cette méthode permet d'apporter une couche discriminante aux SIF pour améliorer les performances de classification par rapport à une modélisation purement intrinsèque.

On peut remarquer que la différence entre les SIF et les RBFN est assez mince. Plusieurs articles [10, 54] traitent de cette ressemblance structurelle et pratique de ces deux classifieurs issus de communautés différentes. Ils ont montrés que sous certaines conditions les SIF et les RBFN sont équivalents, notamment : le type des fonctions à base radiale, le type de conclusion pour les SIF, l'apprentissage des paramètres (prototypes et conclusions) et le type d'inférence choisie. Dans notre cas, les fonctions à base

radiale sont identiques, les conclusions des SIF correspondent aux poids des RBFN et l'apprentissage est identique, mais l'inférence utilisée dans les SIF est différente de la combinaison linéaire utilisée dans les RBFN. Cette petite différence ne change en rien la classification, c'est à dire que pour une même entrée les deux systèmes répondrons le même ordre de classe. Par contre elle a une influence sur la valeur des scores de sortie.

Dans les expérimentations sur le rejet, plus liées au domaine général de la reconnaissance des formes, nous utiliserons des RBFN car ce sont des classifieurs beaucoup utilisés dans ce domaine. Dans les expérimentations sur l'adaptation, les SIF seront utilisés pour leur propriété de compacité (pour les embarquer sur un périphérique mobile) et leur modélisation floue adéquate pour notre problème. Grâce à la leur ressemblance avec les RBFN, les résultats obtenus dans la partie rejet pourront être étendus aux SIF.

Première partie

Rejeter pour éviter les erreurs

Introduction au rejet

Lors de la classification, un classifieur de formes associe normalement une classe à la forme proposée en entrée. Le principe de l'*option de rejet* est d'enrichir ce fonctionnement en permettant au classifieur de ne pas répondre. Dans ce cas on dit que la forme est *rejetée*, il y a *rejet*. Le principal intérêt du rejet est de pouvoir éviter une erreur de classification. Tant que les classifieurs ne seront pas parfaits, le rejet permettra de détecter et de gérer les erreurs potentielles.

Exemples d'applications du rejet

Le rejet est un problème général de reconnaissance des formes et il y a beaucoup de domaines d'application qui ont besoin du rejet. Nous étudierons plus particulièrement les publications déjà nombreuses en rapport avec notre domaine, la reconnaissance de l'écriture manuscrite (en laissant de côté les autres domaines comme la reconnaissance de la parole [114], le diagnostic [12], ...). Dans ce cadre, deux applicatifs sont très représentatifs de la nécessité du rejet : la reconnaissance du montant sur les chèques [43, 45, 44, 52, 55, 57, 65, 82] et la reconnaissance des codes postaux sur le courrier [9, 66]. Dans ces cas, il est important de savoir quand la classification peut être mauvaise car une erreur peut ensuite avoir un coût important. Ces formes rejetées peuvent ensuite faire l'objet soit d'une vérification manuelle comme c'est le cas dans les deux exemples précédents, soit d'une classification par un autre classifieur plus performant car plus complexe [62] ou spécialisé [91]. Le taux de rejet désiré pour ces applications peut être important (supérieur à 20%) car il est choisi de manière à optimiser le coût global du traitement en fonction du coût d'une erreur et du coût d'une vérification manuelle.

Le rejet est aussi très utile lorsqu'un système de reconnaissance comporte une ou plusieurs sous étapes intermédiaires, par exemple dans la reconnaissance de mots [68, 78, 80, 87], de phrases [72, 116], ou de suite de chiffres [22, 66, 82, 95] et plus généralement dans les problèmes nécessitant une étape de segmentation [6, 20, 21]. En effet il est très intéressant de pouvoir détecter le plus tôt possible dans le processus de reconnaissance si une segmentation est mauvaise ou non. Les options de rejet dans ces systèmes complexes peuvent utiliser des sources d'information provenant des différents niveaux du système pour prendre la décision de rejet.

Dans d'autres domaines, les interfaces homme machine orientées stylo ou la reconnaissance de documents complexes par exemple, un grand nombre de formes peut être rencontrées dans un même contexte (chiffres, lettres, symboles, gestes d'édition). Dans

ce cas le rejet est utilisé non pas pour obtenir une option de rejet globale mais pour optimiser le système de reconnaissance en se basant sur le rejet pour choisir le classifieur le plus capable de reconnaître la forme. Ces systèmes prennent la forme d'une cascade de classifieurs : le premier classifieur tente la reconnaissance, s'il n'arrive pas à reconnaître la forme (il y a rejet) on la propose au classifieur suivant, etc. Deux architectures de cascade sont alors possibles : soit tous les classifieurs reconnaissent tous le même ensemble de classes [2, 15, 81] ; soit il s'agit de choisir parmi un ensemble de classifieurs spécialisés (un reconnaisseur de chiffres, un reconnaisseur de lettres...) le bon classifieur à utiliser [6].

Plan de la partie

Comme nous pouvons le constater dans cette introduction, il y a eu de nombreux travaux sur la notion de rejet dans la communauté de l'écrit et il s'agit d'un problème encore ouvert qui n'est pas résolu. Beaucoup de stratégies de rejet sont proposées, utilisant différentes modélisations, différentes sources d'information et différents classifieurs. Il n'existe pas à notre connaissance d'état de l'art rassemblant et comparant avec un peu de recul toutes ces approches[5]. C'est pourquoi, réaliser un état de l'art critique par rapport à notre objectif applicatif faisait partie des objectifs principaux de cette étude. En effet, notre domaine applicatif présenté dans le chapitre introductif nous amène à avoir des préférences pour certains choix. D'abord nous travaillons sur la reconnaissance de caractères isolés, nous nous limiterons donc aux architectures n'utilisant pas d'information de plus haut niveau (reconnaissance de mots, modèles de langage,...) ou de plus bas niveau (segmentation,...). Le système devra pouvoir être embarqué, les systèmes légers seront donc préférés et l'option de rejet devra se contenter des caractéristiques déjà disponibles pour la reconnaissance[6] et des informations données par le classifieur. Nous préférerons aussi les options de rejet compatibles avec les systèmes d'inférence floue que nous savons embarquables (comme les SIF ou les RBFN décrits dans le chapitre précédent). Ces contraintes et préférences feront partie de nos critères de comparaison tout au long de notre étude.

Dans le chapitre 2 nous commençons par présenter un état de l'art sur la notion de rejet. Notre premier apport dans cet état de l'art est de distinguer cinq points importants à considérer lorsque l'on souhaite utiliser ou étudier des classifieurs avec option de rejet :

- La définition des deux natures de rejet,
- Les principales architectures pour les options de rejet,
- L'importance des classifieurs utilisés,
- L'influence des connaissances sur ce qui doit être rejeté,

[5] L'état de l'art en deux parties [69, 70] se limite à un seul type de rejet (le rejet de distance) et reste très descriptif.

[6] Calculer de nouvelles caractéristiques augmente la coût de l'option de rejet mais il pourra parcontre être intéressant de sélectionner le meilleur jeu de caractéristiques parmi celles disponibles pour décider du rejet.

– L'évaluation du rejet.

Dans cet état de l'art nous avons proposé une approche globale et générique de la notion de rejet en détaillant ces cinq points clés à chaque fois selon deux angles de vue : ce qui est présenté dans les publications de l'état de l'art et notre point de vue synthétique, parfois critique, dans le but de formaliser les différentes approches. La section 2.6 présente notre apport sur l'évaluation du rejet, en effet dans le cadre d'une étude comparative des différentes approches du rejet, il faut se munir d'outils de comparaison permettant de mettre en évidence de façon synthétique les différences entre elles.

L'effort de synthèse et de critique que nous avons fait dans l'état de l'art nous a permis de définir ensuite une option de rejet générique compatible avec les contraintes liées à notre applicatif. Le chapitre 3 décrit cette architecture utilisant des seuils multiples sur des fonctions de confiance. Nous présentons dans la section 3.2 notre algorithme AMTL d'apprentissage des seuils que nous comparons à des algorithmes plus classiques.

Enfin le chapitre 4 compare les différentes options de rejet et les différents algorithmes d'apprentissage grâce à une approche pédagogique en deux dimensions et à l'étude d'un cas réel de reconnaissance de caractères isolés.

Chapitre 2

État de l'art

Dans cette section nous présentons l'état de l'art sur la notion de rejet en particulier dans le domaine de la reconnaissance d'écriture. Un effort particulier a été fait pour formaliser les différentes approches et les divers points de vue présents dans la littérature.

2.1 Les deux natures de rejet

Les systèmes de reconnaissance avec option de rejet ont la capacité de ne pas classer une forme afin d'éviter de faire des erreurs. Ces erreurs peuvent avoir principalement deux causes qui correspondent à deux états du classifieur :

- le classifieur hésite entre deux classes possibles, il est indécis,
- le classifieur n'a pas appris à reconnaître la forme à classer, il est surpris.

A ces deux types d'erreurs sont associés deux types de rejet : le *rejet d'ambiguïté* et le *rejet de distance*. Plusieurs auteurs confirment cette vision des choses [28, 32, 34, 37, 53, 67] même si cette différenciation n'est pas faite dans toutes les publications sur ce domaine. De plus, ces mêmes publications expliquent qu'il faut des approches différentes pour apprendre ces deux natures de rejet efficacement. C'est pourquoi il est important de savoir avant toute chose à quel type de rejet l'option de rejet sera confrontée.

Ces deux natures de rejet correspondent à des contextes d'utilisation différents. Par exemple certaines taches nécessitent que très peu d'erreurs de confusion soient faites (reconnaissance de chèques, de codes postaux, etc.), il faut donc que l'on soit sûr que la classification soit bonne, dans ce cas on utilisera le rejet d'ambiguïté [23, 34, 40, 66, 82] pour minimiser le risque d'erreur. Dans un autre contexte, le rejet de distance utilise des connaissances intrinsèques pour délimiter le domaine de validité d'un classifieur spécialisé en rejetant les formes n'appartenant pas aux classes apprises [6, 34]. Le rejet de distance consiste donc à détecter quand une donnée ne correspond pas aux données d'apprentissage ce qui est proche de l'objectif de la détection de nouveauté (*novelty dectetion*) [69, 70]. Ces deux types de rejet peuvent être utilisés (ensemble ou non) pour définir des cascades de classifieurs dans le but de réduire la complexité de la

reconnaissance ou d'en améliorer les performances [1, 2, 6, 74, 81, 62, 91].

Pour illustrer les définitions des deux natures de rejet nous utiliserons un classifieur qui met facilement en valeur leurs différences, un RBFN (*Radial Basis Function Network*, présenté section 1.2.4) entraîné à discriminer trois classes en deux dimensions.

2.1.1 Définition du rejet d'ambiguïté

Le but du rejet d'ambiguïté est d'accroître la fiabilité du classifieur en rejetant les formes pour lesquelles le classifieur a de fortes chances de faire une mauvaise classification, c'est à dire d'associer la forme à une mauvaise classe. Ces erreurs sont proches des frontières de décision car elles activent fortement au moins deux classes de façon équivalente. Un rejet d'ambiguïté peut être vu comme un refus de classification pour éviter une erreur de confusion.

Une technique classique pour réaliser le rejet d'ambiguïté est de définir un couloir autour des frontières de décision. Toutes les formes à l'intérieur sont considérées comme des erreurs potentielles et sont donc rejetées. La figure 2.1 montre un exemple en deux dimensions de rejet d'ambiguïté pour un RBFN appris pour reconnaître trois classes.

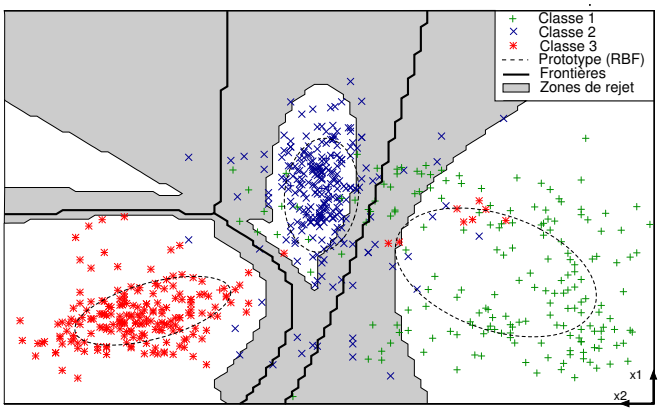


FIG. 2.1 – Exemple de zones de rejet de confusion pour un RBFN.

2.1.2 Définition du rejet de distance

Le rejet de distance est basé sur la délimitation des connaissances du classifieur. En effet un classifieur ne peut classer que ce qu'il a appris à classer. Il faut donc rejeter ce qui n'appartient à aucune classe apprise. Si une forme est trop éloignée des connaissances du classifieur il faut la rejeter.

Une technique classique pour le rejet de distance est de déterminer la probabilité d'appartenir (ou le degré d'appartenance) aux données d'apprentissage. Dans le cas de notre exemple, un RBFN, les activations des neurones cachés permettent d'évaluer cette valeur en décrivant les données d'apprentissage. La figure 2.2 donne un exemple

de rejet de distance sur un RBFN en utilisant un seuil pour chaque fonction à base radiale, dans le but de rejeter une classe 4 inconnue au moment de l'apprentissage.

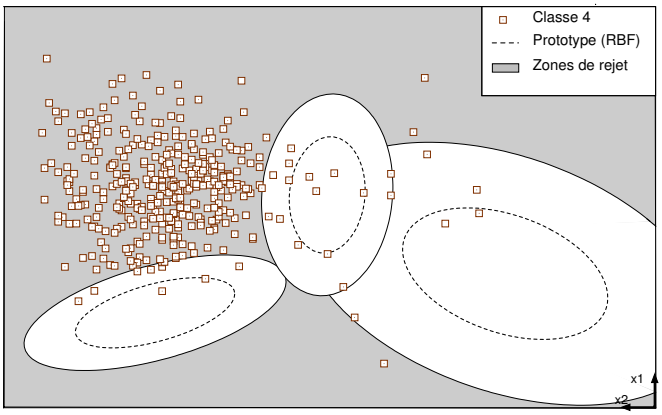


FIG. 2.2 – Exemple de zones de rejet de distance pour le même RBFN que la figure 2.1 pour rejeter une classe 4.

2.1.3 Discussions

Ces définitions ont jusque là été naturellement guidées par les deux sources d'erreurs : la forme n'est pas reconnue soit parce qu'elle ne ressemble pas aux données d'apprentissage, soit parce qu'elle correspond fortement à au moins deux classes. Nous proposons d'ajouter à cette approche la nature des connaissances utilisées pour réaliser le rejet, des connaissances soit discriminantes soit intrinsèques telles que définies dans la section 1.1.2.

Le rejet d'ambiguïté est basé sur la notion de discrimination. En effet les erreurs potentielles du classifieur sont liées aux capacités discriminantes du classifieur. Ces capacités changent d'un classifieur à l'autre, les frontières changent et donc les zones à rejeter changent. Dans la section 4.2.1 nous étudierons les effets des différents choix d'architecture et de classifieur sur la forme des zones de rejet d'ambiguïté.

Le rejet de distance est très lié à la capacité descriptive des classifieurs. En effet l'efficacité du rejet de distance va dépendre des capacités du classifieur utilisé à faire la différence entre les données utilisées lors de l'apprentissage et les autres à rejeter. Les classifieurs basés sur une modélisation intrinsèque des connaissances comme les RBFN, les k-PPV ou les SIF permettent d'avoir facilement ce genre d'information. Dans la section 4.2.2 nous étudierons la forme des zones de rejet de distance en fonction des choix possibles lors de l'apprentissage d'une option de rejet de distance.

Ces deux natures de rejet ne sont pas confrontées aux mêmes difficultés d'apprentissage. Pour le rejet d'ambiguïté, les erreurs à rejeter sont fortement mélangées à des formes bien classées. Pour le rejet de distance, la zone à rejeter est quasiment infinie

puisque qu'il faut rejeter tout ce qui n'est pas à accepter, que ce soit très différent ou peu différent.

2.2 L'influence des connaissances sur ce qui doit être rejeté

La phase d'apprentissage du rejet va dépendre de la tâche future de reconnaissance. La difficulté est de connaître pendant l'apprentissage ce qui devra être rejeté en phase de généralisation (*i.e.* pendant l'utilisation). Ce problème apparaît surtout pour le rejet de distance.

Dans certains contextes il y a un grand nombre de classes à reconnaître comme dans une partition musicale [6] où il y a du texte, des chiffres, des symboles musicaux. Une solution est alors d'utiliser des classifieurs spécialisés pour chaque type de formes : un classifieur de chiffres, un classifieur de lettres, un classifieur de symboles. Chaque classifieur peut rejeter une forme trop différente de ce qu'il a appris (rejet de distance). On peut alors construire une cascade de classifieurs : à chaque fois qu'une forme est rejetée elle est proposée au classifieur suivant jusqu'à l'acceptation ou le rejet global. Dans ce type de problème il est facile de créer une base d'exemples de ce que chaque classifieur doit rejeter dans le but d'apprendre les options de rejet : le classifieur de chiffres rejette les lettres et les symboles...

Par contre dans le contexte de la recherche de champ numérique dans des lettres manuscrites [20] comme dans celui de la coopération entre segmentation et reconnaissance [6, 22, 80, 95] il est beaucoup plus difficile d'obtenir une base représentative de ce qui doit être rejeté. En effet il n'est pas possible de prévoir toutes les configurations de recouvrement ou de sur-segmentation et encore moins les ratures ou taches d'encre. On retrouve le même problème dans notre contexte de reconnaissance en-ligne de formes puisque dans les interfaces homme machine, plusieurs types de formes peuvent être écrits mais l'utilisateur peut aussi entrer des gribouillages ou des dessins non interpretables qui sont difficilement définissables *a priori*. Dans ce dernier cas il faudra mieux rejeter la forme plutôt que d'essayer de reconnaître quelque chose là où il n'y a rien à reconnaître.

2.2.1 État de l'art

Il y a très peu de publications sur l'influence du contenu des bases d'apprentissage des formes à rejeter sur l'efficacité de l'option de rejet. Toutefois des études comparent l'effet de l'utilisation ou non de contre-exemples pendant l'apprentissage du classifieur sur les performances de rejet et de classification [18, 66, 68, 105]. Pour l'apprentissage des classifieurs qui ne permettent pas une définition intrinsèque des classes l'ajout de contre-exemples dans l'apprentissage permet d'améliorer les performances du rejet de distance en fermant les frontières de décision. En effet, les contre-exemples permettent de forcé les scores de sortie à de faibles valeurs en dehors des zones à reconnaître. Par exemple pour les MLP, l'utilisation de contre-exemples pendant l'appren-

tissage permet d'améliorer la qualité du rejet de distance [109]. Toujours dans [109], les auteurs montrent que pour les classifieurs génératifs comme les RBFN, l'ajout de contre-exemples a moins d'influence car ils ont déjà des scores faibles loin des données d'apprentissage. Concernant la qualité des contre-exemples disponibles, dans [63] deux problèmes de rejet sont considérés : rejeter une classe de rejet bien définie et une classe de rejet mal définie dans la même option de rejet. Leur approche permet de gérer ces deux types de rejet grâce à deux seuils, un pour décider de classer la forme suivant les deux classes bien définies (la classe à reconnaître et la classe de rejet connue) et un second pour décider du rejet de distance. Ils obtiennent donc des courbes ROC en 3 dimensions (voir la définition des courbes ROC dans la section 2.6.2).

2.2.2 Discussions : trois problèmes pour le rejet de distance

D'après ces exemples on peut distinguer deux situations correspondant à deux problèmes différents : soit ce qui doit être rejeté est bien défini et une base d'apprentissage représentative des contre-exemples est disponible, soit ce qui doit être rejeté n'est pas bien défini et une base d'apprentissage représentative n'est pas disponible. Une troisième situation intermédiaire, plus réelle, est celle où des contre-exemples d'apprentissage sont disponibles mais pas complètement représentatifs de ce qui devra être rejeté.

Pour formaliser ce problème nous avons défini deux ensembles disjoints de formes à rejeter : l'ensemble A et l'ensemble B. Pour désigner chaque problème, nous utilisons cette notation :

$$\{\textit{ce qui est disponible en apprentissage}\} \rightarrow \{\textit{ce qui est rencontré pendant l'utilisation}\}$$

Les trois problèmes peuvent donc se résumer ainsi :

- Le *problème* $A \rightarrow A$: l'option de rejet est apprise sur l'ensemble A et devra rejeter le même type de formes appartenant aux classes de A, c'est le cas idéal des données d'apprentissage représentatives,
- Le *problème* $A \rightarrow B$: l'option de rejet est apprise sur l'ensemble A et devra rejeter des formes différentes appartenant aux classes de l'ensemble B, c'est le problème des données non représentatives, dans un cas limite A peut être vide et il faut apprendre le rejet sans contre-exemple (on notera dans ce cas le *problème* $\emptyset \rightarrow B$),
- Le *problème* $A \rightarrow A\&B$: l'option de rejet est apprise sur l'ensemble A et devra rejeter des formes appartenant aux classes des deux ensembles A et B, ce problème correspond au cas plus réel des données d'apprentissage non représentatives.

Concernant le rejet d'ambiguïté, nous pouvons considérer qu'il s'agit toujours d'un *problème* $A \rightarrow A$. En effet les données à rejeter dans cette option de rejet sont toujours du même type que celles à classer. Donc si la base d'apprentissage est suffisamment représentative et importante, toutes les possibilités d'erreurs de classification seront représentées.

Dans la suite de cette étude nous utiliserons les notations suivantes pour désigner les bases de données (plus d'explications sont données dans la section 2.6) :

- D_E : base d'exemples à accepter,
- D_A : base de contre-exemples de type A à rejeter,

- D_B : base de contre-exemples de type B à rejeter,
- D_{AB} : union de D_A et D_B,
- D_R : base de contre-exemples à rejeter lorsque la distinction des types A ou B n'est pas faite.

2.3 Les principales architectures pour les options de rejet

La principale architecture qui utilisée dans la littérature est basée sur des seuils de décision, c'est aussi la plus ancienne. Ensuite ce formalisme a été étendu pour s'appliquer à d'autres contextes et d'autres classifieurs. Parallèlement à ces extensions, d'autres architectures sont utilisées qui considèrent le problème sous d'autres angles. Pour présenter cet état de l'art, nous avons choisi de suivre d'abord l'évolution de la principale architecture puis de présenter trois architectures représentatives des autres possibilités. Nous faisons ensuite le bilan d'architectures moins répandues mais connexes à ces quatre principales.

2.3.1 Principale architecture : Utilisation de seuils

2.3.1.1 Première formalisation

Chow [23] a proposé une des premières publications qui traite du rejet. L'option de rejet qu'il propose s'applique à des classifieurs probabilistes. Il y a rejet lorsque la probabilité *a posteriori* que la forme X appartienne à la meilleure classe est plus faible qu'un certain seuil T :

$$\max_i P(i|X) < T. \tag{2.1}$$

Grâce aux propriétés des classifieurs probabilistes il démontre trois propriétés importantes (avec $T = 1 - t$) :

1. le taux d'erreur et le taux de rejet sont des fonctions monotones de t,
2. t est une borne maximale pour le taux d'erreur,
3. t est un compromis différentiel entre le taux d'erreur et le taux de rejet :

$$dE(t)/dR(t) = -t. \tag{2.2}$$

On en déduit le principe suivant : *plus le seuil t est bas, plus il y a de rejet et moins il y a d'erreurs*. La plupart des publications suivantes sur le rejet sont basées sur ce principe. En effet il s'applique à tous les classifieurs probabilistes et Chow a montré que cette option de rejet est le meilleur compromis dans ce cas. Ce principe peut être étendu à tous les classifieurs qui respectent la propriété 1. Cette architecture a donc été beaucoup utilisée comme le montre par exemple l'état de l'art [69]. Par contre nous verrons dans la suite de cette étude comme dans d'autres [109] que les classifieurs qui ne respectent pas cette première propriété ne vérifient pas non plus ce principe.

2.3.1.2 Extension du formalisme

Le rejet proposé par Chow est par construction un rejet d'ambiguïté, dans [34] l'approche de Chow est étendue pour y inclure un rejet de distance. Pour cela les auteurs fixent un seuil sur la probabilité $P(X)$ d'apparition de X, si cette valeur est trop faible, la forme est rejetée.

Plus récemment, les travaux de Fumera *et al* [40] ont montré que l'option de rejet de Chow n'est optimale que lorsque les probabilités *a posteriori* d'appartenir à chaque classe sont bien estimées, ce qui n'est généralement pas le cas dans des contextes réels. C'est pourquoi les auteurs proposent d'utiliser un seuil par classe plutôt qu'un seul seuil global. La règle de rejet devient donc alors :

$$\max_i P(i|X) = P(k|X) < T_k. \tag{2.3}$$

Pour décider du rejet, il faut considérer la classe qui maximise la probabilité d'appartenance de X et la comparer au seuil correspondant. Pour apprendre l'option de rejet il s'agit alors de donner une valeur à chaque seuil $\{T_i\}$. Les auteurs ont montré que pour chaque option de rejet de Chow définie par le seuil T il existe un ensemble $\{T_i\}$ de seuils au moins aussi efficaces. Si $A(T_1, ..., T_k)$ est une fonction d'efficacité du rejet, on a :

$$\forall T, \exists T_1, ..., T_k \ : A(T) \leq A(T_1, ..., T_k). \tag{2.4}$$

La difficulté est de trouver la combinaison de seuils qui permet de faire mieux qu'avec un seul seuil. Nous présenterons dans la section 3.2 plusieurs algorithmes d'apprentissage de ces seuils dont celui proposé par Fumera *et al*[40].

2.3.1.3 Utilisation des fonctions de confiance

L'utilisation des seuils peut être étendue à d'autres types de classifieurs, surtout avec l'utilisation de seuils multiples puisqu'elle permet de faire du rejet même si les scores obtenus pour chaque classe sont une approximation des probabilités *a posteriori*. Dans [37] des seuils sont utilisés sur un classifieur flou pour définir des options de rejet. Des mesures autres que les probabilités *a posteriori* peuvent être utilisées avec des seuils pour réaliser une option de rejet. Dans [86, 87] les scores de sorties, le rapport des meilleurs scores, la sélectivité, l'entropie négative, etc. sont utilisés dans un système de reconnaissance de mots. Dans [66] pour la reconnaissance de chaînes de chiffres, les auteurs font du rejet d'ambiguïté en utilisant la différence des probabilités des meilleures chaînes en réponse. Dans [26] un k-PPV est utilisé avec une distance élastique (DTW), le rejet se fait avec un seuil sur cette distance.

Une solution pour abstraire ces différentes sources d'informations est d'utiliser la notion de *fonctions de confiance* introduite dans [28, 32]. Ces fonctions permettent d'abstraire l'information utilisée pour définir l'option de rejet. De la même manière qu'avec les probabilités, plus la confiance est faible, plus il faut rejeter la forme. On peut alors définir des options de rejet avec un ou plusieurs seuils en fonctions du nombre de fonctions de confiance [28, 32, 86, 87]. Nous présentons dans la section 3.1.1 différentes fonctions de confiance.

2.3.1.4 Apprentissage des seuils

La difficulté pour définir une option de rejet à seuils multiples est d'apprendre ces seuils. En effet avec une option de rejet utilisant la règle de Chow, il suffisait de dessiner la courbe qui donne le taux d'erreur et le taux de rejet en fonction du paramètre t pour choisir la valeur de t qui satisfasse le compromis voulu. Avec le rejet à seuils multiples, pour un même taux de rejet, plusieurs combinaisons de seuils sont possibles. Fumera [40] propose un algorithme de maximisation de $A(T_1, ..., T_k)$ sous la contrainte que le rejet reste plus faible que désiré. Dans [81] les auteurs proposent d'utiliser un algorithme d'optimisation stochastique à base d'une nuée de particules (*Particule Swarm Optmisation*). Ces deux algorithmes seront détaillés dans la section 3.2.1.

2.3.2 Trois autres architectures

L'utilisation de seuils n'est pas la seule solution pour définir une option de rejet. On peut distinguer trois autres architectures principales. En effet le problème du rejet peut être vu comme un problème de classification et donc toutes les solutions du domaine de la reconnaissance de formes sont possibles.

2.3.2.1 Utiliser une classe de rejet

Les formes à rejeter peuvent être considérées comme appartenant à une classe spécifique appelée classe de rejet [65, 80] ou modèle de rebut (*garbage model*) [114]. Cette classe est alors ajoutée dans le problème de classification avec les autres classes et il faudra en tenir compte pendant l'apprentissage.

2.3.2.2 Utiliser un classifieur spécialisé

Le rejet peut aussi être décidé par un classifieur spécialisé à deux classes (rejet et acceptation) [115]. Dans ce cas, on distingue le *classifieur de rejet* qui décide si la forme est acceptée ou rejetée et le *classifieur principal* qui reconnaît les formes acceptées. Dans [115] le classifieur de rejet utilise le même espace que le classifieur principal. Cette stratégie permet d'utiliser des classifieurs différents utilisant les mêmes informations. Dans cet exemple, le classifieur de rejet utilise une description intrinsèque des connaissances par l'utilisation d'une composition de Gaussienne alors que le classifieur principal utilise des capacités discriminantes pour mieux classer les formes.

2.3.2.3 Utiliser un classifieur spécialisé sur les fonctions de confiance

Le classifieur de rejet peut aussi être basé sur l'espace des fonctions de confiance au lieu de l'espace des caractéristiques du classifieur principal. Dans [86, 87] le classifieur de rejet utilise l'espace des fonctions de confiance pour décider du rejet ou non des formes. Ce choix permet d'utiliser les connaissances sur la qualité de la classification apportée par les fonctions de confiance. Normalement dans cet espace, les données à accepter et les données à refuser sont bien séparables si les fonctions de confiance sont efficaces.

2.3.3 Architectures connexes

Ces quatre architectures ne sont pas les seules possibles. Par exemple l'état de l'art sur la détection de nouveautés (comparable au rejet de distance) [69, 70] en décrit plusieurs. Dans [70, 109] certaines techniques cherchent à fermer les frontières de décision des classifieurs pour permettre un rejet de distance. Par exemple, en utilisant pendant l'apprentissage d'un MLP des contre-exemples avec des scores objectifs à zéro, il est possible d'obtenir des scores faibles pour ce qui doit être rejeté et ainsi fermer les frontières de décision. Le rejet est ensuite décider par un seuil. Cette technique correspond à l'utilisation d'une classe de rejet dans le classifieur principal conjointement à l'utilisation de seuils.

L'option de rejet peut aussi être introduite au coeur même du classifieur. Par exemple dans [38] les auteurs proposent un SVM où toutes les formes qui sont à l'intérieur de la marge (score entre -1 et 1) sont rejetées puisqu'il peut y avoir incertitude, le rejet d'ambiguïté est donc pris en compte dans l'apprentissage du SVM. Dans [105] c'est le rejet de distance qui est introduit dans le classifieur : les *Support Vector Data Description* (SVDD) permettent de décrire les données d'apprentissage sur le même principe que les SVM mais il est possible d'apprendre les SVDD avec ou sans contre-exemples. L'inconvénient des SVDD est que ce sont des classifieurs à deux classes (acceptation ou rejet) et qui ne permettent donc pas de faire de la classification en cas d'acceptation.

Dans [104] le classifieur est composé d'un ensemble de reconnaisseurs bi-classes spécialisés pour chaque classe. Chaque classifieur apprend à discriminer une des classes parmi les autres classes et les rebuts. Lors de la classification, la classe résultat est celle du classifieur qui a obtenu le meilleur score, le rejet est réalisé en considérant deux seuils : un seuil minimum sur le meilleur score pour le rejet de distance et un seuil sur le second score pour le rejet de confusion. Cette architecture correspond à l'utilisation d'un seuil sur les scores de classification mais étant utilisée avec un type de classifieur spécial et combinant les deux natures de rejets dans la même option.

Dans [20] l'objectif du rejet est de séparer les suites de chiffres des autres parties du texte (caractères, mauvaises segmentations, ...). Pour cela un classifieur de chiffres (un MLP) est appris puis pendant la reconnaissance on déduit la probabilité d'appartenir à la classe rejet à partir du score de la meilleure classe. Cette correspondance est faite par une table d'association (*LUT, Look Up Table*) apprise sur une base de chiffres et de non-chiffres. Les scores de chaque classe sont ensuite considérés pour la décision. Cette architecture particulière est comprise entre l'utilisation d'un classifieur externe utilisant les scores de sortie (la LUT) et l'insertion d'une classe de rejet dans le classifieur (tous les scores sont ensuite considérés).

Il est aussi possible d'utiliser la notion d'anti-modèle pour évaluer la qualité de la reconnaissance [59, 72]. Un anti-modèle essaie de représenter tout ce qui n'est pas le modèle. Il y a généralement un anti-modèle par classe. Cet anti-modèle peut prendre plusieurs formes suivant la nature des classifieurs utilisés. Le score de confiance de la reconnaissance est évalué en comparant les probabilités du modèle et de l'anti-modèle. La difficulté de cette approche est de définir l'anti-modèle. L'utilisation d'anti-modèles se rapproche d'une architecture utilisant une classe de rejet soit dans le classifieur

spécialisé soit dans le classifieur principal puisque dans ces deux cas on cherche à modéliser ce qui n'est pas à reconnaître.

Les classifieurs à une classe (*One Class Classifiers*) permettent de modéliser chaque classe à reconnaître de façon intrinsèque. Une option de rejet de distance peut alors être définie comme dans [106] sans avoir aucune connaissance sur les formes à rejeter (*problème* $\emptyset \rightarrow B$).

Si on considère l'architecture utilisant un classifieur spécialisé, il est aussi possible d'utiliser un jeu de caractéristiques différent de celui utilisé par le classifieur principal (différent aussi des fonctions de confiance). Cette approche suppose soit une technique de sélection de caractéristiques soit une expertise sur la qualité des caractéristiques pour le rejet considéré.

Toutes ces architectures correspondent soit à une des quatre architectures présentées précédemment, soit ne sont pas utilisables dans notre contexte applicatif qui nécessite de limiter les ressources utilisées. Dans cette étude, nous nous focaliserons donc sur la description et sur la comparaison de ces quatre architectures.

2.3.4 Notre formalisation des quatre architectures

Comme nous l'avons vu, quatre architectures principales pour systèmes de reconnaissance avec les options de rejet peuvent être distinguées. Elles sont illustrées par la figure 2.3. Elles ont toutes le même fonctionnement et objectif final : le système prend en entrée les caractéristiques de la forme à reconnaître et rend en sortie soit la décision de rejet, soit la classe reconnue. Cette section a pour but de formaliser et comparer ces architectures de manière à bien en voir les différences.

2.3.4.1 Classe de rejet

Dans cette architecture notée RC (décrite par la figure 2.3(a)) une classe de rejet est ajoutée au problème de reconnaissance. Cette classe de rejet est considérée de la même manière que les autres classes. Il y a donc rejet si le score obtenu par la classe de rejet est supérieur aux scores obtenus par les autres classes. Dans certaines applications, le classifieur principal existe déjà, il faudra donc le réapprendre pour intégrer la classe de rejet, ce qui peut-être un inconvénient de cette architecture.

Pour *le rejet de distance*, les contre-exemples forment une classe comme les autres. Donc les bases d'apprentissage D_E et D_A sont fusionnées en D_{E+A}. Cette base sert ensuite pour l'apprentissage du nouveau classifieur avec classe de rejet. Cette solution a besoin d'avoir des contre-exemples pour l'apprentissage de la classe de rejet. Donc si D_A est vide dans le *problème* $A \rightarrow B$ (*problème* $\emptyset \rightarrow B$), cette architecture RC n'est pas possible.

Pour *le rejet d'ambiguïté*, la difficulté est de créer une base d'apprentissage des erreurs que va commettre le futur classifieur. Pour cela il faut d'abord apprendre un classifieur normal sur D_E s'il n'existe pas déjà. Ensuite il faut ré-étiqueter les erreurs comme des contre-exemples de type A pour constituer une base d'apprentissage $D_{E'+A}$ qui contient les classes à reconnaître et les erreurs à rejeter. Cette base sert alors

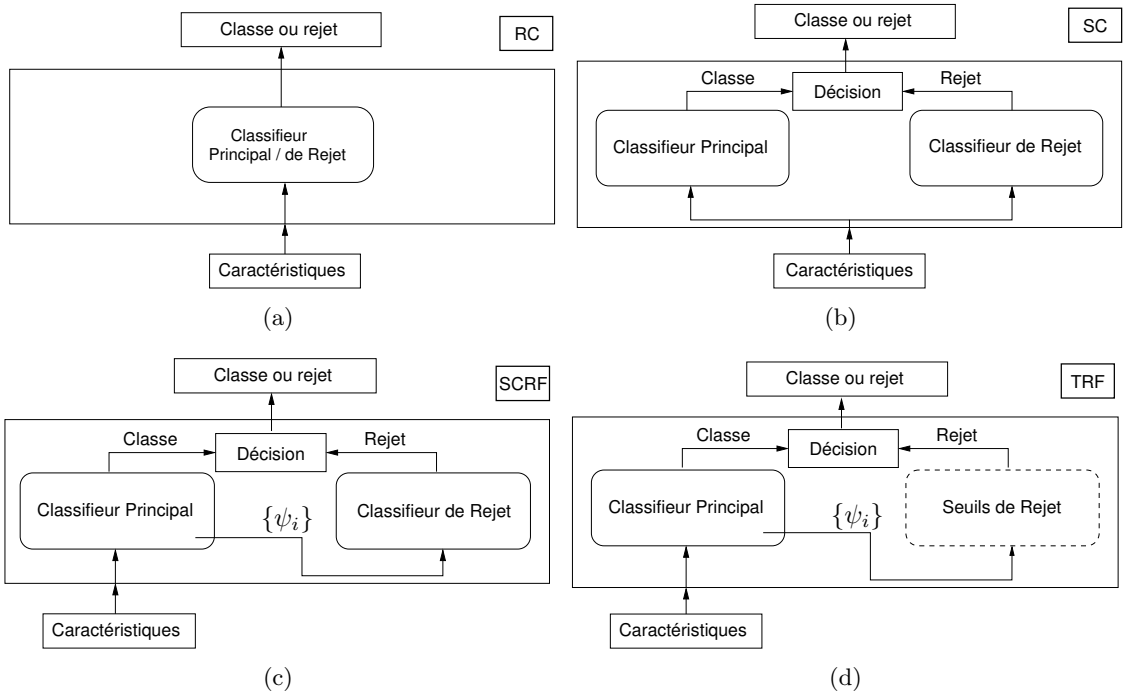


FIG. 2.3 – Les quatre architectures de systèmes avec option de rejet : utilisant (a) une classe de rejet dans le classifieur principal (RC) ; (b) un classifieur spécialisé dans l'espace des caractéristiques (SC) ; (c) un classifieur spécialisé dans l'espace des fonctions de confiance $\{\psi_i\}$ (SCRF) ; (d) des seuils sur les fonctions de confiance $\{\psi_i\}$ (TRF).

pour apprendre le classifieur avec la classe de rejet. Le problème est que même si les erreurs commises par le premier classifieur sont bien rejetées, de nouvelles erreurs vont apparaître à cause des confusions entre les classes principales et la classe de rejet. De plus, les erreurs de classification sont éparpillées dans tout l'espace des caractéristiques le long des frontières de décision, ce qui rend la définition d'une classe de rejet difficile.

Dans cette architecture, les classifieurs avec de bonnes capacités de discrimination seront préférés. En effet l'efficacité du rejet repose entièrement sur les capacités du classifieur à distinguer les classes entre elles, surtout que l'insertion de la classe de rejet rend le problème de classification plus compliqué. De plus les performances du système peuvent diminuer notablement par rapport à celles du classifieur sans rejet, puisque les formes bien reconnues par le système sans rejet peuvent être mal reconnues par le système avec rejet.

2.3.4.2 Classifieur spécialisé

Dans cette architecture appelée architecture avec un classifieur spécialisé (SC), un classifieur indépendant à deux classes est chargé de la décision de rejet (Figure 2.3(b)).

Ce classifieur utilise le même espace de caractéristiques que le classifieur principal. Il y a rejet si le classifieur de rejet classe la forme dans la classe rejet. Si la forme est acceptée, alors c'est le classifieur principal qui décide de la classe d'appartenance.

Séparer la décision de rejet du classifieur principal en utilisant un classifieur de rejet présente quelques avantages. D'abord, le classifieur principal n'est pas modifié s'il existe déjà et la complexité du problème de reconnaissance n'est pas augmentée par l'ajout d'une nouvelle classe. De plus les deux classifieurs peuvent tirer avantage de familles de classifieurs différentes. Par exemple dans [115], un MLP est utilisé comme classifieur principal pour ses capacités discriminantes et un OGMM (*Orthogonnal Gaussian Mixture Model*) est utilisé comme classifieur de rejet de distance pour ses capacités de description intrinsèque.

Pour *le rejet de distance*, cette architecture a les mêmes inconvénients que l'architecture RC. En effet, si D_A est vide, le classifieur de rejet ne pourra pas être appris. La seule solution dans ce cas est d'utiliser un classifieur qui n'a pas besoin des deux classes pour être appris (comme les OGMM dans [115]). De plus pour le *problème* $A \rightarrow B$, le classifieur de rejet doit avoir une bonne capacité de généralisation pour pouvoir rejeter efficacement des formes non rencontrées pendant l'apprentissage. Il donc faudra préférer un classifieur très discriminant comme classifieur principal et un classifieur avec une modélisation intrinsèque et de bonnes capacités de généralisation pour le classifieur de rejet.

Pour *le rejet d'ambiguïté*, l'élaboration de la base de contre-exemples D_A n'est pas compliquée à obtenir : il suffit de classer les exemples de D_E et de séparer les formes bien reconnues des formes mal reconnues. Le premier problème est que les formes à rejeter sont éparpillées dans tout l'espace des caractéristiques, juste à côté des formes à accepter puisque par définition les erreurs sont proches des frontières de décision. La séparation des deux classes va donc être difficile. De plus il y a un problème de représentativité des données. En effet, un classifieur principal performant n'aura que peu de choses à rejeter, et la quantité de données à accepter sera beaucoup plus importante que la quantité de donnée à rejeter. Il faudra donc utiliser un classifieur de rejet capable de gérer ce déséquilibre ou bien ré-équilibrer les bases avant apprentissage.

2.3.4.3 Classifieur spécialisé utilisant les fonctions de confiance

Les fonctions de confiance notées $\{\psi_i\}$ peuvent être utilisées directement comme caractéristiques d'un classifieur de rejet. Cette architecture utilisant un classifieur spécialisé sur les fonctions de confiance est notée SCRF (Speciallized Classifier on Reliability Functions) et décrite par la figure 2.3(c). Dans cet espace de caractéristiques, les données à rejeter et les données à accepter sont normalement plus facilement séparables. De plus il est possible d'utiliser un classifieur de rejet avec de bonne capacités discriminantes (comme dans [87]) puisque les fonctions de confiance ont déjà intégré les informations intrinsèques ou d'ambiguïté.

Cette architecture peut à première vue avoir les mêmes inconvénients que l'architecture SC puisqu'un classifieur de rejet est utilisé.

En effet, pour *le rejet de distance*, cette architecture va être très performante pour

le *problème A→A* mais peut avoir des difficultés de généralisation dans le *problème A→B*.

Pour *le rejet d'ambiguïté*, on retrouvera la difficulté liée au déséquilibre des deux classes acceptation/rejet.

Néanmoins, ces différents problèmes de généralisation peuvent être réduit par la généralisation apportée par les fonctions de confiance, par exemple en réduisant l'espace des caractéristiques ou en regroupant les formes à rejeter plutôt qu'elles soient dispersées dans l'espace des caractéristiques.

2.3.4.4 Seuils sur les fonctions de confiance

Une autre possibilité pour utiliser les fonctions de confiance est d'utiliser directement leur interprétabilité. Plus bas est la confiance, plus il faut rejeter la forme. Une approche simple consiste à utiliser des seuils pour décider du rejet. Ces seuils représentent les bornes inférieures de la confiance que l'on peut accorder. Cette architecture notée TRF (pour *Threshold on Reliability Functions*) est décrite par la figure 2.3(d).

Cette architecture simple a plusieurs avantages. D'abord elle est facilement interprétable et possède une bonne capacité de généralisation. En effet ces seuils de rejet peuvent être vus comme un classifieur simple avec très peu de paramètres, ce qui permet une bonne généralisation. De ce point de vue et aussi grâce à l'abstraction des fonctions de confiance, cette architecture devrait avoir de bons résultats sur le *problème A→A* et le *problème A→B*. De plus elle est très légère puisqu'il n'y a que les fonctions de confiance à évaluer puis à comparer aux seuils, donc pas de classifieur de rejet complexe à utiliser comme dans les architectures SC et SCRF. Enfin le classifieur principal n'est pas modifié comme dans l'architecture RC, donc la difficulté du problème n'est pas augmentée. Ce sont pour ces raisons que nous avons choisi d'étendre cette architecture comme nous le verrons dans le chapitre 3.

Pour cette architecture TRF, la difficulté est d'apprendre automatiquement les seuils de l'option de rejet. C'est pourquoi nous présentons dans la section 3.2 des algorithmes d'apprentissage automatique des seuils.

2.4 L'importance des classifieurs utilisés

Nous avons vu que les différentes architectures ont besoin de classifieurs en différents endroits des systèmes. Les propriétés des classifieurs utilisés peuvent avoir des effets sur l'option de rejet par différentes voies selon le choix de l'architecture et le contexte d'apprentissage. Peu de publications étudient les effets de ces choix.

2.4.1 État de l'art

Dans [66], les auteurs proposent un système de reconnaissance de chaînes de chiffres et utilisent une option de rejet d'ambiguïté en fin de reconnaissance. Ils testent leur système avec plusieurs classifieurs différents et on peut remarquer que les résultats sont

très différents d'un classifieur à l'autre, non seulement sur les performances brutes mais aussi sur l'efficacité du rejet.

Il est donc possible d'utiliser cette influence du choix du classifieur pour obtenir de meilleurs performances. Par exemple dans [115] l'auteur utilise un OGMM (*Orthognonal Gaussian Mixture Model*) comme classifieur de rejet puis un MLP comme classifieur principal. Ainsi le classifieur de rejet utilise sa modélisation intrinsèque pour définir le domaine de définition des classes et le classifieur principal utilise ses capacités discriminantes pour la classification.

2.4.2 Discussions

Dans un classifieur deux propriétés peuvent influencer une option de rejet : sa *capacité de classification* et la *disponibilité d'information sur la qualité de la classification.*

La capacité de classification est importante pour le classifieur principal pour pouvoir discriminer entre les classes cibles (et si nécessaire la classe rejet pour l'architecture RC) et pour le classifieur de rejet pour discriminer les formes à accepter des formes à rejeter (architectures SC et SCRF).

L'information sur la qualité de la classification est importante si l'option de rejet utilise des fonctions de confiance (architectures SCRF et TRF). En effet cette information permettra suivant sa nature de savoir si la forme classée appartient à une des classes apprises pour le rejet de distance ou de savoir s'il y a une possible confusion entre deux classes pour le rejet d'ambiguïté.

Généralement ces propriétés sont liées à deux familles de classifieurs : les classifieurs avec une modélisation *intrinsèque* et les classifieurs avec une modélisation *discriminante* (voir section 1.1.2). La première nous intéressera surtout pour le rejet de distance et la seconde pour le rejet d'ambiguïté. Il n'y a pas de frontière rigide entre ces deux familles car certains classifieurs discriminants rendent disponibles des informations concernant la qualité de classification et certaines modélisations intrinsèques ont des capacités discriminantes intéressantes.

Dans cette étude nous comparons trois types de classifieurs très populaires dans le domaine de la reconnaissance des formes et particulièrement en reconnaissance d'écriture manuscrite. Ils ont été choisis pour représenter différents comportements vis à vis des deux propriétés : description et discrimination. Ainsi nous pourrons comparer les effets du choix du type de classifieur sur les différentes architectures d'options de rejet. Ces classifieurs sont les RBFN (voir section 1.2.4), les MLP (voir section 1.2.2) et les SVM (voir section 1.2.3). Les RBFN ont une description intrinsèque des classes qui pourra être utile pour les architectures utilisant les fonctions de confiance. En effet leur fonctions à base radiale leur offre une bonne capacité de généralisation et une bonne approximation des probabilités a posteriori d'appartenance à chaque classe. Les MLP sont des classifieurs purement discriminants qui pourront servir dans les architectures avec un classifieur spécialisé. Les sorties des MLP n'étant pas fiable en dehors des zones d'apprentissage, les MLP ne seront pas très performants dans le rejet de distance comme expliqué dans [42, 109]. Les SVM sont très réputés pour leurs capacités discriminantes

et donc seront utiles dans les architectures utilisant un classifieur de rejet. Mais ils peuvent aussi avoir des capacités descriptives avec l'utilisation de noyaux Gaussiens ce qui nous permettra de les utiliser aussi comme classifieur principal avec des fonctions de confiance.

2.5 L'évaluation du rejet : état de l'art

Pour comparer efficacement les différentes options de rejet pour les deux natures de rejet, nous présentons ici les outils classiques d'évaluation des options de rejet. Dans cet état de l'art l'évaluation des deux types de rejet est différente, nous commencerons donc par étudier le rejet d'ambiguïté puis le rejet de distance. Dans la section suivante nous verrons comment nous avons étendu ces outils à notre modélisation du rejet.

2.5.1 Évaluation du rejet d'ambiguïté

Le but de l'évaluation d'une option de rejet d'ambiguïté est de vérifier que seul les erreurs de classification sont rejetées. Pour cela les éléments d'une base de test D_E de N_{Tot} exemples à reconnaître sont triés en trois groupes :

- exemples acceptés et correctement classés : N_{Corr} ;
- exemples acceptés mais mal classés : N_{Err} ;
- exemples rejetés : N_{Rej}.

Directement à partir de ces valeurs, qui dépendent des paramètres (ρ) d'apprentissage du rejet, on peut calculer quatre taux :

- le taux d'erreur $E(\rho)$ défini par l'équation (2.5) ;
- le taux de rejet $R(\rho)$ défini par l'équation (2.6) ;
- la performance $Perf(\rho)$ défini par l'équation (2.7) ;
- la fiabilité $Fiab(\rho)$ défini par l'équation (2.8).

On peut remarquer que ces taux sont liés par les équations (2.9) et (2.10).

$$E(\rho) = \frac{N_{Err}}{N_{Tot}} \tag{2.5}$$

$$R(\rho) = \frac{N_{Rej}}{N_{Tot}} \tag{2.6}$$

$$Perf(\rho) = \frac{N_{Corr}}{N_{Tot}} \tag{2.7}$$

$$Fiab(\rho) = \frac{N_{Corr}}{N_{Corr} + N_{Err}} \tag{2.8}$$

$$R(\rho) + E(\rho) + Perf(\rho) = 1 \tag{2.9}$$

$$Fiab(\rho) = \frac{Perf}{1 - R(\rho)} \tag{2.10}$$

Il s'agit de minimiser en même temps le taux d'erreur et le taux de rejet, donc de maximiser la performance globale. A chaque option de rejet apprise correspond un *point de fonctionnement* $[E(\rho), R(\rho)]$ qui dépend de l'architecture choisie, du type des

classifieurs et surtout des paramètres (ρ) d'apprentissage du rejet. Par exemple pour l'option de rejet de Chow ces taux dépendent de la valeur du seuil t (voir l'équation (2.1)). Comme expliqué dans la section 2.3, l'optimisation du taux de rejet et du taux d'erreur est un compromis appelé le compromis *Erreur/Rejet*. Le compromis optimum est étudié dans la publication de Chow [23] mais il se limite au classifieur Bayesien idéal. Plusieurs articles comme [40, 49] ont étudié ce compromis dans le cas général.

2.5.1.1 Les courbes Erreur/Rejet

Pour comparer entre elles les options de rejet, la courbe Erreur/Rejet (*courbe ER*) est utilisée [23]. Elle se construit en faisant varier les paramètres (ρ) d'apprentissage et en mesurant pour chaque valeur les taux $E(\rho)$ et $R(\rho)$ pour construire la courbe correspondante. Cette courbe permet de choisir le point de fonctionnement du rejet qui correspond le mieux au besoin de l'application cible. La figure 2.4 présente un exemple de courbe ER.

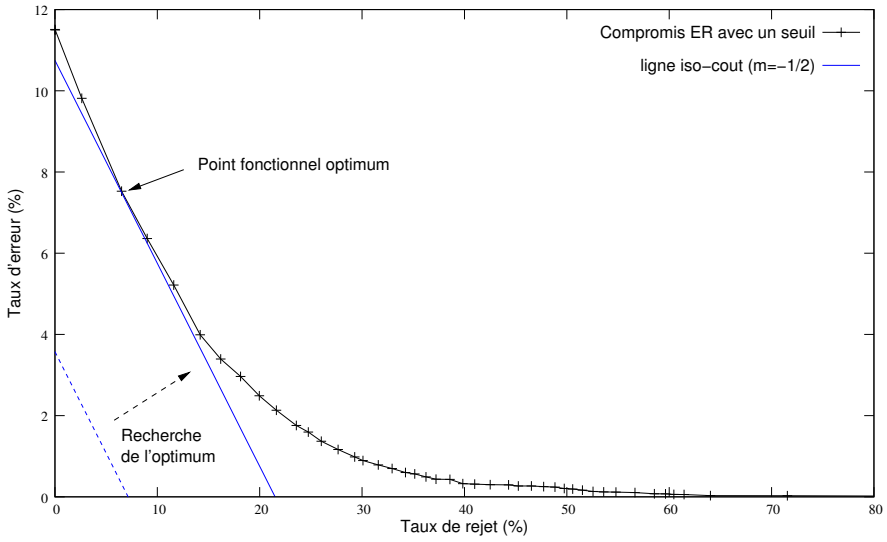


FIG. 2.4 – Exemple de courbes Erreur/Rejet (pour un RBFN avec un seuil sur la différence relative des deux meilleurs scores) avec un exemple de ligne d'iso-coût ($m = -1/2$).

Le classifieur parfait se trouve à l'origine de ce graphique (0% de rejet et 0% d'erreur). Mais tant que ce classifieur n'existe pas, il faudra choisir parmi les autres compromis le point de la courbe correspondant à l'option la plus intéressante. Pour aider l'utilisateur dans ce choix il est possible de donner un coût de l'erreur ($C_E > 0$), un coût du rejet ($C_R > 0$, généralement $C_E > C_R$) et éventuellement un coût de la bonne réponse ($C_C \leq 0$). Dans les applications industrielles ces coûts sont facilement éva-

luables, mais moins dans les applications du type IHM. On peut alors tracer la courbe du coût total en fonction des paramètres (ρ) :

$$C(\rho) = C_E.E(\rho) + C_R.R(\rho) \tag{2.11}$$

On voit que le coût des bonnes réponses n'apparaît pas dans l'équation (2.11). Effet, les équations (2.12) à (2.15) montrent qu'il n'est pas nécessaire de définir ce coût puisque pour un C_C donné, il existe un couple (C'_E, C'_R) qui donne le même compromis.

$$\begin{aligned} C(\rho) &= C_E.E(\rho) + C_R.R(\rho) + C_C.Perf(\rho) && (2.12)\\ C(\rho) &= C_E.E(\rho) + C_R.R(\rho) + C_C.(1 - E(\rho) - R(\rho)) && (2.13)\\ C(\rho) &= (C_E - C_C).E(\rho) + (C_R - C_C).R(\rho) + C_C && (2.14)\\ C'(\rho) &= C'_E.E(\rho) + C'_R.R(\rho) && (2.15) \end{aligned}$$

Cette approche utilisant des coûts est très utilisée pour l'apprentissage du compromis ER [28, 32, 38, 53, 97]. Cette notion de coût peut se retrouver sur la courbe ER en dessinant des courbes d'iso-coût. En effet l'équation (2.11) se traduit dans l'espace ER par une droite de pente m :

$$m = -\frac{C_R}{C_E} \tag{2.16}$$

La hauteur de cette droite donnant la valeur du coût (l'intersection avec l'axe des ordonnées). Cette droite permet de trouver le point de fonctionnement minimisant le coût pour un couple (C_R, C_E) donné : tracer la droite de pente m proche de l'origine, puis la faire glisser jusqu'à ce qu'elle tangente la courbe ER, le point de contact est l'optimum. La figure 2.4 donne un exemple de droite d'iso-coût pour $m = -1/2$. Dans le cas idéal de l'option de rejet de Chow [23], cette pente est égale à la valeur de $-t$ (voir équation (2.2)).

2.5.1.2 Autres compromis

Il existe d'autres compromis utilisés pour évaluer le rejet d'ambiguïté. Un premier exemple est l'utilisation des courbes DET [72, 86][1] qui considèrent deux types d'erreur : le taux E_{type1} de ce qui est accepté alors que c'est mal reconnu et le taux E_{type2} de ce qui est rejeté alors que c'est bien reconnu. Pour définir ces taux il faut différencier dans les N_{Rej} exemples rejetés les N_{Rej}^{Corr} exemples rejetés qui auraient été bien reconnus et les N_{Rej}^{Err} exemples rejetés qui auraient été mal reconnus. On peut alors définir les deux types d'erreurs :

$$\begin{aligned} E_{type1}(\rho) &= \frac{N_{Err}}{N_{Err} + N_{Rej}^{Err}} && (2.17)\\ E_{type2}(\rho) &= \frac{N_{Rej}^{Corr}}{N_{Corr} + N_{Rej}^{Corr}} && (2.18) \end{aligned}$$

[1] Dans [86] ces courbes sont appelées courbe ROC, mais ce sont des courbes DET.

La courbe DET est la courbe donnant E_{type1} en fonction de E_{type2}, la figure 2.5(a) en montre un exemple. Cette courbe correspond plus à un rejet de distance, comme nous le verrons dans la section suivante, même si elle permet de comparer efficacement les options de rejet d'ambiguïté. En effet, appliquée au rejet d'ambiguïté, elle considère les erreurs et bonnes reconnaissances comme deux classes séparées. Par contre, il devient alors difficile à l'utilisateur d'évaluer l'effet du rejet sur l'applicatif.

Certains auteurs [40] utilisent le compromis *Fiabilité/Rejet* pour présenter les résultats du rejet d'ambiguïté. Dans ce cas il s'agit de minimiser le rejet et de maximiser la fiabilité. Le classifieur optimal est en haut à gauche du graphe (0% de rejet et 100% de fiabilité). La figure 2.5(b) donne un exemple de ce type de courbe pour le même exemple que la courbe 2.4. L'avantage d'utiliser la fiabilité est de pouvoir apprécier l'effet du rejet sur la réponse du classifieur sans prendre en compte ce qui est rejeté. En effet puisque les exemples rejetés n'interviennent plus dans la réponse du classifieur, il peut être intéressant de ne pas les prendre en compte. Par exemple, dans les applications comme la reconnaissance de chèques, ce qui compte c'est de savoir que presque tout ce qui est reconnu est bien reconnu, quelque soit le taux de rejet global.

2.5.2 Évaluation du rejet de distance

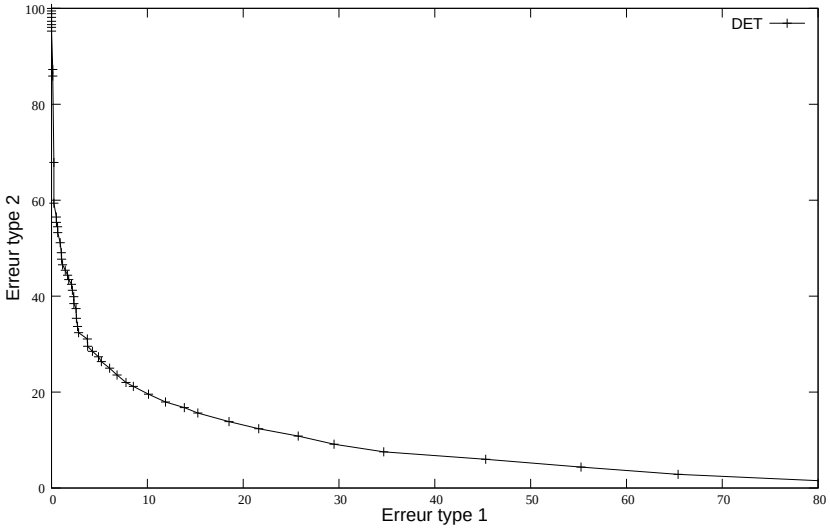
Pour évaluer une option de rejet de distance nous avons besoin de deux bases de test :

- D_E : une base d'exemples à accepter de taille N_E,
- D_R : une base de contre-exemples à rejeter de taille N_R.

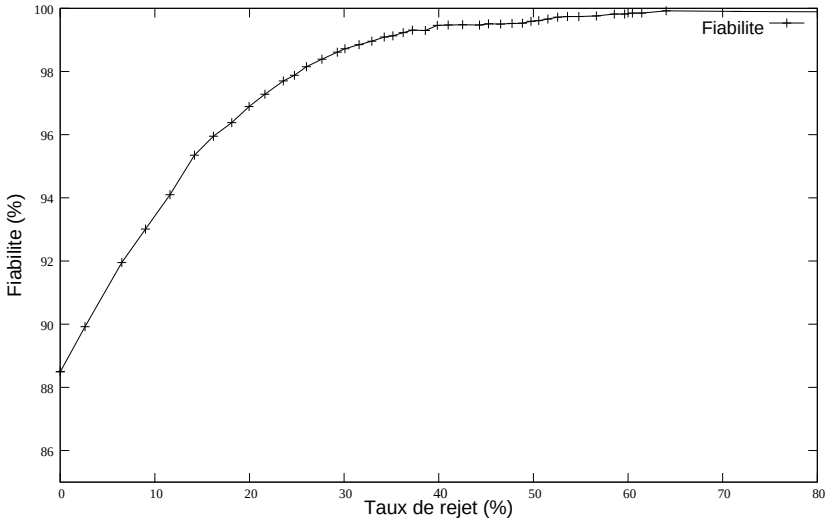
Les données de chacune de ces bases peuvent être soit acceptées soit rejetées. Le but du rejet de distance étant d'accepter tous les exemples de D_E et de rejeter tous les contre-exemples de D_R. Comme l'option de rejet parfait n'est généralement pas atteinte, il faut faire un compromis entre ces deux objectifs. Pour mesurer ce compromis on mesure la proportion d'exemples de D_E qui sont acceptés par le taux de vraie acceptation (*True Acceptance Rate* : TAR) et de ceux qui sont refusés par le taux de faux rejet (*False Reject Rate* : FRR). De même on mesure dans la base D_R la proportion de contre-exemples qui sont acceptés par le taux de fausse acceptation (*False Acceptance Rate* : FAR) et la proportion de contre-exemples qui sont rejetés par le taux de vrai rejet (*True Reject Rate* : TRR). La table 2.1 récapitule ces quatre mesures.

Tab. 2.1 – Les différentes mesures pour évaluer le rejet de distance.

		Acceptés		Rejetés	
	Nombre	Nombre	Mesure	Nombre	Mesure
Exemples	N_E	N_E^A	$TAR = N_E^A/N_E$	N_E^R	$FRR = N_E^R/N_E$
Contre-exemples	N_R	N_R^A	$FAR = N_R^A/N_R$	N_R^R	$TRR = N_R^R/N_R$



(a)



(b)

FIG. 2.5 – Exemple de courbes (a) DET et (b) Fiabilité/Rejet pour le même problème que la courbe de la figure 2.4.

2.5.2.1 Les courbes ROC

Pour comparer entre elles les options de rejet de distance, on considère le TAR et le FAR de chacune d'elles. Dans cet espace (TAR, FAR) l'option de rejet parfait est située en haut à gauche (100% de vraie acceptation et 0% de fausse acceptation). Ces taux dépendent en réalité des paramètres ρ d'apprentissage du rejet. Dans ce cas on peut tracer la courbe paramétrique continue $(TAR(\rho), FAR(\rho))$. Cette courbe est

appelée *courbe ROC* (*Receiver Operating Characteristics*) [36]. L'avantage des courbes ROC est de pouvoir comparer visuellement les classifieurs sans être perturbé par des distributions inégales entre les classes ou les coûts différents entre les erreurs.

Pour les options de rejet avec un paramétrage discontinu cette courbe n'est pas accessible directement. On commence par placer les points $(TAR(\rho_1), FAR(\rho_1))$ à $(TAR(\rho_n), FAR(\rho_n))$ correspondants aux différents paramètres possibles. On peut alors considérer le problème d'optimisation du rejet comme un problème d'optimisation multi-objectifs, les objectifs étant de minimiser le FAR et de maximiser le TAR. L'algorithme de Pareto [85] est une technique classique pour sélectionner dans un ensemble de solutions celles qui optimisent deux ou plusieurs critères. Cet algorithme conserve seulement les points d'un ensemble pour lesquels il n'y a pas de points meilleurs sur tous les critères en même temps. On obtient alors l'ensemble des options de rejet les plus performantes parmi celles disponibles. Pour obtenir une courbe ROC il faut alors relier les points entre eux. Cette opération est réalisable par interpolation entre deux classifieurs avec option de rejet comme expliqué dans [36][2].

Pour mieux représenter les performances atteignables par une option de rejet sur la courbe ROC obtenue, il faut considérer la courbe convexe qui englobe la courbe expérimentale [36]. En effet on peut continuer à faire l'interpolation entre tous les points de la courbes ROC et à chaque fois que la courbe est convexe, l'interpolation aura de meilleures performances. Nous utiliserons donc dans cette étude cette courbe convexe.

Pour comparer deux courbes ROC il y a plusieurs possibilités. La première et la plus utilisée est l'aire sous la courbe appelée *AUROCC* (*Area Under ROC Curve*) [16]. Cette aire est calculée par l'intégrale de l'équation (2.19) qui considère le TAR comme une fonction du FAR.

$$AUROCC = \int_0^1 TAR(FAR)\, dFAR \tag{2.19}$$

Pour le calcul expérimental de l'aire, la courbe étant discrète et interpolée, on utilise donc pour le calcul de l'aire sous la courbes ROC une intégrale trapézoïdale (équation (2.20), avec N le nombre de points sur la courbes).

$$AUROCC = \sum_{i=0}^{N-1} (TAR_{i+1} + TAR_i)(FAR_{i+1} - FAR_i)/2 \tag{2.20}$$

D'après cette équation, on peut considérer que l'AUROCC correspond à la moyenne du TAR sur toutes les valeurs de FAR. Une option de rejet est meilleure qu'une autre si elle a en moyenne un meilleur taux de vraie acceptation.

La figure 2.6 donne un exemple artificiel complet de calcul des courbes à partir des points de fonctionnement expérimentaux. La figure 2.7 présente les courbes ROC de

[2] N'importe quelle option de rejet entre deux points de fonctionnement peut être simulé en choisissant aléatoirement pour chaque décision de rejet une des deux options. La position du point généré dépend de la probabilité de choisir une des deux options.

deux options de rejet issue des expérimentations réelles. On peut voir ainsi la construction de la courbe à partir des données expérimentales avec un paramétrage continu ou discret.

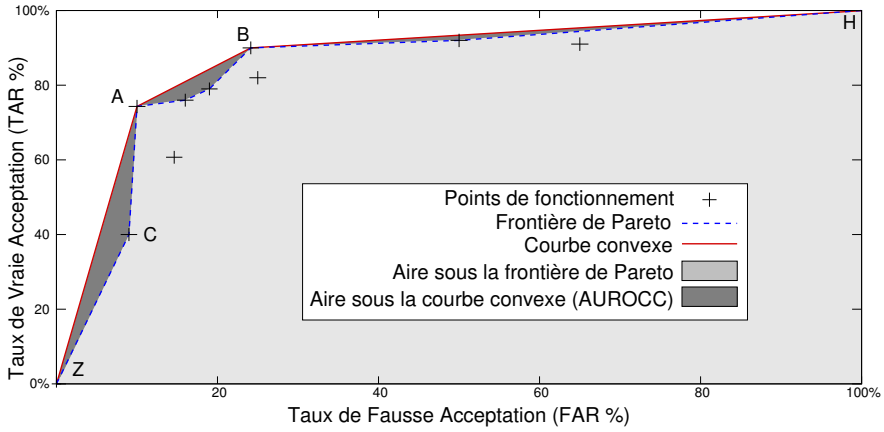


FIG. 2.6 – Exemple artificiel de courbe ROC ainsi que le front de Pareto et la courbe convexe associés. Les aires sous les courbes sont grisées.

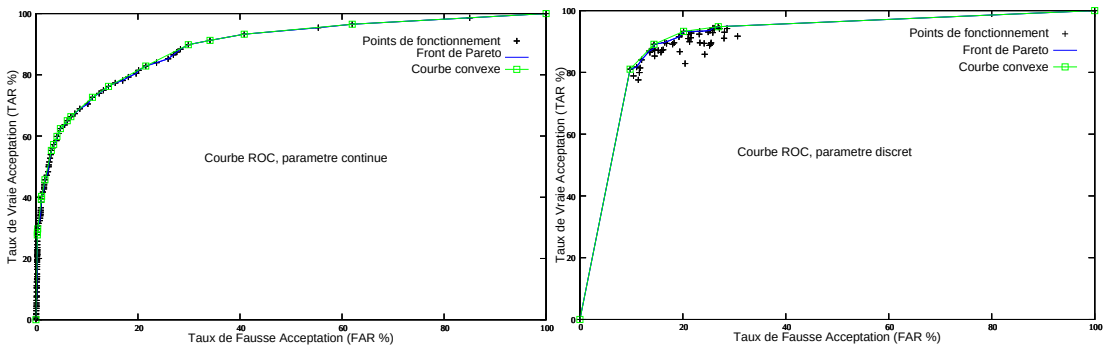


FIG. 2.7 – Exemple de courbes ROC pour deux options de rejet avec un paramétrage continu (à gauche) et un discret (à droite) ainsi que leurs fronts de Pareto et leurs courbes convexes associés.

L'utilisation de l'AUROCC a comme principal avantage de pouvoir comparer de façon globale différentes options de rejet simplement, mais c'est aussi son principal inconvénient. En effet deux courbes peuvent avoir la même AUROCC et pour autant ne pas être identiques : elles peuvent se croiser (ce qui est le cas des courbes ROC de la figure 2.7). Il faut alors considérer les besoins de l'application pour décider quelle est la meilleure pour cette application. Par exemple on peut savoir que l'application peut accepter un taux de faux rejet de plus de $FAR1$ mais pas plus de $FAR2$. Pour

départager les deux options on peut alors utiliser l'aire sous une partie de la courbe de $FAR1$ à $FAR2$:

$$subAUROCC = \int_{FAR1}^{FAR2} TAR(FAR)\, dFAR \tag{2.21}$$

Cette sous-aire sous la courbe est plus difficilement interprétable car il faut diviser cette aire par $FAR2 - FAR1$ pour obtenir le taux de vraie acceptation moyen sur cette zone. En extrême limite, $FAR2 = FAR1$ et on peut alors ne considérer qu'un seul point sur la courbe pour départager les options de rejet.

Il est aussi possible de considérer le coût d'une erreur de rejet (comme pour le rejet d'ambiguïté) et la proportion d'exemples à rejeter. En effet ces paramètres peuvent être non négligeables dans certaines applications. On définit alors le coût d'un faux rejet C_{FR} et d'une fausse acceptation C_{FA} ainsi que la probabilité *a priori* des exemples à rejeter $p(n)$ et celle des exemples à accepter $p(p)$. On introduit alors la notion de *lignes d'iso-coût* (appelées *iso-perfomance line* dans [36]). Ces lignes ont une pente définie par m :

$$m = \frac{C_{FP} p(n)}{C_{FA} p(p)}. \tag{2.22}$$

Deux points sur cette ligne ont un coût identique. Plus la ligne est haute, plus le coût total est faible. Le point de la courbe ROC qui minimise ce coût est le point tangent entre le courbe ROC et la droite de pente m la plus haute. De la même manière que sur la courbe ER pour le rejet d'ambiguïté, la recherche du point de fonctionnement optimal se fait en faisant glisser une droite de pente m depuis le point le plus en haut à gauche jusqu'au contact avec la courbe ROC, ce point tangent est le point de fonctionnement optimal.

La figure 2.8 présente un exemple de comparaison de deux options de rejet (celles présentées dans la figure 2.7). L'option 2 a une AUROCC supérieure à celle de l'option 1 (0.907 contre 0.888), mais l'option 1 est meilleure pour les fausses acceptations inférieures à 8%. Les points M1 et M2 sont respectivement les points de fonctionnement optimaux pour les deux options de rejet avec un compromis de $m = 1/2$ (soit par exemple : $C_{FP} = C_{FA}, p(p) = 2p(n)$). Pour ce compromis, l'option 2 est moins coûteuse.

Comme toutes mesures de performance, l'AUROCC dépend de la base de test utilisée, il est donc utile d'effectuer une validation croisée pour pouvoir évaluer l'incertitude de la mesure. Dans [36], il est expliqué comment construire une courbe ROC en visualisant les écarts-types sur les courbes.

2.5.2.2 Autres mesures

Il existe dans la littérature d'autres mesures de performance du rejet de distance. Ces mesures sont communes à d'autres domaines de recherche qui sont en rapport avec la détection d'information : la recherche d'information, reconnaissance du locuteur [71], ... Les autres mesures sont le *rappel* (*recall*) qui est synonyme du taux de vraie acceptation, la *précision* (*precision*) qui mesure le taux de bonnes réponses parmi les réponses

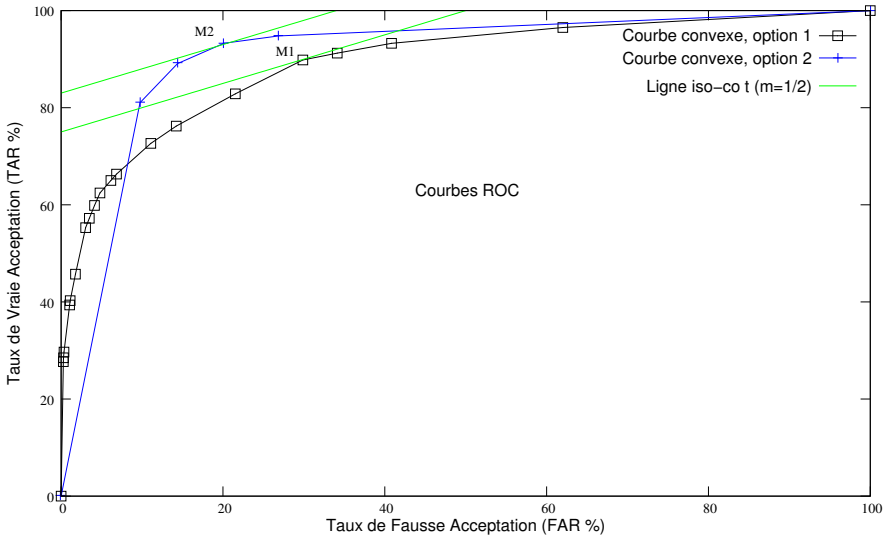


FIG. 2.8 – Courbes ROC des deux options de la figure 2.7 avec les lignes d'iso-coût pour $m = 1/2$, M1 et M2 sont respectivement les deux meilleurs points de fonctionnement pour ce compromis.

positives, la *fiabilité* (*accuracy*) qui mesure le taux de bonnes réponses globales (mesure différente de la fiabilité définie précédemment par l'équation (2.8)), le taux de *fausses alarmes* (*false alarm probability*, aussi appelé *erreur de type 1*) qui est synonyme du taux de fausse acceptation, le taux de *détection manquées* (*miss probability*, aussi appelé *erreur de type 2*) qui est le taux de rejet parmi les exemples à accepter :

$$Rappel = TAR = \frac{N_E^A}{N_E} \tag{2.23}$$

$$Précision = \frac{N_E^A}{N_E^A + N_R^A} \tag{2.24}$$

$$Fiabilité = \frac{N_E^A + N_R^R}{N_E + N_R} \tag{2.25}$$

$$Fausse\ alarme = FAR = \frac{N_R^A}{N_R} \tag{2.26}$$

$$Detections\ manquées = FRR = \frac{N_E^R}{N_E} \tag{2.27}$$

A partir de ces mesures, deux courbes classiques sont disponibles : la courbe *précision/rappel* et la courbe *détections manquées/fausses alarmes* (courbe DET, déjà présentée pour le rejet d'ambiguïté section 2.5.1.2). Les courbes précision/rappel sont plus utilisées dans le domaine de la recherche d'information comme dans [20, 21]. Nous n'avons pas utilisé cette courbe car la précision mélange les exemples et contre-exemples

acceptés et devient donc sensible à leur proportion relative dans les bases. Les courbes DET sont très proches des courbes ROC et représentent exactement la même information. Dans [71] les courbes DET sont utilisées dans le domaine du traitement de la parole pour permettre une meilleure séparation des courbes par l'usage d'une échelle logarithmique. Nous avons choisi d'utiliser les courbes ROC car elles représentent des taux d'acceptation sur les deux axes alors que les axes des courbes DET ne sont pas homogènes.

Les exemples acceptés par le rejet de distance sont ensuite classés par le classifieur principal. Même si l'objectif du rejet de distance n'est pas forcément d'éviter les erreurs de classification, ce rejet a des effets sur les performances du classifieur principal [63][3]. C'est pourquoi il est aussi intéressant d'utiliser un graphe *performance/FAR* en utilisant la définition de la performance de l'équation (2.7). Cette courbe va ressembler beaucoup à la courbe ROC TAR/FAR si le classifieur cible est très performant. Plus le rejet augmente, plus la performance diminue, mais elle diminuera moins si ce sont surtout des erreurs de classification qui sont rejetées. Par contre, si l'option rejette surtout des exemples bien classés, la performance va nettement diminuer. Pour certaines applications on pourra donc choisir une option de rejet avec un peu plus de faux rejet. En effet ce n'est pas grave si on rejette plus de formes si ces formes sont de toutes façons mal classées. La figure 2.9 présente les deux courbes pour la même option de rejet. Dans cet exemple, l'utilisateur peut choisir en considérant sur la courbe ROC le point de fonctionnement A car il n'a que 10% de diminution du TAR. Si on considère la courbe *Performance/FAR*, il pourra modifier son choix par le point B qui réduit la performance de 10% par rapport au classifieur sans rejet avec un TAR plus petit que le point A. La différence entre le point A et le point B est de savoir si 10% des exemples sont rejetés (point A) ou si 10% des exemples bien reconnus sont rejetés (point B).

2.6 Extension de l'évaluation du rejet

Les différentes possibilités d'évaluation du rejet ont été déjà beaucoup développées dans l'état de l'art dans la section 2.5. Dans cette section nous proposons de compléter les outils d'évaluation des options de rejet d'ambiguïté et de distance. En effet pour pouvoir comparer toutes les options de rejet que nous avons présentées, il nous faut des outils permettant leur comparaison rapide dans les différents contextes. Pour le rejet d'ambiguïté, nous faisons une étude complète de la courbe ER et nous proposons une mesure d'aire sous la courbe pour la comparaison rapide. Pour le rejet de distance nous proposons d'étendre le test de la courbe ROC à nos trois problèmes. Dans chacun des cas, nous faisons le bilan des mesures utilisées par la suite dans les résultats.

[3]Nous avons vu que dans cette publication, les auteurs considèrent en fait 2 classes de rejet : une bien définie et une mal définie. Si on considère la première comme une classe principale à reconnaître, leur courbes ROC en trois dimensions deviennent des courbes Performance/ROC.

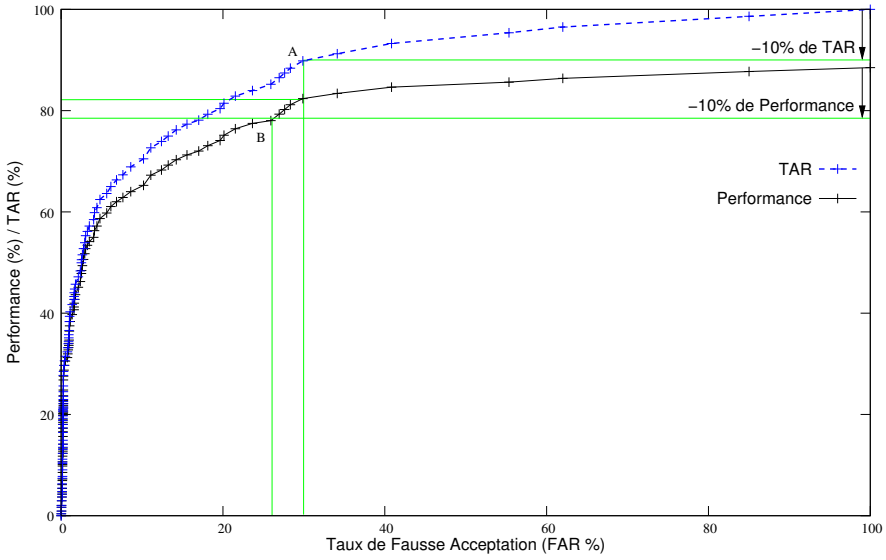


FIG. 2.9 – Comparaison d'une courbe ROC et de la courbe Performance/FAR de la même option de rejet.

2.6.1 Évaluation du rejet d'ambiguïté

Nous avons vu dans la section 2.5.1 comment construire une courbe *Erreur/Rejet* (noté *courbe ER*) pour une option de rejet d'ambiguïté utilisant un paramètre continu comme un seuil sur une fonction de confiance en utilisant une base de test D_E contenant des exemples à reconnaître. Nous proposons maintenant d'étendre cette possibilité aux options de rejet d'ambiguïté avec un paramétrage discret, comme par exemple celles utilisant un classifieur de rejet. Nous considérons donc que l'apprentissage d'une option de rejet d'ambiguïté donne un ensemble de points non ordonnés dans l'espace Erreur/Rejet. A partir de ce nuage de points nous pouvons appliquer la même approche que celle vue pour les courbes ROC précédemment. En effet, il est possible, en considérant l'erreur et le rejet comme deux objectifs à minimiser, de calculer le front de Pareto de cet ensemble. De plus il est possible d'interpoler entre chaque couple de points pour obtenir n'importe quelle option intermédiaire. Nous pouvons donc calculer l'enveloppe convexe (qui est en fait concave dans l'espace ER) de ce nuage pour former une *courbe ER* optimale.

A partir de l'enveloppe convexe de la courbe ER, nous proposons de calculer l'aire sous cette courbe (AUERC pour *Area Under Error/Rejet Curve*) de la même manière que pour une courbe ROC. Comme il s'agit ici de minimiser au plus l'erreur et le rejet, plus l'aire sera petite, plus l'option sera efficace. La figure 2.10 donne un exemple fictif de courbe convexe et de l'AUERC correspondante.

L'AUERC possède les mêmes avantages et inconvénients que l'AUROCC si on compare des options de rejet qui ont le même taux d'erreur sans rejet. Dans ce cas une

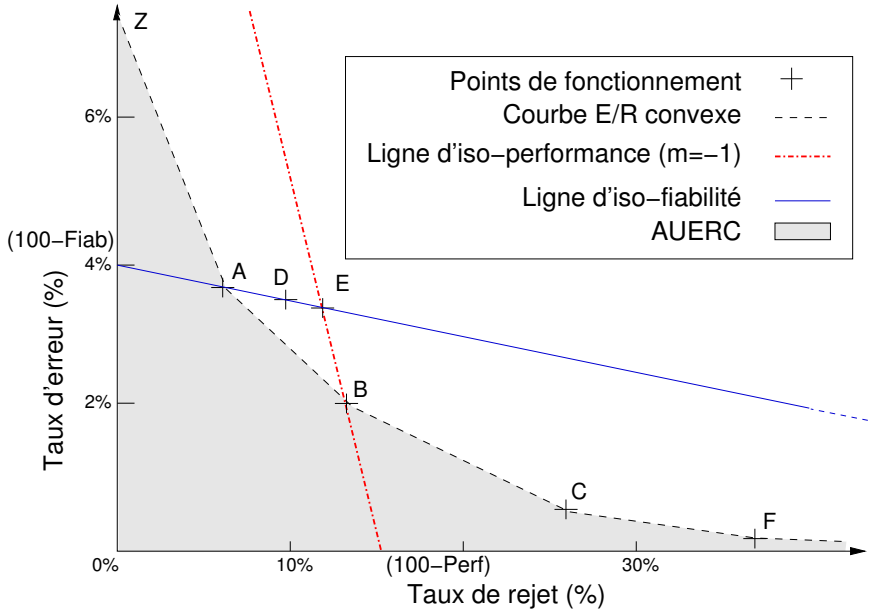


FIG. 2.10 – Exemple de courbes Erreur/Rejet avec les lignes d'iso-performance et d'iso-coût.

aire plus petite signifie que l'option de rejet est meilleure mais des aires trop proches ne permettent pas de conclure (si les courbes se croisent). L'AUERC mesure en effet un comportement global de l'option de rejet. De plus, lorsque nous comparons deux courbes ER qui n'ont pas le même taux d'erreur sans rejet, celle qui a le faible taux est avantagée (mais n'aura pas forcément la plus faible AUERC dans le cas où elles se croisent). Ce cas arrive lorsque l'on compare des options de rejet d'ambiguïté utilisant des classifieurs principaux différents. En effet dans ce cas, on compare non seulement l'efficacité du rejet à éviter les erreurs, l'efficacité du couple *classifieur principal/option de rejet* (puisque certaines options seront plus efficaces avec certains classifieurs) mais aussi l'efficacité du classifieur à éviter les erreurs. Pour pouvoir comparer seulement l'effet du rejet, il est possible d'utiliser une courbe normalisée par le taux d'erreur initial, mais on continue de mesurer l'effet du couple *classifieur principal/option de rejet*.

Nous avons vu qu'il existait d'autres mesures possibles pour évaluer le rejet d'ambiguïté, notamment les mesures de Fiabilité et de Performance. Nous n'utilisons pas de courbes Fiabilité/Performance dans nos résultats car si elles peuvent être utiles à l'utilisateur final, elles n'apportent rien de plus dans la comparaison des options de rejet. Nous pouvons toutefois noter que dans l'espace Erreur/Rejet, il existe des droites d'iso-Fiabilité et d'iso-Perfomance.

En effet d'après les équations (2.11) et (2.16), si on a $C_R = C_E = 1$ alors $m = -1$ et $C(\rho) = 1 - Perf(\rho)$. Donc la droite de pente -1 qui coupe l'axe des ordonnées en $1 - Perf(\rho)$ est une droite où la performance est constante avec une valeur de $Perf(\rho)$

(sans rejet, $Perf(\rho) = 1 - E(\rho)$). De même avec les équations (2.8) et (2.9) on peut déduire que $E = -(1 - Fiab)R + (1 - Fiab)$. Donc dans le graphe ER, la droite de pente $-(1-Fiab)$ et qui coupe l'axe des ordonnées en $1-Fiab$ correspond à une droite où la fiabilité est constante. On remarque que pour 0% de rejet l'erreur est bien égale à $1-Fiab$ et que pour 100% de rejet, l'erreur est nulle. La figure 2.10 montre un exemple fictif de courbe ER avec ses lignes d'iso-Fiabilité et d'iso-Performance.

Pour chercher le point où la fiabilité est la plus forte, il faut donc partir de la droite horizontale (de pente $(1-Fiab) = 0$) et la faire monter (augmenter la pente en gardant fixe le point $(1, 0)$) jusqu'à ce qu'elle coupe la courbe ER. Comme la courbe ER est par construction convexe, le premier point rencontré est toujours le point le plus à droite sur la courbe. De plus si l'applicatif impose un taux de fiabilité et une performance, il n'y a qu'une seule option de rejet qui corresponde, celle qui est à l'intersection des deux droites correspondantes (le point E dans la figure 2.10).

Bilan des mesures utilisées pour le rejet d'ambiguïté

Dans cette étude nous utiliserons l'AUERC (sans normalisation) pour comparer les options de rejet d'ambiguïté entre elles en tenant compte des limitations citées, notamment la cas où le classifieur principal est différent dans les options considérées. Nous utiliserons aussi le taux d'erreur pour un taux de rejet donné (5%, noté ERR5) pour permettre d'avoir un aperçu local de l'efficacité du rejet d'ambiguïté. Ces deux valeurs permettent d'apprécier l'allure générale de la courbe sans aller voir chaque graphique. Pour plus de précision nous effectuerons une validation croisée (sur la base D_E) pour obtenir une valeur moyenne moins sensible aux bases de test et d'apprentissage ainsi qu'un écart-type qui permettra de dire si deux valeurs sont comparables ou non.

2.6.2 Évaluation du rejet de distance

Pour évaluer le rejet de distance, nous avons vu dans l'état de l'art (section 2.5.2) comment construire une courbe ROC convexe à partir d'un nuage de points de fonctionnement calculés en utilisant deux bases de test D_E et D_R. Nous avons défini dans la section 2.2.2 trois problèmes pour le rejet de distance, le *problème* $A \rightarrow A$, le *problème* $A \rightarrow B$ et le *problème* $A \rightarrow A\&B$, qui correspondent à trois situations différentes. Il faut donc aussi définir trois mesures pour évaluer les options de rejet suivant ces trois problèmes.

Nous considérons donc ici trois bases pour l'évaluation du rejet de distance :

- D_E : base d'exemples à accepter (et à classer), qui sert en partie pour l'apprentissage de l'option de rejet (D_E^{App}) et en partie pour son test (D_E^{Tst}),
- D_A : base de contre-exemples de type A à rejeter, comme vu dans la section 2.2.2 ces contre-exemples sont disponibles pendant l'apprentissage et le test, une partie de la base sert donc à l'apprentissage (D_A^{App}) et une autre au test (D_A^{Tst}),
- D_B : base de contre-exemples de type B à rejeter, comme vu dans la section 2.2.2 ces contre-exemples ne sont pas disponibles pendant l'apprentissage, seulement pendant le test, cette base ne sert que pour le test des options de rejet.

– D_{AB} : base de contre-exemples à rejeter utilisée seulement en test, soit il s'agit de l'union de D_A^{Tst} avec D_B, soit d'une base indépendante dont une partie des exemples « ressemble » aux exemples de D_A

Pour l'apprentissage des options de rejet (apprentissage des classifieurs principaux et de rejet, apprentissage des seuils) nous utiliserons donc D_E^{App} et de D_A^{App}. Pour le test du *problème* $A \rightarrow A$, nous testerons sur D_E^{Tst} et D_A^{Tst}. Le test du *problème* $A \rightarrow B$ se fera sur D_E^{Tst} et tout D_B. Enfin le *problème* $A \rightarrow A\&B$ sera testé sur D_E^{Tst} et l'union de D_A^{Tst} avec D_B (ou D_{AB} si une base cette base existe). Le tableau 2.2 récapitule ce protocole.

TAB. 2.2 – Utilisation des bases pour l'évaluation du rejet de distance (*App* = utilisation en apprentissage, *Test* = utilisation en test).

	Exemples	Contre-Exemples	
Bases	$D_E = D_E^{App} \cup D_E^{Tst}$	$D_A = D_A^{App} \cup D_A^{Tst}$	D_B
Test $A \rightarrow A$	App et Test	App et Test	
Test $A \rightarrow B$	App et Test	App	Test
Test $A \rightarrow AB$	App et Test	App et Test	Test

Lors de l'évaluation d'une option de rejet, nous obtenons donc une courbe ROC pour chacun des trois problèmes, auxquelles correspondent trois AUROCC. Ces aires permettent de comparer les options de rejet entre elles en considérant chacun des problèmes séparément. Par contre il n'est pas possible de comparer l'aire sous la courbe $A \rightarrow A$ avec l'aire sous la courbe $A \rightarrow B$ ou $A \rightarrow AB$ car les bases de test sont différentes.

Nous avons vu qu'il est aussi possible de considérer la performance du système par rapport au rejet par un graphe *Performance/FAR*. Dans ce cas il faudra choisir sur quelle base calculer le FAR : sur la base D_A, D_B ou D_{AB} suivant le problème considéré. En effet il faut faire la même distinction que pour les courbes ROC. Si c'est la performance en fonction du rejet des exemples qui intéresse l'utilisateur, il ne faut pas considérer le graphe *Performance/TAR* mais *fiabilité/TAR*. En effet la performance et le TAR sont tellement corrélés que le graphe n'est pas lisible, alors que le graphe *Fiabilité/TAR* permet de bien voir les changements de performance dans ce cas.

Bilan des mesures utilisées pour le rejet de distance

Nous utiliserons dans la partie résultats ces trois mesures gardant en mémoire les limitations liées à l'utilisation des AUROCC (explicitées dans la section 2.5.2). Dans certains cas nous considérerons, lorsque ce sera nécessaire, le TAR pour un FAR fixé ainsi que les effets du rejet de distance sur la performance du système. Pour plus de précision nous effectuerons une validation croisée (sur D_E et D_A). Ainsi nous obtiendrons une valeur moyenne plus robuste aux bases d'apprentissage et de test avec un écart-type qui permettra de comparer les valeurs entre elles.

Chapitre 3

Extension générique de l'architecture TRF

Compte tenu de notre applicatif, l'option TRF est la plus appropriée des approches parmi les quatre présentées précédemment. En effet, le tableau suivant récapitule les contraintes dues à la reconnaissance à la volée embarquée de caractères et quelles architectures les respectent en comparaison d'un classifieur sans option de rejet.

Tab. 3.1 – Contraintes liées au domaine applicatif.

Contraintes	RC	SC	SCRF	TRF
Utiliser les mêmes caractéristiques	Oui	Oui	Oui	Oui
Utiliser un classifieur existant	Non	Oui	Oui	Oui
Surcoût en ressources mémoire faible	Oui/Non	Non	Non	Oui
Surcoût en calcul faible	Oui/Non	Non	Non	Oui

L'architecture TRF utilise des informations sur le classifieur principal via les fonctions de confiance. Ces approches ne modifient donc pas le classifieur principal ce qui est un avantage lorsqu'il existe déjà. La difficulté est dans la disponibilité et le choix des fonctions de confiance, mais aussi dans la façon de les utiliser dans le processus de décision. Le but des fonctions de confiance est de considérer les formes dans un autre espace de représentation que celui des caractéristiques d'origine où les formes à accepter et les formes à rejeter seront plus facilement séparables.

Les fonctions de confiance ont été utilisées dans beaucoup de travaux sous diverses formes [32, 40, 53, 63, 67, 86, 87] avec différentes architectures d'option de rejet et diverses fonctions de confiance. Grâce aux fonctions de confiance, nous pouvons abstraire le classifieur et l'information utilisée dans ce dernier et donc le formalisme que nous proposons devient indépendant du type de classifieur, du choix des fonctions et du type de rejet appris.

Dans la section 3.1 nous proposons une nouvelle option de rejet pour l'architecture TRF. Puis la section 3.1.1 présente plusieurs fonctions de confiance possibles pour cette approche utilisable sur différents classifieurs.

3.1 Nouvelle option de rejet pour l'architecture TRF

L'option de rejet que nous proposons est une approche multi-seuils générique, c'est-à-dire indépendante de la nature du rejet et du type de classifieur, généralisant celle proposée par Fumera (équation (2.3)).

L'option de rejet est donc définie par un ensemble de N seuils $\{t_i\}$ chacun associé à une fonction de confiance ψ_i. Pour qu'il y ait rejet il faut que chaque fonction de confiance soit inférieure à son seuil :

$$\forall i = 1..N, \psi_i \leq t_i. \tag{3.1}$$

Inversement il y a acceptation si au moins une fonction de confiance dépasse son seuil :

$$\exists i = 1..N, \psi_i > t_i. \tag{3.2}$$

Cette approche généralise celle de Fumera [40] définie par l'équation (2.3). En effet, la décision de l'option de rejet de Fumera dépend directement du résultat de classification. De ce fait, si les fonctions confiance sont utilisées avec l'option de Fumera, il faut qu'elles correspondent à la classification (*i.e.* une fonction par classe, par exemple le score de la classe), et il n'est pas possible d'utiliser des fonctions de confiance qui n'ont pas de rapport avec la classification. Donc grâce à notre approche, il est possible d'utiliser de nouvelles fonctions de confiance plus riches. Nous verrons dans la section suivante que le choix de ces fonctions de confiance peut permettre de retrouver exactement l'option de rejet de Fumera, il s'agit donc bien d'une généralisation.

L'utilisation de N fonctions de confiance permet une meilleure précision dans la discrimination du rejet nous l'avons vu dans la section 2.3.1.1. Nous noterons cet ensemble $\{\psi_i\}$. Notre formalisme permet aussi d'utiliser des fonctions de confiance avec $N = 1$ pour formaliser par exemple l'approche de Chow (équation (2.1)).

L'utilisation des fonctions de confiance permet à notre approche de pouvoir modéliser du rejet de confusion aussi bien que du rejet de distance pour n'importe quel type de classifieur et de type d'information utilisé pour décider du rejet. Ainsi le processus de décision et d'apprentissage peut être unifié entre les différents types de classifieurs et différentes natures de rejet. C'est le choix des fonctions de confiance qui est important puisqu'il va dépendre du classifieur utilisé et de la nature du rejet désirée.

3.1.1 Les fonctions de confiance

Une fonction de confiance ψ dépend du classifieur principal utilisé et du type de rejet désiré. Une fonction de confiance permet d'évaluer la confiance que l'on peut avoir dans la réponse du classifieur. La contrainte forte sur les fonctions de confiance est : *plus la forme doit être rejetée, plus la confiance est faible.* Les fonctions de confiance

permettent d'abstraire les connaissances et informations issues du classifieur principal pour obtenir des options de rejet génériques et performantes.

Nous avons vu que les fonctions de confiance sont déjà utilisées dans plusieurs travaux, mais la plupart du temps seules : la décision de rejet n'est basée que sur une seule fonction. Nous avons vu que dans notre approche nous permettons d'utiliser un ensemble de N fonctions de confiance $\{\psi_i\}$ qui permet plus de précision dans la décision de rejet. Le nombre de fonctions de confiance utilisé est important. Un nombre trop faible donnera un rejet moins performant mais un nombre trop élevé impliquera un nombre de seuils plus important à apprendre et donc un problème plus complexe nécessitant plus de données d'apprentissage. Le compromis dépendra du contexte d'apprentissage du rejet (quantité de données d'apprentissage, type de classifieur, nature du rejet, ...). C'est pourquoi nous proposons à chaque fois des fonctions de confiance qui représentent la même information mais avec différents niveaux de précision (de $N = 1$ à $N =$ plusieurs dizaines).

Dans cette section nous présentons des fonctions de confiance dédiées au rejet d'ambiguïté et d'autres dédiées au rejet de distance. Ces fonctions dépendent fortement des classifieurs sur lesquelles elles sont appliquées, nous présentons ici des fonctions pour les RBFN, les MLP et les SVM. Certaines peuvent être transposées directement d'un classifieur à l'autre, mais leur efficacité varie d'un type de classifieur à l'autre.

3.1.1.1 Fonctions de confiance pour le rejet d'ambiguïté

Le rejet d'ambiguïté détermine si la forme est proche d'une frontière de décision ou non et donc potentiellement une erreur de classification. Il s'agit donc de trouver des fonctions de confiance qui traduisent cette notion. Toutes les fonctions de confiance que nous allons présenter ne dépendent pas du type de classifieur principal utilisé. La seule contrainte est de disposer d'un score (comme un degré ou une probabilité d'appartenance) pour chaque classe.

Une première possibilité est d'utiliser l'approche de Fumera dont l'option de rejet d'ambiguïté est donnée par l'équation (2.3). En utilisant notre formalisme, cette option de rejet équivaut à utiliser cet ensemble de fonctions de confiance :

$$\psi_i^{Fumera} = \begin{cases} s_i & \text{si } i = C_1, \\ 0 & \text{sinon.} \end{cases} \tag{3.3}$$

avec C_1 la meilleure classe et s_i le score de la classe i. Il y a donc autant de fonctions de confiance que de classes. Comme toutes les fonctions de confiance sont à zéro sauf celle de la classe gagnante, on obtient bien la même option de rejet que celle de Fumera.

Une autre approche est de comparer le score obtenu par la meilleure classe C_1 et la seconde meilleure classe C_2. On obtient alors une seule fonction de confiance utilisée dans [32, 53, 67] :

$$\psi_{1\ abs}^{Amb} = s_{C_1} - s_{C_2}. \tag{3.4}$$

On peut aussi utiliser une différence relative comme dans [32] :

$$\psi_1^{Amb} = \begin{cases} \frac{s_{C_1} - s_{C_2}}{s_{C_1}}, & \text{si } s_{C_1} \neq 0, \\ 0 & \text{sinon.} \end{cases} \tag{3.5}$$

L'avantage d'une différence relative plutôt que d'une différence absolue est qu'elle traite de la même manière des formes avec de forts scores que celles avec de faibles scores : ce qui compte c'est que le premier soit nettement au dessus du second pour pouvoir lui faire confiance d'un point de vue ambiguïté. La différence simple introduit artificiellement une notion de rejet distance dans la confiance (même si le premier score est bien plus grand que le second, la confiance peut être faible si le premier score est faible).

Ces fonctions de confiance traitent de la même manière toutes les classes. Or cette confiance peut dépendre des classes en concurrence. C'est pourquoi nous proposons d'associer une fonction de confiance par frontière de décision. Si N_c est le nombre de classes reconnues par le classifieur principal, il y aura $N = N_c * (N_c - 1)/2$ fonctions de confiance définies par cette équation :

$$\psi_{\{i,j\}}^{ConfSym} = \begin{cases} \frac{s_i - s_j}{s_i} & \text{si } s_i \neq 0, (i,j) = (C_1, C_2) \vee (i,j) = (C_2, C_1), \\ 0 & \text{sinon.} \end{cases} \tag{3.6}$$

Il faut remarquer que toutes les fonctions de confiance sont à zéro sauf une, celle qui est concernée par la classification en cours. Cet ensemble de fonctions permet une vision précise de la confiance car elles permettent de ne pas avoir le même comportement pour chaque couple de classes.

On peut encore augmenter cette précision en associant une fonction de confiance à chaque couple ordonné de classes. La fonction activée sera donc différente suivant de quel coté de la frontière la forme se trouve. Si N_c est le nombre de classes reconnues par le classifieur principal, il y aura $N = N_c * (N_c - 1)$ fonctions de confiance :

$$\psi_{i,j}^{Amb} = \begin{cases} \frac{s_i - s_j}{s_i} & \text{si } s_i \neq 0, i = C_1, j = C_2, \\ 0 & \text{sinon.} \end{cases} \tag{3.7}$$

Le problème de ces dernières fonctions de confiance est que si le nombre de classes N_c devient grand, le nombre de fonctions de confiance augmente en $O(N_c^2)$ ce qui peut poser des problèmes ensuite pour l'apprentissage des options de rejet comme nous le verrons par la suite. Pour réduire ce problème, il est possible d'utiliser une fonction par classe en utilisant ψ_i^{AmbCl} :

$$\psi_i^{AmbCl} = \begin{cases} \frac{s_i - s_{C_2}}{s_i} & \text{si } i = C_1, s_i \neq 0, \\ 0 & \text{sinon.} \end{cases} \tag{3.8}$$

3.1.1.2 Fonctions de confiance pour le rejet de distance

Le but du rejet de distance est de délimiter les connaissances du classifieur principal. Donc les fonctions de confiance du rejet de distance devront donner une information concernant la description intrinsèque des classes.

La solution généralement utilisée dans la littérature est de considérer que les sorties du classifieur approximent les densités de probabilité des données d'apprentissage (ou quelque chose de comparable comme un degré d'appartenance floue). Cette supposition est faite pour la plupart des classifieurs et permet d'utiliser les sorties du classifieur comme source d'information pour les fonctions de confiance. Malheureusement, comme nous l'avons vu dans la section 1.2 ce n'est pas le cas pour tous les classifieurs : pas de problème pour les RBFN dont les sorties combinent les activations des fonctions à base radiale de la couche cachée ; les sorties des SVM utilisant des noyaux gaussiens contiennent une part de description puisqu'elles combinent les noyaux mais ces derniers ne sont pas choisis dans ce but contrairement aux fonctions à base radiale des RBFN ; les sorties des MLP ne sont valides qu'à l'intérieur des données d'apprentissage, rien ne garantit les valeurs en dehors de ces domaines [42, 109], de plus elles sont apprises dans un but discriminant.

La première fonction de confiance est donc d'utiliser directement le score de la classe gagnante C_1 :

$$\psi_{C1}^{Dist} = s_{C1}, \tag{3.9}$$

Cette solution classique est utilisée dans [32, 63, 67] mais aussi dans l'option de rejet de Chow [23].

Ensuite il est possible, pour avoir plus de précision, d'utiliser une fonction de confiance par classe ($N = N_c$) :

$$\psi_i^{Dist} = s_i. \tag{3.10}$$

Ainsi il est possible de définir un comportement différent suivant chacune des classes.

Ces fonctions de confiance ne correspondent pas tout à fait à celles utilisées par Fumera *et al.* dans [40] puisque les auteurs utilisent seulement l'activation de la classe gagnante. Nous définissons donc un autre ensemble de fonctions de confiance :

$$\psi_i^{Fumera} = \begin{cases} s_i & \text{si } i = C_1, \\ 0 & \text{sinon.} \end{cases} \tag{3.11}$$

Cet ensemble de fonctions est exactement le même que celui défini par l'équation (3.3) pour le rejet d'ambiguïté. Nous avons choisi de le considérer aussi comme une possibilité pour le rejet de distance puisqu'il utilise les scores de sortie du classifieur.

Il est aussi possible d'utiliser des connaissances plus spécifiques au classifieur utilisé. Ainsi pour un RBFN les activations μ_k des fonctions à base radiale sont une source intéressante d'informations puisqu'elles sont directement issues des données d'apprentissage. Nous proposons donc des fonctions de confiance dédiées aux RBFN.

La première utilise directement les activations des neurones cachés :

$$\psi_i^{DistRBF} = \mu_i. \tag{3.12}$$

La seconde considère que la confiance que l'on doit accorder à la reconnaissance d'une forme correspond à l'activation de la fonction à base radiale la plus proche, celle

qui est censée la représenter. Ainsi seule la plus activée est différente de zéro :

$$\psi_i^{DistNRBF} = \begin{cases} \mu_i & \text{si } i = \text{argmax}_k(\mu_k), \\ 0 & \text{sinon.} \end{cases} \tag{3.13}$$

Il est aussi possible de réduire le nombre de fonctions de confiance à 1 en n'utilisant que la meilleure activation des fonctions à base radiale :

$$\psi^{DistRbfW} = \max_k(\mu_k). \tag{3.14}$$

Ces listes de fonctions de confiance pour le rejet de d'ambiguïté et pour le rejet de distance ne sont pas exhaustives. En effet d'autres sont possibles comme celles utilisées dans [86, 87]. De plus il est possible de combiner entre elles les fonctions de confiance pour prendre en compte plusieurs sources d'information, mais cette approche n'est pas développée dans cette étude.

3.2 Apprentissage des seuils de l'architecture TRF

L'architecture TRF est utilisée avec un classifieur principal déjà appris et avec un ensemble de fonctions de confiance $\{\psi_i\}$ adaptées à ce classifieur et à la nature de rejet désirée. L'utilisation de plusieurs seuils exige une étape d'apprentissage pour l'option de rejet. En effet si un seul seuil est utilisé, il suffit de le faire varier pour tracer une courbe ROC ou ER suivant le cas et ainsi choisir le compromis intéressant. Avec plusieurs seuils, soit il faut tracer la courbe en N dimensions (une pour chaque seuil) ce qui est ensuite difficilement utilisable, soit utiliser un algorithme qui cherche quelle est la meilleure combinaison de seuils pour chaque compromis. L'apprentissage de cette option de rejet consiste donc à fixer une valeur pour chaque seuil associé aux fonctions de confiance.

Dans cette section nous présentons deux algorithmes originaux. Le premier appelé AMTL est basé sur des heuristiques définissables par un expert, le second appelé TGD est l'application de la méthode de la descente du gradient de l'erreur à notre problème d'apprentissage des seuils. A titre de comparaison nous présentons d'abord deux algorithmes de l'état de l'art que nous avons transposé à notre formalisme.

3.2.1 Les algorithmes de l'état de l'art

Les deux algorithmes auxquels nous nous comparons ensuite sont l'*optimisation par nuée de particules* (proposé par Oliveira *et al.* [81]) et le *problème de maximisation sous contraintes* (proposé par Fumera *et al.* [40]). La transposition à notre formalisme de ces algorithmes consiste surtout à utiliser des mesures de performance adéquates.

3.2.1.1 Optimisation par nuée de particules

Le problème de trouver l'ensemble optimal de seuils peut être vu comme un problème d'optimisation multi-objectifs. Dans [81] les auteurs ont choisi d'utiliser un algorithme d'optimisation appelé optimisation par nuée de particules (PSO pour *Particle Swarm Optimization*). Nous comparerons dans les résultats nos algorithmes AMLT et TGD, présentés dans les sections suivantes, à PSO.

Comme expliqué dans [56, 85], PSO est une technique d'optimisation stochastique multi-objectifs. Elle simule le comportement d'un groupe d'oiseaux en vol ou d'un banc de poissons cherchant de la nourriture. En effet PSO utilise un ensemble de particules dont le déplacement dépend du mouvement du groupe. La direction du déplacement d'une particule dépend de sa meilleure position trouvée jusqu'alors et de la meilleure position trouvée par le groupe.

Dans notre problème, l'espace de recherche est l'espace des valeurs possibles pour les N seuils $\{t_i\}$, c'est-à-dire $\Re^N$. La position de chaque particule correspond à une option de rejet *i.e.* une valeur pour chaque seuil. La mesure de *performance*[1] (*fitness function*) F utilisée pour évaluer les positions des particules dépendra de la nature du rejet désirée. Dans [81], les auteurs utilisent comme fonction F pour le rejet d'ambiguïté la fiabilité avec la contrainte que l'erreur reste inférieure à un seuil (presque comme dans l'approche de Fumera[2], présentée dans la section suivante). Nous avons choisi de modifier cette fonction F pour qu'elle représente mieux la notion de compromis entre ce qu'il faut rejeter et ce qu'il faut accepter. De plus nous en proposons une fonction F adaptée au rejet de distance.

Si un rejet d'ambiguïté est appris, nous proposons d'utiliser cette fonction utilisant la base d'apprentissage D_E :

$$F(\{t_i\}, D_E) = 1 + \alpha(1 - Err_{\{t_i\}}(D_E)) - Rej_{\{t_i\}}(D_E). \tag{3.15}$$

Le paramètre α permet d'ajuster le coût d'une erreur par rapport à celui d'un rejet.

Si un rejet de distance est appris, la mesure de F utilise les bases d'apprentissage D_E et D_A :

$$F(\{t_i\}, D_E, D_A) = \alpha TRR_{\{t_i\}}(D_A) - FRR_{\{t_i\}}(D_E). \tag{3.16}$$

De la même manière, le paramètre α permet de régler le compromis entre le vrai rejet et le faux rejet.

Il faut remarquer que PSO est un algorithme stochastique et donc deux exécutions successives de l'algorithme peuvent donner des résultats différents. De plus, il y a plus de chances pour qu'une solution optimale soit trouvée si l'algorithme tourne plus longtemps et s'il est répété plusieurs fois.

Dans l'approche présentée ici les deux objectifs sont fusionnés dans la même mesure de F, dans [85] les auteurs proposent de considérer les deux objectifs séparément pour générer en une seule exécution un ensemble de solutions tout le long de la frontière de Pareto, au lieu de générer une seule solution par valeur de α.

[1] Pour qu'il n'y ait pas de confusion avec la performance de l'équation 2.7, nous désignerons cette *performance* par F.

[2] Oliveira utilise le taux d'erreur alors que Fumera utilise le taux de rejet.

Le principal inconvénient de PSO est le coût en temps de calcul. En effet les calculs des taux d'erreur et de rejet sur les bases d'apprentissage doivent être effectués un grand nombre de fois tout au long de l'algorithme. De plus PSO ne fonctionnera pas si D_A est vide dans le rejet de distance.

3.2.1.2 Problème de maximisation sous contraintes

Fumera propose dans [39, 40] un algorithme d'apprentissage noté CMP (pour *Constrained Maximization Problem*) pour les options de rejet de confusion à seuils multiples. Cet algorithme essaye de résoudre itérativement un problème de maximisation sous contraintes. Nous avons ré-écrit ce problème pour qu'il soit applicable à la fois au rejet d'ambiguïté et au rejet de distance :

$$\begin{cases} \max_{\{\sigma_i\}} F\left(\{t_i\}, D_E, D_A\right) \\ FRR_{\{t_i\}}(D_E) \leq FRR_{MAX} \end{cases} . \tag{3.17}$$

A l'initialisation, les valeurs de $\{t_i\}$ sont nulles pour un rejet nul. Puis à chaque pas de l'algorithme la valeur d'un seuil est augmentée de manière à améliorer la valeur de $F\left(\{t_i\}, D_E, D_A\right)$ jusqu'à ce que le taux de rejet dépasse la valeur de FRR_{MAX}. Concrètement dans [39] les auteurs expliquent que pour faire varier un seuil ils utilisent un pas Δt et une variable d'itération k comprise entre 1 et k_{MAX}. Pour chaque valeur $t_i + k\Delta t$ du seuil, F est mesurée et la valeur k_i de k qui maximise F pour ce seuil est conservé. Le seuil qui est modifié à chaque pas est celui qui maximise F. Et ainsi de suite jusqu'à ce que pour chaque seuil aucune valeur de k n'améliore F sous la contrainte du FRR_{MAX}. L'Algorithme 1 décrit cet apprentissage.

Comme Fumera nous avons choisi d'utiliser la fiabilité (équation 2.8) comme mesure F pour le rejet d'ambiguïté. Pour le rejet de distance, la mesure F est le taux de vrai rejet TRR calculé sur la base d'apprentissage de contre-exemples D_A. Ces mesures F sont différentes de celles utilisées pour PSO car la contrainte sur le faux rejet permet déjà de parcourir tous les compromis.

Il y a plusieurs limitations à cette approche CMP. Premièrement, toutes les fonctions de confiance ne peuvent pas être utilisées. En effet, l'algorithme suppose qu'en modifiant un seul seuil, les taux de rejet vont être modifiés ce qui n'est pas toujours le cas dans notre modélisation. En effet avec des fonctions de confiance comme $\psi_i^{DistRBF}$, si tous les seuils sont à zéro comme à l'initialisation, modifier un seul seuil ne fera pas augmenter le taux de rejet car les formes seront acceptées par les autres seuils restés à zéro. Il faut donc qu'il y ait toujours une seule fonction de confiance qui soit non nulle, par exemple toutes les fonctions de confiance pour le rejet d'ambiguïté de la section 3.1.1.1 peuvent être utilisées, tandis que pour le rejet de distance seules les fonctions 3.9, 3.11 et 3.13 sont utilisables.

Deuxièmement, le taux de faux rejet FRR n'est atteint que s'il existe une meilleure valeur de F pour une valeur plus faible de FRR. En effet les seuils ne sont augmentés que si la valeur de F augmente aussi. Si F présente un minimum local qui ne peut être franchi (à cause d'un k_{MAX} trop petit par exemple ou si F devient décroissante)

Algorithme 1 : Algorithme CMP d'apprentissage des seuils.

Entrées : F fonction d'évaluation
Entrées : D_E base d'exemples et D_A base de contre-exemples
Entrées : FRR_{MAX} paramètre du cas d'arrêt
Entrées : k_{MAX} valeur maximale de recherche de k
Entrées : Δt pas de recherche des seuils
Résultat : N seuils de rejet t_i
début
  `//Initialisation`
  pour chaque $i = 1..N$ **faire**
    $t_i = 0$
  `//Apprentissage itératif des seuils`
  continue $\leftarrow vrai$
  tant que *continue* **faire**
    pour chaque $i = 1..N$ **faire**
      `//Cherche le meilleur k pour chaque seuil`
      soit $T_k = \{t_1, .. t_i + k\Delta t, .., t_N\}$
      $k_i \leftarrow \text{argmax}_{0 \leq k \leq k_{MAX}} \left(F\left(T_k, D_E, D_A\right) | FRR(T_k, D_E) \leq FRR_{MAX}\right)$
      $F_i \leftarrow F\left(T_{k_i}, D_E, D_A\right)$
    si $\exists k_i \neq 0$ **alors**
      `//Mise à jour du seuil permettant la meilleure` F
      $j \leftarrow argmax_i(F_i)$
      $t_j \leftarrow t_j + k_j \Delta t$
    sinon
      continue $\leftarrow faux$
fin

alors l'algorithme s'arrête. Donc toute la frontière de Pareto peut parfois ne pas être complètement explorée par cette approche notamment pour les fortes valeurs de FRR.

Troisièmement, si k_{MAX} n'est pas assez grand pour permettre de commencer à rejeter des formes, l'algorithme ne commence pas car il n'a pas trouvé de meilleure valeur F. Les auteurs ne proposent pas de solution pour ce cas particulier. Pour les expérimentations nous avons choisi $\Delta t = 0.01$ et $k_{MAX} = 20$. Cette configuration permet de rejeter les formes qui ont moins de $k_{MAX} \times \Delta t = 0.2$ de confiance, il est ainsi assez rare que l'algorithme n'arrive pas à démarrer.

Enfin, le temps d'exécution de CMP est très long puisqu'il faut calculer F et le FRR à chaque essai de valeur de k pour chaque seuil, tant que le critère d'arrêt n'est pas vérifié.

3.2.2 Automatic Multiple-Thresholds Learning algorithm (AMTL)

Compte tenu des algorithmes existants, il était nécessaire de proposer un nouvel algorithme d'apprentissage. En effet CMP et PSO sont très coûteux en temps de calcul,

CMP n'est pas adapté à toutes les fonctions possibles de notre approche, PSO est difficilement interprétable car il est stochastique. De plus aucun de ces deux algorithmes ne peut apprendre le rejet de distance sans contre-exemples. Toutes ces raisons nous ont poussé à développer un algorithme générique, efficace et interprétable.

L'algorithme que nous présentons dans cette section se nomme AMTL pour apprentissage automatique de seuils multiples (*Automatic Multiple-Thresholds Learning Algorithm*). Cette approche est un cadre générique pour l'apprentissage des seuils des différentes natures de rejet. AMTL est un algorithme glouton basé sur des heuristiques définissables par l'utilisateur. Différentes variantes de cet algorithme donc sont possibles et nous présentons les deux plus intéressantes AMTL1 et AMTL2.

AMTL utilise un classifieur principal déjà appris et un ensemble de fonctions de confiance déjà choisies. De plus il a besoin de deux bases d'apprentissage : une base d'exemples notée D_E et une base de contre-exemples notée D_A. Le contenu de ces deux bases dépend de la nature du rejet appris comme expliqué dans les sections précédentes : pour le rejet de distance, D_E contient les formes à accepter et D_A contient les formes à rejeter disponibles ; pour le rejet d'ambiguïté D_E contient les formes bien reconnues et D_A contient les formes mal reconnues par le classifieur principal. Le choix des fonctions de confiance et des heuristiques utilisées dépendant du problème considéré, le seul paramètre de AMTL est θ le taux de faux rejet maximal permis.

Comme on peut le voir dans l'Algorithme 2, AMTL est composé de cinq étapes :

1. A l'initialisation les valeurs des fonctions de confiance sont calculées pour tous les exemples et contre-exemples puis les seuils sont fixés de manière à rejeter tous les contre-exemples et tous les exemples, E et C contiennent la liste des exemples et contre-exemples rejetés ;
2. Les étapes suivantes sont répétées tant que le critère d'arrêt n'est pas vérifié par la fonction `cas_d_arrêt` ;
3. Le seuil qu'il va falloir modifier est choisi suivant la politique de choix des seuils par la fonction `choisir_le_seuil` ;
4. Le seuil choisi est ensuite diminué suivant la politique de descente des seuils grâce à la fonction `diminuer_le_seuil` pour accepter de plus en plus d'exemples (et de contre-exemples) ;
5. Mise à jour des exemples et contre-exemples encore rejetés, on retire de E et C ceux qui sont acceptés par un des seuils.

En considérant l'espace $\Re^N$ des valeurs possibles des seuils, AMTL est basé sur le fait qu'il y a deux points remarquables : le point de rejet nul P_0 où tous les seuils sont à zéro, et le point de rejet total P_T où tous les seuils sont suffisamment hauts pour tout rejeter. Cet algorithme cherche un chemin pour aller de P_T à P_0 en suivant itérativement une dimension (un seuil) à chaque étape. Ce chemin ressemble à un escalier en N dimensions. La dimension parcourue à chaque étape est choisie par la fonction `choisir_le_seuil` en essayant de minimiser le FRR et maximiser le TRR plus ou moins localement suivant cette fonction. Ensuite la fonction `diminuer_le_seuil` diminue le seuil choisi. Il faut noter que le taux de rejet est une fonction qui diminue de façon monotone tout au long

Algorithme 2 : Algorithme AMTL d'apprentissage des seuils.

Entrées : Le classifieur
Entrées : Ψ ensemble des N fonctions de confiance ψ_i
Entrées : D_E base d'exemples
Entrées : D_A base de contre-exemples
Entrées : θ paramètre du cas d'arrêt
Résultat : T ensemble des N seuils de rejet t_i
début
`//Initialisation`
1 **pour chaque** $\psi_i \in \Psi$ **faire**
$E_i \leftarrow \{\psi_i(e_k), \forall e_k \in D_E, \psi_i(e_k) > 0\}$
$E \leftarrow E \cup \{E_i\}$
$C_i \leftarrow \{\psi_i(c_k), \forall c_k \in D_A, \psi_i(c_k) > 0\}$
$C \leftarrow C \cup \{C_i\}$
$\sigma_i \leftarrow \max(C_i \cup E_i)$
`//Apprentissage itératif des seuils`
2 **tant que** *NON* `cas_d_arrêt`$(\theta, \Psi, T, D_E, D_A)$ **faire**
3 $s \leftarrow$ `choisir_le_seuil`(T, E, C)
4 $T \leftarrow$ `diminuer_le_seuil`(s, T, E, C)
`//Mise à jour`
5 **pour chaque** $i = 1..N$ **faire**
$E_i \leftarrow E_i - \{\psi_i(e_k), \exists s\ \psi_s(e_k) > t_s\}$
$C_i \leftarrow C_i - \{\psi_i(c_k), \exists s\ \psi_s(c_k) > t_s\}$
fin

du parcours de P_T à P_0 puisqu'il n'y a pas de remise en cause des choix (pas de retour arrière).

Ce sont les deux fonctions `choisir_le_seuil` et `diminuer_le_seuil` contenant les heuristiques définies par l'utilisateur qui déterminent l'apprentissage des seuils. La fonction `cas_d_arrêt` ne change pas le parcours de l'espace des possibilités mais seulement le moment de l'arrêt du parcours. C'est pourquoi nous avons choisi arbitrairement de considérer le taux de faux rejet comme critère d'arrêt : l'apprentissage s'arrête lorsque $FRR(D_E) < \theta$.

Il est donc possible de définir différentes variantes de AMTL et nous en proposons deux, chacune avec des objectifs différents. D'un côté AMTL1 a besoin de contre-exemples pour son apprentissage et est dédié au *problème* $A \rightarrow A$. D'un autre côté AMTL2 n'utilise pas de contre-exemples et est donc dédié au *problème* $A \rightarrow B$. Nous présentons ensuite dans la section 3.2.2.3 d'autres possibilités pour chacune des trois fonctions qui pourraient être utilisées dans d'autres contextes d'utilisation que le nôtre.

3.2.2.1 AMTL1 utilisant les contre-exemples

Le but de AMTL1 est de trouver le meilleur compromis entre le rejet des exemples et le rejet des contre-exemples. Pour cela il utilise une fonction `choisir_le_seuil` qui sélectionne le seuil qui minimise le nombre de contre-exemples acceptés pour l'acceptation d'un nouvel exemple. Ensuite la fonction `diminuer_le_seuil` diminue le seuil choisi pour accepter autant de contre-exemples nécessaires pour accepter un exemple de plus. Elle fixe donc le seuil sur le contre-exemple suivant le premier exemple. Les Algorithmes 3 et 4 formalisent ces fonctions d'apprentissage.

Algorithme 3 : Fonction `choisir_le_seuil` de AMTL1.

Entrées : Le classifieur
Entrées : E confiances des exemples e_k rejetés
Entrées : C confiances des contre-exemples c_k rejetés
Entrées : T ensemble des N seuils de rejet t_i
Résultat : s seuil choisi
début
  pour chaque $i = 1..N$ **faire**
    `//Compter le nombre de contre-exemples à accepter`
    $n_i \leftarrow \text{card}\{\psi_i(c_k) \in C_i, \psi_i(c_k) \geq \max E_i\}$
  $s \leftarrow \text{argmin}_i(n_i)$
fin

Algorithme 4 : Fonction `diminuer_le_seuil` de AMTL1.

Entrées : Le classifieur
Entrées : E confiances des exemples e_k rejetés
Entrées : C confiances des contre-exemples c_k rejetés
Entrées : s seuil choisi
Entrées : T ensemble des N seuils de rejet t_i
Résultat : T' nouveaux seuils
début
  `//Le seuil est fixé sur le premier contre-exemple`
  `//juste après le premier exemple`
  $t_s \leftarrow \max_k\{\psi_s(c_k) \in C_s, \psi_s(c_k) < \max E_s\}$
fin

Ainsi AMTL1 prend en compte la localisation des contre-exemples par rapport aux exemples et fera plus de rejet là où il y aura une forte proportion de contre-exemples. Donc AMTL1 est bien adapté au *problème* $A \rightarrow A$.

3.2.2.2 AMTL2 sans contre-exemples

L'objectif de AMTL2 est d'obtenir une meilleure description des connaissances du système par rapport à AMTL1. Pour cela il n'utilise aucune information à propos des contre-exemples. La fonction `choisir_le_seuil` choisit le seuil t_i qui maximise la pseudo-densité d_i d'exemples activant la fonction ψ_i. La pseudo-densité d_i d'exemples est définie en considérant la variation relative du seuil t_i pour accepter la moitié des exemples restant à accepter. Soit M_i le nombre d'exemples restant à accepter par ψ_i : $M_i = \|E_i\|$. Soit V_i la variation nécessaire de t_i pour accepter $M_i/2$ exemples. On a alors :

$$d_i = \frac{M_i t_i}{2V_i}. \tag{3.18}$$

Utiliser la moitié des exemples restant à accepter permet d'avoir une vision plus globale de la densité des données sans être perturbé par les effets instables des formes très éloignées. Ensuite la fonction `diminuer_le_seuil` diminue le seuil choisi pour accepter un seul exemple de plus. Elle fixe donc le seuil sur l'exemple suivant. Les Algorithmes 5 et 6 formalisent ces fonctions d'apprentissage.

Algorithme 5 : Fonction `choisir_le_seuil` de AMTL2.

Entrées : Le classifieur
Entrées : E confiances des exemples e_k rejetés
Entrées : T ensemble des N seuils de rejet t_i
Résultat : s seuil choisi
début
  pour chaque $i = 1..N$ **faire**
    `//Calcule de la pseudo densité d'exemples à accepter`
    $V_i \leftarrow \max E_i - \min\{\psi_i(e_k) \in E_i, \psi_i(e_k) > \text{median}(E_i)\}$
    $d_i \leftarrow \frac{\text{card}\{E_i\} t_i}{2V_i}$
  $s \leftarrow \text{argmax}_i(d_i)$
fin

Algorithme 6 : Fonction `diminuer_le_seuil` de AMTL2.

Entrées : Le classifieur
Entrées : E confiances des exemples e_k rejetés
Entrées : s seuil choisi
Entrées : T ensemble des N seuils de rejet t_i
Résultat : T' nouveaux seuils
début
  `//Le seuil est fixé juste après le premier exemple`
  $t_s \leftarrow \max_k\{\psi_s(e_k) \in E_s, \psi_s(e_k) < \max E_s\}$
fin

Il faut remarquer que AMTL2 n'utilisant pas les contre-exemples permet d'ap-

prendre une option de rejet même lorsque D_A est vide. C'est la seule option de rejet présentée dans cette étude qui permet l'apprentissage dans ces conditions. Cette aptitude donne à AMTL2 une très bonne capacité de généralisation et un avantage pour le *problème* $A \rightarrow B$.

3.2.2.3 Exemples d'autres variantes

D'autres variantes des fonctions de AMTL ont été développées mais pas retenues parce qu'elles présentent moins d'intérêt dans notre contexte. Nous présentons ici brièvement quelques exemples qui peuvent être utiles dans d'autres contextes.

Pour la fonction `cas_d_arrêt` nous proposons deux variantes en plus de l'utilisation du $FRR(D_E)$:

- utiliser le $TRR(D_A)$ pour s'assurer de rejeter au moins un minimum de contre-exemples : utile lorsqu'il y a de fortes contraintes sur le TRR dans l'applicatif ;
- s'arrêter lorsque toutes les classes à reconnaître ont un minimum de taux d'acceptation : pour être sûr qu'il n'y ait pas de déséquilibre entre les classes, utile par exemple pour accepter le chiffre '0' alors qu'il y a la lettre 'O' à rejeter.

Pour `choisir_le_seuil` deux autres possibilités sont proposées en plus de celles de AMTL1 et AMTL2 :

- choisir le seuil qui nécessite la plus petite variation pour accepter un exemple de plus. Comme cette variation est la plus petite, peu de contre-exemples seront acceptés : variante pour AMTL2 mais plus locale ;
- choisir le seuil avec une autre politique déjà présentée mais en imposant que la variation du seuil choisi augmente l'acceptation de la classe la plus rejetée, ainsi toutes les classes auront le même taux de rejet : pour être sûr qu'il n'y ait pas de déséquilibre entre les classes, utile par exemple pour accepter le chiffre '0' alors qu'il y a la lettre 'O' à rejeter.

3.2.2.4 Remarques sur l'implémentation

AMTL étant un algorithme glouton basé sur les fonctions de confiance quelques optimisations simples lui permettent d'être assez performant en terme de vitesse d'apprentissage. D'abord l'utilisation des ensembles E et C permet de ne pas avoir à réévaluer les valeurs des ψ_i à chaque boucle. De plus comme il n'y a pas de retour arrière, les formes acceptées sont retirées des ensembles E et C dans l'étape 5 ce qui permet d'accélérer le déroulement de l'algorithme. De plus si ces ensembles sont triés, le choix des seuils et la variations des seuils sont faciles et donc plus rapides à exécuter.

Par contre les heuristiques que nous avons présentées ici ne traitent que les cas généralement rencontrés pendant le déroulement de l'algorithme. En effet il y a des cas particuliers qui se présentent nécessitant des heuristiques plus complexes.

3.2.3 Descente du gradient

Les principaux inconvénients de AMTL sont qu'il s'agit d'un algorithme glouton et qu'il est basé sur des heuristiques. Nous présentons donc dans cette étude un algorithme

qui n'est pas du tout basé sur les mêmes principes. Cet algorithme appelé descente de gradient sur les seuils (TGD pour *Threshold Gradient Descent*) est une descente de gradient d'une fonction de coût. Cette approche a l'avantage d'être indépendante d'heuristiques. De plus TGD n'est pas glouton, donc les choix faits à un moment peuvent être ensuite remis en cause au court de l'apprentissage. En contrepartie, TGD comme tout les algorithmes basés sur ce principe est plus coûteux en terme de temps de calcul et présente le risque de converger vers un minimum local.

Les algorithmes de descente de gradient sont basés sur la dérivation d'une fonction de coût. Dans notre cas, la fonction de coût représente le compromis entre le FRR et le TRR calculé sur une base D_E d'exemples à accepter et une base D_A de contre-exemples à rejeter. La fonction de coût (équation (3.19)) correspond à la différence entre le FRR et le TRR. La fonction $C(X)$ vaut 1 si la forme est rejetée et 0 sinon, X_E étant un exemple et X_A un contre-exemple :

$$E(D_E, D_A) = \frac{\sum_{X_E \in D_E} C(X_E)}{card(D_E)} - \alpha \frac{\sum_{X_A \in D_A} C(X_A)}{card(D_A)} \ . \tag{3.19}$$

Le compromis entre le FRR et le TRR peut être ajusté avec le paramètre α en utilisant le coût d'un vrai rejet par rapport à un faux rejet. L'avantage d'utiliser directement les taux est que l'apprentissage n'est pas influencé par la taille respective des bases d'apprentissage, ce qui est utile lorsque les bases sont très déséquilibrées. S'il y a besoin de prendre en compte ces tailles, le paramètre α peut incorporer cette information par exemple en utilisant l'équation (2.22).

La principale difficulté est de trouver une fonction dérivable pour $C(X)$. Nous utilisons une fonction sigmoïde $Sig(x)$ sur chaque seuil pour savoir si la confiance est inférieure au seuil et $C(X)$ devient alors la conjonction de toutes les sigmoïdes :

$$Sig(x) = \frac{1}{1 + e^{-\lambda x}} , \tag{3.20}$$

$$Sig_i(X) = Sig(-(\psi_i(X) - t_i)) , \tag{3.21}$$

$$C(X) = \prod_{i=1}^{N} Sig_i(X) . \tag{3.22}$$

Le paramètre λ permet de définir la pente de la transition entre 0 et 1 de la sigmoïde. Plus λ est grand, plus la fonction Sig_i et donc aussi $C(X)$ se rapproche de la décision binaire *accepté/rejeté* définie par l'équation (3.1). De même, $E(D_E, D_A)$ se rapproche de $FRR(D_E) - TRR(D_A)$.

L'apprentissage des seuils se fait en faisant varier les seuils dans le sens contraire du gradient de l'erreur :

$$\Delta t_i = -\frac{\partial E(D_E, D_A)}{\partial t_i} \tag{3.23}$$

Les équations (3.24) à (3.31) montrent comment est calculé Δt_i.
On a d'abord :

$$\frac{\partial Sig(u)}{\partial u} = \lambda Sig(u)(1 - Sig(u)) , \tag{3.24}$$

donc d'après (3.21)

$$\frac{\partial Sig_i(X)}{\partial t_i} = -\lambda Sig_i(X)(1 - Sig_i(X)) \ , \tag{3.25}$$

d'après (3.22) on obtient :

$$\frac{\partial C(X)}{\partial t_i} = \frac{\partial Sig_i(X)}{\partial t_i} \prod_{\substack{j=1 \\ j \neq i}}^{N} Sig_j(X) \ . \tag{3.26}$$

D'après (3.19), on peut séparer la dérivée de l'erreur en deux sous-dérivées :

$$\frac{\partial E(D_E, D_A)}{\partial t_i} = \frac{\partial E_E(D_E)}{\partial t_i} - \alpha \frac{\partial E_A(D_A)}{\partial t_i} \ , \tag{3.27}$$

avec pour les exemples :

$$\frac{\partial E_E(D_E)}{\partial t_i} = \frac{\sum_{X_e \in D_E} \frac{\partial C(X_e)}{\partial t_i}}{card(D_E)} \ , \tag{3.28}$$

donc :

$$\frac{\partial E_E(D_E)}{\partial t_i} = \frac{\sum_{X_e \in D_E} -\lambda Sig_i(X_e)(1 - Sig_i(X_e)) \prod_{\substack{j=1 \\ j \neq i}}^{N} Sig_j(X_e)}{card(D_E)} \ , \tag{3.29}$$

qui se simplifie en :

$$\frac{\partial E_E(D_E)}{\partial t_i} = -\lambda \frac{\sum_{X_e \in D_E} (1 - Sig_i(X_e)) C(X_e)}{card(D_E)} \ . \tag{3.30}$$

De même pour les contre-exemples :

$$\frac{\partial E_A(D_A)}{\partial t_i} = -\lambda \frac{\sum_{X_a \in D_A} (1 - Sig_i(X_a)) C(X_a)}{card(D_A)} \ . \tag{3.31}$$

La variation du seuil prend donc en compte tous les exemples de la base d'apprentissage en fonction des sigmoïdes des autres seuils et de la dérivée du seuil concerné. La figure 3.1 donne un exemple de fonction sigmoïde pour $\lambda = 10$ et $t_i = 0.5$ et de la fonction dérivée correspondante.

Il faut remarquer que c'est la valeur de λ qui détermine la valeur de la pente et donc la largeur de la fonction dérivée. Plus λ est grand, plus il y aura d'exemples et de contre-exemples pris en compte dans le calcul (i.e. avec une participation non nulle) et plus la variation Δt_i tiendra compte de ces données d'apprentissage d'un point de vue global. Nous avons donc choisi de faire varier λ au cours de l'apprentissage : une valeur forte au début pour prendre en compte d'un maximum d'exemples et de contre-exemples pour trouver une solution globale qui évite les minimum locaux et ensuite un λ qui diminue pour trouver une solution de plus en plus locale.

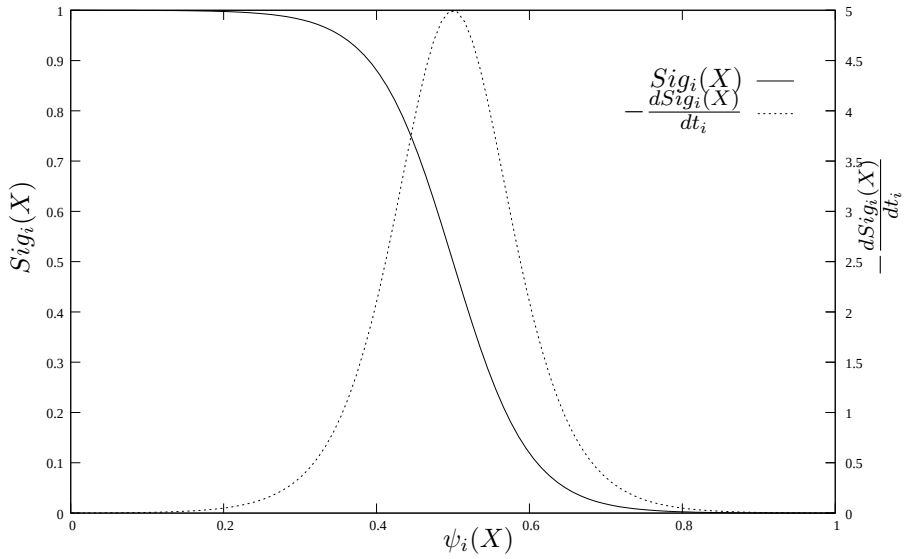


FIG. 3.1 – Exemple de courbe sigmoïde et de sa dérivée (seuil à 0.5).

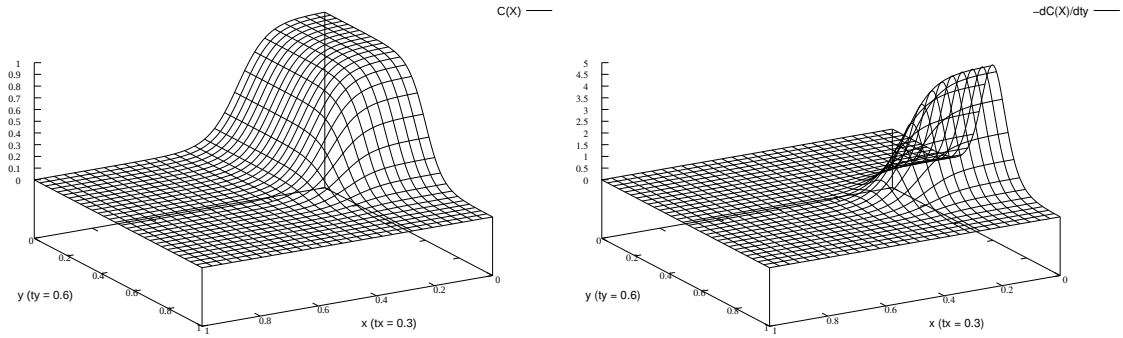
L'Algorithme 7 montre le déroulement en quatre étapes de TGD. L'initialisation des seuils est importante car il ne faut pas que les seuils soient initialisés quelque part où il n'y a pas d'exemples ou de contre-exemples, c'est pourquoi à l'étape 1 nous initialisons les seuils à la moyenne des fonctions de confiance non nulles. Nous n'avons pas montré que cette initialisation était la meilleure solution mais elle est généralement suffisante. Les variations de chaque seuil sont ensuite toutes calculées à l'étape 3 puis les seuils sont tous mis à jour à l'étape 4. Ces deux étapes sont bien séparées pour éviter qu'une modification du premier seuil n'ait d'impact sur la modification des suivants. Ces étapes sont répétées jusqu'à ce que le cas d'arrêt de l'étape 2 soit vérifié, c'est à dire que le nombre maximum d'itérations soit atteint, ou que la fonction d'erreur arrête de diminuer. Enfin à l'étape 5 la valeur de λ est ajustée si l'erreur se stabilise c'est-à-dire lorsque qu'un minimum local est trouvé (T étant la demi-période de cette variation et ϵ étant le seuil à partir duquel on considère que l'erreur est stable).

Pour bien comprendre le fonctionnement de TGD, prenons l'exemple d'une option de rejet avec l'architecture TRF utilisant deux fonctions de confiance (une représentée par l'axe x et l'autre par l'axe y). Il y a donc deux seuils à fixer, tx et ty. La figure 3.2 montre la fonction $C(X)$ correspondant avec $tx = 0.3, ty = 0.6$ ainsi que sa dérivée par rapport au seuil ty. La fonction dérivée est non nulle seulement dans l'espace proche de la partie de la frontière de décision correspondante au seuil ty. Ainsi, seuls les exemples et contre-exemples dans cette zone participeront à l'apprentissage. Plus la valeur de λ sera élevée, plus la pente de $C(X)$ sera forte et plus la fonction $\frac{\partial C(X)}{\partial ty}$ sera étroite et donc seuls les exemples très proches de la frontière seront considérés.

Les avantages de l'algorithme TGD sont une approche très rigoureuse du problème

Algorithme 7 : Algorithme TGD d'apprentissage des seuils.

Entrées : le classifieur
Entrées : ensemble des N fonctions de confiance ψ_i
Entrées : D_E base d'exemples et D_A base de contre-exemples
Entrées : α paramètre du compromis
Entrées : $\lambda_{min}, \lambda_{max}, T$ paramètres de l'apprentissage
Entrées : Nb_{max} nombre max d'itérations
Entrées : ϵ critère de stabilisation de l'erreur
Résultat : ensemble des N seuils de rejet t_i
début
  `//Initialisation`
  pour chaque t_i **faire**
1 $t_i \leftarrow \frac{\sum_{X \in D_E \cup D_A} \psi_i(X)}{card(X \in D_E \cup D_A, \psi_i(x) \neq 0)}$,
  `//Apprentissage itératif des seuils`
  $N_{iter} \leftarrow 0$
  $N_\lambda \leftarrow 1$
  $E_{anc} \leftarrow +\infty$
2 **tant que** $N_{iter} < Nb_{max}$ ***ET*** $E(D_E, D_A) < E_{anc}$ **faire**
    $E_{anc} \leftarrow E(D_E, D_A)$
    pour chaque t_i **faire**
3 $\Delta t_i = -\frac{\partial E(D_E, D_A)}{\partial t_i}$
    `//Mise à jour`
    pour chaque t_i **faire**
4 $t_i \leftarrow t_i + \Delta t_i$
    $N_{iter} \leftarrow N_{iter} + 1$
    si $E_{anc} - E(D_E, D_A) < \epsilon$ **alors**
5 $\lambda \leftarrow \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{1 + T/N_\lambda}$
      $N_\lambda \leftarrow N_\lambda + 1$
fin



(a) Fonction de rejet $C(X)$ (b) Dérivée de la fonction de rejet $\frac{\partial C(X)}{\partial ty}$

FIG. 3.2 – Exemple de fonction de rejet $C(X)$ utilisant deux fonctions de confiance, et sa dérivée par rapport à un des seuils

avec une technique d'apprentissage éprouvée, la possibilité de passer d'une vision globale du problème qui se focalise sur une solution locale et enfin la possibilité de remettre en cause au fur et à mesure de l'apprentissage les choix déjà faits. L'inconvénient majeur de cette approche est l'obligation de la présence d'une base D_A. Il serait possible de proposer une variante qui se passerait de cette base de contre-exemples en proposant une fonction d'erreur dérivable qui n'utiliserait pas ces contre-exemples. La difficulté étant de pouvoir régler le compromis comme le fait le paramètre α dans l'équation (3.19).

Nous avons présenté cinq algorithmes d'apprentissage des seuils de l'architecture TRF : CMP et PSO qui sont des approches de l'état de l'art, AMTL1 et AMTL2 qui sont des algorithmes utilisant des heuristiques s'appuyant sur les propriétés des fonctions de confiance et TGD qui est une approche plus formelle de ce problème d'apprentissage. Nous allons maintenant comparer ces algorithmes d'apprentissage entre eux ainsi qu'avec les trois autres architectures RC, SC et SCRF.

Chapitre 4

Expérimentations et résultats

Les expérimentations de cette section ont deux objectifs principaux. Le premier est d'aider un utilisateur à choisir, parmi les options de rejet et leurs algorithmes d'apprentissage décrits précédemment, la combinaison qui correspond le mieux à son contexte d'utilisation (rejet d'ambiguïté, type de problème dans le rejet de distance). Le second objectif est de choisir parmi ces possibilités la mieux adaptée à notre contexte applicatif, c'est à dire dans un contexte de ressources limitées. Ces résultats sont résumés dans un tableau dans la conclusion.

Il y a ensuite des objectifs secondaires. Le premier est de voir l'intérêt d'utiliser des options de rejet différentes pour le rejet de distance et pour le rejet d'ambiguïté. Le second est de bien comprendre les différences entre les différentes architectures et les effets des différents choix possibles (classifieurs utilisés, algorithmes d'apprentissage, paramètres ...). Parmi ces possibilités nous ferons particulièrement attention aux effets des propriétés des classifieurs utilisés, pour cela nous considérons trois types de classifieurs différents les MLP, les RBFN et les SVM. Enfin les résultats montreront les points forts de l'option de rejet et des algorithmes d'apprentissage, AMTL et TGD, que nous proposons par rapport à ceux l'état de l'art.

La section 4.1.1 commencera par présenter les deux types de bases de données que nous avons utilisées pour ces tests : une base artificielle en deux dimensions dans un but d'illustration des différentes approches et une base de données réelles qui correspond à notre problème applicatif, la reconnaissance de l'écriture manuscrite. La section 4.1.2 explique comment pour chaque architecture nous générons les points de fonctionnement qui servent ensuite à obtenir les courbes ER ou ROC. Ensuite la section 4.2 montrera le fonctionnement des protocoles de test et les différences principales entre les architectures en utilisant la base artificielle. Le problème en deux dimensions étant trop simple pour tirer des conclusions, la section 4.3 présente les résultats sur la base réelle et mettra en évidence les différences dues au passage à l'échelle. Compte tenu du nombre d'options de rejet à comparer, nous avons dans ces deux sections sélectionné les résultats les plus représentatifs, les résultats complets étant à chaque fois donné en annexe (cf. Annexe A).

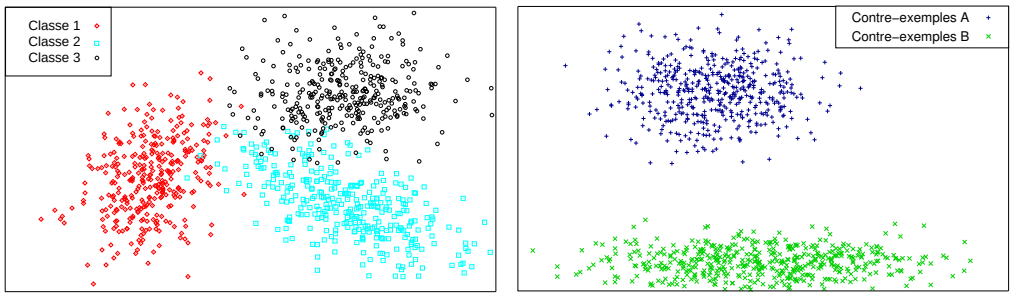
4.1 Protocoles de tests

Les tests consistent à comparer les quatre architectures apprises avec les différents algorithmes pour les mêmes problèmes de rejet (rejet d'ambiguïté et les trois problèmes du rejet de distance). Pour comparer les options de rejet obtenues, nous utilisons les mesures présentées dans la section 2.6. Nous présentons maintenant les bases que nous avons utilisées pour ces tests puis la façon d'obtenir les différents points opérationnels.

4.1.1 Les bases utilisées

Comme expliqué dans la sections 2.6, il nous faut définir quatre types de bases pour réaliser nos tests. Une base d'exemples D_E pour le rejet d'ambiguïté. Pour le rejet de distance, il faut en plus les bases de contre-exemples connus D_A, inconnus D_B et mal connus D_{AB} correspondant à chacun des trois problèmes. Nous avons définis ces bases autour de deux contextes très différents destinés à tester et expliquer nos approches : un contexte artificiel pédagogique et un contexte réel d'utilisation.

La première base utilisée est générée artificiellement dans le but d'illustrer cette étude. Elle se place dans un espace à deux dimensions ce qui permettra de visualiser les données et les zones de rejet. La base d'exemples D_E est composée de trois classes (notées 1, 2 et 3) chacune générées à partir d'une distribution gaussienne. Il y a volontairement un léger recouvrement entre ces trois classes. Chacune des bases de contre-exemples D_A et D_B est aussi générée à partir d'une gaussienne. Les gaussiennes sont toutes différentes les unes des autres. Il y a aussi un recouvrement partiel avec les trois classes principales. La figure 4.1 présente la localisation des trois classes de D_E et les deux distributions de D_A et D_B. La base D_{AB} sera constituée de l'union des deux bases de contre-exemples.



(a) (b)

FIG. 4.1 – Les données artificielles en deux dimensions D_E (a), D_A et D_B (b).

La seconde base utilisée correspond à des problèmes réels de rejet rencontrés dans notre contexte applicatif : la reconnaissance de chiffres pour le rejet d'ambiguïté et la reconnaissance de chiffres avec rejet des lettres minuscules pour le rejet de distance. Les classes de D_E sont les chiffres isolés manuscrits en-ligne de la base UNIPEN [46] (soit 15798 chiffres après nettoyage de la base). La base de contre-exemples à rejeter

D_A sont les lettres minuscules isolées manuscrites en-ligne de UNIPEN (soit 61017 lettres). La base de contre-exemples D_B est composée d'un sous-ensemble des lettres majuscules isolées de UNIPEN choisies pour être très différentes des chiffres et des lettres minuscules. Les lettres pour D_B choisis sont B, H, K, P, Q, R, X, Y et Z (soit 1789 lettres tirées au hasard parmi toutes les majuscules correspondantes de UNIPEN). Ce choix s'est fait en considérant la nature des caractéristiques utilisées par la suite. Enfin la base D_{AB} est composée de toutes les lettres majuscules de UNIPEN donc certaines ressembleront aux chiffres, d'autres aux lettres minuscules et d'autres à aucun des deux (soit 6000 lettres tirées au hasard parmi toutes les majuscules de UNIPEN). Les caractéristiques utilisées pour cette bases sont celles décrites dans la section 1.1.1.2. Ce problème beaucoup plus compliqué que celui avec les données artificielles permet de tester les options de rejet dans un contexte réel avec plus de classes (les 10 chiffres), plus de dimensions (21 caractéristiques) et plus de variabilité (caractères manuscrits).

Comme nous l'avons vu dans la section 2.6 nous utilisons pour nos tests une validation croisée. Nous avons donc séparé D_E et D_A en quatre parties : un quart sert au test de l'option de rejet, la moitié sert de base d'apprentissage avec le dernier quart utilisé comme base de validation (pour les apprentissage nécessitant une base de validation, comme l'apprentissage des classifieurs). Les expérimentations sont donc répétées quatre fois en faisant une rotation des quatre bases. La séparation entre base d'apprentissage, base de validation et base de test a été fait de manière aléatoire sans considérer les scripteurs, c'est donc un test multi-scripteurs c'est à dire que les scripteurs présents dans la base d'apprentissage peuvent être dans la base de test.

La table 4.1 résume le contenu de chaque base et présente la taille de chacune d'elles pour le test, pour l'apprentissage et pour la validation.

Tab. 4.1 – Taille et nature des bases de données utilisées.

Bases		Taille			Classes
		Apprentissage	Validation	Test	
Données artificielles	D_E	3000	1500	1500	1 à 3
	D_A	1000	500	500	contre-exemples A
	D_B	-	-	1000	contre-exemples B
	D_{AB}	-	-	2000	contre-exemples A et B
Caractères manuscrits	D_T	≈7900	≈3950	≈3950	10 chiffres
	D_A	≈30500	≈15254	≈15254	26 lettres minuscules
	D_B	-	-	1789	9 lettres majuscules
	D_{AB}	-	-	6000	26 lettres majuscules

4.1.2 Génération des points de fonctionnement

Nous expliquons dans cette section comment générer une diversité de points de fonctionnement pour chaque architecture avec leurs algorithmes d'apprentissage respectifs. En effet pour obtenir les courbes ER et ROC utilisées pour les résultats il faut pouvoir générer des points de fonctionnement explorant tous les compromis.

Pour les architectures RC, SC et SCRF nous avons modifié les tailles respectives des bases d'exemples et de contre-exemples. Faire varier la taille des bases permet de donner plus ou moins d'importance à chacune des deux classes. Nous avons donc diminué la taille de la base des exemples pour mieux rejeter les contre-exemples, puis diminué celle des contre-exemples pour mieux accepter les exemples. La variation du rapport entre exemples et contre-exemples a été faite de façon logarithmique : le rapport des contre-exemples par rapport aux exemples est donné par $log_2(k)$, k variant de -5 à 5. Suivant la complexité de l'apprentissage, cette variation se fait en 101 ou 51 pas.

Pour l'architecture TRF, chaque algorithme a un paramètre qui fait varier le compromis appris : θ pour AMTL1 et AMTL2 ; α pour PSO et TGD ; FRR_{MAX} pour CMP. Nous avons donc fait varier θ de 0 à 1 avec un pas de 0.01 (101 valeurs), pour le paramètre α de PSO seulement 41 valeurs sont utilisées à cause de la complexité de l'algorithme (les bornes dépendent des expérimentations) et pour le paramètre α de TGD 51 valeurs sont utilisées. Nous avons aussi ajouté les deux points de fonctionnement extrêmes : 100% de rejet et 0% de rejet (classifieur principal sans option de rejet), points qui ne sont pas forcément atteignables par les algorithmes d'apprentissage.

Le nombre de points de fonctionnement est noté NOP (pour *Number of Operating Points*).

4.2 Résultats sur la base artificielle

Dans cette section nous comparons les différentes options de rejet utilisant les quatre architectures présentées dans la section 2.3. Les données en deux dimensions permettent de comprendre les principales différences de comportement des différentes options de rejet et d'illustrer le protocole de test sur des données simples. Il faut avoir conscience des limites d'un test en deux dimensions. En effet d'un point de vue classification, ce type de problème est beaucoup plus simple que les problèmes réels. C'est pourquoi nous utilisons cette base artificielle dans un but pédagogique et non pour effectuer une comparaison précise. Nous commençons par étudier le rejet d'ambiguïté puis le rejet de distance.

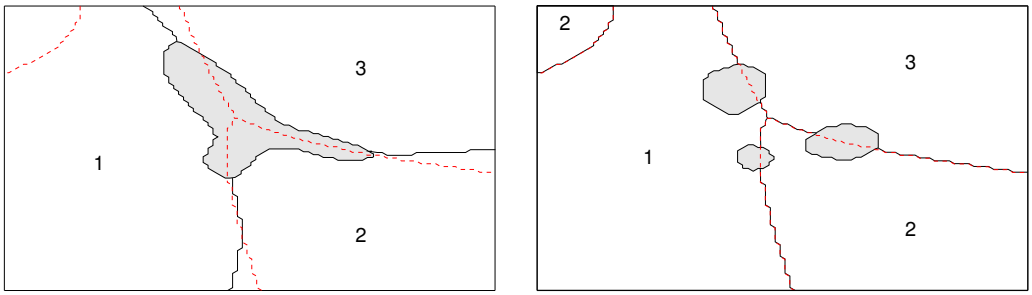
4.2.1 Pour le rejet d'ambiguïté

Le rejet d'ambiguïté cherche à limiter les erreurs de classification. Dans cette section nous étudions le comportement des différentes options de rejet face à ce problème. Nous commençons par étudier la forme en deux dimensions des zones de rejet produites par ces options puis leur efficacité globale pour résoudre ce problème.

4.2.1.1 La forme des zones de rejet

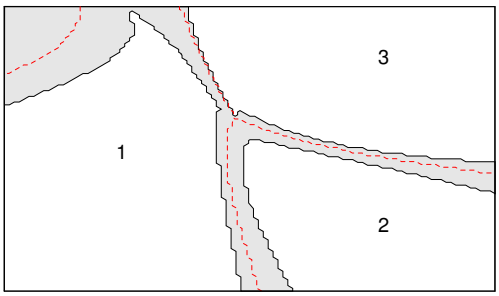
La figure 4.2 présente les zones de rejet d'ambiguïté apprises à partir de la base artificielle pour les architectures RC (avec une classe de rejet, figure 4.2(a)), SC (avec un classifieur spécialisé, figure 4.2(b)), TRF (avec des seuils sur les fonctions de confiance, figure 4.2(c)) et SCRF (avec un classifieur spécialisé sur les fonctions de confiance, figure 4.2(d)). Pour ces quatre exemples nous avons choisi comme classifieur principal un RBFN avec un prototype par classe car il s'agit d'un classifieur facile à observer en deux dimensions. Le type de classifieur de rejet pour les architectures SC et SCRF ayant peu d'importance pour la visualisation des zones de rejet (il s'agit d'un SVM). Les architectures TRF et SCRF sont basées sur les fonctions de confiance $\psi_{i,j}^{Amb}$ de l'équation (3.7). Les taux de rejet de ces options de rejet sont approximativement identiques (entre 3% et 5%) . Le chiffre dans les zones d'acceptation (zones blanches) correspond au numéro de la classe reconnue. Les pointillés correspondent aux frontières de décision du classifieur principal sans rejet. Les zones grisées sont les zones rejetées.

Il est intéressant de remarquer que les formes des zones de rejet des quatre options de rejet sont assez différentes les unes des autres. Les deux premières utilisant directement l'espace des caractéristiques (RC et SC) ont les zones de rejet placées là où il y a le plus de confusions entre les trois classes. Pour l'architecture RC les frontières de décision ont changé par rapport au classifieur sans rejet (les pointillés ne se superposant pas aux nouvelles frontières) alors que ce n'est pas le cas pour les autres architectures puisqu'elles conservent le classifieur principal d'origine. Dans l'architecture SC les zones de rejet sont très spécialisées et la petite zone où les trois classes se rejoignent n'est pas rejetée car il doit y avoir moins de données d'apprentissage ici. Les deux autres architectures (TRF et SCRF) utilisent les fonctions de confiance pour décider du rejet et c'est pourquoi les formes de leur zones de rejet se ressemblent un peu. Ce qu'il faut

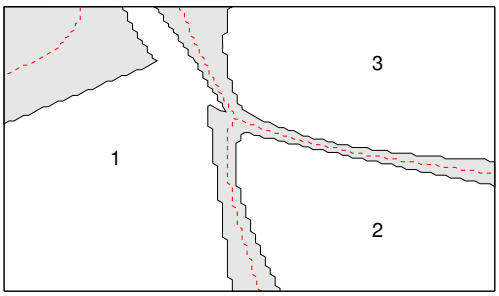


(a) RC sur RBFN

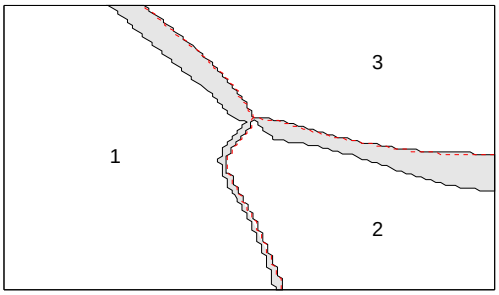
(b) SC sur RBFN avec SVM



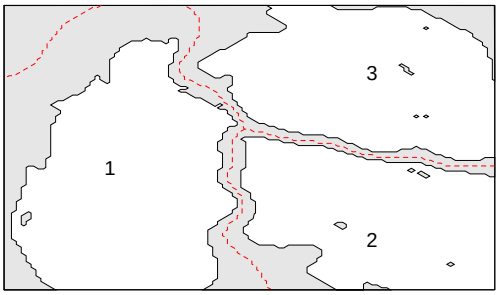
(c) TRF sur RBFN avec $\psi_{i,j}^{Amb}$



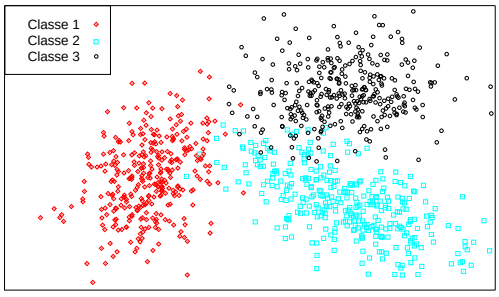
(d) SCRF sur RBFN avec SVM et $\psi_{i,j}^{Amb}$



(e) TRF sur MLP avec $\psi_{i,j}^{Amb}$



(f) TRF sur SVM avec $\psi_{i,j}^{Amb}$



(g) Données de D_E

FIG. 4.2 – Les zones de rejet des différentes options de rejet d'ambiguïté. Les zones de rejet sont en gris et les zones acceptées sont en blanc. Les frontières de décision sans rejet sont en pointillés, les numéros correspondent à la décision.

remarquer en premier c'est que les zones de rejet suivent toutes les frontières de décision et ne se focalisent pas sur des petites zones. Par conséquent ces architectures ont une bonne généralisation de la zone de rejet même là où il y a très peu d'erreurs puisque les fonctions de confiance apportent une information comme quoi ce qu'il faut rejeter est plutôt proche des frontières. Nous pouvons observer dans le coin Nord-Est des figures que la zone sans données d'apprentissage est rejetée. Cette zone de rejet est présente car les scores des classes 1 et 2 sont proches (et faible) dans cette zone éloignée des données d'apprentissage. Mais la présence de cette frontière est due à la nature du classifieur.

La forme des zones de rejet d'ambiguïté dépend de la nature du classifieur principal utilisé pour les architectures TRF et SCRF (et évidement aussi pour RC). Les figures 4.2(e) et 4.2(f) montrent les zones de rejet d'une option avec l'architecture TRF basée sur un MLP et un SVM avec noyaux Gaussiens. Cette architecture permet de bien voir les différences dans les formes des fonctions de confiance par rapport à l'architecture SCRF qui ajoute l'étape du classifieur de rejet. On peut voir que avec un SVM il y a le même effet dans le coin Nord-Est comme avec le RBFN. Par contre il n'y a pas de rejet avec le MLP dans cette zone. Cette zone de rejet est donc due aux informations intrinsèques apportées par la modélisation des connaissances dans les RBFN et ce type de SVM qui ne se retrouve pas dans les MLP. D'un autre point de vue, on ne peut pas dire que la présence de cette zone de rejet isolée soit bénéfique puisqu'il n'y a pas de données dans cette zone qui correspond en vérité plus à une notion de rejet de distance. Par contre on peut voir que la zone de rejet du SVM est beaucoup plus découpée que celle du MLP ou du RBFN. Cette différence vient de la modélisation des connaissances dans le SVM qui utilise les vecteurs support issus des données d'origine, il n'y a peu de lissage des connaissances comme avec les RBFN et les MLP.

4.2.1.2 Comportement global

Considérons maintenant l'option de rejet dans sa globalité. Nous allons donc utiliser les courbes Erreur/Rejet (ER) ainsi que l'aire sous cette courbe (AUERC) comme expliqué dans la section 2.6.1. La table 4.2 présente l'AUERC, le taux d'erreur à 5% de rejet (ERR5) et le nombre de points de fonctionnement (NOP) calculés par l'apprentissage de l'option pour les quatre architectures sur un RBFN (l'algorithme d'apprentissage utilisé, l'éventuelle fonction de confiance et le classifieur de rejet utilisé sont indiqués entre parenthèses). Les résultats complets pour chaque classifieur et chaque architecture sont présentés en Annexe A.1. Rappelons que plus l'AUERC et l'ERR5 sont petits, meilleure est l'option de rejet.

D'après les résultats en annexe, la première observation est que, pour ce problème simple, le choix de la fonction de confiance ou de l'algorithme utilisés n'a pas toujours d'influence. C'est pourquoi nous avons sélectionné ces huit résultats représentatifs. Les AUERC des lignes 1, 6, 7 et 8 montrent que les quatre architectures ont des résultats comparables pour ce problème d'un point de vue global compte tenu des écarts-types (AUERC entre 0.22 et 0.25). Par contre avec l'utilisation de l'architecture TRF, l'utilisation de fonctions de confiance non adaptées au rejet d'ambiguïté (par exemple ψ_i^{Dist} ligne 2) diminue nettement les résultats. On voit que l'AUERC ne permet pas toujours

TAB. 4.2 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur le RBFN (extrait de la table A.1, A.4, A.5 et A.6)

Option de rejet		100*AUERC	100*ERR5(%)	NOP
1	TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.248± 0.035	1.45± 0.23	103
2	TRF (AMTL1, ψ_{i}^{Dist})	0.697± 0.15	2.67± 0.41	103
3	TRF (CMP, $\psi_{i,j}^{Amb}$)	0.433± 0.045	1.58± 0.21	103
4	TRF (PSO, $\psi_{i,j}^{Amb}$)	0.360± 0.09	1.34± 0.26	41
5	TRF (CMP, ψ_{i}^{Fumera})	1.030± 0.35	2.67± 0.50	103
6	RC	0.225± 0.033	1.57± 0.19	103
7	SC (SVM)	0.234± 0.038	1.79± 0.28	103
8	SCRF (SVM)	0.222± 0.038	1.30± 0.26	103

de comparer les options de rejet comme le montre la figure 4.3 qui compare pour un exemple d'apprentissage de l'architecture TRF les algorithmes AMTL1 et PSO. En effet dans le tableau précédent AMTL1 a une meilleure AUERC que PSO pourtant nous voyons sur les courbes ER que PSO est meilleur pour les taux de rejet plus petits que 12%. Donc si l'utilisateur veut un taux de rejet de 5% ou 10% il devra choisir l'option apprise avec PSO plutôt qu'avec AMTL1 contrairement à ce que laissait supposer les AUERC. C'est pourquoi dans le tableau 4.2 nous présentons aussi le taux d'erreur avec

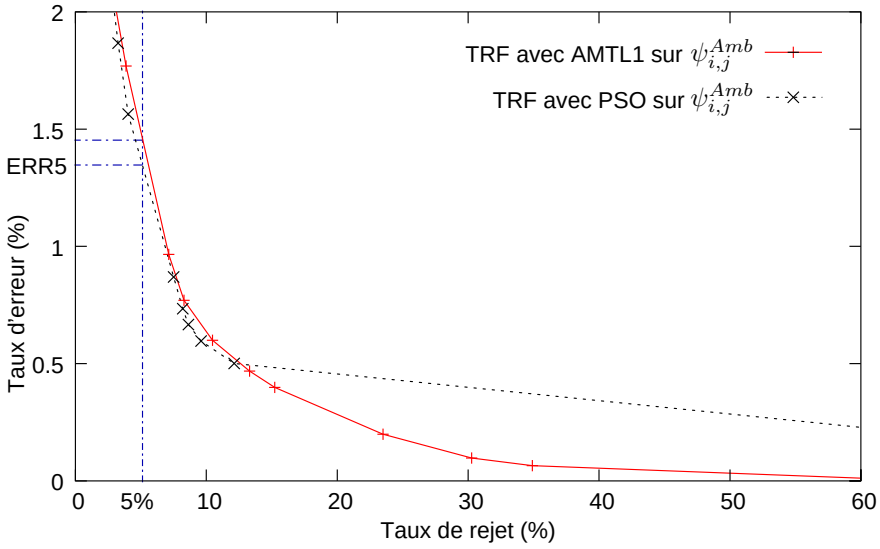


FIG. 4.3 – Comparaison entre les courbes ER des options de rejet d'ambiguïté d'architecture TRF appris avec les algorithmes AMTL1 et avec PSO.

5% de rejet, nous avons choisi arbitrairement ce taux souvent utilisé dans la littérature pour se mettre à la place d'un utilisateur. Avec ce critère, les architectures utilisant les fonctions de confiance (SCRF et TRF) sont les meilleures (entre 1.3% et 1.5%) et l'utilisation de fonctions de confiance inadaptées avec TRF est la pire solution. Remarquons que la solution proposée par Fumera (TRF avec CMP et ψ_i^{Fumera} ligne 5) obtient de très mauvais résultats par rapport à nos solutions.

Le tableau 4.3 résume les résultats obtenus pour l'architecture TRF avec les classifieurs MLP et SVM (avec normalisation SM des sorties).

Tab. 4.3 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur le MLP et le SVM (extrait des tables A.2 et A.3)

Option de rejet		100*AUERC	100*ERR5(%)	NOP
1	MLP TRF (AMTL1, ψ_1^{Amb})	0.217±0.027	1.31± 0.25	103
2	MLP TRF (AMTL1, ψ_i^{Dist})	0.224±0.038	1.46± 0.26	103
3	MLP TRF (CMP, ψ_i^{Fumera})	1.61 ± 0.19	3.05± 0.36	103
4	SVM TRF (TGD, $\psi_{i,j}^{Amb}$)	0.498± 0.11	1.39± 0.32	83
5	SVM TRF (AMTL1, ψ_i^{Dist})	0.683± 0.19	2.03± 0.63	103
6	SVM TRF (CMP, ψ_i^{Fumera})	1.65 ± 0.27	3.14± 0.51	103

Plusieurs différences sont à noter par rapport aux résultats du RBFN. Pour le MLP il n'y a pas de différence entre l'utilisation des fonctions de confiance dédiées au rejet d'ambiguïté et celles pour le rejet de distance pour ce problème simple. La différence est plus marquée pour les SVM. Il est aussi intéressant d'expliquer les très mauvais résultats de l'algorithme CMP : une étude des scores obtenus par les fonctions de confiance a montré que nous nous trouvons dans le cas particulier décrit dans la section 3.2.1.2 où CMP n'arrive pas à trouver de valeur de seuil pour commencer à rejeter.

4.2.1.3 Conclusion partielle

Le problème en deux dimensions ne permet pas de discriminer les architectures entre elles puisque les meilleurs résultats pour chaque architecture sont très proches compte tenu des écarts-types. Mais ces premiers résultats montrent que pour le rejet d'ambiguïté les fonctions de confiance permettent d'obtenir de bons résultats si elles sont prévues pour ce type de rejet. En effet les architectures TRF et SCRF ont, pour un AUERC équivalent aux autres architectures, un meilleur ERR5. De plus nous avons montré que notre algorithme AMTL permet de mieux parcourir les différents compromis que PSO et CMP.

4.2.2 Pour le rejet de distance

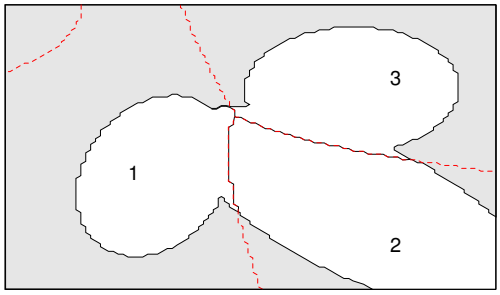
Le rejet de distance a pour but de rejeter les formes qui ne correspondent pas aux classes apprises. Nous avons vu que ce type de rejet peut être confronté à trois types de problèmes suivant les contre-exemples disponibles pendant l'apprentissage de l'option de rejet. L'objectif de ces tests est de comprendre le comportement des différentes options de rejet face à ces problèmes pour choisir les plus adaptées. Nous commençons par présenter les résultats (vu en deux dimensions puis comportement global), ensuite nous commenterons ces résultats d'abords pour les architectures utilisant les fonctions de confiance puis pour celles utilisant directement les caractéristiques.

4.2.2.1 La forme des zones de rejet et comportement global

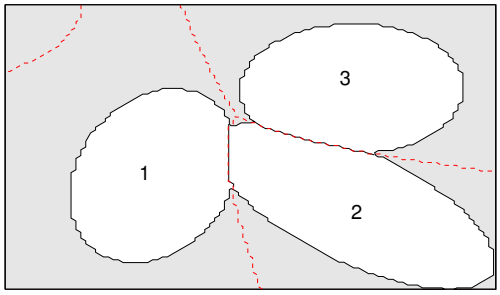
La figure 4.4 compare les formes des zones de rejet de distance pour différentes options de rejet apprises sur la base artificielle D_A. Les taux de rejet des options de rejet présentées sont à peu près identiques. Nous pouvons constater que les zones de rejet sont très différentes d'une architecture à l'autre. La table 4.4 présente pour les mêmes options de rejet (et quelques autres) les aires sous la courbe ROC (AUROCC) pour les trois problèmes $A \rightarrow A$, $A \rightarrow B$ et $A \rightarrow AB$ et le nombre de points de fonctionnement appris (NOP). Rappelons que plus l'AUROCC est grand, meilleure est l'option de rejet. Les résultats complets de cette expérience sont donnés en annexe A.2. Quelques remarques générales à propos de l'utilisation de l'aire sous la courbe sont les mêmes que pour le rejet d'ambiguïté, par exemple le fait que l'égalité des AUROCC n'implique pas l'égalité des courbes ROC.

Tab. 4.4 – AUROCC et NOP pour le rejet de distance sur les données artificielles (extraits des tables A.7 à A.12.

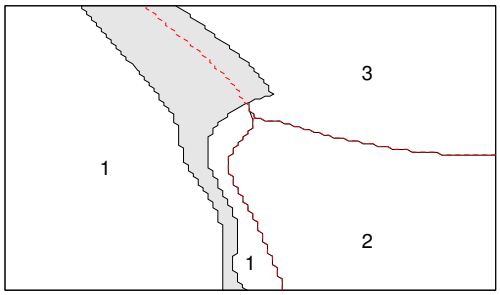
	Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
1	TRF (RBFN, AMTL1, ψ_i^{Dist})	91.0±0.48	91.8±0.53	91.1±0.15	103
2	TRF (RBFN, AMTL2, ψ_i^{Dist})	89.5±0.79	94.9± 0.2	92.2±0.27	103
3	TRF (RBFN, AMTL1, ψ_{C1}^{Dist})	89.3±0.77	95.1±0.21	92.2±0.28	103
4	TRF (MLP, PSO, ψ_i^{Dist})	84.9±0.46	57 ± 1	68.7±0.58	43
5	TRF (SVM, TGD, ψ_i^{Dist})	81.5± 1.2	63.5± 2.8	70.6± 1.4	103
6	TRF (RBFN, AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	93.4±0.52	81.1±0.66	87.1±0.48	103
7	SCRF(RBFN avec $\psi_i^{DistRBF}$, SVM)	96.1±0.18	82.1± 1.3	88.4±0.62	53
8	SC (RBFN avec SVM)	97.1±0.23	55.1± 5.1	73.7± 3.5	103
9	RC (SVM)	97.0±0.21	59.3± 7.5	76.8± 3.2	103



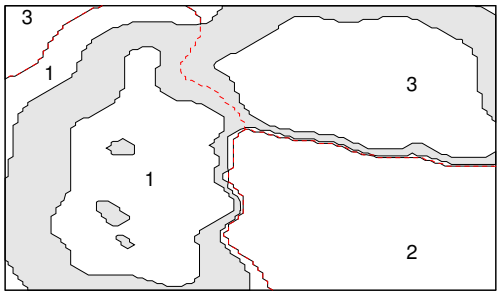
(a) TRF (AMTL1) sur RBFN avec ψ_i^{Dist}



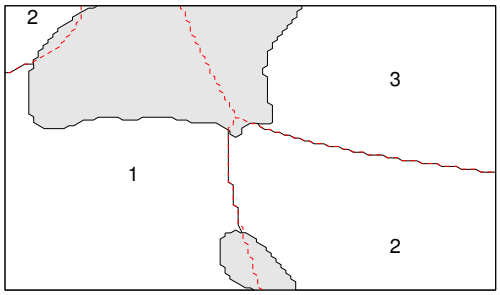
(b) TRF (AMTL2) sur RBFN avec ψ_i^{Dist}



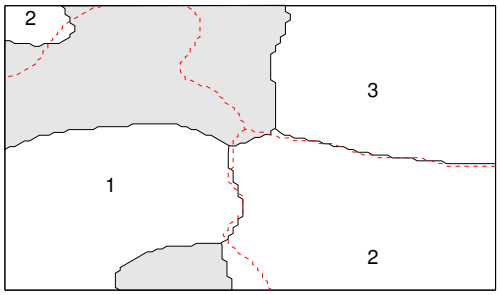
(c) TRF (AMTL1) sur MLP avec ψ_i^{Dist}



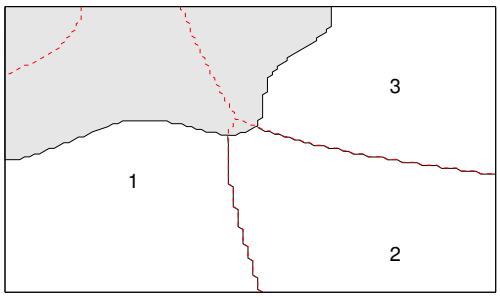
(d) TRF (AMTL1) sur SVM avec ψ_i^{Dist}



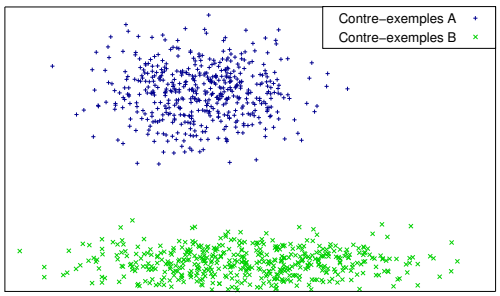
(e) SCRF (avec un MLP) sur RBFN avec ψ_i^{Dist}



(f) RC sur SVM



(g) SC (avec un SVM) sur RBFN



(h) Bases de contre-exemples D_A et D_B

FIG. 4.4 – Les zones de rejet des différentes options de rejet de distance. Les zones de rejet sont en gris et les zones acceptées sont en blanc. Les frontières de décision sans rejet sont en pointillés. Les chiffres correspondent aux décisions de classification.

4.2.2.2 Architectures TRF et SCRF

Les figures 4.4(a) et 4.4(b) permettent de comprendre la différence entre les deux variantes AMTL1 et AMTL2 de notre algorithme d'apprentissage des seuils de l'architecture TRF. AMTL1 prend en compte les contre-exemples et donc les zones d'acceptation 1 et 3 sont plus petites que celles apprises par AMTL2 car elles sont proches des contre-exemples de type A. D'un autre coté la zone de la classe 2 est plus large avec AMTL1 qu'avec AMTL2 car les contre-exemples en sont éloignés. AMTL2 en ne prenant en compte que la densité des exemples généralise mieux les zones de rejet de façon homogène alors que AMTL1 se spécialise dans le rejet d'un type de contre-exemples. Cette différence se retrouve dans la table 4.4 (lignes 1 et 2) : l'AUROCC du *problème* $A \rightarrow A$ est le même pour AMTL1 et AMTL2 mais pour les problèmes $A \rightarrow B$ et $A \rightarrow AB$ les AUROCC sont bien meilleures pour AMTL2. La ligne 3 du tableau montre les résultats en utilisant une seule fonction de confiance ψ_{C1}^{Dist} et on peut constater que ce problème simple ne permet pas de montrer l'intérêt du multi seuillage. En effet le fait de n'avoir qu'un seul seuil à fixer permet facilement de généraliser, ce qui dans ce problème est un handicap pour le *problème* $A \rightarrow A$ mais un avantage pour les problèmes $A \rightarrow B$ et $A \rightarrow AB$.

Les bons résultats de l'architecture TRF dépendent en grande partie du classifieur principal utilisé comme le montrent les figures 4.4(c) et 4.4(d) qui utilisent respectivement un MLP et un SVM. On peut voir que le MLP n'est pas du tout adapté à ce type de rejet puisqu'il essaye de rejeter seulement dans les zones proches des frontières. La ligne 4 du tableau confirme ces résultats. Les résultats du SVM (ligne 5 du tableau) sont faibles malgré l'information intrinsèque contenue dans les vecteurs supports. La figure 4.4(d) montre que l'utilisation de la normalisation soft-max n'est pas adaptée au rejet de distance mais d'après les résultats complets en annexe il n'y pas beaucoup de différence. D'une façon générale, pour un classifieur principal donné et une fonction de confiance donnée, ces tests ne permettent pas de discriminer les différents algorithmes d'apprentissage.

L'architecture SCRF utilisée dans la figure 4.4(e) arrive à bien utiliser les fonctions de confiance pour localiser les contre-exemples de type A mais n'arrive pas à bien généraliser au type B. Ces résultats se confirment dans le tableau 4.4 ligne 7 où cette architecture a une meilleure AUROCC pour le *problème* $A \rightarrow A$ que l'architecture TRF mais nettement moins bonne pour le *problème* $A \rightarrow B$ par rapport à l'utilisation de AMTL2 sur TRF.

Nous avons aussi essayé d'utiliser une fonction de confiance dédiée au rejet d'ambiguïté avec l'algorithme TRF sur un RBFN. Les résultats obtenus ligne 6 sont très bons pour le *problème* $A \rightarrow A$ mais pas pour le *problème* $A \rightarrow B$. Cette efficacité inattendue sur le premier problème s'explique par la présence de la frontière de décision traversant la classe A. Ces observations seront confirmées par le passage à l'échelle. Ce bon résultat est dû au hasard puisque si les contre-exemples de type A n'avait pas été positionnés sur cette frontière, le problème aurait été plus compliqué pour cette fonction de confiance.

4.2.2.3 Architectures RC et SC

Les architectures RC (figure 4.4(f)) et SC (figure 4.4(g)) ont de bonnes performances pour le *problème* $A \rightarrow A$. En effet elles rejettent bien les contre-exemples de type A. Par contre les zones de rejet ne sont pas généralisées aux contre-exemples de type B. Ces observations sont confirmées par les lignes 8 et 9 du tableau 4.4 : l'AUROCC pour le *problème* $A \rightarrow A$ est la meilleure de toutes les solutions mais les AUROCC des deux architectures sont faibles pour le *problème* $A \rightarrow B$ par rapport aux résultats de l'architecture TRF. De plus nous pouvons voir sur la figure 4.4(f) que les frontières de décision sont modifiées pour l'architecture RC par rapport au classifieur sans rejet. Cette modification peut entraîner des dégradations des performances par rapport à une architecture ne modifiant pas le classifieur principal. Néanmoins cette variation n'est pas significative pour ce problème en deux dimensions.

4.2.2.4 Conclusion partielle

Les architectures SC et RC obtiennent la meilleure AUROCC pour le *problème* $A \rightarrow A$. L'architecture TRF avec l'algorithme d'apprentissage AMTL2 permet d'obtenir les meilleurs résultats pour le *problème* $A \rightarrow B$ et le *problème* $A \rightarrow A\&B$. Nous avons aussi montré qu'avec l'architecture TRF les classifieurs discriminants comme les MLP et les SVM sont moins adaptés au *problème* $A \rightarrow B$ et qu'il faut choisir les fonctions de confiance utilisées en accord avec la nature du rejet désirée.

4.2.3 Conclusions intermédiaires

Dans cette section nous avons présenté le comportement des différentes architectures pour les deux problèmes de rejet : le rejet d'ambiguïté puis le rejet de distance. Une vision en 2 dimensions permet de comprendre les fonctionnements des architectures et de tirer quelques premières conclusions. D'une façon générale l'utilisation des fonctions de confiance dans les architecture TRF et SCRF est intéressante pour ses capacités de généralisation. Cette capacité est un avantage : pour le rejet d'ambiguïté où les erreurs sont éparpillées et en forte concurrence avec les formes bien reconnues ; et pour le rejet de distance pour les problèmes $A \rightarrow B$ et $A \rightarrow AB$ où il faut délimiter les connaissances du classifieur avec peu d'information sur la localisation de ce qu'il faut rejeter.

D'un autre coté, les architecture SC et RC tirent partie des performances des classifieurs pour obtenir de bons résultats : pour le rejet d'ambiguïté les résultats sont comparables à l'utilisation des fonctions de confiance ; pour le *problème* $A \rightarrow A$ du rejet de distance ces architectures sont bien meilleures mais sont pénalisées pour les autres problèmes.

Néanmoins ces expérimentations en deux dimensions ne permettent pas de comparer les options de rejet et leurs algorithmes d'apprentissage plus en détail. C'est pourquoi les expérimentations dans un contexte réel sont nécessaires.

4.3 Résultats sur la base réelle

Dans cette section nous comparons les différentes architectures d'option de rejet dans un contexte réel d'utilisation pour le rejet d'ambiguïté puis pour le rejet de distance. Pour le rejet d'ambiguïté, nous utilisons la reconnaissance de chiffres isolés manuscrits. Pour le rejet de distance, nous utilisons la reconnaissance de chiffres manuscrits et le rejet des caractères manuscrits (majuscules ou minuscules). Les bases utilisées sont celles décrites dans la section 4.1.1.

Les trois classifieurs utilisés dans ces expérimentations sont les RBFN, les MLP et les SVM. A partir des données d'exemples D_E quatre classifieurs principaux ont été appris pour toutes les expérimentations (un par validation croisée). Ces classifieurs principaux sont appris comme expliqué dans la section 1.2 avec les paramètres suivants : le RBFN a quatre prototypes par classe, le MLP utilise 75 neurones cachés dans son unique couche cachée et le SVM utilise des noyaux Gaussiens avec une normalisation soft-max des sorties (déviation standard à 4000, $C = 100$). Ces paramètres ont été choisis pour optimiser le taux de reconnaissance moyen des classifieurs : 93.50% pour le RBFN, 92.64% pour les MLP et 95.50% pour le SVM.

4.3.1 Pour le rejet d'ambiguïté

Pour le rejet d'ambiguïté, les tables de résultats présentent l'AUERC, le taux d'erreur à 5% de rejet (ERR5) et le nombre de points de fonctionnement calculés pendant l'apprentissage de l'option de rejet (NOP).

4.3.1.1 L'architecture TRF

Les tables A.13, A.14 et A.15 en annexe comparent les AUERC ainsi que l'ERR5 pour les options de rejet sur les trois types de classifieurs principaux (respectivement les RBFN, les MLP et les SVM) avec l'architecture TRF utilisant différentes fonctions de confiance et différents algorithmes d'apprentissage.

Considérons tout d'abord les résultats utilisant AMTL1 sur le RBFN présentés dans la table A.13 en annexe. Nous pouvons constater qu'utiliser plusieurs seuils est bénéfique par rapport à n'en utiliser qu'un seul puisque l'AUERC et l'ERR5 de $\psi_{i,j}^{Amb}$ sont meilleures que ψ_1^{Amb}, ψ_i^{AmbCl} et $\psi_{\{i,j\}}^{AmbSym}$. On remarque de plus qu'il n'est pas du tout intéressant d'utiliser une fonction de confiance destinée au rejet de distance tel que ψ_i^{Fumera} ou ψ_i^{Dist}. Considérons maintenant les autres algorithmes d'apprentissage TDG, PSO et CMP toujours sur les RBFN dans la table A.13. On remarque que les résultats des options de rejet utilisant un seul seuil avec ψ_1^{Amb} sont presque identiques quelque soit l'algorithme d'apprentissage ce qui est normal puisque il n'y a qu'une seule possibilité et qu'un algorithme n'est donc pas nécessaire. Les différences viennent donc des points de fonctionnement choisis par les cas d'arrêts et la capacité de chaque algorithme à parcourir tout l'espace des compromis. Les différences ne sont pas assez grandes pour être significatives. Pour les fonctions de confiance à seuils multiples, AMTL1 est le plus performant dans tous les cas. Nous avons donc reporté dans la table 4.5 les résultats de

AMTL1 avec $\psi_{i,j}^{Amb}$ en ligne 1 puisqu'il s'agit de l'option obtenant les meilleurs résultats. La ligne 2 utilisant ψ_1^{Amb} correspond à une option à un seul seuil couramment utilisée dans la littérature. L'utilisation de CMP avec ψ_i^{Fumera} en ligne 3 correspond à l'option de rejet de Fumera.

Considérons maintenant aussi les résultats sur les autres classifieurs principaux dans les tables A.14 pour le MLP et A.15 pour le SVM. L'utilisation de plusieurs seuils reste l'approche la plus intéressante pour les MLP puisque c'est ψ_i^{Fumera} qui obtient le meilleur résultat. Cette fonction de confiance étant orientée pour le rejet de distance cela montre que les sorties des MLP permettent un rejet le long des frontières de décision. Bien que AMTL1 obtienne l'AUERC la plus faible, l'ERR5 est identique pour AMTL1 et CMP pour les MLP. AMTL1 permet donc de mieux parcourir les différents compromis. Nous avons donc reporté ces deux résultats lignes 4 et 5 de la table 4.5. Pour les SVM, c'est l'utilisation des fonctions de confiance dédiées au rejet de confusion qui permettent les meilleurs scores, l'effet de cette fonction de confiance est sûrement accentué par la normalisation *soft max* qui augmente le coté discriminant des scores des SVM. Mais ces fonctions de confiance ont toutes des résultats à peu près identiques que ce soit ψ_1^{Amb} avec un seul seuil ou les autres avec plusieurs seuils, c'est à dire que l'utilisation de plusieurs seuils n'apporte rien pour les SVM. Cela est probablement un effet de l'apprentissage des marges des SVM qui permettent de bien placer les frontières de décision entre les classes. Nous avons donc reporté les options de rejet AMTL1 avec $\psi_{i,j}^{Amb}$ en ligne 6 qui est un bon compromis entre le meilleur AUERC et le meilleur ERR5 et l'option proposée par Fumera CMP avec ψ_i^{Fumera} en ligne 7.

Il est intéressant de constater que pour ces trois classifieurs, même si AMTL1 et TGD ne donnent pas toujours les meilleurs ERR5, c'est toujours eux qui ont la meilleure AUERC pour une configuration donnée. AMTL1 est en premier et TGD en second.

Tab. 4.5 – AUERC, ERR5 (en %) et NOP pour le rejet d'ambiguïté utilisant l'architecture TRF (extrait des tables A.13, A.14 et A.15)

	Principal	Option de Rejet	100*AUERC	100*ERR5(%)	NOP
1	RBFN	TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.643±0.069	3.84± 0.47	103
2	RBFN	TRF (AMTL1, ψ_1^{Amb})	0.682± 0.06	4.08± 0.4	103
3	RBFN	TRF (CMP, ψ_i^{Fumera})	1.01 ±0.072	4.47± 0.32	103
4	MLP	TRF (AMTL1, ψ_i^{Fumera})	0.655±0.053	3.66± 0.45	103
5	MLP	TRF (CMP, ψ_i^{Fumera})	0.909± 0.1	3.66± 0.37	103
6	SVM	TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.507±0.026	2.47± 0.13	103
7	SVM	TRF (CMP, ψ_i^{Fumera})	1.82 ± 0.18	3.43± 0.34	103

Comme le montre la table 4.5, on peut voir que notre algorithme AMTL1 combiné avec des fonctions de confiance adaptées au rejet d'ambiguïté permet d'obtenir de meilleurs résultats que les approches existantes. Si on compare maintenant les trois classifieurs entre eux, c'est le SVM qui obtient le meilleur AUERC et ERR5, ce qui

n'est pas étonnant puisqu'il est aussi le meilleur classifieur sans rejet.

4.3.1.2 L'architecture SC

La table A.16 en annexe présente les résultats pour l'architecture SC pour les trois classifieurs principaux en utilisant comme classifieur de rejet les trois types de classifieurs. Nous pouvons constater que l'utilisation des SVM en tant que classifieur de rejet est la meilleure solution quelque soit le classifieur principal (meilleures ERR5 et AUERC proches dans les trois cas). Il faut remarquer qu'à chaque fois les classifieurs de rejet sont triés dans l'ordre de leur pouvoir discriminant : les SVM sont meilleurs que les MLP qui sont meilleurs que les RBFN. Nous avons donc reporté dans la table 4.6 les résultats obtenus pour les trois classifieurs avec un SVM comme classifieur de rejet.

Tab. 4.6 – AUERC, ERR5 (en %) et NOP pour le rejet d'ambiguïté utilisant l'architecture SC, extrait de la table A.16.

Principal	Rejet	100*AUERC	100*ERR5(%)	NOP
RBFN	SVM	0.969±0.087	4.19± 0.31	103
MLP	SVM	0.966± 0.12	3.77± 0.43	103
SVM	SVM	1.29 ±0.056	3.46± 0.14	103

Si on compare les ERR5 et les AUERC avec les résultats obtenus par l'architecture TRF, on peut en déduire que TRF est nettement meilleure que l'architecture SC. Ces résultats s'expliquent par le fait que, comme nous l'avons vu en deux dimensions, l'architecture SC se spécialise aux zones où il y a le plus de rejet et a donc du mal à généraliser les zones de rejet le long des frontières de décision.

4.3.1.3 L'architecture RC

La table 4.7 présente les résultats pour l'architecture RC. Une classe de rejet est ajoutée au classifieur principal qui est alors réappris.

Tab. 4.7 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture RC

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
RC (RBFN)	2.09± 0.23	5.18± 0.17	103
RC (MLP)	2.20± 0.11	4.22± 0.32	103
RC (SVM)	2.25± 0.15	4.28± 0.29	103

Les résultats sont nettement moins bons que ceux obtenus par les architectures SC et TRF. En effet l'architecture RC cumule ici deux handicaps. D'abord, comme SC, cette architecture a du mal à généraliser les zones de rejet. Ensuite la classe de rejet

entre en confusion avec toutes les autres classes et même les SVM n'arrivent pas à faire mieux que l'architecture SC qui elle sépare le problème de classification du problème de rejet.

4.3.1.4 L'architecture SCRF

La table A.17 présente les résultats pour l'architecture SCRF pour les trois types de classifieurs principaux, différents classifieurs de rejet et différentes fonctions de confiance dédiées au rejet d'ambiguïté.

Si on considère d'abord le RBFN en classifieur principal, il faut remarquer que les résultats sont meilleurs lorsque les SVM sont utilisés en classifieur de rejet que les MLP et les RBFN. Il faut aussi noter que la dimension de l'espace des fonctions de confiance peut être grande (avec $\psi_{i,j}^{Amb}$) et que le nombre de données disponibles pour apprendre la classe de rejet (les erreurs) est relativement faible. Donc l'apprentissage des classifieurs de rejet devient difficile (voir impossible pour les RBFN en grande dimension). Nous avons donc utilisé des SVM comme classifieur de rejet pour les deux autres classifieurs principaux.

Ensuite on peut constater que avec le RBFN comme classifieur principal, c'est l'utilisation de $\psi_{i,j}^{Amb}$ qui est la plus efficace comme avec l'architecture TRF. Par contre pour les MLP et SVM, la dimension de la fonction de confiance a moins d'effet. C'est pourquoi nous avons reporté dans la table 4.8 les résultats utilisant comme classifieur de rejet un SVM et la fonction de confiance $\psi_{i,j}^{Amb}$.

Tab. 4.8 – AUERC, ERR5 (en %) et NOP pour le rejet d'ambiguïté utilisant l'architecture SCRF (avec un SVM comme classifieur de rejet et $\psi_{i,j}^{Amb}$, extrait de la table A.17).

Principal	Option de Rejet	100*AUERC	100*ERR5(%)	NOP
RBFN	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	0.598±0.046	3.71± 0.41	103
MLP	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	1.25 ±0.096	4.57± 0.44	53
SVM	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	0.579±0.026	2.57± 0.049	53

Pour le RBFN, les résultats de cette architecture sont légèrement meilleurs que ceux de l'architecture TRF sur le même classifieur. Dans ce cas, ces résultats s'expliquent par le fait que cette architecture arrive à tirer parti de l'information des fonctions de confiance mais aussi des performances du classifieur de rejet. Par contre les résultats sont moins bons pour les MLP et les SVM. Les difficultés de généralisation du classifieur de rejet cumulées avec des fonctions de confiance moins performantes sur ces classifieurs (à cause de leur nature) doivent être la cause de ces résultats moins bons que l'utilisation de seuils.

Si on compare ces résultats aux deux architectures RC et SC utilisant directement l'espace des caractéristiques du classifieur principal, on constate que l'architecture SCRF est meilleure quelque soit le classifieur principal. Ce qui montre que les fonctions de confiance permettent une meilleure discrimination des erreurs.

4.3.1.5 Conclusion pour le rejet d'ambiguïté

Nous pouvons conclure cette première partie des résultats en constatant que l'utilisation des fonctions de confiance (dans les architectures TRF et SCRF) est plus efficace que d'utiliser directement l'espace des caractéristiques (dans les architectures SC et RC) pour le rejet d'ambiguïté. Cette différence vient du fait qu'il est difficile de généraliser la localisation des erreurs dans l'espace des caractéristiques puisqu'elles sont réparties un peu partout le long des frontières. L'utilisation des fonctions de confiance permet de changer l'espace de représentation où les erreurs sont rassemblées dans une même zone où la confiance est faible. Dans ce nouvel espace, l'utilisation d'un classifieur de rejet discriminant (architecture SCRF) ou de seuils (architecture TRF) permettent une bonne généralisation (bonne AUERC) et de bonnes performances pour un problème donné (bonne ERR5).

Si on se replace dans notre contexte applicatif avec ses limitations, on choisira l'architecture TRF appliquée au RBFN avec AMTL1 et $\psi_{i,j}^{Amb}$. Ce choix permet d'utiliser un classifieur léger avec une option de rejet des plus efficaces en n'augmentant que très peu la complexité grâce aux seuils. Si on se place dans un contexte sans contraintes, il sera plus intéressant d'utiliser un SVM avec l'architecture TRF avec AMTL1 et $\psi_{i,j}^{Amb}$ puisque c'est l'option qui a les plus faibles AUERC et ERR5.

4.3.2 Pour le rejet de distance

Dans cette section nous analysons les résultats obtenus pour le rejet de distance avec les quatre architectures sur les données réelles : acceptation des chiffres, apprentissage du rejet sur les lettres minuscules (*problème A→A*) et test de généralisation sur les lettres majuscules (*problème A→B* et *problème A→A&B*). Notons que pour ce problème il est possible de comparer entre eux les types de classifieurs puisque le problème est identique pour chacun d'eux.

4.3.2.1 L'architecture TRF

La table 4.9 présente les AUROCC pour l'architecture TRF sur le classifieur principal de type RBFN dans le contexte de chacun des trois problèmes. Seuls les résultats obtenus par les meilleures fonctions de confiance pour chaque algorithme sont présentés.

Considérons d'abord le *problème A→A*, première colonne de la table A.18 en annexe. $\psi_i^{DistRBF}$ avec l'algorithme d'apprentissage AMTL1 obtient les meilleurs résultats pour ce problème (reportés ici ligne 1). D'une manière plus générale, l'utilisation de plusieurs seuils est plus performante que l'utilisation d'un seul seuil puisque pour AMTL1, TGD et PSO il y a toujours une fonction de confiance meilleure que ψ_{C1}^{Dist} et $\psi^{DistRbfW}$. Ce n'est pas le cas pour AMTL2 (reporté ligne 3 avec ψ_i^{Fumera}) puisqu'il est optimisé pour le *problème A→B*, il a donc forcément de moins bons résultats sur le *problème A→A*. De plus les fonctions de confiance prévues pour le rejet d'ambiguïté ($\psi_{i,j}^{Amb}$, $\psi_{\{i,j\}}^{AmbSym}$ et ψ_1^{Amb}) sont moins intéressantes que les fonctions adaptées au rejet de distance ($\psi_{i,j}^{Amb}$ est reportée ici ligne 2). Si on compare les différents algorithmes d'apprentissage, on

Tab. 4.9 – AUROCC et NOP pour le rejet de distance avec l'architecture TRF sur un RBFN, extrait de la table A.18 en annexe.

Option de rejet		$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
1	AMTL1, $\psi_i^{DistRBF}$	92.6±0.069	95 ±0.28	92.1±0.19	103
2	AMTL1, $\psi_{i,j}^{Amb}$	89.1± 0.68	89.9±0.78	88.5±0.68	103
3	AMTL2, ψ_i^{Fumera}	87.9± 0.36	94.9±0.26	88.1±0.31	103
4	TGD, ψ_i^{Dist}	90.5±0.064	94.2±0.082	89.8±0.33	103
5	CMP, ψ_i^{Fumera}	89.6± 0.11	93.3±0.38	89.1±0.32	53
6	PSO, ψ_i^{Dist}	88.5± 0.17	93.6±0.28	87.7±0.24	43

remarque que AMTL1 est loin devant les autres : PSO (avec ψ_i^{Dist}, reporté ligne 6) et CMP (avec ψ_i^{Fumera}, reporté ligne 5) font au mieux 89.6, puis vient TGD avec 90.5 (avec ψ_i^{Dist}, reporté ligne 4), la différence étant assez significative compte tenu des écarts-types, et enfin AMTL1 fait mieux avec ψ_i^{Fumera}, $\psi_i^{DistNRBF}$, ψ_i^{Dist} et $\psi_i^{DistRBF}$, la différence avec ce dernier (à 92.6, ligne 1) étant plus significative.

Pour le *problème* $A \rightarrow B$ (seconde colonne du tableau), AMTL1 obtient aussi la meilleure AUROCC mais cette fois la différence avec les autres algorithmes est moins significative. En effet l'utilisation d'un seul seuil permet une bonne généralisation qui profite à PSO et TGD. Si on ignore les deux fonctions de confiance ψ_{C1}^{Dist} et $\psi^{DistRbfW}$, AMTL1, AMTL2 et TGD obtiennent de meilleurs résultats que PSO et CMP. Ce qui est très intéressant c'est de remarquer que si aucun contre-exemple n'est disponible (*problème* $A \rightarrow B$ avec A vide) seul AMTL2 est disponible et il obtient des résultats très comparables voire meilleurs qu'avec les autres algorithmes qui eux utilisent les contre-exemples. Pour départager AMTL1 et AMTL2 sur le *problème* $A \rightarrow B$ nous avons comparé leur fausse acceptation sur la base D_B (FAR) pour une vraie acceptation fixée ($TAR = 0.85$) avec $\psi_i^{DistRBF}$. La figure 4.5 présente ces deux courbes en mettant en valeur le TAR fixé. AMTL2 obtient $FAR = 8.2 \pm 0.4$ et AMTL1 obtient $FAR = 9.5 \pm 0.9$. Donc dans cette configuration AMTL2 accepte moins de contre-exemples que AMTL1.

Pour le *problème* $A \rightarrow A\&B$, AMTL1 avec $\psi_i^{DistRBF}$ obtient les meilleures performances. En effet nous avons vu qu'il arrive à bien apprendre la localisation des contre-exemples de type A tout en généralisant aux contre-exemples de type B, cette capacité lui permet d'être le meilleur sur le problème mixte.

La table 4.10 présente un extrait des meilleurs résultats pour l'architecture TRF appliquées aux deux autres classifieurs : le MLP et le SVM.

Nous pouvons d'abord remarquer que pour les trois problèmes, ces deux classifieurs sont beaucoup moins efficaces que les RBFN. Ensuite on constate dans les tables A.19, A.20 et A.21 en annexes que pour chacun d'eux l'utilisation de plusieurs seuils est plus performante que l'utilisation d'un seul seuil. Mais ce qu'il y a de plus

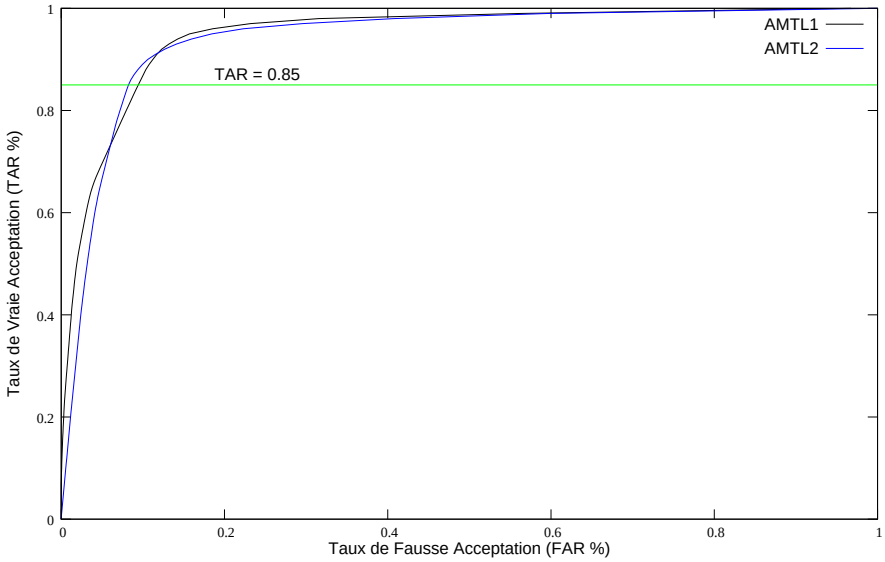


FIG. 4.5 – Comparaison entre les courbes ROC moyennes pour le *problème* $A \rightarrow B$ des options de rejet de distance d'architecture TRF apprise avec les algorithmes AMTL1 et AMTL2.

TAB. 4.10 – AUROCC et NOP pour le rejet de distance avec l'architecture TRF sur MLP et SVM, extrait des tables A.19, A.20 et A.21.

	Classifieur	Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
1	MLP	PSO, ψ_i^{Dist}	81.2±0.61	83.7± 2	81.5± 0.55	43
2	MLP	AMTL1, $\psi_{i,j}^{Amb}$	82.1± 1.5	83.7± 2.1	81.7± 1.8	103
3	MLP	AMTL2, ψ_i^{Dist}	75.4±0.78	74.3± 3.0	75.6± 1.0	103
4	SVM avec SM	AMTL1, $\psi_{i,j}^{Amb}$	87.9±0.24	89.3±0.45	88.1±0.25	103
5	SVM avec SM	TGD, ψ_i^{Fumera}	85.4±0.16	88.6±0.69	85.5±0.67	53
6	SVM avec SM	AMTL2, ψ_i^{Fumera}	81.6±0.46	86.3± 2.1	82.7±0.81	103
7	SVM	TGD, ψ_i^{Dist}	83.8±0.52	88.7±0.93	84.3± 0.9	53

inattendu est que l'utilisation d'une fonction de confiance dédiée au rejet de confusion ($\psi_{i,j}^{Amb}$ ici) soit plus efficace que l'utilisation des scores de sortie du classifieur. Nous avons donc reporté dans la table 4.10 les résultats des meilleurs options pour chaque type de fonctions de confiance (ligne 1 et 2 pour le MLP et ligne 4 et 5 pour le SVM avec normalisation). En considérant les résultats médiocres de AMTL2 utilisant ψ_i^{Dist} sur le MLP (ligne 3) et ψ_i^{Fumera} sur le SVM (ligne 6) on comprend mieux la cause : les scores de sorties du MLP et du SVM normalisé ne sont pas de bons indices de confiance pour déterminer si la forme appartient ou non aux formes à accepter. En effet nous avons vu que pour le MLP en dehors de l'espace des données d'apprentissage des classes prin-

cipales, plusieurs classes peuvent avoir comme scores de sortie une valeur élevée, il est donc plus intéressant de considérer une fonction de confiance qui détecte si les scores sont proches. De même pour les SVM utilisant la normalisation. Sans normalisation du SVM, on peut voir dans la table A.20 qu'il n'y a pas cet effet et la meilleure option de rejet utilise des fonctions de confiance dédiées au rejet de distance (ψ_i^{Dist} reporté ligne 7). Les SVM ont donc bien une information de description intrinsèque dans leurs scores de sortie mais pas suffisante par rapport à l'utilisation de la normalisation avec une fonction de confiance pour le rejet d'ambiguïté.

Ces mauvais résultats sur les SVM et les MLP se retrouve aussi sur les problèmes $A{\rightarrow}B$ et $A{\rightarrow}AB$ pour les mêmes raisons.

Remarquons qu'en utilisant les fonctions de confiance dédiées au rejet de distance sur les SVM, c'est l'algorithme TGD qui obtient les meilleurs résultats (lignes 5 et 7).

4.3.2.2 L'architecture SCRF

La table 4.11 présente les meilleurs résultats obtenus pour le rejet de distance avec l'architecture SCRF pour les trois types de classifieurs.

Tab. 4.11 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture SCRF, extrait de la table A.22.

Principal	Option de rejet	$A{\rightarrow}A$	$A{\rightarrow}B$	$A{\rightarrow}AB$	NOP
RBFN	SCRF (SVM sur ψ_i^{Dist})	95 ±0.15	95.6±0.094	93.3±0.22	53
MLP	SCRF (SVM sur ψ_i^{Dist})	84.7± 1.1	85.9± 2	84.6± 1.6	53
SVM	SCRF (SVM sur ψ_i^{Dist})	93.5±0.17	94.1± 0.15	91.9±0.22	53

D'après les résultats de la table A.22 en annexe, c'est l'utilisation d'un SVM comme classifieur principal qui permet les meilleurs résultats pour les trois problèmes, ce qui s'explique par les capacités discriminantes supérieures du SVM sur les deux autres classifieurs.

L'architecture SCRF a de meilleurs résultats pour le *problème* $A{\rightarrow}A$ avec chacun des trois classifieurs que l'architecture TRF. En effet le classifieur de rejet permet un meilleur apprentissage de la zone de rejet dans l'espace des fonctions de confiance que l'utilisation de simples seuils. Les résultats pour le *problème* $A{\rightarrow}B$ sont comparables (95.6 contre 95 pour le RBFN par exemple) aux résultats obtenus par TRF. En effet l'utilisation d'un classifieur par rapport aux seuils n'apporte que peu de capacités de généralisation (la part la plus importante de l'information étant apportée par les fonctions de confiance). Notons que pour un *problème* $A{\rightarrow}B$ avec A vide, cette architecture n'est pas disponible.

Ce qui est très intéressant concernant SCRF sur les MLP et les SVM comme classifieurs principaux par rapport aux résultats obtenus par TRF, c'est que l'utilisation d'un classifieur de rejet permet de dépasser les difficultés liées à la mauvaise qualité des fonctions de confiance pour le rejet sur ces classifieurs. Même la normalisation pour

les SVM n'est plus nécessaire. Il faut donc nuancer les conclusions données dans la section précédente : les fonctions de confiance de ces classifieurs ne sont pas assez informatives d'un point de vue description intrinsèque pour être utilisées efficacement avec simplement des seuils (architecture TRF) mais elles portent quand même suffisamment d'informations pour être utilisées avec des outils plus puissants comme un classifieur de rejet.

4.3.2.3 L'architecture SC

La table 4.12 présente les résultats pour l'architecture SC pour les trois types de classifieur de rejet. Il faut noter que le type du classifieur principal n'a ici aucune influence sur les résultats du rejet.

Tab. 4.12 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture SC.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
SC (RBFN)	88.5± 0.11	85.5± 0.8	84.4±0.21	53
SC (MLP)	86.4± 0.87	87.3± 1.5	82.9±0.75	53
SC (SVM)	96.7±0.052	95.3±0.28	93.7±0.25	53

L'utilisation d'un SVM comme classifieur de rejet est un meilleur choix que d'utiliser un RBFN ou un MLP. En effet c'est le SVM qui a les meilleures capacités de discrimination entre ces trois classifieurs.

Par rapport aux architectures présentées précédemment, les résultats de SC avec un SVM sont meilleurs pour le *problème* $A \rightarrow A$ (96.7 contre 95 pour SCRF et 92.6 pour TRF) et équivalent pour le *problème* $A \rightarrow B$ au résultat obtenu par TRF sur les RBFN (95.3 contre 95). Pour le *problème* $A \rightarrow A\&B$ c'est l'architecture SC qui est la meilleure.

4.3.2.4 L'architecture RC

La table 4.13 présente les résultats obtenus pour l'architecture RC pour les trois types de classifieurs.

Tab. 4.13 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture RC.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
RC (RBFN)	93.4±0.26	92.9±0.28	91.2±0.36	53
RC (MLP)	94.8±0.32	94.3±0.53	91.7±0.16	53
RC (SVM)	96.8±0.12	96.2±0.19	94.4±0.12	53

L'architecture RC sur un SVM est la plus performante pour tous les problèmes, tous classifieurs confondus. Concernant le RBFN, l'AUROCC pour le *problème A→B* est moins bonne que celle des architectures TRF et SCRF conformément à ce qui était envisagé dans le cas en deux dimensions. Ce n'est par contre pas le cas pour le SVM et le MLP : l'architecture TRF doit être handicapée par la mauvaise qualité des fonctions de confiance de ces classifieurs mais l'architecture RC arrive à tirer parti de la présence de toutes les classes en même temps pour généraliser le rejet.

Il y a deux inconvénients à cette architecture RC. Premièrement si aucun contre-exemple n'est disponible, il n'est pas possible d'apprendre une classe de rejet. Deuxièmement le classifieur principal est modifié et la complexité du problème de classification est augmenté, il peut donc y avoir une baisse des performances.

Nous avons donc comparé la Performance pour les architecture SC (SVM en classifieur principal et en classifieur de rejet) et RC (SVM en classifieur principal) pour le *problème A→A* qui obtiennent des AUROCC très proches (96.8 et 96.7) dans la figure 4.6.

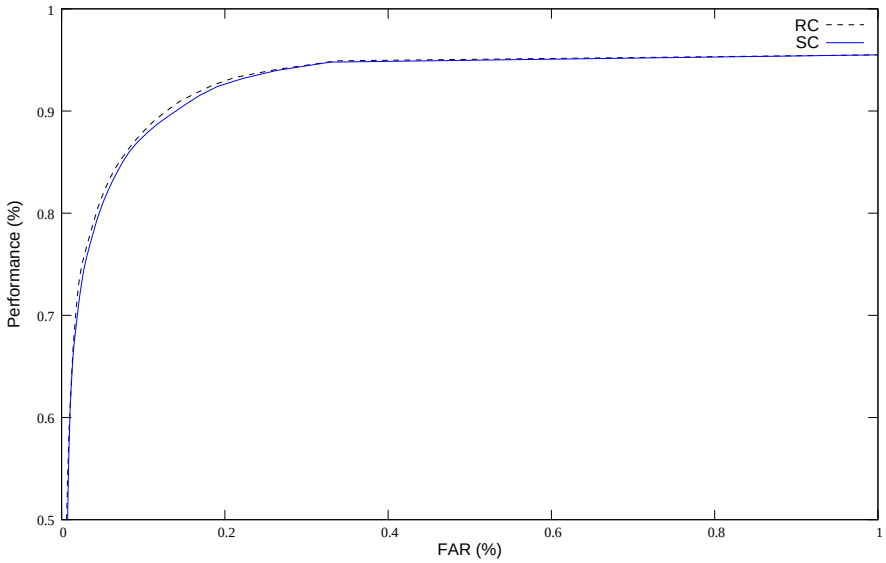


FIG. 4.6 – Comparaison entre les courbes de Performance moyennes pour le *problème A→A* des options de rejet de distance d'architecture RC et SC, avec un SVM en classifieur principal (SVM en classifieur de rejet).

Nous pouvons constater qu'il n'y a pas de différence significative de performance entre les deux options de rejet. Ce comportement quasi identique des deux options peut s'expliquer par l'architecture *un contre tous* du classifieur SVM utilisé. En effet dans l'option RC, le SVM *contient* un sous-classifieur SVM bi-classe *Classe de rejet contre toutes les autres* qui correspond exactement au classifieur spécialisé de rejet de l'architecture SC. La différence étant entre le SVM principal de SC et les autres sous-SVM de RC, puisque les contre-exemples sont dans l'un mais pas dans l'autre.

L'architecture SC utilisant des SVM reste donc la plus performante lorsqu'il n'y a aucune contrainte.

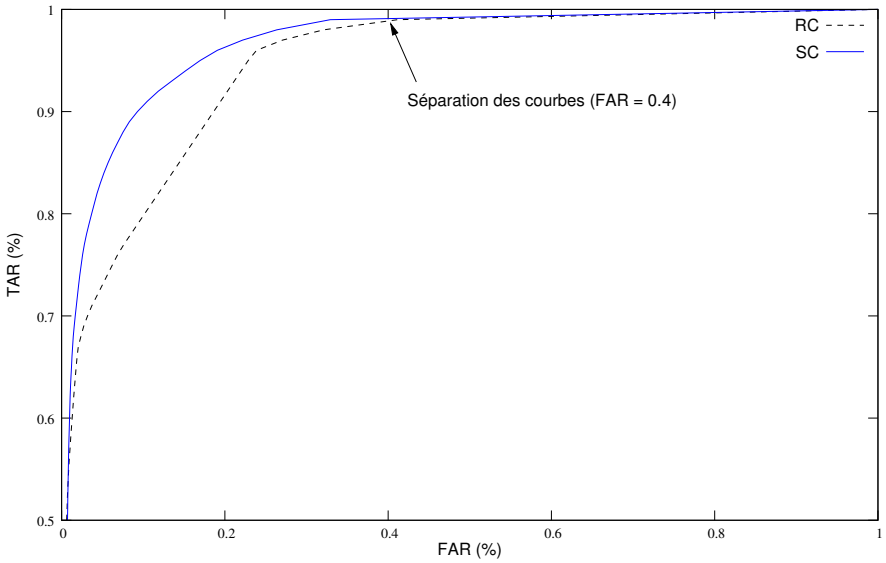
Les différences d'AUROC obtenues par ces architectures pour les classifieurs RBFN et MLP sont assez importantes pour qu'une modification des performances ne soit pas significative (96.7 pour SC contre 93.4 pour RC(RBFN) et 94.8 pour RC(MLP)). Malgré cela si nous comparons pour le MLP les courbes ROC du *problème* $A \rightarrow A$ pour ces architectures SC et RC (figure 4.7(a)) nous pouvons constater que les TAR sont équivalents jusqu'à 0.4 de FAR (séparation mise en valeur par la flèche sur le graphique). L'utilisateur qui désire ce compromis TAR/FAR, choisira l'architecture RC qui est moins coûteuse que l'architecture SC (qui utilise un SVM en classifieur de rejet). Si on considère maintenant la performance obtenue par ces architecture en ce point (figure 4.7(b)), on voit que l'architecture RC a une moins bonne performance. L'utilisateur devra donc changer son choix pour l'architecture SC.

4.3.2.5 Conclusion pour le rejet de distance

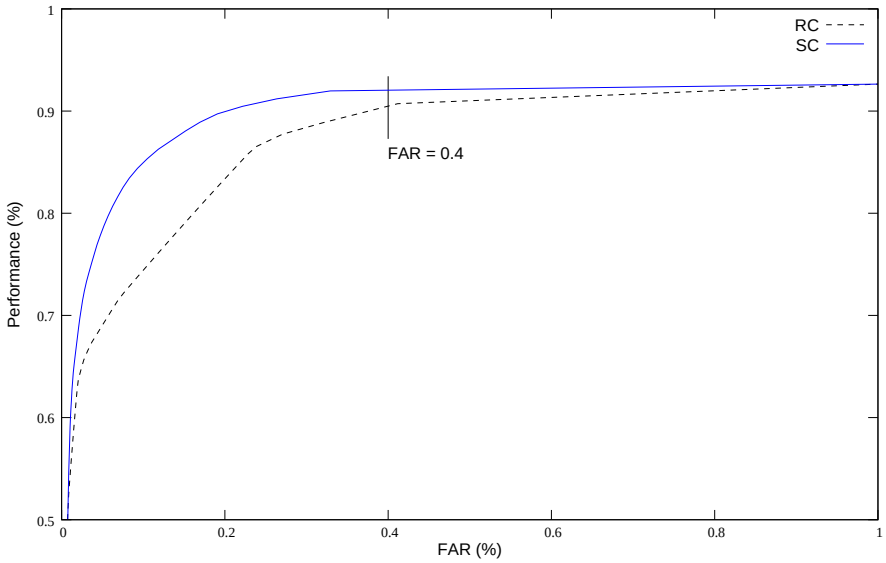
Nous avons montré que, pour le rejet de distance, l'architecture RC utilisée avec un classifieur très performant comme les SVM obtient les meilleures résultats quelque soit le problème considéré. Utiliser avec d'autres classifieurs plus légers et moins performants, cette architecture diminue son efficacité de rejet mais peut aussi avoir des effets sur les performances de classification du système.

Cette architecture optimale étant très coûteuse en ressources, elle ne convient pas à notre contexte applicatif. Nous lui préférerons donc dans notre contexte l'architecture TRF. Nous avons confirmé que pour le rejet de distance avec l'architecture TRF il est plus intéressant d'utiliser plusieurs seuils qu'un seul pour améliorer les performances et ce quelque soit le classifieur. Par contre nous avons montré concernant cette architecture que son efficacité dépend beaucoup de la qualité des fonctions de confiance disponibles. D'un côté le RBFN met à disposition de très bons indices de confiance et les fonctions dédiées au rejet de distance sont les plus efficaces. D'un autre côté pour les MLP et les SVM, qui sont des classifieurs plus discriminants, les fonctions de confiance ne sont pas aussi efficaces.

L'architecture TRF garde l'avantage de mettre à disposition des algorithmes d'apprentissage permettant d'apprendre le rejet de distance lorsqu'aucun contre-exemple n'est disponible et avec des performances de rejet très bonnes, comparables à celle utilisant des contre-exemples (dans un contexte de ressources limitées).



(a) Courbe ROC



(b) Courbe Performance/FAR

FIG. 4.7 – Comparaison entre les courbes ROC (a) et Performance (b) moyennes pour le *problème* $A \rightarrow A$ des options de rejet de distance avec un MLP en classifieur principal d'architecture RC et SC (SVM en classifieur de rejet).

4.4 Bilan des résultats

Le tableau 4.14 résume les différentes conclusions intermédiaires données tout au long des résultats en mettant en confrontation le problème à résoudre et les contraintes liées au contexte d'utilisation.

TAB. 4.14 – Bilan des choix conseillés pour les options de rejet suivant le contexte d'utilisation.

<table>
<tr><td colspan="2" rowspan="2">Problème</td><td colspan="2">Contexte</td></tr>
<tr><td>Sans contraintes</td><td>Avec contraintes</td></tr>
<tr><td rowspan="4">Rejet de distance</td><td>$A \rightarrow A$</td><td rowspan="3">SVM avec RC</td><td rowspan="3">RBFN avec
TRF AMTL1
sur $\psi_i^{DistRBF}$</td></tr>
<tr><td>$A \rightarrow B$</td></tr>
<tr><td>$A \rightarrow AB$</td></tr>
<tr><td>$\emptyset \rightarrow B$</td><td colspan="2">RBFN avec
TRF AMTL2 sur ψ_i^{Fumera}</td></tr>
<tr><td colspan="2">Rejet
d'ambiguïté</td><td>SVM avec
TRF AMTL1 $\psi_{i,j}^{Amb}$</td><td>RBFN avec
TRF AMTL1 $\psi_{i,j}^{Amb}$</td></tr>
</table>

Nous avons considéré deux contextes d'utilisation du rejet tout au long de cette étude. Soit il n'y a aucune contrainte sur les ressources mises à disposition de l'option de rejet, soit on se place dans le cadre de notre domaine applicatif avec des restrictions sur les ressources en termes de temps de calcul et en mémoire. Pour le rejet de distance, on peut distinguer les trois problèmes $A \rightarrow A$, $A \rightarrow B$ et $A \rightarrow AB$ plus un cas particulier où A est vide dans le *problème* $A \rightarrow B$. Pour ce cas particulier, seule l'option TRF avec l'algorithme d'apprentissage AMTL2 est disponible et c'est sur les RBFN que cette option de rejet est la plus efficace. Pour les trois autres problèmes, dans le cas avec contraintes, c'est l'utilisation de TRF avec AMTL1 sur un RBFN qui est le plus avantageux par rapport à l'architecture RC sur un SVM conseillée sans contraintes. En effet le RBFN couplé avec TRF demande peu de ressources par rapport au SVM tout en ayant de bons résultats grâce aux propriétés descriptives du RBFN et à la capacité de généralisation des seuils. Par contre l'architecture RC tire parti des capacités discriminantes du SVM. Pour le rejet d'ambiguïté, c'est l'architecture TRF qui est la plus avantageuse dans les deux contextes d'utilisation : soit avec un RBFN lorsqu'il y a des contraintes, soit avec un SVM lorsqu'un classifieur plus coûteux peut être utilisé.

4.5 Conclusion et discussion

Dans cette partie nous avons étudié avec précision la notion de rejet dans le but d'aider un utilisateur à élaborer une option de rejet en adéquation avec son problème.

Nous avons d'abord montré qu'il est important de définir la nature du rejet désiré : rejet d'ambiguïté et de distance. En effet suivant le cas il ne faudra pas faire les mêmes choix d'architectures et les méthodes de test du rejet seront différentes.

Ensuite nous avons présenté quatre architectures simples et génériques : TRF qui utilise des seuils sur des fonctions de confiance, SCRF qui utilise un classifieur de rejet sur ces fonctions de confiance, SC basée sur un classifieur spécialisé indépendant et RC ajoutant une classe de rejet au classifieur principal. Il existe beaucoup d'autres architectures que nous n'avons pas étudiées, soit par manque de temps (utilisation d'un ensemble de caractéristiques différentes pour le classifieur de rejet ou sélection parmi les caractéristiques du classifieur principal, apprentissage du classifieur principal avec des rebuts puis utilisation de seuils pour la décision, ...), soit parce que ces architectures dépendent d'un problème bien spécifique (utilisation d'un modèle de langage, ...). Pour chaque architecture, les algorithmes d'apprentissage ont été détaillés et expliqués et pour l'architecture TRF nous avons proposé trois algorithmes innovants AMTL1, AMTL2 et TGD.

Pour comparer ces options de rejet et leurs différents algorithmes d'apprentissage, nous avons décrit et étendu les protocoles classiques de test du rejet. Notamment nous avons expliqué comment calculer la courbe ROC pour les options de rejet qui ont un paramétrage discret comme avec les architectures SC ou RC. Pour le rejet d'ambiguïté nous introduisons la notion d'aire sous la courbe Erreur/Rejet qui permet (par analogie avec l'aire sous la courbe ROC) de comparer des options de rejet d'un point de vue global.

Pour le rejet de distance nous avons montré qu'il existe plusieurs types de problèmes suivant que les contre-exemples sont disponibles ou non. C'est pourquoi nous avons utilisé trois bases de tests différentes pour ce rejet.

Les tests exhaustifs que nous avons effectués, combinant les différentes architectures, leurs différents algorithmes d'apprentissage et les différents types de classifieurs, ont été effectués sur deux bases : une artificielle en deux dimensions qui permet de visualiser facilement les effets des différents choix, et une traitant un cas réel, la reconnaissance des chiffres et le rejet des lettres.

Ces tests nous ont permis de proposer pour chaque problème de rejet (rejet d'ambiguïté, de distance, avec ou sans contre-exemples) et deux contextes d'utilisation (avec ou sans contraintes de ressources) l'option de rejet la plus appropriée parmi celles présentées. L'extension de l'architecture TRF que nous proposons avec nos algorithmes d'apprentissage permet d'apprendre une option de rejet sans contre-exemples et d'obtenir des options de rejet de distance et d'ambiguïté efficaces avec un faible coût en ressources. C'est donc cette architecture que nous avons retenue pour la suite de nos expérimentations.

Deuxième partie

S'adapter pour corriger les erreurs

Introduction à l'adaptation

Avec l'émergence des PDA et autres *smartphones* utilisant des interfaces orientées stylo et n'ayant plus de clavier, une forte fiabilité des systèmes de reconnaissance de l'écriture manuscrite est nécessaire. En effet pour rendre possible l'utilisation d'interfaces utilisant la reconnaissance de l'écriture, il faut que les performances de ces systèmes soient supérieures à l'utilisation d'un clavier (virtuel ou réel), sinon l'utilisateur se lasse et revient aux anciennes interfaces. Une première solution est d'utiliser des systèmes omni-scripteurs très complexes par la quantité de connaissances et les processus de reconnaissance qu'ils utilisent. Il est aussi possible de considérer le problème de la reconnaissance de mots ou de phrases, qui permet de résoudre des ambiguïtés au niveau des caractères en ajoutant des connaissances sur le lexique utilisé ou le modèle de langage. Mais ces approches sont très coûteuses en ressources qui ne sont pas disponibles en quantité sur les petits périphériques mobiles.

Il nous faut alors concevoir des systèmes qui doivent combiner un fort taux de reconnaissance et un faible coût en ressources (en terme de calcul et de mémoire) et les systèmes omni-scripteurs ne permettent pas d'allier ces deux propriétés, même en se limitant à la reconnaissance de caractères isolés. Il y a deux raisons pour lesquelles les systèmes omni-scripteurs sont aussi complexes. D'abord il existe un grand nombre de styles d'écriture différents comme l'ont montré les travaux sur la vérification de scripteurs [102]. De plus, parmi les différents styles d'écriture, certains caractères appartenant à des classes différentes se ressemblent beaucoup. En effet comme le montre la figure 4, en passant d'un scripteur à l'autre, les formes des lettres de différentes classes peuvent être très proches. Il est même possible de passer progressivement de la lettre `u` à la lettre `h` en passant par les lettres `v`, `r` puis `n`.

D'un autre côté, cet exemple montre aussi qu'un même scripteur utilise très rarement deux allographes proches pour deux classes différentes. En effet, si le `u` du scripteur 1 peut ressembler au `v` du scripteur 2, les caractères `u` et `v` de chacun des deux scripteurs ne se ressemblent pas. Dans un contexte d'utilisation réel, les systèmes de reconnaissance de caractères utilisés sur les petits périphériques mobiles n'ont à reconnaître l'écriture que d'un seul scripteur à la fois, le propriétaire de l'appareil. Ces deux constatations nous ont amené à proposer l'utilisation de systèmes de reconnaissance mono-scripteurs. Ces systèmes pourront être plus simples, et donc plus légers, puisque la discrimination des différents caractères d'un même scripteur est plus facile que dans le cas omni-scripteur.

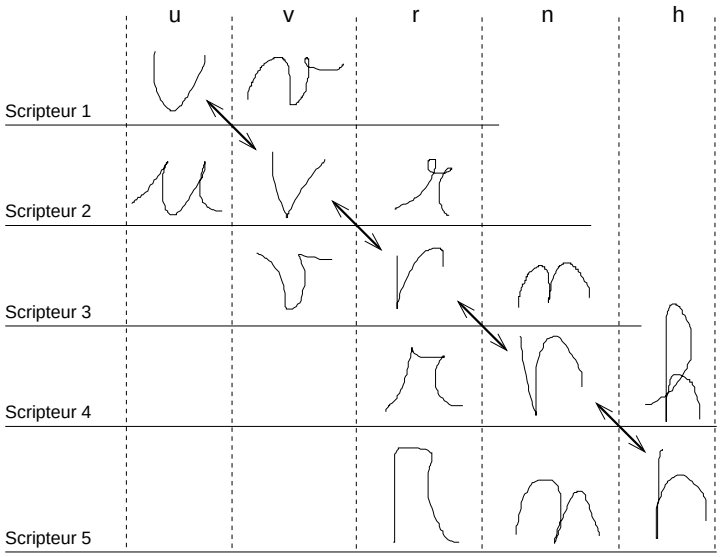


FIG. 4: Exemple de caractères ambigus d'un scripteur à l'autre, mais pas pour un scripteur donné.

Pour obtenir un classifieur mono-scripteur, il y a plusieurs possibilités. La première consiste à demander au scripteur de saisir plusieurs dizaines de fois chaque caractère, puis d'apprendre un classifieur avec ses données d'apprentissage. Cette approche n'est pas envisageable dans notre cas puisque la saisie initiale est très fastidieuse et donc un obstacle à l'utilisation du système pour un utilisateur lambda. La seconde spécialise un système omni-scripteur pour la reconnaissance de l'écriture d'un seul scripteur, ce processus est appelé *adaptation*. Nous pouvons distinguer deux modes d'adaptation :

- l'*enrôlement* qui correspond à une adaptation hors-ligne en deux étapes : acquérir les données de l'utilisateur puis spécialiser le système omni-scripteur à ces données. Le classifieur n'est ensuite plus modifié.
- l'*incrémental* qui correspond à une adaptation en-ligne : la spécialisation se fait à la volée, au fur et à mesure de la saisie de nouvelles données par l'utilisateur.

Dans ces deux cas, l'utilisation du système omni-scripteur initial permet de profiter d'une partie de ses connaissances pour ne pas commencer l'apprentissage depuis zéro. Par contre, l'enrôlement nécessite toujours la saisie par l'utilisateur de caractères d'apprentissage avant de pouvoir utiliser le système. C'est pourquoi nous proposons dans cette partie une stratégie incrémentale en-ligne d'adaptation au scripteur. Ainsi l'adaptation est transparente pour l'utilisateur qui n'a pas d'effort particulier à faire à part utiliser normalement le système.

L'adaptation incrémentale est confrontée à plusieurs problèmes auxquels nous tentons de répondre. D'abord la classe des nouveaux caractères de l'utilisateur doit être connue. Ces nouvelles données peuvent être étiquetées par l'utilisateur lui même via une interface adéquate (adaptation supervisée) ou automatiquement par le système via un lexique par exemple (adaptation non supervisée). Nous avons supposé dans ce

chapitre que les étiquettes des données sont validées par l'utilisateur via l'interface de saisie, donc connues. Le second problème est la faible quantité de données disponibles. En effet pour que l'adaptation soit rapide, il ne faut pas attendre que l'utilisateur ait saisi beaucoup de caractères pour modifier le système. Enfin il faut que l'adaptation soit stable, c'est-à-dire qu'il ne faut pas que les performances du système se dégrade au cours du temps à force de modifications.

Dans cette partie nous présentons une stratégie d'adaptation rapide, stable et transparente pour l'utilisateur. Le chapitre 5 étudie la possibilité de synthétiser des caractères artificiels pour palier le manque de données disponibles pour l'adaptation. Le chapitre 6 présente notre stratégie d'adaptation incrémentale de Système d'Inférence Flous, basée sur notre méthode ADAPT. Le chapitre 7 présente les résultats de tests simulant l'adaptation au scripteur d'une façon de plus en plus réaliste jusqu'à la mise en application réelle du processus sur un périphérique mobile, puis conclura cette partie.

Chapitre 5

Synthèse de caractères en-ligne

Avant de présenter notre approche pour l'adaptation, nous abordons un problème très lié : comment palier le manque de données disponibles dans un apprentissage à partir de peu d'exemples ? En effet, lors de l'adaptation à la volée d'un système, ce qui prend le plus de temps c'est de rassembler suffisamment de données pour permettre un apprentissage efficace. Un processus d'adaptation doit être capable d'apprendre un nouveau style à partir de très peu d'exemples. En effet il ne faut pas devoir demander à l'utilisateur de saisir plusieurs fois chaque caractère avant de pouvoir utiliser un système performant. Dans notre contexte d'adaptation au scripteur, nous proposons d'utiliser la synthèse de caractères manuscrits en-ligne pour augmenter la quantité de données disponibles pour l'adaptation.

Cette synthèse de caractères manuscrits en-ligne doit permettre d'apprendre un classifieur capable de reconnaître l'écriture d'un seul scripteur, il s'agit donc d'un problème d'apprentissage mono-scripteur. Les données générées doivent donc être représentatives de l'écriture de ce scripteur c'est-à-dire qu'elles doivent ressembler aux caractères saisis par le scripteur mais doivent aussi contenir suffisamment de variabilité pour absorber la variabilité de l'écriture du scripteur.

L'approche que nous proposons dans ce chapitre consiste à utiliser quelques exemples (de deux à dix) saisis par le scripteur que nous déformons ou combinons pour en générer un grand nombre. Nous présentons donc ici plusieurs techniques de déformation de caractères : des techniques couramment utilisées dans le domaine de la reconnaissance d'image (section 5.2) et deux déformations innovantes dédiées à l'écriture en-ligne (section 5.3). Pour combiner les caractères entre eux nous utilisons le principe intuitif de l'analogie (section 5.4) : les caractères synthétisés doivent être analogues aux exemples d'origine. Ces trois approches de la synthèse de caractères étant complémentaires, nous proposons dans la section 5.5 trois stratégies pour les combiner et obtenir plus de variabilité autour du style du scripteur.

Pour évaluer nos processus de synthèse de caractère de façon indépendante du processus d'adaptation présenté au chapitre suivant, nous considérons dans la section 5.6 le problème de l'apprentissage d'un système de reconnaissance mono-scripteur à partir des données générées. Si les systèmes obtenus sont performants, nos stratégies de syn-

thèse de caractères en-ligne seront capables de générer des caractères représentatifs du style d'écriture du scripteur pour améliorer l'efficacité de l'adaptation.

5.1 État de l'art sur la synthèse de caractères

La synthèse de caractères est un problème qui a été souvent abordé dans le domaine de la reconnaissance de l'écriture hors-ligne. La synthèse se fait à partir d'un ou plusieurs exemples qui sont déformés par différentes opérations. En hors-ligne le classifieur travaille sur une image du caractère à reconnaître, donc les déformations utilisées sont des déformations d'image. Nous appellerons *déformations hors-ligne* les déformations issues de ce domaine.

On peut distinguer dans cette communauté deux objectifs à cette synthèse. Le premier assez proche du nôtre est d'augmenter la quantité de données disponibles pour l'apprentissage d'un classifieur (k-PPV [19], réseaux de neurones [101], Modèles de Markov Cachés [50, 107]...). Les déformations appliquées peuvent être assez simples comme l'inclinaison, l'étirement et la variation de l'épaisseur du trait dans [19] ou plus compliquées comme la distortion élastique (pour créer des « vagues » sur l'image) utilisée dans [101]. Dans un contexte de reconnaissance de textes manuscrits, les lignes de textes peuvent aussi être déformées, comme dans [50, 107] en utilisant les déformations classiques différemment tout au long de la ligne de texte.

Le deuxième objectif des déformations hors-ligne est d'annuler une déformation subie par le caractère avant l'acquisition. Dans ce but, certains auteurs [48] étudient les différentes déformations d'image pour essayer de retrouver l'image avant déformation afin de faciliter la reconnaissance. En plus des précédentes, deux nouvelles déformations peuvent être alors utilisées : la mise en perspective et la rotation.

À notre connaissance, il y a peu de travaux portant sur la synthèse de données spécifiques aux caractères manuscrits en-ligne. Néanmoins il existe des travaux connexes. Rappelons que dans le domaine en-ligne, le tracé est considéré comme une fonction paramétrique $p(t)$ (voir section 1.1.1.1). Nous appellerons *déformations en-ligne* les déformations spécifiques à l'écriture en-ligne.

On peut d'abord considérer les travaux sur la modélisation de l'écriture manuscrite [88, 89]. Dans [88, 89] la modélisation utilise une fonction paramétrique pour simuler le mouvement de la main de l'homme. Ils expliquent que des modifications de cette fonction entraînent des variations dans le style de l'écriture.

La synthèse d'écriture en-ligne a été aussi utilisée [47, 113] dans les interfaces homme-machine pour transformer des chaînes de caractères numériques (texte ASCII) en écriture manuscrite. Dans [113] un modèle Bayesien des graphèmes, des liaisons entre graphèmes et entre caractères sont appris à partir d'une base d'exemples d'écriture. Puis ces modèles sont appliqués pour générer le texte manuscrits. Dans [47], l'utilisateur doit d'abord écrire un ensemble de groupe de lettres appelés *glyphes* (lexique contenant 1000 glyphes de 1 à 4 lettres, dépendant de la langue). Les glyphes sont ensuite déformés et concaténés pour obtenir le texte manuscrit. Les déformations utilisées sont du type

hors-ligne (rotation, étirement, inclinaison, épaisseur et type de crayon).

Les déformations en-ligne peuvent aussi être utilisées pour rendre un classifieur plus robuste à la variabilité de l'écriture. Dans [100] des transformations du signal en-ligne sont étudiées pour élaborer une distance qui tient compte de ces possibles transformations. Cette distance est alors utilisée dans le classifieur qui devient robuste à ces transformations.

Dans [108] les travaux de [89] sont utilisés pour augmenter la taille de la base d'apprentissage d'un Modèles de Markov Cachés qui reconnaît des phrases hors-ligne. Les auteurs utilisent un modèle unique de caractères en-ligne qu'ils déforment et concatènent pour générer ensuite des images de phrases à partir d'un texte ASCII. Dans ces travaux un seul modèle de caractères, obtenu manuellement par un expert, est utilisé. Cette dernière approche est assez proche de la nôtre mais n'est pas directement applicable à nos travaux. En effet dans le cadre de l'adaptation en-ligne au scripteur, il faut que la synthèse de caractères soit automatique et doit se faire à partir des caractères saisis par l'utilisateur.

Les déformations présentées dans cette section ne sont pas toutes utilisables dans le contexte de l'écriture en-ligne. Par exemple la variation de l'épaisseur du tracé n'a pas de sens ici. De plus certains classifieurs effectuent des normalisations sur les caractères avant de les traiter pour rendre plus robuste la classification, donc les déformations correspondantes ne permettront pas d'apporter de nouvelles données. De plus il nous faut des approches qui n'utilisent pas un modèle unique de caractère mais les exemples disponibles du scripteur. Nous étudierons dans ce manuscrit certaines déformations inspirées du domaine hors-ligne dans la section 5.2 car celles-ci sont simples et pertinentes par rapport à nos besoins : l'étirement et l'inclinaison des caractères. Nous proposons dans la section 5.3 des déformations spécifiques à l'écriture en-ligne qui permettent des modifications du tracé non atteignables par les déformations hors-ligne. Enfin dans la section 5.4 nous utilisons le principe de l'analogie pour combiner plusieurs caractères d'origine en un nouveau caractère respectant le style des originaux.

5.2 Déformations du type image

Les déformations issues de la génération de données hors-ligne sont basées sur la déformation d'image. Nous présentons dans cette section, après une brève description du tracé en-ligne disponible, les plus utilisées : l'étirement et l'inclinaison de l'image.

Même si nous étudions des déformations du type hors-ligne, le signal disponible est en-ligne. Nous ne considérons pas les mouvements du crayon lorsqu'il ne touche pas la surface sensible. Chaque point $p(t)$ est défini par ses coordonnées $x_{(t)}$ et $y_{(t)}$. Le tracé n'est pas modifié après la saisie, à une translation près pour plus de simplicité. Nous supposerons donc que :

$$\min_t(x_{(t)}) = 0, \tag{5.1}$$

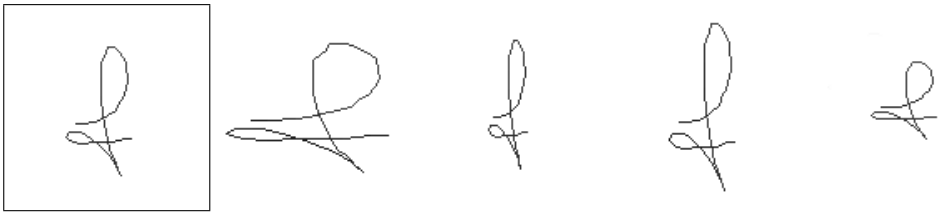
$$\min_t(y_{(t)}) = 0. \tag{5.2}$$

5.2.1 Étirements

Une image peut être étirée suivant l'axe x ou l'axe y. L'opération d'étirement dépend donc de deux paramètres α_x et α_y respectivement le coefficient de déformation suivant l'axe x et le coefficient de déformation suivant l'axe y. Pour un tracé en ligne, les coordonnées des points sont modifiées suivant les équations (5.3) et (5.4), $x'_{(t)}$ et $y'_{(t)}$ étant les coordonnées du tracé déformé :

$$x'_{(t)} = \alpha_x x_{(t)}, \tag{5.3}$$
$$y'_{(t)} = \alpha_y y_{(t)}. \tag{5.4}$$

La figure 5.1 donne l'exemple d'un caractère déformé par étirement suivant les axes x et y. On peut constater qu'une déformation par étirement conserve le style du scripteur si la déformation est raisonnable. Il faudra donc choisir les valeurs des coefficients pas trop éloignées de l'unité.



(a) Original, (b) Étiré suivant l'axe x, (c) Étiré suivant l'axe y.

FIG. 5.1 – Exemple d'un caractère déformé par étirement.

5.2.2 Inclinaison

L'inclinaison d'une image est différente de la rotation d'une image. En effet cette déformation se rapproche plus de l'écriture « penchée ». L'inclinaison dépend d'un paramètre α_i. Comme le montrent les équations (5.5) et (5.6) la coordonnée en y ne change pas mais la variation de $x_{(t)}$ dépend de la valeur de $y_{(t)}$:

$$x'_{(t)} = x_{(t)} + \alpha_i * y_{(t)}, \tag{5.5}$$
$$y'_{(t)} = y_{(t)}. \tag{5.6}$$

La figure 5.2 montre une lettre déformée par inclinaison dans les deux directions (vers la gauche et vers la droite). On peut constater que la forme de la lettre ne change pas mais que les lignes verticale sont inclinées. Le style est donc bien conservé si la valeur de α_i n'est pas trop éloignée de l'unité. En effet l'inclinaison est une propriété importante de l'écriture d'un scripteur et même si elle varie d'un caractère à l'autre, un scripteur ne change pas son sens d'inclinaison radicalement.

(a) Original, (b) Vers la gauche, (c) Vers la droite.

FIG. 5.2 – Exemple d'un caractère déformé par inclinaison.

5.3 Déformations en-ligne

Les déformations dites en-ligne utilisent des informations sur le tracé à déformer qui ne sont pas disponibles dans l'image correspondant à ce même tracé. En effet le tracé manuscrit n'est pas ré-échantillonné. Si les points sont plus éloignés à certains moments c'est que le scripteur a accéléré son tracé. Ainsi nous conservons la cinétique du tracé. L'ordre des points du tracé et leur cinétique peuvent être exploités dans les stratégies de déformation. Nous présentons dans cette section des déformations permettant de modifier la vitesse du tracé suivant sa direction ou de modifier la courbure du tracé.

D'autres déformations non exploitées ici sont possibles. Par exemple avec des caractères composés de plusieurs tracés il est possible de faire varier la position relative des deux tracés. Il est aussi possible de déphaser les fonctions $x(t)$ et $y(t)$ pour obtenir des courbes différentes comme expliqué dans [89].

Les déformations présentées dans cette section peuvent bien sûr être utilisées conjointement avec les déformations du type hors-ligne, comme c'est le cas dans nos expérimentations.

5.3.1 Modification de la vitesse

L'objectif de cette déformation est de déplacer certaines parties du caractère les unes par rapport aux autres, ce qui devrait permettre par exemple de fermer ou d'ouvrir des lettres comme le `e` ou le `s`. Pour cela nous proposons de modifier la longueur des traits verticaux ou horizontaux sans modifier les parties orientées différemment. Nous considérerons donc ici qu'un même scripteur peut faire un trait vertical ou horizontal plus ou moins long d'un caractère à l'autre sans changer son style.

Une des possibilités offertes par la connaissance de la cinétique du tracé est de pouvoir modifier la vitesse de ce tracé. Si la vitesse est modifiée de façon homogène sur l'ensemble de la suite des points il en résultera une simple homothétie. La vitesse du tracé à l'instant t est donnée par le vecteur séparant les deux points $p_{(t)}$ et $p_{(t+1)}$. Nous avons donc choisi de modifier la taille des vecteurs $\vec{V}_{(t-1)} = \left(x_{(t)} - x_{(t-1)}, y_{(t)} - y_{(t-1)}\right)$ suivant leur direction. Ainsi les vecteurs dont les directions sont proches des axes principaux pourront augmenter ou diminuer selon un paramètre α_v. Les vecteurs seront considérés comme proches d'un axe principal s'ils forment un angle inférieur à $\frac{\pi}{8}$ avec

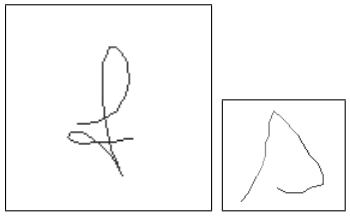
un des axes. Les équations (5.7) à (5.9) donnent les coordonnées des points du tracé après cette déformation :

$$x'_{(t)} = x'_{(t-1)} + \beta * (x_{(t)} - x_{(t-1)}), \tag{5.7}$$

$$y'_{(t)} = y'_{(t-1)} + \beta * (y_{(t)} - y_{(t-1)}) \tag{5.8}$$

$$\beta = \begin{array}{ll} 1 & \text{si } \arg(\vec{V}_{(t-1)}) \left[\frac{\pi}{2}\right] \in [\frac{\pi}{8}, \frac{3\pi}{8}] \\ \alpha_v & \text{sinon} \end{array} \tag{5.9}$$

La figure 5.3 donne un exemple du type de déformation obtenue en modifiant ainsi la vitesse du tracé. On peut constater sur le `f` déformé que la longueur du segment vertical du tracé change suivant la déformation ce qui a pour effet de rapprocher ou d'éloigner les deux boucles du `f`. De plus, le caractère `s` peut être plus ouvert ou même fermé suivant la déformation effectuée. Ce type de déformation n'aurait pas pu être obtenu par des déformations hors-ligne.



(a) Originaux,

(b) Vitesse augmentée,

(c) Vitesse diminuée.

FIG. 5.3 – Exemples de caractères déformés par modification de la vitesse.

5.3.2 Modification de la courbure

La déformation précédente permet d'agir surtout sur les lignes droites horizontales ou verticales du tracé. Nous proposons donc ici de déformer les boucles et les liaisons entre deux parties rectilignes, qui sont des éléments structurants de l'écriture. Les boucles pourront s'ouvrir et se fermer et les liaisons pourront modifier légèrement le positionnement relatif des différentes parties rectilignes. Pour cela nous proposons une déformation qui agit sur les courbes du caractère et non sur les parties rectilignes.

La modification de la courbure du tracé se fait en fonction d'un paramètre α_c. La courbure du tracé à l'instant t correspond à l'angle $\hat{\theta}$ formé par le point $p_{(t)}$ avec le point

précédent et le point suivant (équation (5.10)). Pour ne pas complètement modifier le caractère et conserver sa structure, il y a certaines parties du tracé qu'il ne faut pas modifier, par exemple les lignes droites et les points de rebroussement du tracé. Ainsi la déformation sera maximale pour des angles proches de $\frac{\pi}{2}$ et nulle pour les angles 0 et π. Les équations (5.10) à (5.12) donnent la position des points $p'_{(t)}$ après déformation :

$$\hat{\theta}_{(t-1)} = (p_{(t-2)}, \widehat{p_{(t-1)}, p_{(t)}}) \in]-\pi, \pi], \tag{5.10}$$

$$p'_{(t)} = p^* \text{ tel que } (p'_{(t-2)}, \widehat{p'_{(t-1)}, p^*}) = \hat{\theta}_{(t-1)} - \beta, \tag{5.11}$$

avec :

$$\beta = \alpha_c * 4 * \frac{\left|\hat{\theta}_{(t-1)}\right|}{\pi} * (1 - \frac{\left|\hat{\theta}_{(t-1)}\right|}{\pi}). \tag{5.12}$$

Le degré de cette déformation est donnée par la valeur β, la figure 5.4 présente la courbe de valeur de β pour $\alpha_c = 1$ en fonction de $\hat{\theta}_{(t-1)}$. On peut remarquer que la déformation est bien maximale pour les angles droits et minimale pour les angle nuls et les angles plats.

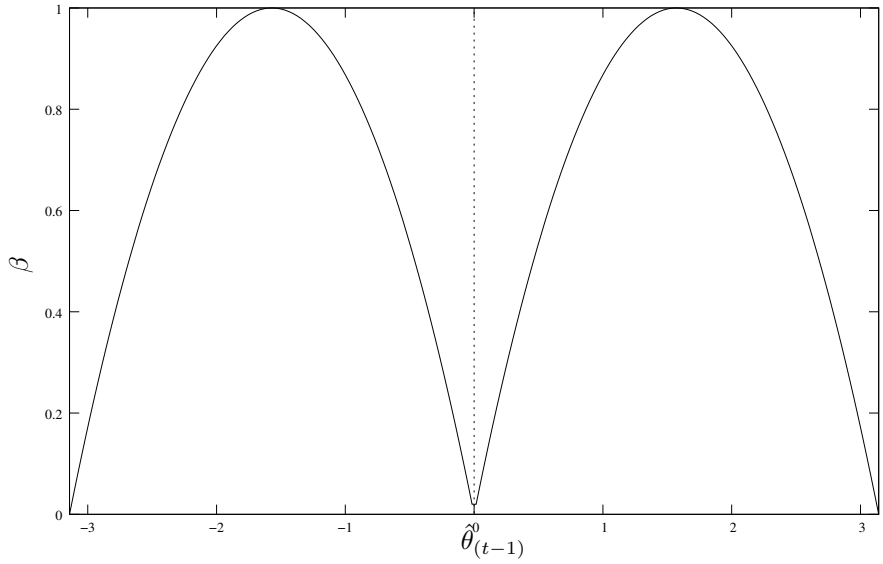
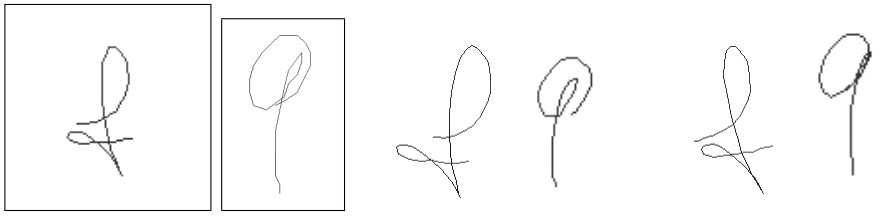


FIG. 5.4 – Valeur de β pour $\alpha_c = 1$ en fonction de $\hat{\theta}_{(t-1)}$ (en radiant).

La figure 5.5 donne deux exemples de caractères déformés par la modification de la courbure. Ainsi on voit que les boucles du `f` ont changé de forme et la boucle du `q` se ferme ou s'ouvre suivant la déformation. Ces déformations n'auraient pas été possibles avec les déformations hors-ligne ni avec la variation de vitesse précédemment présentée. En effet la boucle de la lettre `q` n'ayant pas comme celles du `f` de longues parties horizontales ou verticales, la variation de la vitesse ne l'aurait presque pas déformée.



(a) Originaux, (b) Courbure diminuée, (c) Courbure augmentée.

FIG. 5.5 – Exemples de caractères déformés par modification de la courbure.

5.4 Utilisation de l'analogie

Dans cette section nous expliquons d'une façon générale la notion d'analogie et comment l'utiliser dans la synthèse de caractères en-ligne. Cette partie des travaux a été réalisée en collaboration avec Sabri Bayoudh et Laurent Miclet. Pour plus d'information sur l'analogie nous renvoyons le lecteur à leurs travaux [7, 8, 33].

D'une façon générale, une analogie met en relation quatre objets (ou termes) ainsi : « *le second terme est au premier ce que le quatrième est au troisième* » d'après la définition d'Aristote. L'analogie a été beaucoup utilisée comme modèle de raisonnement en philosophie [51], en Intelligence Artificielle [41] et en linguistique [64].

5.4.1 La proportion analogique

Pour être plus précis, une Proportion Analogique (*Analogical Proportion* (*AP*)) entre quatre objets A, B, C et D, dans le même univers, est exprimée ainsi : « A `est à` B `comme` C `est à` D ». Selon la nature des quatre objets, une AP peut avoir différentes significations. Par exemple : « *jument* `est à` *poulain* `comme` *vache* `est à` *veau* » est une AP dans le domaine sémantique. D'un autre côté si nous considérons seulement la séquence de lettres nous pouvons créer des analogies comme : « recommencer `est à` commencera `comme` rechercher `est à` cherchera ». L'analogie est aussi possible dans des espaces mathématiques, par exemple dans l'espace euclidien $\mathbb{R}^m$, quatre points en AP forment un parallélogramme.

Si un élément x d'une proportion analogique est inconnu, trouver une valeur à x est appelé résoudre une équation analogique. Par exemple « la meringue[1] `est à la` tarte au citron `comme` la chantilly `est à` x », la solution x peut-être *Saint Honoré*[2] dans le domaine sémantique. Dans l'espace euclidien $\mathbb{R}^m$, soit trois points A, B et C, résoudre « $A : B :: C : x$ » consiste à trouver le point D qui forme le parallélogramme $ABCD$.

[1] meringue sur tarte : battre trois blancs d'œufs avec 150 g de sucre glace, jusqu'à ce que le mélange durcisse. Étaler sur la tarte précuite. Four à 200° sans chaleur tournante jusqu'à ce que la meringue brunisse (environ 15 min).

[2] Saint Honoré : gâteau composé de choux fourrés à la crème, armés par du caramel, le tout recouvert de chantilly.

Définition 5.1 *Une AP sur un ensemble* $\mathbf{X}$ *est une relation entre quatre éléments de* $\mathbf{X}$ *(i.e. un sous ensemble de* $\mathbf{X}^4$*). Un élément de cette relation est noté « a : b :: c : d » et se lit « a* `est à` *b* `comme` *c* `est à` *d », ou « a, b, c et d sont en AP ». D'après [64], une AP doit vérifier ces trois propriétés :*

$$\begin{aligned}
&\textit{Symétrie de la relation « comme » :} && (a : b :: c : d) \Leftrightarrow (c : d :: a : b)\\
&\textit{Échange des termes moyens :} && (a : b :: c : d) \Leftrightarrow (a : c :: b : d)\\
&\textit{Déterminisme :} && (a : a :: b : x) \;\; \Rightarrow (x = b)
\end{aligned}$$

Comment utiliser cette définition pour la synthèse de caractères ? Supposons que nous ayons trois exemples du caractère manuscrit « `a` ». Nous pouvons générer de nouveaux caractères « a » en résolvant des équations analogiques comme : « $a_1 : a_2 :: a_3 : x$ » ou « $a_3 : a_2 :: a_1 : x$ ». Si n exemples d'un caractère sont disponibles, le nombre de nouveaux caractères synthétisables est $n^2 \times (n-1)/2$ correspondant au nombre d'analogies différentes possibles compte tenu des relations d'équivalence de la définition 5.1. Nous verrons dans les sections suivantes comment en synthétiser plus en relâchant la définition de la solution de l'équation analogique.

5.4.2 Dissimilarité analogique entre objets

Dans cette section nous donnons une définition précise de la proportion analogique approximative « a `est à` b `à peu près comme` c `est à` d ». Pour cela nous introduisons une quantité qui représente à quel point quatre objets sont en AP. Cette mesure est appelée *Dissimilarité Analogique* (*AD*) [7]. Pour étendre correctement une *AP*, une *AD* doit vérifier les propriétés suivantes :

1. $\forall u, v, w, x, AD(u, v, w, x) = 0 \Leftrightarrow u : v :: w : x$
2. $\forall u, v, w, x, AD(u, v, w, x) = AD(w, x, u, v) = AD(v, u, x, w)$
3. $\forall u, v, w, x, z, t, AD(u, v, z, t) \leq AD(u, v, w, x) + AD(w, x, z, t)$
4. En général, $\forall u, v, w, x, AD(u, v, w, x) \neq AD(v, u, w, x)$

Par exemple dans $\mathbb{R}^m$ nous pouvons définir $AD(u, v, w, x) = \|(u - v) - (w - x)\|$. Il est facile de prouver que les quatre propriétés précédentes sont vérifiées. Lorsque les objets en analogie ne sont pas dans $\mathbb{R}^m$, par exemple avec des symboles une solution consiste à associer à chaque symbole un point dans $\mathbb{R}^m$.

Résoudre une équation analogique approximée consiste à trouver l'objet x qui minimise la *AD* avec u, v et w. Il est aussi possible de chercher les k meilleurs objets.

5.4.3 L'analogie entre séquences d'objets

Soit Σ l'alphabet des objets de l'espace, pour l'exemple on prendra $\Sigma = \{1, 2, 3, ..., 9\}$. Les lettres de Σ peuvent avoir une sémantique, dans notre exemple Σ est ordonné et cyclique (1 est avant 2 et après 9). L'analogie se fera entre mots composés de lettres de Σ. Nous introduisons une nouvelle lettre à cette alphabet $\backsim$ (la lettre vide), qui

peut être introduite n'importe où dans un mot sans en changer le sens sémantique. Par exemple « 3 $\smile$ 5 $\smile\smile$ 4 » signifie la même chose que « 354 ». Grâce à cet alphabet augmenté, nous pouvons définir un alignement entre quatre séquences de lettres de tailles différentes qui permettra de définir la dissimilarité analogique de ces séquences.

5.4.3.1 Alignement

Un alignement de quatre séquences de différentes longueurs est réalisé en insérant des lettres $\smile$ de manière à ce que les quatre mots aient la même taille finale. Une fois cet alignement effectué, il faut considérer la dissimilarité dans chaque colonne de l'alignement dans l'alphabet augmenté.

Le principe de la synthèse d'une séquence est d'aligner trois séquences et de résoudre colonne après colonne chaque équation analogique. Le coût d'un alignement est défini comme la somme des AD de chaque colonne et l'alignement optimal comme celui de coût minimal. La *Dissimilarité Analogique entre quatre Séquences* (ADS) est le coût de l'alignement optimal. Cette dissimilarité analogique a toutes les propriétés d'une AD données plus haut exceptée la troisième (l'inégalité triangulaire).

La difficulté est de trouver parmi tous les alignements possibles celui de coût minimal. Pour cela nous pouvons utiliser soit la programmation dynamique (DP) dans le cube représentant les trois séquences, soit l'algorithme A^* qui permet de trouver les k meilleurs alignements. Nous ne présentons pas ces algorithmes dans cette étude mais renvoyons le lecteur intéressé aux références [BMMA07] et [8].

5.4.4 Codage des caractères

Nous présentons dans cette section comment les caractères manuscrits sont codés et décodés pour permettre l'utilisation de l'analogie de façon efficace. Il s'agit en fait de définir l'alphabet Σ dans lequel coder la séquence de points du tracé en ligne, un caractère manuscrit étant un mot de cet alphabet.

5.4.4.1 Codage du tracé

Nous avons vu dans la section 1.1.1.1 que le signal en-ligne est défini par une fonction $p(t)$ décrivant la position $[x(t), y(t)]^T$ du point ainsi que la pression du crayon sur la surface sensible. Nous ne considérons ici que les points avec une pression non nulle, seul le dernier point d'une trace a une pression nulle correspondant à la position du lever de crayon. Donc un caractère est composé d'une ou plusieurs traces ordonnées (pour les caractères effectués en plusieurs traits), chacune commençant par un poser de crayon et finissant par un lever de crayon.

L'utilisation de l'AP nécessite une représentation plus abstraite du tracé. C'est pourquoi le signal est d'abord ré-échantillonné puis transformé en chaîne de Freeman. Le ré-échantillonnage produit un nouveau signal dans lequel la distance euclidienne entre deux points consécutifs du tracé est constante. Lorsqu'il y a plusieurs traces dans le caractère, les points entre le lever de crayon et le poser de crayon suivant sont interpolés en ligne droite (la position des lever et poser est quand même mémorisée). Un

tracé peut maintenant être vu comme une séquence unique de vecteurs de même longueur. L'encodage de Freeman transforme cette séquence de vecteur en une séquence de symboles représentant les directions de façon discrète. L'alphabet du code de Freeman est un ensemble de 4, 8, 16 ou plus de directions équitablement réparties dans les 360°. Nous avons ajouté la direction 0 qui correspond à un point isolé pour lequel un vecteur direction n'est pas calculable (par exemple le point de certain 'i' peut donc modélisé par un 0). Dans cette étude nous avons utilisé 16 directions de Freeman, donc $\Sigma = \{0, 1, .., 16\}$. La génération par AP produit donc une chaîne de Freeman. La séquence générée est ensuite décodée en un signal en-ligne en suivant les directions du code de Freeman en utilisant la même distance inter-point que celle utilisée pour le codage.

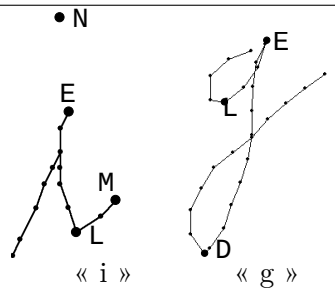
5.4.4.2 Ajout des points d'ancrage

Utiliser directement le codage de Freeman pour la génération AP ne permet pas un alignement des séquences suffisamment correct et mène parfois à un alignement sans aucun sens. En effet l'alignement doit non seulement générer une séquence correspondant à un caractère mais aussi conserver le style du scripteur. Pour guider l'alignement, nous avons utilisé des connaissances sur l'écriture manuscrite pour ajouter des *points d'ancrage* dans le codage. L'alphabet est donc enrichi avec un ensemble de 8 lettres majuscules $\{C, D, E, H, K, L, M, N\}$ chacune correspondant à une caractéristique particulière du tracé manuscrit. En définissant de façon adéquate les AD il est possible de forcer l'étape d'alignement à aligner d'abord ces points particuliers puis a réaliser l'alignement entre ces points sur le code de Freeman. Ainsi la génération AP est de meilleure qualité.

Ces points d'ancrage correspondent à des points d'*ancrage visuel* sur l'écriture manuscrite latine comme définis dans [3] : lever/poser de crayon internes (pour les caractères multi-traces), extrema en y, points anguleux (points avec une forte variation angulaire) et extrema en y dans une boucle. Ces points particuliers permettent donc de qualifier le tracé en fonction du style du scripteur (par exemple présence d'une boucle au lieu d'un simple de point anguleux...). La table 5.1 décrit ces huit points d'ancrage. Ils sont organisés de façon hiérarchique car un extremum en y peut aussi être un point angulaire, un extremum en y d'une boucle ou un lever de crayon. La priorité de l'étiquetage des points d'ancrage est la suivante (en commençant par la plus basse) : extremum en y, point anguleux, extremum en y dans une boucle et lever/poser de crayon.

5.4.5 Exemple de génération par Analogie

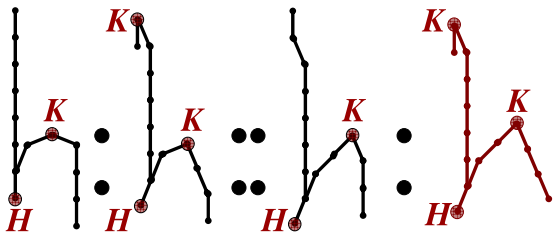
La figure 5.6 donne un exemple de résolution d'une AP dans le domaine du code de Freeman enrichi des points d'ancrage. Nous avons utilisé trois exemples manuscrits du caractère « `h` » : h_1, h_2 et h_3. Nous avons généré grâce à l'algorithme DP une séquence x_0 représentant un nouveau caractère « `h` » avec la plus faible dissimilarité analogique. Cette AD est non nulle dans cet exemple puisque pour plusieurs colonnes (celles encadrées) l'analogie n'est pas parfaite.

Symbole	Description	Exemples
C / D	extremum en y de boucle haut / bas	
E / H	point anguleux haut / bas	
K / L	extremum en y haut / bas	
M / N	lever/poser de crayon	

TAB. 5.1 – Description des points d'ancrage.

$h_1=$	9	∽	9	∽	9	9	9	9	9	∽	H	1	2	∽	4	K	6	9	9	9
$h_2=$	1	K	∽	8	9	9	9	9	9	10	H	∽	2	2	4	K	∽	8	8	9
$h_3=$	∽	∽	9	8	9	9	9	9	9	10	H	2	2	3	3	K	8	9	9	∽
$x=$	1	K	∽	8	9	9	9	9	9	10	H	2	2	3	3	K	8	8	8	∽

(a)



$h_1 \;:\; h_2 \;::\; h_3 \;:\; x$

(b)

FIG. 5.6 – (*a*) Résolution par *AP* sur le code Freeman enrichi des points d'ancrage, (*b*) représentation graphique des caractères correspondants.

Nous pouvons constater que le caractère généré est bien différent des trois premiers tout en conservant le style du scripteur. De plus, ce type de caractère n'aurait pas pu être synthétisé par les déformations présentées précédemment.

5.5 Trois stratégies de synthèse de caractères

Nous avons vu que les trois processus que nous proposons pour la synthèse de caractères sont complémentaires, c'est-à-dire ne génèrent pas des déformations identiques. Comme notre but est d'obtenir le plus possible de variabilité autour du style du scripteur nous proposons ici trois stratégies pour combiner ces trois processus. Ces trois stratégies permettent de plus en plus de variabilité en respectant le style du scripteur.

La figure 5.7 présente d'abord le processus global de génération utilisant les déformations du tracé en-ligne et la génération par AP utilisant le codage de Freeman enrichi.

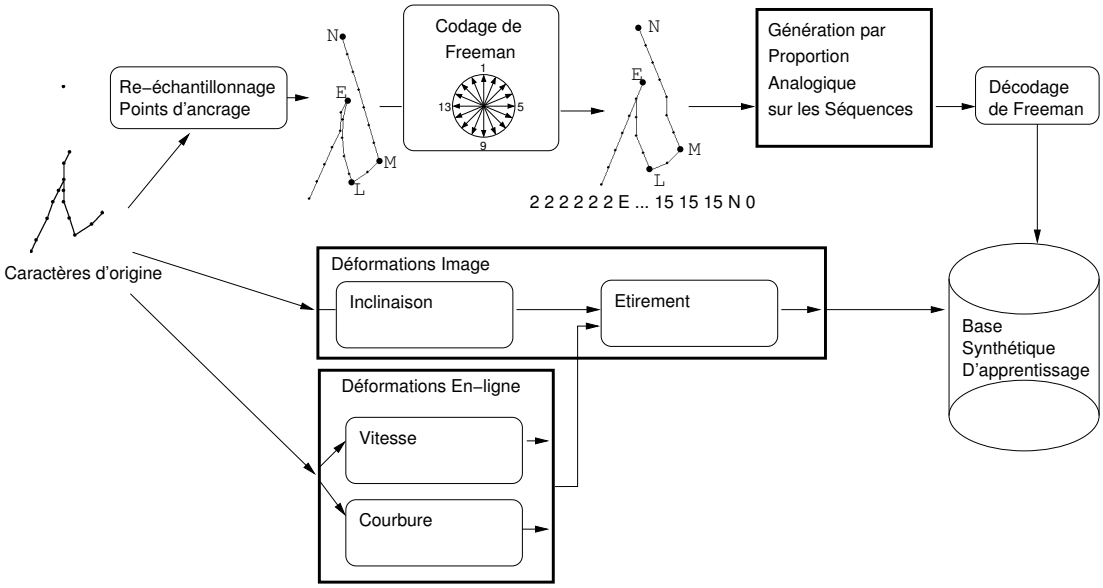


FIG. 5.7 – Vue d'ensemble du processus de synthèse d'une base caractères artificiels d'apprentissage.

À partir de ce schéma général, nous comparons trois stratégies pour générer une base synthétique d'apprentissage. Ces trois stratégies fonctionnent de la même façon : elles prennent en entrée une base de n caractères d'origine et donnent en sortie une base de $N - n$ caractères synthétisés. Ces deux bases constituent la base d'apprentissage de N caractères. Ces trois stratégies diffèrent par le choix des modes de génération des caractères, *i.e.* par le chemin pris dans le schéma de la figure 5.7. Pour chaque caractère, les paramètres des déformations (α_x, α_y, α_i, α_v et α_c) sont chacun tirés aléatoirement dans un intervalle de valeurs possibles. Cet intervalle fixé grâce à nos connaissances sur

l'écriture manuscrite, permet de ne pas générer des caractères trop déformés et donc de conserver le style du scripteur. Pour l'analogie, toutes les analogies possibles avec les n caractères originaux sont effectuées, chacune générant un nombre égal de caractères.

La stratégie *« Déformations Image »* applique consécutivement les deux déformations du type image au caractère : étirement et inclinaison. La stratégie *« Déformations Image et En-ligne »* choisit au hasard l'inclinaison, la modification de la vitesse ou la modification de la courbure puis applique l'étirement (ainsi les déformations en-ligne s'effectuent sur les caractères non déformés par l'étirement). Enfin la stratégie *« Analogie et Déformations »* génère deux tiers de la base avec la stratégie précédente et le tiers restant avec la génération AP. Cette proportion a été choisie expérimentalement comme étant le meilleur compromis.

5.6 Expérimentations

Dans cette section nous commençons par présenter le protocole de test et des exemples de caractères générés. Le but de ces expérimentations est de trouver une stratégie de génération de caractères manuscrits pour améliorer les performances de l'adaptation décrite au chapitre suivant. Nous nous plaçons donc dans un contexte d'apprentissage et de test mono-scripteur. Nous montrons donc que nos stratégies de génération permettent d'améliorer le taux de reconnaissance mono-scripteur de trois classifieurs communément utilisés en reconnaissance de formes lorsqu'ils sont appris avec peu de données.

5.6.1 Protocole expérimental

Douze scripteurs différents ont écrits 40 fois les 26 caractères minuscules (1040 caractères) sur un PDA. La base de chaque scripteur est coupée aléatoirement en quatre parties de 10 fois chaque caractère. Dans le but d'améliorer la robustesse de nos tests, nous utilisons ce découpage pour une validation croisée stratifiée en 4 parties : un quart pour la base notée $D10$ (260 caractères) et trois quarts pour la base $D30$ (780 caractères). Ainsi l'expérimentation est faite quatre fois en inter-changeant le rôle de chaque partie.

L'expérimentation est composée de deux phases dans lesquelles trois classifieurs mono-scripteurs sont appris : un RBFN, un K-PPV et un SVM multi-classes. Ces classifieurs, présentés dans la section 1.2, ont un comportement différent vis à vis des données d'apprentissage : modélisation intrinsèque pour le RBFN, utilisation directe de la base d'apprentissage pour le k-PPV et modélisation discriminante pour le SVM. Il est donc intéressant de voir l'effet de la synthèse de caractères sur l'apprentissage de ces trois classifieurs. L'apprentissage des classifieurs et les tests se font dans l'espace en 21 dimensions des caractéristiques déjà présentées en section 1.1.1.2.

Dans la première phase, nous calculons deux *taux de Reconnaissance de Référence* sans génération de données artificielles : $RR10$ et $RR30$. Pour $RR10$, les classifieurs mono-scripteur sont appris pour chaque scripteur sur sa base $D10$ et testés sur la base $D30$ complémentaire, pour les quatre validations. De la même manière, pour $RR30$ les

classifieurs sont appris sur $D30$ et testés sur $D10$. Dans un cas d'utilisation réelle, il n'est pas raisonnable de demander à l'utilisateur de saisir plus de 10 fois chaque caractère, c'est pourquoi nous avons choisi cette taille de base. L'apprentissage sur $D30$ permet de mesurer le taux obtenu dans le cas idéal où plus de données seraient demandées à l'utilisateur. La table 5.2 présente les résultats obtenus pour ces deux références pour chaque type de classifieur.

Tab. 5.2 – Taux de reconnaissance de référence sans génération de données d'apprentissage (en %).

	RBFN	K-PPV	SVM
$RR10$	82.3	92.2	92.9
$RR30$	94.5	95.4	96.2

On constate que l'apprentissage est plus efficace avec 30 données par classe qu'avec 10. Le manque de données peut même être handicapant par exemple pour le RBFN puisque son $RR10$ est bien plus faible que celui des deux autres classifieurs. Ces taux de références donnent un aperçu de ce qu'il est possible de gagner comme taux de reconnaissance en augmentant la quantité de données d'apprentissage.

Dans la seconde phase, les stratégies de génération de caractères manuscrits sont testées. Si l'utilisation d'une base d'apprentissage synthétique est plus performante que l'utilisation des seuls caractères d'origine, alors c'est que les stratégies de synthèse respectent le style du scripteur. Pour chaque scripteur, un à dix caractères par classe sont choisis aléatoirement dans sa base $D10$. À partir de ces caractères, une base d'apprentissage synthétique de 300 caractères par classe est générée en utilisant une des stratégies présentées précédemment. Un classifieur mono-scripteur est alors appris sur cette base et testé sur la base $D30$ correspondante du scripteur. Ce test est répété trois fois pour chaque validation croisée (12 tests par scripteur) dans le but de varier les caractères originaux choisis. La moyenne et l'écart-type de ces 12 taux de reconnaissance sont calculés pour chaque scripteur. Enfin les moyennes de ces deux mesures sont calculées pour obtenir le taux de reconnaissance mono-scripteur moyen et l'écart-type mono-scripteur moyen.

Nous donnons d'abord des exemples de caractères synthétisés par ces trois stratégies puis nous présentons les résultats obtenus par ce test.

5.6.2 Exemples de caractères manuscrits générés

La figure 5.8 montre des exemples de caractères manuscrits générés par chaque déformation et par proportion analogique.

Nous pouvons constater que chaque génération permet des déformations des caractères d'origine qui ne sont pas atteignables par les autres stratégies. Les déformations du type image permettent d'étirer et d'incliner le caractère. Les déformations en-ligne permettent de déplacer des parties du caractère en modifiant la vitesse du tracé et de

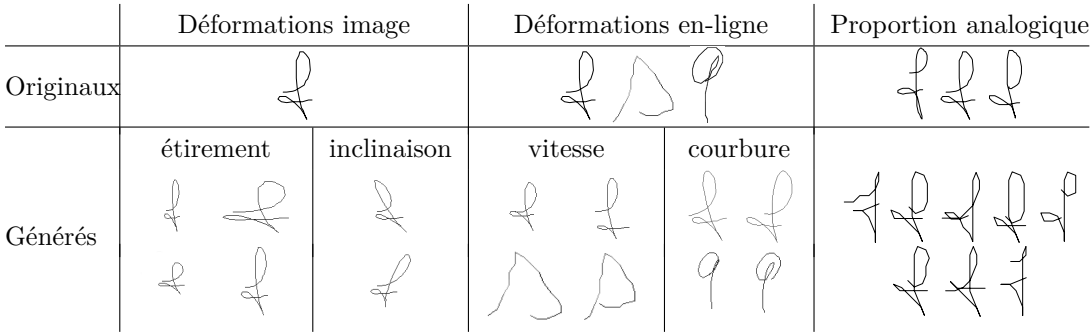


FIG. 5.8 – Exemples de caractères manuscrits générés.

fermer ou d'ouvrir les boucles du caractère en modifiant la courbure du tracé, ceci en conservant la structure globale du caractère. L'analogie permet de générer des caractères par rapport aux transformations trouvées entre les trois caractères originaux. Ces stratégies permettent bien de créer des caractères différents des originaux en conservant le style du scripteur.

5.6.3 Résultats

La figure 5.9 compare les taux de reconnaissance atteints par les trois stratégies de génération pour les trois classifieurs en mettant en valeur les taux de référence $RR10$ et $RR30$.

Premièrement, nous pouvons constater que le comportement pour les trois classifieurs est le même. Donc les conclusions suivantes ne dépendent pas du type de classifieur utilisé.

Ensuite, ces résultats montrent que les trois stratégies de générations sont complémentaires car utiliser la stratégie *« Déformations Image et En-ligne »* est plus performante que d'utiliser la stratégie *« Déformations Image »* seule et la stratégie *« Analogie et Déformation »* est plus intéressante que d'utiliser les déformations seules.

Enfin, en utilisant la stratégie de génération *« Analogie et Déformation »* qui utilise toutes les méthodes de génération et seulement quatre caractères originaux, le taux de reconnaissance obtenu dépasse la référence $RR10$. De plus la référence $RR30$ est atteinte avec 9 ou 10 caractères suivant le classifieur.

Nous pouvons donc conclure que notre stratégie de génération permet d'apprendre un classifieur avec très peu de données aussi efficacement qu'avec une longue étape de saisie pour l'utilisateur : nous avons besoin à peu près de trois fois moins de données pour atteindre le même taux de reconnaissance.

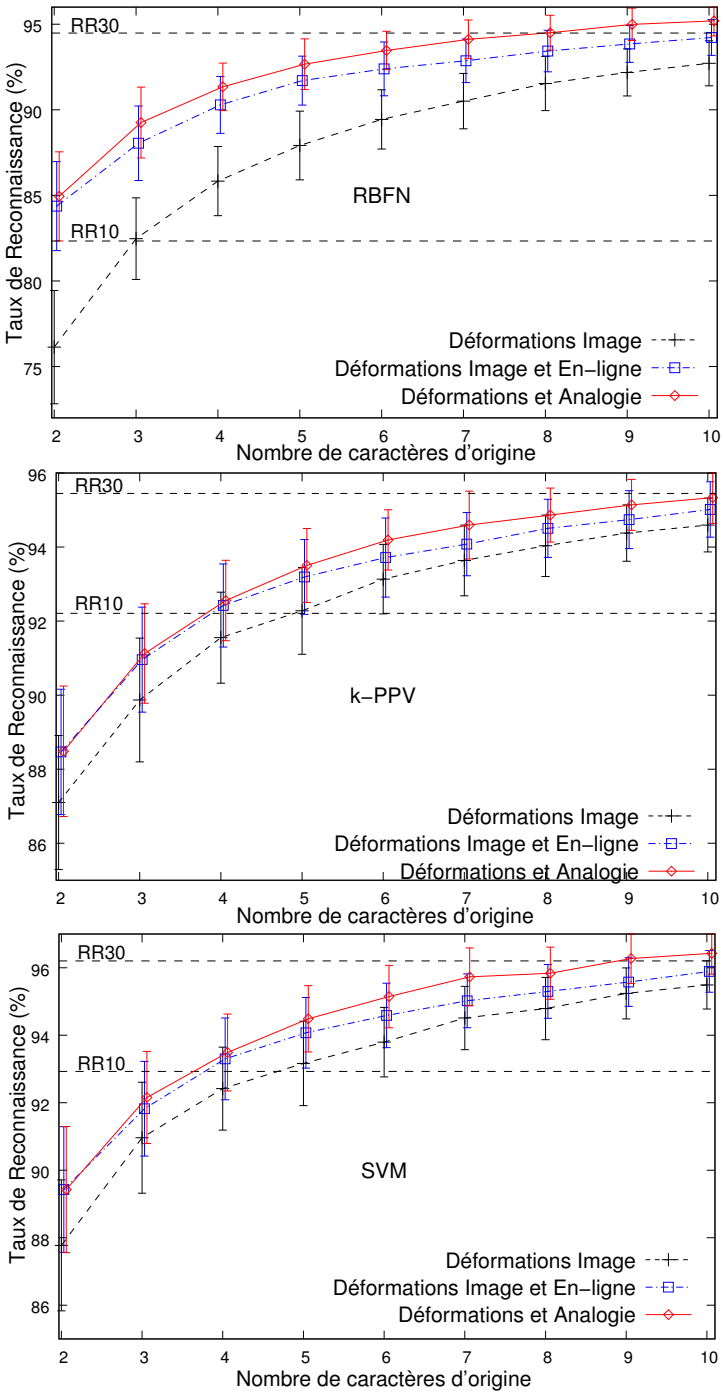


FIG. 5.9 – Taux de reconnaissance moyen (et écart type) mono-scripteur en fonction du nombre de caractères utilisés comparés aux références RR10 et RR30, pour trois types de classifieurs.

5.7 Conclusion et discussions

Dans ce chapitre, nous avons présenté comment synthétiser des exemples de caractères artificiels pour l'apprentissage rapide d'un classifieur de caractères manuscrits d'un nouvel utilisateur. Nous avons présenté trois méthodes complémentaires pour générer des caractères à partir d'un ou plusieurs exemples originaux. La première considère le caractère comme une image, la seconde utilise des connaissances spécifiques à l'écriture manuscrite et la troisième utilise le concept d'analogie. Chacune d'elles génère des déformations originales par rapport aux autres en respectant le style de l'écriture de l'utilisateur. Ces trois méthodes sont combinées dans trois stratégies cumulant ainsi leurs avantages respectifs.

Nous avons montré qu'utiliser la combinaison des trois méthodes permet de diminuer nettement la quantité de données nécessaire pour obtenir un apprentissage de qualité. Nous pouvons aussi affirmer que les données générées par nos stratégies sont fidèles au style d'écriture des données d'origines puisqu'elles permettent d'améliorer les reconnaisseurs mono-scripteurs.

Dans de futures recherches sur la génération de caractères manuscrits, il sera très intéressant d'utiliser une modélisation de plus haut niveau du tracé en ligne et du caractère comme l'approche *delta log-normale* [89] pour proposer de nouvelles déformations et une nouvelle modélisation pour l'analogie.

Si nous replaçons cette étude dans le contexte de la thèse, la synthèse de caractères pourra être une approche efficace à utiliser dans le processus d'adaptation. En effet, si dès les premiers caractères saisis par l'utilisateur, il est possible d'en générer plusieurs dizaines fidèles à son style d'écriture, l'adaptation pourra être beaucoup plus rapide. Dans notre contexte applicatif, nous avons aussi une contrainte sur les ressources de calcul disponibles. Il faut donc veiller à ce que nos techniques de synthèse ne soient pas trop coûteuses de ce point de vue. Les méthodes de déformations d'image et de déformations en-ligne sont assez peu coûteuses alors que l'analogie utilise des algorithmes de recherche très complexes. Nous utiliserons dans le cadre de notre étude seulement la stratégie *« Déformations Image et En-ligne »* qui, d'après nos résultats présentés précédemment, est un très bon compromis entre la qualité des caractères générés et le coût de cette génération.

Chapitre 6

Adaptation à la volée au scripteur

Nous avons vu dans le chapitre précédent qu'il est difficile d'apprendre un classifieur mono-scripteur avec très peu de données, même avec une extension de la base d'apprentissage en utilisant la synthèse de caractères, les taux de reconnaissance atteints ne sont pas comparables avec ceux des systèmes omni-scripteur appris avec une grande quantité de données. Nous proposons donc ici la possibilité d'adapter un système omni-scripteur à la reconnaissance de l'écriture du scripteur courant.

L'apprentissage et l'adaptation sont deux processus d'acquisition des connaissances différents. Lors d'un apprentissage le classifieur acquiert toutes ses connaissances d'un seul coup à partir d'une seule base d'apprentissage. Lors d'un processus d'adaptation, le classifieur part de connaissances générales initialement apprises sur une base d'apprentissage, puis ces connaissances sont remises en cause par l'arrivée de nouvelles données spécifiques à l'utilisation courante du système.

Il y a plusieurs façons d'adapter un classifieur et elles dépendent principalement de deux aspects du problème : le type de classifieur utilisé et la quantité de données disponibles pour l'adaptation. Une première solution, l'*enrôlement*, est d'utiliser une base d'adaptation constituée par l'utilisateur pour effectuer une adaptation hors-ligne c'est à dire en une seule fois sans remise en cause ultérieure. Cela peut se réaliser en ré-apprenant une partie du classifieur ou en terminant son apprentissage comme dans [17, 27] avec des MMC (Modèle de Markov Caché). Cette approche nécessite que l'apprentissage du classifieur puisse être cumulatif comme c'est le cas avec les MMC. Le problème de cette approche est qu'elle nécessite un apprentissage coûteux et qui n'est pas ensuite remis en cause, ce qui ne correspond pas à nos contraintes d'applicatif.

Dans cette étude nous nous intéressons à une seconde approche du problème, une *adaptation incrémentale* ou *adaptation en-ligne* qui peut être réalisée avec peu de ressources au fur et à mesure que l'utilisateur entre de nouveaux caractères. Ce type d'adaptation permet de modifier en continu le système de reconnaissance par de petits ajustements. Il devient de plus en plus performant en acquérant de plus en plus d'expérience. Les travaux [83, 112] proposent ce type d'approche dans un cadre proche du

nôtre puisqu'ils utilisent des systèmes à base de prototypes (des K-ppv) et une adaptation incrémentale effectuée à chaque nouveau caractère. Nous détaillerons donc ces travaux dans l'état de l'art.

Dans notre approche, nous supposons que les caractères disponibles pour l'adaptation sont bien étiquetés. La classe de ces caractères peut être obtenue soit par validation implicite de l'utilisateur (comme dans notre interface IREMA présentée dans la section 7.3) soit en corrigeant automatiquement la reconnaissance par l'utilisation d'un dictionnaire [83].

L'objectif omni-scripteur des systèmes de reconnaissance poussait jusqu'alors à utiliser des systèmes complexes capables d'absorber toutes les difficultés du problème. La reconnaissance mono-scripteur étant un problème de classification plus simple, on peut donc se permettre d'utiliser des systèmes de reconnaissance plus simples. Les Systèmes d'Inférence Floue (SIF, présentés dans la section 1.2.5) sont des systèmes simples, efficaces et interprétables. Ces atouts nous permettent de présenter ici un système d'adaptation en-ligne nommé ADAPT dédié au SIF à base de prototypes. Cette approche est inspirée des LVQ (*Learning Vector Quantization*) [58] et des EFCL (Elliptical Fuzzy Competitive Learning) [31] qui sont des techniques d'apprentissage incrémentales. ADAPT est conçue pour respecter les contraintes liées à notre applicatif *i.e.* d'un côté une adaptation en-ligne progressive et stable effectuée tout au long de l'utilisation, de l'autre l'utilisation de peu de ressources comme sur les petits périphériques mobiles. Pour cela, ADAPT est basée sur l'interprétation des connaissances contenues dans les SIF.

La suite de ce chapitre est organisée de la façon suivante. La section 6.1 présente un état de l'art des différentes techniques d'adaptation existantes dans différents contextes avec différents classifieurs (MMC, réseaux de neurones, classifieurs à base de prototypes). Puis nous présentons notre approche ADAPT pour l'adaptation en ligne d'un SIF dans la section 6.2 et ses différents aspects, en montrant ses originalités par rapport aux techniques qui nous ont inspirées. La section 6.3 montre ensuite comment utiliser ADAPT de façon optimale en fonction des ressources disponibles. Ensuite le chapitre 7 présentera nos résultats expérimentaux simulant un contexte réel d'utilisation et la mise en place d'une plate-forme de test en conditions réelles d'utilisation baptisée IREMA.

6.1 État de l'art

Nous pouvons distinguer deux modes d'adaptation déjà évoqués en introduction : l'enrôlement (ou adaptation hors-ligne) et l'adaptation incrémentale (ou adaptation en-ligne). Même si notre objectif est clairement une stratégie d'adaptation en-ligne pour un SIF, il est intéressant d'étudier les stratégies d'adaptation hors-ligne ainsi que celles en-ligne utilisant d'autres classifieurs afin de savoir si nous pourrions transposer ces méthodes existantes à notre contexte.

6.1.1 Adaptation hors-ligne

Les systèmes que nous présentons ici ont en commun d'utiliser une base d'exemples de caractères du scripteur pour s'adapter en une seule fois. Ces exemples du scripteur peuvent soit être saisis avant utilisation du système, soit enregistrés pendant le début de l'utilisation.

Citons d'abord les travaux utilisant des MMC [17, 27] qui s'inspirent de ce qui est fait en reconnaissance de la parole sur l'adaptation au locuteur. Dans ces études, les MMC sont d'abord entraînés sur une base omni-scripteur puis ré-entraînés sur la base du scripteur. Plus précisément, dans [17] un MMC est appris pour chaque caractère, puis ces modèles sont utilisés pour de la reconnaissance de mots. L'adaptation se fait en continuant l'apprentissage de tous les modèles en comparant trois méthodes d'apprentissage (maximum Likelihood (ML), Maximum a Posteriori (MAP) et Maximum Likelihood Linear Regression (MLLR)) et en ne modifiant que un ou plusieurs types de paramètres (les moyennes, les variances, les poids ou les transitions). Il faut noter que les meilleurs résultats sont obtenus en ne modifiant que les moyennes et les poids. Dans [27] les MMC sont appris sur les graphèmes (déterminés par classification non supervisée) puis sont étiquetés automatiquement. Lors de l'adaptation, seul les graphèmes utilisés par l'utilisateur suffisamment souvent sont ré-entraînés. La base omni-scripteur est censée représenter tous les graphèmes possibles, il n'y a donc pas d'ajout de nouveaux graphèmes.

Dans [98] le système de reconnaissance de mots en-ligne commence par modéliser l'ensemble des primitives du tracé (*strokes*) omni-scripteur par une carte de Kohonen (classification non-supervisée). Les cellules de cette carte sont ensuite étiquetées avec les données d'apprentissage et une matrice de transitions entre primitives est apprise pour la reconnaissance des lettres. Cette étape propose alors à la reconnaissance de mot un ensemble de segmentations possibles. L'adaptation se fait en utilisant un ensemble de mots du scripteur. Les mots déjà partiellement reconnus (présents dans la liste TOP-20) sont utilisés pour ré-évaluer la matrice de transitions entre primitives. Les autres permettent de ré-étiqueter (manuellement) certaines primitives du scripteur. Enfin la matrice de transition est alors ré-évaluée en utilisant toutes les données du scripteur. Cette approche permet d'adapter la segmentation en primitives et leur sémantique au style du scripteur. Ainsi le nombre de possibilité de lettres est réduit et la phase de reconnaissance de mots est plus rapide et plus fiable.

Les travaux [79, 80] se placent dans le contexte de la reconnaissance de texte hors-ligne, ils utilisent le concept d'adaptation en considérant que l'écriture manuscrite au sein d'une page est stable. Le principe de leur système repose sur le schéma des algorithmes EM (*Espectation Maximisation*) : le texte est d'abord reconnu avec les connaissances omni-scripteur (avec éventuellement un rejet de certain mots), puis le système s'adapte en utilisant ces nouvelles données étiquetées et ainsi de suite. La phase d'adaptation consiste à prendre en compte les graphèmes utilisés par le scripteur mais aussi à modifier les étiquettes portées par ces graphèmes en fonction de la précédente phase de reconnaissance.

Bien que le principe de l'adaptation soit aussi utilisé dans le domaine de la recon-

naissance de la parole, nous avons limité notre état de l'art au domaine de la reconnaissance de l'écriture qui est déjà assez riche. Citons tout de même l'approche UBM (*Universal Background Model*) [96] utilisée en reconnaissance du locuteur car elle est comparable à la nôtre. La modélisation des connaissances omni-locuteur, l'UBM, est réalisé par un modèle de mélange de Gaussiennes (*Gaussian Mixture Model*, GMM). Ensuite l'UBM est spécialisé pour le locuteur en utilisant l'algorithme EM, ce qui permet de conserver une partie de la connaissance initiale pour ne pas avoir de problème de sur-apprentissage. La reconnaissance se fait en comparant l'activation du modèle universel avec le modèle mono-locuteur. L'analogie avec notre approche se limite au type de modélisation des connaissances (le GMM/les prototypes flous des SIF) et au fait de déplacer et déformer ces prototypes (EM/ADAPT), car avec l'UBM le modèle du locuteur n'est plus remis en cause.

6.1.2 Adaptation en-ligne

Lors d'une adaptation en-ligne, les connaissances sur le style du scripteur sont prises en compte au fur et à mesure de l'arrivée de nouveaux exemples. Il y a deux solutions pour modifier le comportement du système : soit une étape de post-traitement des résultats de classification est ajoutée sans modifier le classifieur, soit le classifieur est modifié pour prendre en compte les nouvelles informations. Nous commençons par présenter les techniques ajoutant un post-traitement puis celles modifiant le classifieur car cette dernière stratégie est plus proche de la nôtre.

Dans [1] les auteurs utilisent un système de reconnaissance (un k-PPV avec une distance DTW) donnant comme résultat de classification un couple de classes. Une option de rejet permet de détecter les couples avec un fort risque d'erreur. Le couple est alors transmis à un second classifieur adaptatif. Ce dernier est un DEC (*Dynamically Expanding Context*, technique issue du domaine de reconnaissance de la parole) qui utilise un ensemble de règles qui associent une sortie (le couple de classes en confusion *i.e.* le contexte) avec une correction. Lors de la classification, si aucune règle ne correspond, alors la classification n'est pas changée, sinon le résultat est modifié par la correction (le système global n'a pas de rejet). Une nouvelle règle est ajoutée à chaque erreur détectée. L'inconvénient de l'ajout des règles dans cet article est que leur nombre est limité par le nombre de couples de classes et qu'elles ne sont ensuite jamais remises en question. Ainsi cette adaptation est bien incrémentale mais le système se fige après un certain temps d'utilisation.

Dans [90] le post-traitement pour l'adaptation noté OAM (*Output Adaptation Module*) est basé sur l'utilisation d'un RBFN pour ajuster les sorties du classifieur principal. Le classifieur principal peut être de n'importe quel type avec la contrainte de donner un score pour chaque classe. Ces scores de sortie servent d'entrées à l'OAM. Les sorties du classifieur principal et celle de l'OAM sont additionnées pour obtenir les scores corrigés. À chaque nouvelle erreur détectée un neurone caché est ajouté pour associer la sortie du classifieur avec la correction à effectuer. Si la nouvelle erreur est trop près d'une existante (utilisation du rejet de distance ici), le centre de la fonction à base radiale existante est déplacé vers la nouvelle erreur. L'avantage de ce système

est qu'il reste adaptatif tout au long de l'utilisation même si beaucoup de corrections ont été effectuées, mais il y a une augmentation du coût de la classification à chaque nouvelle correction car il faut à chaque fois activer l'ensemble des neurones cachés de l'OAM.

Les approches suivantes modifient le classifieur principal pour prendre en compte les spécificités du scripteur. L'avantage commun de ces approches est de pouvoir continuer de remettre en question les connaissances apprises pendant l'adaptation.

Dans [73] le classifieur est un TDNN (*Time Delay Neural Network*) dans lequel la couche de sortie est remplacée par un *Optimal Hyperplane* (OH, sorte de SVM simple sans noyau) dans le but d'obtenir une adaptation plus facile. L'apprentissage omni-scripteur se fait en deux étapes : entraîner le TDNN de façon classique puis retirer la dernière couche et entraîner l'OH sur la même base omni-scripteur. L'adaptation se fait à chaque fois qu'une erreur est commise en ajoutant aux supports vecteurs cet exemple en erreur et en ré-apprenant la combinaison linéaire optimale. Le reste du classifieur (le TDNN d'origine) n'est pas modifié. Les inconvénients de cette approche sont d'une part la phase coûteuse d'apprentissage du OH à chaque adaptation, d'autre part le fait que cette approche ne prévoit pas de supprimer des vecteurs supports qui seraient sources d'erreurs.

Les références suivantes [77, 83, 112] adaptent un classifieur à base de prototypes en ajoutant, en inactivant ou en modifiant des prototypes.

Dans [83] un classifieur k-PPV avec une distance DTW (*Dynamic Time Warping*, alignement élastique de séquences) est utilisé. L'adaptation se fait en ajoutant les lettres du scripteur pour lesquelles il y a eu une mauvaise classification. Pour atteindre de meilleures performances il propose de détecter aussi les prototypes qui sont sources d'erreurs pour les inactiver. L'utilisation d'une analyse lexicale permet de définir une stratégie d'adaptation non-supervisée ou plus précisément auto-supervisée : c'est la reconnaissance lexicale qui détecte les erreurs de reconnaissance de lettres.

Dans [77] l'adaptation permet d'ajouter et de supprimer des prototypes mais aussi de modifier le poids des prototypes existants dans la décision. Ils étudient aussi trois stratégies d'adaptation : ajouter tous les caractères, seulement les erreurs ou tous en donnant plus d'importance aux erreurs. Dans ces travaux, les expérimentations sur les caractères japonais (plus de 1500 classes) ont montré qu'il est intéressant d'utiliser tous les caractères mais avec différentes priorités suivant qu'ils sont bien classés ou non.

Dans [111, 112] les auteurs utilisent un classifieur k-PPV avec DTW comme distance et comparent trois stratégies d'adaptation et leur combinaison : ajout de prototypes, inactivation de prototypes et déformation de prototypes. La déformation des prototypes est faite en appliquant le principe de LVQ (*Learning Vector Quantization*) sur les caractères eux-mêmes. Ainsi un prototype qui a servi à bien reconnaître un caractère va se déformer en se rapprochant de ce dernier et en s'éloignant en cas de mauvaise classification.

Dans cet état de l'art nous pouvons constater que le problème de l'adaptation a été traité en utilisant différentes stratégies sur différents types de classifieurs. Notre système

de reconnaissance de base étant un SIF, nous nous inspirerons plus des techniques d'adaptation utilisant des prototypes. La différence entre nos SIF et les classifieurs utilisés dans [83, 112] est que nos prototypes portent plus d'informations grâce à la modélisation floue choisie, il sera donc plus délicat d'en ajouter et d'en supprimer. Par contre cet état de l'art confirme qu'il est intéressant de modifier les prototypes en utilisant tous les caractères à disposition. Nous allons donc dans un premier temps transposer certaines techniques à nos SIF, que nous compléterons ensuite pour obtenir l'approche ADAPT.

6.2 Adaptation en-ligne par la méthode ADAPT

La méthode ADAPT (pour *ADaptation par Ajustement de ProTotypes*) est une approche floue du problème de l'adaptation en-ligne. Cette approche floue permet d'interpréter les connaissances du classifieur et de les modifier. Dans cette étude nous l'avons appliquée à un système de reconnaissance floue simple : les Systèmes d'Inférence Floue de Takagi-Sugeno d'ordre 0 (SIF) présentés dans la section 1.2.5. L'adaptation est faite pendant l'utilisation du SIF, ADAPT correspond donc à un apprentissage itératif c'est à dire qu'il n'utilise que le dernier caractère courant (ou un tampon des derniers exemples).

6.2.1 Principe de l'adaptation d'un SIF

Nous avons vu en introduction qu'un SIF est composé de règles. Chaque règle est composée d'une prémisse et d'une conclusion. La prémisse est un prototype flou représentant le sous-ensemble flou de l'espace d'entrée pour lequel la règle est valide. La conclusion représente le degré d'appartenance du prototype à chaque classe.

D'après cette architecture, il existe plusieurs possibilités pour adapter un SIF que nous détaillerons dans les sections suivantes. Ces différentes étapes de l'adaptation constituent ce nous appelons un *cycle d'adaptation*. Il est d'abord possible de modifier les règles existantes en adaptant les prémisses ou en adaptant les conclusions. Les prémisses étant des prototypes nous proposons deux modifications possibles : les déplacer (section 6.2.2.1) ou les déformer (section 6.2.2.2) pour mieux représenter la répartition des données dans l'espace des caractéristiques de façon cohérente avec les conclusions. Les conclusions peuvent être modifiées pour ré-estimer la participation du prototype à chaque classe par la technique classique de descente de gradient (section 6.2.3). Il est aussi possible d'ajouter ou de supprimer des règles dans le SIF (section 6.2.4) si les nouveaux caractères ne sont pas représentés par les prototypes existants, ou si certaines règles ne sont pas utilisées, ou si elles sont sources d'erreurs.

6.2.2 Adaptation des prémisses des règles

Les prototypes flous des prémisses peuvent être soit déplacés dans l'espace des caractéristiques, soit leur zone d'influence peut être déformées.

6.2.2.1 Déplacement des prototypes

Quelle que soit l'approche utilisée, le déplacement d'un prototype P_r consiste à modifier la position de son centre μ_r par le vecteur $\Delta\mu_r$:

$$\vec{\mu_r}(t+1) = \vec{\mu_r}(t) + \Delta\vec{\mu_r} \ . \tag{6.1}$$

Dans toutes les approches, $\Delta\vec{\mu_r}$ dépend d'un paramètre d'adaptation λ compris entre 0 et 1 qui règle la vitesse d'adaptation (sa valeur est discutée dans la section 6.3).

Une première approche est d'utiliser les techniques d'apprentissage itératif de classifieurs à base de prototypes comme les k-PPV, ces techniques appartiennent à la famille des apprentissages compétitifs (*Competitive Learning*, CL) utilisés par exemple dans les algorithmes de type C-Moyennes. Comme nous nous intéressons aux apprentissages supervisés, nous pouvons considérer l'algorithme LVQ (*Learning Vector Quantization*) utilisé pour l'apprentissage des cartes de Kohonen [58]. Différentes versions de LVQ sont comparées pour l'adaptation de classifieurs k-PPV dans [112]. La version la plus simple, LVQ1, consiste à déplacer le prototype le plus proche vers le nouvel exemple s'ils sont de la même classe ou dans l'autre sens sinon. Cette approche du type *tout-pour-le-gagnant* est facilement transposable à nos SIF comme nous le verrons dans la section suivante. Une version floue FLVQ [24, 31] (*Fuzzy Learning Vector Quantization*) utilisée pour l'apprentissage des C-Moyennes Floues modifie tous les prototypes en fonction de leur activation.

Pour déplacer les prototypes flous du SIF une autre possibilité est d'utiliser les techniques itératives d'apprentissage des ensembles flous telles que l'algorithme EM. Dans [76] le carré de l'erreur est minimisé par une descente de gradient pour l'apprentissage de SIF dans un contexte d'approximation de fonctions. L'architecture des SIF utilisés dans [76] étant différente, on ne peut donc pas utiliser directement leur formule de déplacement. Cette technique est aussi utilisée pour apprendre les centres des fonctions à base radiale des RBFN [84].

Pour le déplacement des prototypes de nos SIF, une première étape de nos travaux a consisté à transposer ces approches existantes à notre classifieur. La section suivante présente donc comment nous appliquons les algorithmes LVQ, FLVQ et de descente de gradient aux SIF. Nous présentons aussi les limites de ces approches qui nous ont poussé à proposer notre méthode ADAPT qui prend en compte toutes les spécificités des SIF pour le déplacement des prototypes.

Transposition des approches existantes

Dans un k-PPV, LVQ1 déplace le prototype le plus proche vers le nouvel exemple s'ils sont de la même classe ou l'éloigne sinon. LVQ1 nécessite donc que les prototypes soient étiquetés par la classe qu'ils représentent. Nous avons vu que lors de l'apprentissage des SIF, les prototypes sont appris classe par classe. Nous avons donc étiqueté chaque prototype par la classe sur laquelle il a été appris. Ainsi on obtient le déplacement suivant :

$$\vec{\Delta\mu_r} = \lambda * \delta * (\vec{X} - \vec{\mu_r}), \tag{6.2}$$

avec δ valant 0 si P_r n'est pas le plus activé, 1 si l'exemple X et le prototype P_r sont de la même classe et -1 sinon. Remarquons que seul le prototype le plus activé est déplacé. Nous verrons dans les résultats que cette approche utilisant un étiquetage dur des prototypes n'est pas très efficace lorsqu'elle est utilisée sur nos SIF car les prototypes participent à la reconnaissance de toutes les classes.

Une extension de ce mécanisme est de déplacer tous les prototypes en fonction de leur activation et de la classe qu'ils représentent. C'est le principe de FLVQ qui utilise l'activation β_r de la prémisse de règle r et un score objectif β_r^* pour cette prémisse (R étant le nombre de règle du SIF) :

$$\Delta\vec{\mu}_r = \lambda * (\beta_r^* - \frac{\beta_r}{\sum_{q=1}^{R} \beta_q}) * (\vec{X} - \vec{\mu_r}). \tag{6.3}$$

Le score objectif β_r^* vaut 1 si le prototype P_r et X sont de la même classe et 0 sinon. Ainsi tous les prototypes qui peuvent influencer la reconnaissance de X sont modifiés en fonction de leur activation. Le problème de FLVQ est que les prototypes ont toujours un étiquetage dur et que le score objectif est très difficile à atteindre. En effet si deux prototypes éloignés sont utilisés pour reconnaître une classe, et que X active beaucoup un des deux prototypes, le second sera beaucoup déplacé car il est peu activé et ce déplacement non nécessaire peut être préjudiciable.

Pour éviter ce genre d'inconvénient, la descente du gradient de l'erreur permet de prendre en compte tous les paramètres qui peuvent influencer la reconnaissance d'un exemple. Nous avons donc utilisé l'erreur quadratique E que nous avons dérivée par rapport aux centres des prototypes (équation (6.4)). Le détail du calcul est donné en Annexe B.1. On obtient la mise à jour $\Delta\vec{\mu_r}$ suivante (R étant le nombre de règle du SIF et C le nombre de classe) :

$$\Delta\vec{\mu_r} = -\lambda \frac{\partial E}{\partial \vec{\mu_r}} , \tag{6.4}$$

$$\Delta\vec{\mu_r} = 2\lambda \frac{\beta_r^2}{\sum_k^R \beta_k} \left(\sum_{c=1}^{C} (b_c - s_c)(s_c^r - s_c) \right) (\vec{X} - \vec{\mu_r})^T Q_r^{-1} , \tag{6.5}$$

b_c étant le score objectif de la classe c *i.e.* 1 si X est de la classe c et 0 sinon. Nous pouvons remarquer que cette mise à jour prend en compte tous les paramètres de la règle correspondante : l'activation de la prémisse, la forme du prototype flou à travers l'utilisation de Q_r, la participation de la règle à chaque classe. De plus elle considère l'erreur commise sur chaque classe. Malgré ces avantages, la descente de gradient est prévue pour un apprentissage hors-ligne par l'algorithme EM effectuant de multiples itérations sur toutes les données et donc n'est pas adaptée à une modification à la volée des prototypes. Cette observation sera confirmée par les expérimentations. C'est pourquoi il est nécessaire de proposer notre approche ADAPT qui prend en compte les spécificités des SIF tout en étant performantes pour l'adaptation.

L'approche ADAPT

Nous confirmerons dans les résultats que les méthodes LVQ1 et FLVQ transposées aux SIF ne permettent pas de prendre en compte la participation de chaque prototype dans la reconnaissance de chaque classe et que la descente de gradient sur les centres n'est pas adaptée à un apprentissage à la volée. L'objectif de l'approche ADAPT est de cumuler les avantages de simplicité des méthodes du type LVQ et de complétude de la descente de gradient. Pour cela nous utilisons l'interprétabilité des connaissances contenues dans les SIF.

L'approche ADAPT consiste donc à ne pas étiqueter les prototypes mais à utiliser leur participation dans la reconnaissance de chaque classe ainsi que leur degré d'activation par l'exemple. Ainsi le déplacement du prototype doit être d'autant plus important :

- qu'il est fortement activé par l'exemple,
- qu'il participe à la reconnaissance de chaque classe,
- que l'erreur commise pour chaque classe est grande.

Ces principes intuitifs sont traduits par un facteur d'apprentissage ADAPT noté $\delta_r^{'}$ défini grâce à l'interprétabilité des SIF par l'équation (6.6). Le déplacement $\Delta\vec{\mu}_r$ du prototype P_r par l'approche ADAPT est donc définie ainsi :

$$\delta_r^{'} = \beta_r \sum_{c=1}^{C} \left((b_c - s_c)\, s_c^r \right) \ , \tag{6.6}$$

$$\Delta\vec{\mu}_r = \lambda \delta_r^{'} (\vec{X} - \vec{\mu_r}) \ , \tag{6.7}$$

avec b_c le score objectif pour s_c : 1 si c est la classe de X et 0 sinon.

Nous pouvons ré-écrire l'équation (6.7) comme une somme de déplacements où chaque déplacement tente d'améliorer le score d'une classe, vers 1 pour la classe de X et vers 0 pour les autres :

$$\Delta\vec{\mu}_r = \lambda \sum_{c=1}^{C} \left(\beta_r s_c^r (b_c - s_c)(\vec{X} - \vec{\mu_r}) \right) . \tag{6.8}$$

Ainsi ADAPT effectue un compromis entre améliorer le score de la classe de l'exemple et ne pas augmenter le score des autres classes. La figure 6.1 montre un exemple fictif de ce compromis pour un problème à trois classes en deux dimensions.

Dans cet exemple, le prototype est initialement appris sur la classe 1 et un exemple de cette classe est présenté pour l'adaptation. La première idée est d'approcher le prototype de l'exemple (méthodes LVQ1 et FLVQ), mais comme le prototype participe aussi à la reconnaissance des autres classes (2 et 3 dans cet exemple) ce déplacement peut avoir un effet sur les scores de ces classes suivant la participation du prototype. Il faut donc aussi considérer le déplacement nécessaire pour diminuer le score de ces deux classes dans le déplacement global. Si la participation du prototype pour ces classes est positive, le déplacement correspondant sera dans le sens inverse de celui de la classe 1 (comme dans l'exemple). Le déplacement final du centre est la somme de tous les déplacements optimisant ainsi le score de chaque classe.

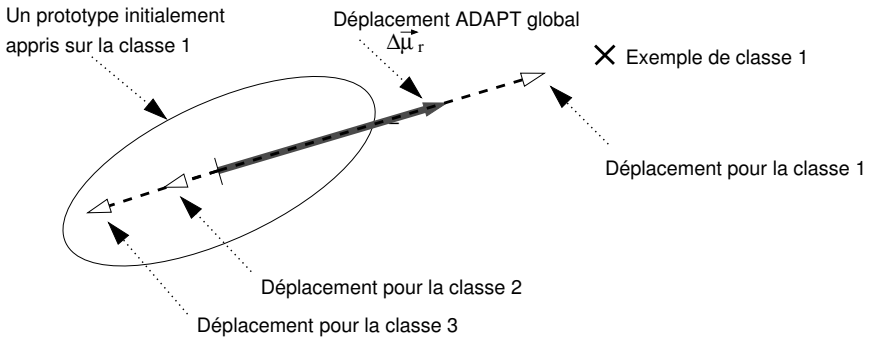


FIG. 6.1 – Principe du compromis effectué par ADAPT pour l'optimisation de chaque classe (ici pour un problème à trois classes).

Déplacer les prototypes dans l'espace d'entrée permet donc d'ajuster la représentativité des données du nouveau scripteur dans cet espace mais aussi d'améliorer les capacités discriminantes du classifieur en prenant en compte le résultat de classification. Nous allons maintenant voir comment améliorer encore ces capacités en déformant les prototypes.

6.2.2.2 Déformation des prototypes

Si le déplacement des prototypes permet de recentrer les prototypes sur les données du scripteur, la déformation de ces prototypes flous permet ensuite de mieux représenter la répartition des données décrites. La déformation ne peut donc pas être utilisée seule contrairement au déplacement. Nous présentons d'abord la transposition aux SIF de deux approches classiques pour déformer les prototypes, puis nous présentons notre approche ADAPT qui étend ces approches.

La forme du prototype P_r dans l'espace des caractéristiques est donnée par la matrice de covariance Q_r. Donc déformer un prototype consiste à réévaluer cette matrice. Néanmoins, la distance de Mahalanobis utilise la matrice inverse Q_r^{-1}, il sera donc plus efficace de ré-évaluer directement cette matrice inverse plutôt que de modifier Q_r pour ensuite l'inverser à chaque calcul de distance de Mahalanobis.

Une première approche est d'appliquer la méthode de la descente du gradient de l'erreur quadratique à ce problème comme dans [76] pour l'apprentissage d'ensembles flous. Chaque case de la matrice Q_r^{-1} est considérée comme une variable, et l'erreur est dérivée par chacune de ces variables.

Une seconde approche est d'utiliser une technique d'estimation itérative de matrice de covariance et ensuite d'inverser la formule pour obtenir une estimation itérative de la matrice inverse [31, 99]. L'estimation itérative de matrice de covariance fonctionne de façon analogue à LVQ. En effet la matrice de covariance n'est jamais qu'un calcul de moyenne, il est donc facile de cumuler les valeurs de façon itérative.

Transposition des approches existantes

L'application de la descente du gradient au calcul des matrices de covariance inverse est en grande partie identique à celle faite pour les centres. Le détail du calcul est donné en Annexe B.1. On obtient la mise à jour de la matrice Q_r^{-1} du prototype P_r suivante :

$$\Delta Q_r^{-1} = -2\frac{\beta_k^2}{\sum_r^R \beta_r}\left(\sum_{c=1}^{C}(b_c - s_c)(s_c^k - s_c)\right)\vec{w_k}\vec{w_k}^T \ , \tag{6.9}$$

avec $\vec{w} = \vec{X} - \vec{\mu_r}$. On peut remarquer qu'une grande partie de cette formule est similaire à celle de la mise à jour des centres par la même méthode (équation (6.5)).

Pour l'estimation en-ligne de matrice de covariance nous avons choisi d'utiliser celle proposée par [99][1] :

$$Q_r \Leftarrow (1-\alpha)\left(Q_r + \alpha(\vec{X} - \vec{\mu_r})(\vec{X} - \vec{\mu_r})^T\right) \ , \tag{6.10}$$

avec α étant le facteur d'apprentissage (analogue à λ pour le déplacement). Comme nous l'avons vu, il est plus intéressant pour nous de travailler sur la matrice inverse. L'inversion de cette mise à jour est expliquée dans [99] et on obtient :

$$Q_r^{-1} \Leftarrow \frac{Q_r^{-1}}{1-\alpha} - \frac{\alpha}{(1-\alpha)}\frac{(Q_r^{-1}\vec{w})(Q_r^{-1}\vec{w})^T}{1+\alpha(\vec{w}^T Q_r^{-1}\vec{w})} \ , \tag{6.11}$$

avec $\vec{w} = \vec{X} - \vec{\mu_r}$ et α le facteur d'apprentissage compris entre 0 et 1. La valeur de α est discutée dans la section 6.3. Le problème de cet apprentissage est qu'il est prévu pour être utilisé lorsqu'il y a seulement une seule matrice de covariance à estimer (une pour toutes les données) et non lorsqu'il y en a plusieurs comme dans notre cas (une par prototype). Nous ne pouvons donc pas utiliser cette approche directement sur nos SIF.

L'approche ADAPT

Les inconvénients de l'estimation itérative de la matrice de covariance sont que cet apprentissage n'est pas supervisé, il ne prend pas en compte l'activation des prototypes, leur participation dans la classification et le résultat de cette classification. Par contre ces paramètres sont pris en compte dans la descente de gradient, mais comme pour le déplacement cette approche classique n'est pas appropriée pour un apprentissage itératif, rapide et avec peu de données.

Nous proposons donc de modifier l'estimation de la matrice de covariance comme cela a déjà été fait dans [31] pour l'EFCL (Elliptical Fuzzy Competitive Learning) en remplaçant le α par un facteur d'apprentissage prenant en compte tous les paramètres des SIF. Pour cela nous utilisons le facteur d'apprentissage $\delta_r^{'}$ de l'équation (6.6) en remplacement dans l'équation (6.10) α par $\alpha\delta_r^{'}$. Ainsi nous pouvons réécrire l'équation (6.11) pour ré-estimer la matrice de covariance inverse de manière supervisée en

[1] celle de [31] étant légèrement différente mais équivalente.

utilisant les connaissances disponibles dans le SIF :

$$Q_r^{-1} \Leftarrow \frac{Q_r^{-1}}{1-\alpha\delta_r'} - \frac{\alpha\delta_r'}{1-\alpha\delta_r'} \cdot \frac{(Q_r^{-1}\vec{w}) \cdot (Q_r^{-1}\vec{w})^T}{1+\alpha\delta_r'(\vec{w}^T Q_r^{-1}\vec{w})} \ . \tag{6.12}$$

Ainsi la déformation, comme le déplacement, devient un compromis entre l'optimisation de toutes les classes.

D'une manière générale, la déformation des prototypes est beaucoup plus coûteuse que leur déplacement, quelque soit la méthode employée, ce qui peut poser des problèmes dans notre contexte d'adaptation en-ligne embarquée. Nous verrons dans la section 6.3 comment prendre en compte cet aspect du problème.

6.2.3 Adaptation des conclusions des règles

Pour effectuer l'adaptation des conclusions des SIF, nous utilisons la méthode classique de descente du gradient de l'erreur sur les conclusions des règles. Dans ce cas, cette approche est simple, peu coûteuse en ressources et permet une interprétation cohérente. Plus une règle est activée et plus l'erreur est grande sur le score, plus il faut modifier la conclusion.

La mise à jour des poids est donnée par l'équation (6.13).

$$\Delta s_c^r = n * (b_c - s_c) * \beta_r \ , \tag{6.13}$$

avec n le facteur d'apprentissage des poids compris entre 0 et 1. Il faut remarquer que cette mise à jour ne tient pas compte du score existant s_c^r liant la règle à la classe et encore moins de la classe sur laquelle le prototype de la règle a été appris. Ainsi, les prototypes peuvent facilement changer de classe principalement décrite. Nous verrons des exemples de ce phénomène dans la section 7.1.3.2.

6.2.4 Ajout/Suppression de règles

Nous avons vu dans l'état de l'art [77, 83, 112] que l'ajout et la suppression de prototypes peuvent permettre une adaptation rapide. Dans ces références les classifieurs utilisés sont des k-PPV utilisant une distance du type DTW, donc l'influence de chaque prototype est réduite à son voisinage. L'ajout ou la suppression d'un prototype a donc une influence très locale, suffisante pour le problème d'adaptation dans ce type de classifieur. Par contre dans le cas d'un SIF les prototypes peuvent avoir une importance beaucoup plus grande. En effet pour décrire chaque classe le nombre de règles est faible (généralement compris entre 1 et 4), chaque prototype décrit donc une grande quantité de données. De plus enlever/ajouter un prototype signifie enlever/ajouter une règle qui influence la reconnaissance de toutes les classes. Il devient donc dangereux de les supprimer et difficile d'en ajouter.

La suppression de prototypes a deux objectifs différents. Le premier est d'enlever les prototypes qui ne sont pas utilisés, i.e. très peu activés, pour alléger le système. Nous avons choisi dans cette étude de ne pas explorer cette piste, une solution étant de surveiller l'activation des règles au fur et à mesure de l'utilisation [31, 83]. Le second

objectif est de supprimer les prototypes qui engendrent des erreurs de classification. Dans notre contexte, les règles générant des erreurs sont déjà déplacées, déformées et leur participation aux différentes classes modifiées. Si un prototype est mal placé, soit l'adaptation des conclusions va changer suffisamment les poids pour qu'il participe à reconnaître une autre classe (celle avec laquelle il y avait concurrence), soit le déplacement et la déformation des prototypes vont l'écarter des nouvelles données et il n'y aura plus d'influence. Les premières expérimentations comme celle présentées dans la section 7.1.3.2 confirmant cette intuition, nous n'avons pas exploré plus loin la possibilité de supprimer les règles des SIF.

L'ajout de prototype répond au problème des styles d'écriture suffisamment rares pour qu'ils ne soient pas au moins un peu représentés par les prototypes existants. En effet les nouvelles données d'un scripteur avec un style très différent seront toutes (ou seulement pour certaines classes) éloignées des prototypes omni-scripteurs. Donc aucun prototype ne sera assez activé pour être déplacé ou déformé pour ensuite le reconnaître correctement, de même pour l'adaptation des conclusions, puisque toutes ces méthodes utilisent l'activation comme facteur d'apprentissage. La création d'une nouvelle règle va permettre de créer un prototype dans un secteur de l'espace des caractéristiques où aucune règle n'est applicable. La première étape est de détecter la nécessité de création d'un prototype, il s'agit typiquement d'un problème de rejet de distance, nous verrons dans la section suivante comment nous avons utilisé le rejet pour cela. La seconde étape est de créer le prototype au bon endroit et avec la bonne forme, or nous avons vu dans le chapitre précédent que l'apprentissage de prototypes flous avec peu de données est assez difficile voir impossible. Nous expliquons donc dans la section 6.2.4.2 comment nous utilisons la génération de caractères pour créer les nouveaux prototypes.

6.2.4.1 Utilisation du rejet

Notons tout d'abord que le système global n'a pas d'option de rejet. L'option de rejet est interne au processus d'adaptation mais le système global répond toujours par la classe la plus activée.

D'après la définition des types de rejet précédemment donnée, il est clair qu'il s'agit d'un problème de rejet de distance. De plus, nous n'avons pas d'exemples à rejeter qui soient disponibles, nous devons donc pouvoir apprendre le rejet sans base de contre-exemples. En outre, comme l'adaptation interagit déjà beaucoup avec le classifieur principal il ne faut pas que le rejet perturbe l'adaptation, ni que le rejet soit perturbé par l'adaptation.

Nous avons donc choisi d'utiliser l'architecture TRF[2] avec une seule fonction de confiance sur les sorties du classifieur. En effet l'adaptation va modifier les prototypes, voire en ajouter, il ne va pas être facile d'ajuster les seuils de rejet au fur et à mesure de l'adaptation. Il est donc plus simple d'utiliser des fonctions de confiances portant sur les sorties du classifieur. De même les scores de sortie vont évoluer au fur et mesure de la résolution des conflits et il est donc plus simple d'avoir une approche globale

[2]Architecture utilisant des seuils sur des fonctions de confiance, voir chapitre 3.

du problème en utilisant une seule fonction de confiance pour toutes les classes *i.e.* la fonction ψ_{C1}^{Dist}.

Pendant l'adaptation, nous séparons les formes en quatre catégories en fonction de la validation de la classification et du rejet :

- bien classées et acceptées,
- bien classées et rejetées en distance,
- mal classées et acceptées,
- mal classées et rejetées en distance.

Seules les erreurs rejetées sont conservées pour créer un nouveau prototype. En effet les formes bien reconnues, même rejetées, ne sont pas ici intéressantes puisque déjà bien reconnues et les formes mal classées acceptées sont déjà traitées par l'adaptation standard. Donc les erreurs rejetées en distance sont conservées classe par classe. Lorsque le nombre d'erreurs pour une classe dépasse un seuil (entre 2 et 4), un prototype est créé en utilisant la synthèse de caractères. Pour limiter le nombre de créations, ce seuil est incrémenté à chaque création d'une certaine valeur (par exemple pour un incrément de 0.25, il faut une erreur de plus après 4 créations pour déclencher la prochaine création)

6.2.4.2 Utilisation de la synthèse de caractères

Pour créer un prototype représentatif dans un espace avec autant de dimensions, il faut un nombre conséquent de données d'apprentissage. Nous utilisons donc la stratégie *« Déformations Image et En-ligne »* décrite dans la section 5.5 pour générer à partir des quelques exemples disponibles plus d'une centaine de caractères. Ces caractères servent ensuite à calculer le centre et la matrice de covariance du prototype flou. Il faut ensuite calculer les conclusions de cette nouvelle règle.

6.2.4.3 Calcul des conclusions de la nouvelle règle

Les conclusions représentent la participation du prototype flou à la reconnaissance de chacune des classes. L'adaptation étant supervisée, nous connaissons la classe des exemples qui ont permis de créer le prototype flou mais pas sa participation pour les autres classes. Nous initialisons donc tous les poids à 0 sauf celui de la classe correspondante à 1. Puis pour ajuster cette première approximation nous effectuons une rétro-propagation de l'erreur seulement sur les poids de cette règle en utilisant les derniers caractères entrés par l'utilisateur (la fenêtre glissante décrite dans la section suivante) et les erreurs qui ont servi à créer la règle.

Maintenant que nous avons décrit les différentes possibilités d'adapter un SIF, nous allons montrer comment les utiliser et les combiner pour effectuer une adaptation en-ligne.

6.3 Stratégies d'adaptation en-ligne

Le but des stratégies d'adaptation est d'obtenir une adaptation rapide et robuste en respectant les contraintes d'une adaptation en-ligne embarquée sur des petits périphériques mobiles comme les smartphones.

6.3.1 Utilisation d'une fenêtre glissante

Une adaptation robuste peut être obtenue en stockant tous les exemples précédemment saisis par l'utilisateur pour adapter ensuite le système à tous ces exemples à chaque fois. Plus il y a de données disponibles, meilleure est l'adaptation. Cependant ce n'est pas possible compte tenu des limitations en ressources de mémoire et en temps de calcul puisque s'adapter à beaucoup de données peut être long. Au contraire, ne s'adapter qu'au dernier caractère entré permet une adaptation très légère mais très lente puisqu'il y a moins de diversité des données à chaque fois. Donc pour trouver un compromis entre la quantité et la diversité des exemples nécessaires à une adaptation rapide et robuste et le temps de calcul nécessaire à une adaptation légère, nous proposons de conserver les F derniers exemples saisis par l'utilisateur dans un tampon (par exemple $F = 20$ dans nos expérimentations). Ce tampon appelé *fenêtre glissante* se comporte comme une file FIFO[3], dès que la taille maximale de la fenêtre est atteinte, à chaque nouveau caractère ajouté, le dernier est retiré. Pour chaque nouveau caractère arrivant dans la fenêtre, un *cycle d'adaptation* (comme défini dans la section 6.2.1) est effectué sur chaque caractère de la fenêtre. Donc le paramètre F permet de régler le compromis entre rapidité d'adaptation et temps de calcul nécessaire.

6.3.2 Facteurs d'adaptation décroissants

Les valeurs des facteurs d'apprentissage permettent aussi d'influencer la vitesse et la robustesse de l'adaptation. En effet les paramètres λ et α utilisés pour le déplacement et la déformation des prototypes contrôlent l'influence de chaque exemple sur l'apprentissage des paramètres du système : une valeur forte permet une adaptation rapide mais instable, une valeur faible permet une adaptation stable mais lente. Nous utilisons donc un mécanisme classique utilisé de différentes manières dans [24, 58, 99] consistant à faire décroître au fur et à mesure les valeurs du facteur d'apprentissage. Ainsi nous pouvons bénéficier d'une adaptation rapide au début et robuste dans le temps. Initialement le but de cette technique est de donner la même importance à tous les exemples au cours de l'apprentissage. Néanmoins dans notre contexte d'adaptation cette stratégie permet de stabiliser l'apprentissage. Pour cela nous utilisons une courbe décroissante en demi-cloche pour que la diminution des facteurs soit faible au début de l'adaptation et converge vers une valeur minimale dans le temps pour conserver un minimum d'adaptation même après une longue utilisation. Ainsi la valeur de λ décroît

[3]FIFO : First In First Out.

de λ_{max} à λ_{min} en suivant cette équation :

$$\lambda(t) = \frac{\lambda_{max} - \lambda_{min}}{1 + t/T} + \lambda_{min} \ , \tag{6.14}$$

avec t le nombre de caractères entrés par l'utilisateur. Lorsque $t = T$ nous avons $\lambda = \frac{1}{2}(\lambda_{max} + \lambda_{min})$. Le paramètre T permet donc de régler la vitesse de décroissance de la courbe (par exemple $T = 100$ dans nos expérimentations). De la même façon le facteur de déformation α décroît de α_{max} à α_{min} en suivant la même équation :

$$\alpha(t) = \frac{\alpha_{max} - \alpha_{min}}{1 + t/T} + \alpha_{min} \ . \tag{6.15}$$

6.3.3 Utilisation de la synthèse de caractères

La diversité des caractères auxquels le système s'adapte permet une meilleure rapidité et robustesse d'adaptation puisqu'elle évite le sur-apprentissage d'un nombre réduit d'exemples et permet d'avoir diverses occurrences d'une même classe. Nous avons vu que l'utilisation de la fenêtre glissante permet d'obtenir une certaine diversité dans les caractères d'adaptation. Un même caractère est utilisé F fois avant de quitter la fenêtre glissante. Donc pour augmenter encore la diversité des données nous proposons d'utiliser la synthèse de caractères pour diversifier les données d'apprentissage tout en respectant le style du scripteur. Dans cette stratégie, le premier (*i.e.* le nouveau) caractère de la fenêtre est utilisé tel quel mais les $F - 1$ suivants sont déformés avant d'être utilisés pour l'adaptation. La stratégie *« Déformations Image et En-ligne »* est utilisée pour cette synthèse.

Chapitre 7

Expérimentations et résultats

Les expérimentations se décomposent en trois étapes qui correspondent de plus en plus à une situation réelle. Les deux premières étapes sont des expérimentations faciles à mettre en place et peuvent être répétées un grand nombre de fois. La dernière étape est plus coûteuse car elle fait intervenir l'utilisateur et servira donc à valider les résultats théoriques.

Dans la section 7.1 les expérimentations comparent les différentes stratégies présentées précédemment pour dégager la plus performante et pour bien comprendre son fonctionnement. L'expérience de la section 7.2 étudie le comportement de la stratégie d'adaptation la plus performante dans la situation simulée de saisie d'un texte réel. Enfin la section 7.3 présente la mise en application réelle du processus d'adaptation sur un petit périphérique mobile ainsi que les premiers résultats obtenus pour cette expérimentation.

7.1 Simulations expérimentales

Ces expérimentations ont pour but de valider et de comprendre les différentes approches pour l'adaptation. La section 7.1.1 commence par présenter le protocole de test utilisé. Ce protocole permet de comparer ces approches dans un contexte expérimental idéal qui n'est biaisé ni par l'utilisateur, ni par la séquence d'adaptation. Ensuite la section 7.1.2 présente les résultats globaux sur l'adaptation en comparant ADAPT avec les stratégies existantes de déplacement des prototypes transposées aux SIF mais aussi en montrant l'apport de la déformation des prototypes, de l'ajout de prototypes et de l'utilisation de la synthèse de caractères. Ensuite la section 7.1.3 fait une analyse plus poussée de l'adaptation en considérant la vitesse d'adaptation et en prenant un exemple en deux dimensions du processus d'adaptation à un nouveau style.

7.1.1 Premier protocole expérimental

Ces expériences portent sur la reconnaissance des 26 caractères latins minuscules isolés en-ligne, sans aucune autre contrainte pour les scripteurs. L'apprentissage omni-scripteur initial du système de reconnaissance utilise 5287 caractères de la base IRo-

noff [110] saisie par environ 400 scripteurs. Les bases de caractères mono-scripteur ont été écrites sur un PDA par douze scripteurs différents de ceux qui ont servi à constituer la base IRonoff. Chaque scripteur a saisi 40 fois chaque caractère *i.e.* 1040 caractères par scripteur. Dans cette expérience, les caractères sont saisis dans un ordre aléatoire. De plus, dans ce protocole expérimental, il n'y a pas de reconnaissance des caractères au moment de la saisie, donc le scripteur ne peut pas s'adapter au système de reconnaissance.

Pour estimer les performances de l'adaptation sur chaque scripteur, nous procédons à un test mono-scripteur par scripteur en utilisant le principe de la validation croisée stratifiée[1] en quatre parties. Trois quarts de la base du scripteur (780 lettres) sont utilisés pour adapter le système à son style et un quart (260 lettres) est utilisée pour évaluer le taux de reconnaissance du système tout au long de l'adaptation. Pour observer les effets de l'adaptation sur un plus long terme, la base d'adaptation est utilisée deux fois consécutivement. Pour chacun des quatre découpages de la base du scripteur, le test est fait cinq fois avec une séquence des caractères d'adaptation différente pour éviter des effets dus à l'ordre des caractères.

Les résultats que nous présentons correspondent à la moyenne (et à l'écart type le cas échéant) pour chaque scripteur de ces 20 tests (cinq fois les quatre validations). Lorsque nous présentons les résultats du scripteur moyen, il s'agit de la moyenne sur les douze scripteurs du taux de reconnaissance moyen et de la moyenne des douze écarts types correspondants.

Dans ces expériences, les caractères sont décrits par les 21 caractéristiques décrites dans la section 1.1.1.2. Ces caractéristiques peuvent être calculées à la volée dans le cas de l'utilisation de synthèse de caractères lors de l'adaptation. La description de chaque classe lors de l'apprentissage omni-scripteur utilise un ou deux prototypes suivant les expériences. Le système contient donc 26 ou 52 règles suivant les cas.

7.1.2 Résultats globaux

Nous commençons par comparer les méthodes de déplacement de prototypes transposées aux SIF avec notre approche ADAPT. Ensuite nous considérons les apports de la déformation des prototypes, de la synthèse de caractère et de l'ajout de règles.

7.1.2.1 Comparaison des méthodes de déplacements

Le tableau 7.1 présente les résultats obtenus pour chaque scripteur avec les différentes stratégies de déplacement de prototypes avec un SIF utilisant une seule règle par classe ou deux règles par classe. La dernière ligne donne les résultats du scripteur moyen. La colonne *« Avant »* correspond aux taux de reconnaissance avant adaptation. *« DG »* signifie l'utilisation seule de l'adaptation des conclusions par la méthode de descente du gradient (équation (6.13)). *« DG+X »* correspond au déplacement des prémisses par la stratégie X et à la modification des conclusions par la stratégie DG

[1] Une validation croisée stratifiée force la proportion de chaque classe à être la même dans chaque base.

(avec le facteur d'apprentissage $n = 0.14$ quelque soit X). La stratégie X peut être une des quatre stratégies de déplacement que nous comparons :

- *DERIV* pour la descente de gradient sur les centres des prototypes (équation (6.5), avec $\lambda_{max} = 0.05$ et $\lambda_{min} = 0.005$) ;
- *FLVQ* pour la transposition de FLVQ (équation (6.3), avec $\lambda_{max} = 0.005$ et $\lambda_{min} = 0.0005$) ;
- *LVQ1* pour la transposition de LVQ1 (équation (6.2), avec $\lambda_{max} = 0.05$ et $\lambda_{min} = 0.005$) ;
- *ADAPT* pour notre approche de déplacement ADAPT (équation (6.7), avec $\lambda_{max} = 0.05$ et $\lambda_{min} = 0.005$).

Pour toutes ces stratégies, la demi-période de décroissance de λ est $T = 100$ et la taille de la fenêtre glissante est $F = 20$ ce qui est un bon compromis entre vitesse de calcul, encombrement mémoire et performance.

TAB. 7.1 – Comparaison des taux de reconnaissance mono-scripteur après adaptation par les différentes stratégies de déplacement, le SIF est appris avec 1 prototype ou 2 prototypes par classe.

Scripteur	Avant	DG	DERIV +DG	FLVQ +DG	LVQ1 +DG	ADAPT +DG
1	86.8 ±1.0	90.4 ±1.2	90.4 ±1.2	91.2 ±1.8	92.1 ±1.2	94.2 ±0.7
2	87.7 ±1.2	93.5 ±0.9	93.5 ±0.8	94.8 ±0.8	95.9 ±1.0	97.3 ±0.6
3	92.0 ±1.2	93.4 ±0.5	93.3 ±0.5	91.7 ±1.9	94.0 ±0.5	94.9 ±0.6
4	88.5 ±0.9	91.4 ±1.4	91.4 ±1.5	92.7 ±1.7	91.7 ±1.9	95.6 ±0.3
5	89.1 ±1.8	92.0 ±1.1	92.0 ±1.0	91.2 ±1.0	93.8 ±2.0	94.4 ±1.2
6	91.2 ±1.8	93.7 ±1.5	93.6 ±1.5	94.9 ±0.8	95.6 ±0.8	97.2 ±1.0
7	79.9 ±1.6	81.3 ±1.0	81.5 ±1.2	84.5 ±0.7	88.0 ±1.1	89.6 ±1.2
8	93.3 ±0.8	96.7 ±1.3	96.6 ±1.3	96.9 ±1.4	97.2 ±1.7	97.9 ±1.1
9	92.7 ±1.4	95.7 ±1.0	95.5 ±1.2	94.6 ±1.3	96.1 ±0.3	96.6 ±0.9
10	87.5 ±1.0	89.3 ±1.2	89.3 ±1.1	93.0 ±0.7	92.3 ±0.9	96.3 ±1.0
11	88.1 ±1.8	91.0 ±1.8	90.8 ±1.8	92.5 ±2.2	91.5 ±2.2	93.8 ±1.9
12	85.7 ±0.5	88.7 ±1.5	88.7 ±1.5	92.9 ±1.8	96.1 ±0.4	96.6 ±0.6
Moyen 1 proto.	**88.5** ±1.3	**91.4** ±1.2	**91.4** ±1.2	**92.6** ±1.3	**93.7** ±1.2	**95.3** ±1.0
Moyen 2 proto.	**88.8** ±1.4	**91.8** ±1.2	**91.8** ±1.2	**92.0** ±1.5	**93.2** ±1.3	**95.6** ±1.0

Considérons dans un premier temps les résultats sur les SIF n'utilisant qu'une règle par classe. Nous pouvons tout d'abord constater que le déplacement des prototypes améliore les résultats par rapport à l'adaptation des conclusions *DG* seule. Ensuite le déplacement par la méthode ADAPT obtient les meilleurs résultats pour tous les scripteurs. En moyenne ADAPT atteint 95.3% de reconnaissance soit une réduction de l'erreur de 59% contre une réduction de 45% pour *LVQ1*. La transposition de *FLVQ* et celle de *LVQ1* sont moins performantes mais aussi moins stables que ADAPT puisque l'écart type moyen de *ADAPT* est le plus faible. Ces observations montrent que les approches du type *LVQ1* et *FLVQ* ne sont pas optimales pour notre système de reconnaissance. En effet nous avons vu que ces méthodes étiquettent les prototypes de façon unique (un prototype appartient à une classe) au lieu de prendre en compte dans le déplacement les conclusions floues des règles comme le fait *ADAPT*. Les mauvais résultats de FLVQ (surtout pour le scripteur 3) peuvent être dus au fait que FLVQ prend plus de risque que *LVQ1* en déplaçant tous les prototypes. L'utilisation de la descente de gradient sur les centres (notée *DERIV*) ne permet pas une amélioration significative des résultats par rapport à la modification des conclusions utilisée seule. En effet l'utilisation de *DERIV* nécessite plus de données (une fenêtre beaucoup plus grande) et de présenter plusieurs fois l'ensembles des données (comme dans l'algorithme EM)[2].

Pour les SIF utilisant deux prototypes par classe, les résultats sont à peu près identiques. Seul *LVQ* et *FLVQ* ont un score légèrement inférieur qui peut s'expliquer par une augmentation, due à la présence de plus de prototypes, de leurs difficultés déjà évoquées. ADAPT a le même comportement avec deux prototypes par classe qu'avec un seul. Mais l'amélioration faible des résultats obtenus avec deux prototypes par classe ne justifie pas l'encombrement double de ces classifieurs. Dans notre contexte applicatif, il sera plus intéressant de se limiter à un seul prototype par classe.

Donc ADAPT est la méthode de déplacement des prototypes la plus appropriée à notre système. Nous utiliserons donc cette approche dans la suite des expérimentations pour déplacer les prototypes.

7.1.2.2 Contribution de la déformation, de la synthèse de caractères, de l'ajout de règles

Le tableau 7.2 présente les taux de reconnaissance moyens pour les combinaisons des stratégies suivantes :

- *ADAPT +DG* : déplacement des prototypes et adaptation des conclusions (résultats extraits du tableau 7.1) ;
- *ADAPT +DG +Déform.* : comme *ADAPT +DG* avec la déformation des prototypes par la méthode ADAPT (équation (6.12), avec $\alpha_{max} = 0.005$, $\alpha_{min} = 0.001$) ;

[2]Des tests d'apprentissage en batch ont montré que dans les conditions d'un apprentissage du type EM, *DERIV* permet d'améliorer les résultats par rapport à *DG* seule, mais pas dans notre contexte d'adaptation.

- *ADAPT +DG +Synth.* : comme *ADAPT +DG* avec utilisation de la synthèse de caractères pour les exemples de la fenêtre ;
- *ADAPT +DG +Déform. +Synth.* : comme la précédente avec la déformation des prototypes ;
- *ADAPT +DG +Déform. +Synth. +Création* : comme la précédente avec utilisation de la création de règles (seuil de création à 2 erreurs, incrément à 0.25, taux de rejet à 20%, rejet à un seuil).

TAB. 7.2 – Comparaison des taux moyens de reconnaissance mono-scripteur après adaptation par les différentes stratégies d'adaptation, pour des SIF appris avec un ou deux prototypes par classe.

	ADAPT +DG	ADAPT +DG +Synth.	ADAPT +DG +Déform.	ADAPT +DG +Déform. +Synth.	ADAPT +DG +Déform. +Synth. +Création
1 proto. / classe	95.3 ± 1.0	95.5 ± 1.0	96.9 ± 0.8	97.2 ± 0.7	97.3 ± 0.6
2 proto. / classe	95.6 ± 1.0	95.8 ± 0.9	97.0 ± 0.8	97.4 ± 0.7	97.6 ± 0.6

Considérons tout d'abord les résultats sur les SIF appris avec un seul prototype par classe. L'utilisation de la déformation en plus du déplacement des prototypes pour l'adaptation des prémisses permet une diminution de 31% de l'erreur par rapport au déplacement seul soit 73% par rapport au classifieur avant adaptation. La synthèse de caractères ne permet une augmentation que de 0.2% des performances de l'adaptation avec le déplacement seul, mais cette amélioration reste présente lorsque la déformation et la synthèse sont utilisées conjointement ce qui permet d'atteindre 97.2% de reconnaissance après adaptation. La création de règles permet d'améliorer encore les résultats pour atteindre une diminution de l'erreur de 77% par rapport au classifieur avant adaptation. A la fin du processus d'adaptation, la création de prototype a créé en moyenne de 1.4 prototypes (soient 27.4 règles par SIF) ce qui est une augmentation raisonnable par rapport à nos limites de ressources. Par contre l'utilisation de deux prototypes par classe fait monter à 52 règles utilisées (en moyenne 53.2 avec la création) pour une très faible amélioration des résultats. Nous ne considérerons donc dans la suite des résultats que les SIF appris initialement avec une seule règle par classe.

L'apport de la création de prototypes peut sembler très faible, mais comme le montre le tableau 7.3 cet apport dépend des scripteurs.

Selon les scripteurs trois situations se présentent : soit les prototypes existants sont suffisants il n'y pas besoin de création de nouveaux prototypes comme c'est le cas pour

TAB. 7.3 – Effet de la création de prototypes suivant les scripteurs, SIF appris avec un prototype par classe.

	ADAPT +DG +Déform. +Synth.	ADAPT +DG +Déform. +Synth. +Création	Nb. Proto.
1	96.6 ± 0.8	96.7 ± 0.6	27.4
2	98.7 ± 0.2	98.7 ± 0.2	26
3	**96.3** ± 0.7	**97.0** ± 0.6	**29.5**
4	96.3 ± 0.4	96.3 ± 0.3	26.7
5	**95.9** ± 1.8	**96.9** ± 1.2	**28.8**
6	98.1 ± 0.7	98.1 ± 0.7	26.1
7	94.7 ± 0.7	94.6 ± 0.8	28.5
8	98.9 ± 0.9	99.0 ± 0.8	27.8
9	98.1 ± 0.3	98.1 ± 0.3	26.5
10	98.3 ± 0.4	98.4 ± 0.4	26.6
11	95.6 ± 1.2	95.7 ± 1.2	29.2
12	98.3 ± 0.3	98.4 ± 0.3	26.4
Taux moyens	97.2 ± 0.7	97.3 ± 0.6	27.4

la moitié des scripteurs, soit la création n'apporte rien de plus (scripteurs 1, 7, 8 et 11), soit la création permet une nette amélioration des résultats (+0.7% pour le scripteur 3 et +1% pour le scripteur 5) en ajoutant peu de prototypes (entre 2 et 4).

Les stratégies d'adaptation que nous proposons permettent donc d'éviter plus de trois erreurs sur quatre pour atteindre un taux d'une erreur seulement pour 40 caractères reconnus et ceci avec un faible encombrement mémoire.

7.1.3 Analyse de l'adaptation

7.1.3.1 Comparaison de la vitesse d'adaptation

La figure 7.1 présente le taux moyen de reconnaissance mono-scripteur au cours du temps pour les différentes stratégies d'adaptation vues dans la section précédente appliquées à un SIF initialement appris avec 1 prototype par classe.

Nous pouvons d'abord constater que notre approche permet une adaptation stable c'est à dire que le taux de reconnaissance n'oscille pas dans le temps. De plus la méthode ADAPT est la plus rapide. En effet, avec 250 caractères utilisés pour l'adaptation

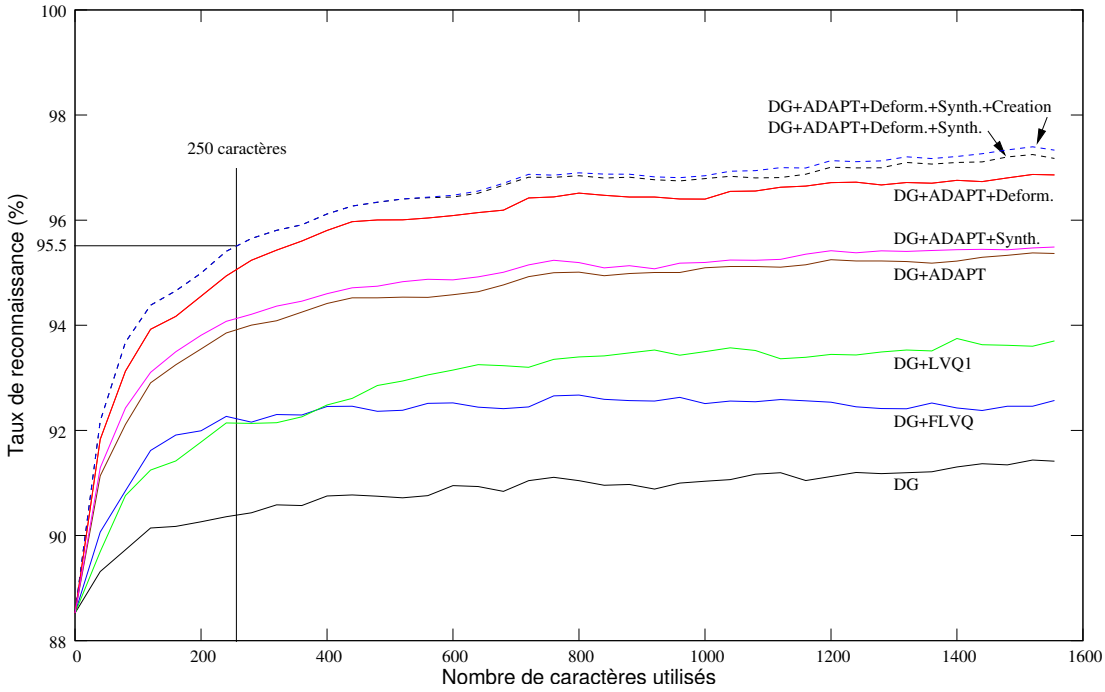


FIG. 7.1 – Comparaison des vitesses d'adaptation des différentes stratégies d'adaptation, (SIF appris avec 1 prototype par classe).

(environ 50 mots), *ADAPT+DG* atteint 93.9% alors que *LVQ+DG* atteint 92.1%. La figure montre aussi que la déformation des prototypes et l'utilisation de la synthèse de caractères permettent non seulement d'améliorer les résultats finaux de l'adaptation mais aussi d'augmenter la vitesse d'adaptation. Avec seulement 250 caractères, la stratégie *ADAPT +DG +Déform. +Synth.* passe de 88.5% à 95.5% ce qui représente 80% de la progression totale. Nous pouvons aussi constater que l'utilisation de la création de prototype n'a d'influence sur l'adaptation qu'à partir de 500 caractères environ. En effet il faut attendre que suffisamment d'erreurs d'une classe soient rejetées pour qu'il y ait création.

7.1.3.2 L'adaptation en deux dimensions

Pour comprendre le comportement du déplacement et de la déformation de la stratégie d'adaptation ADAPT, nous avons réalisé une partie des expérimentations en deux dimensions (*i.e.* avec deux des caractéristiques) avec trois classes (« `a` », « `f` » et « `x` »). Ces classes et caractéristiques ont été choisies pour obtenir une répartition intéressante en deux dimensions (légèrement dispersée, avec un peu de confusions et plusieurs styles facilement distinguables). Ainsi nous pouvons observer la répartition des lettres pour chaque scripteur en fonction de son style d'écriture.

Les figures 7.2, 7.3 et 7.4 représentent les prototypes des SIF et les frontières de décision obtenues par les conclusions. Chaque classe est initialement apprise avec deux prototypes : la classe « `a` » est décrite par les prototypes `Pa1` et `Pa2`, la classe « `f` » par les prototypes `Pf1` et `Pf2` et la classe « `x` » par les prototypes `Px1` et `Px2`. Les frontières de décision sont représentées par une ligne continue et les prototypes flous par des ellipses correspondantes à l'α coupe à 0.5. La position des caractères représentés est donnée par le calcul des deux caractéristiques.

La figure 7.2 représente le SIF initial avant adaptation et une partie de la base d'apprentissage IRonoff pour les trois classes. Les figures 7.3 et 7.4 correspondent au même SIF mais après adaptation par la stratégie *ADAPT +DG +Déform.* aux scripteurs 6 et 12 respectivement. Les caractères sur ces deux figures sont des exemples saisis par ces scripteurs.

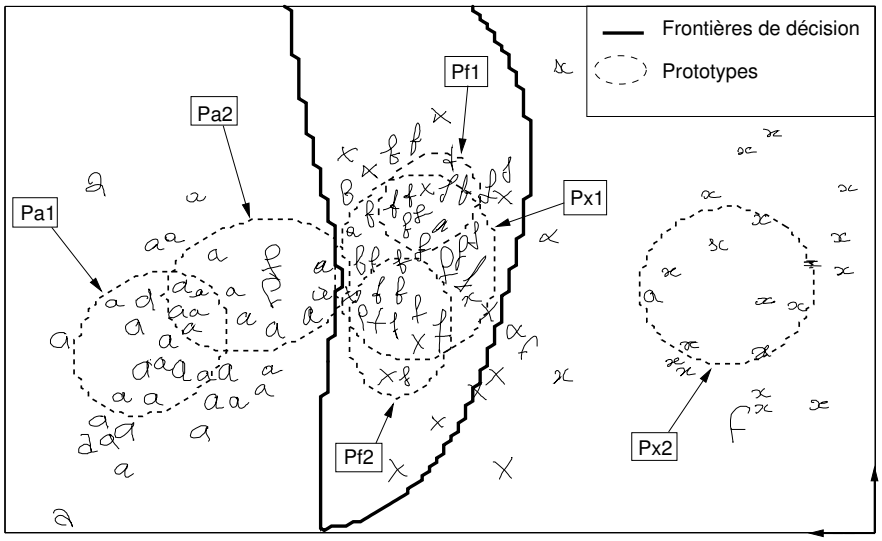


FIG. 7.2 – SIF initial appris en deux dimensions avec des exemples des trois classes et les six prototypes les décrivant (par exemple `Pa1` et `Pa2` sont les prototypes appris sur la classe « `a` »).

Nous pouvons voir sur la figure 7.2 qu'il existe différents styles de chaque caractère et que certains styles portent à confusion entre eux (les `x` en croix avec les `f` par exemple). Par contre la figure 7.3 montre que le scripteur 6 a un style d'écriture assez régulier (`f` à deux boucles, `x` cursifs en deux traits). Après adaptation à ce scripteur, les prototypes se sont bien recentrés sur la nouvelle distribution des caractères : `Pa2`, `Px1` et `Px2` se sont déplacés, `Pa1` et `Pf1` ont très peu bougé car ils sont déjà bien placés mais `Pf2` n'a pas bougé car aucune donnée ne l'utilise (donc jamais activé). De plus `Px1` a changé de classe principalement décrite puisqu'il représente maintenant les `f`. `Pf2` semble quant à lui ne pas être utilisé car il n'influence pas la frontière toute proche. Dans des travaux futurs ce type de cas pourrait être détecté pour supprimer ce prototype.

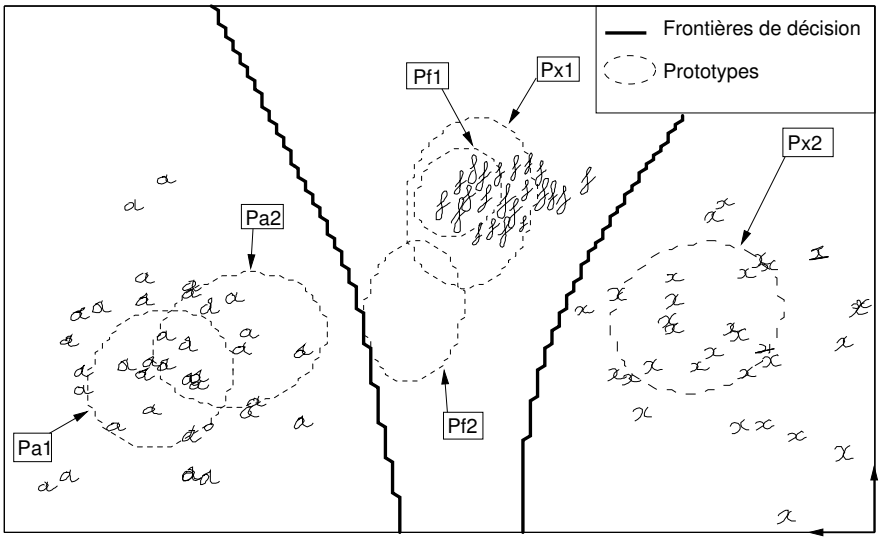


FIG. 7.3 – Frontières de décision et prototypes du SIF après adaptation au scripteur 6, avec des exemples de ses caractères.

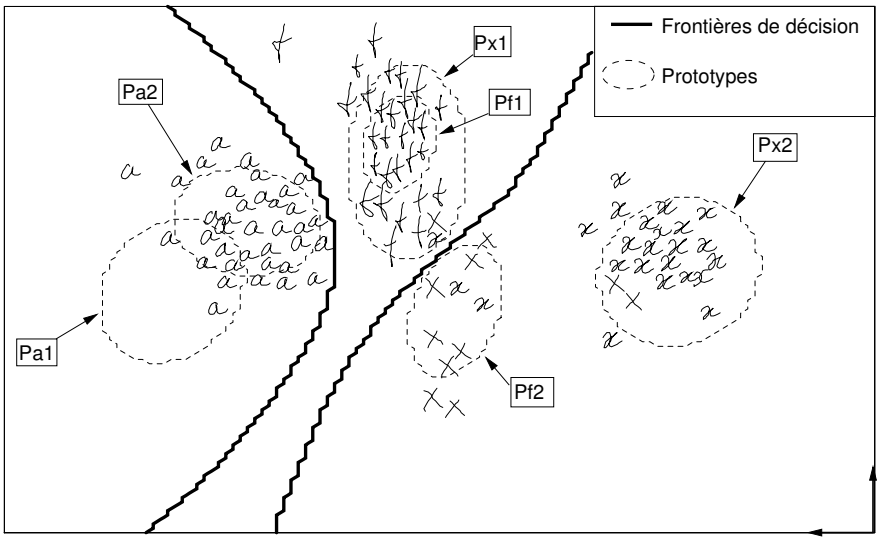


FIG. 7.4 – Frontières de décision et prototypes du SIF après adaptation au scripteur 12, avec des exemples de ses caractères.

La figure 7.4 montre le résultat de l'adaptation au scripteur 12 qui a un style très

différent du scripteur 6 : ses `a` sont plus homogènes, les `f` n'ont qu'une seule boucle ou sont en deux traits, les `x` sont soit cursifs en un seul trait soit en croix. Le plus intéressant est que après adaptation, deux prototypes différents décrivent les deux types de `x` et qu'un des deux (`Pf2`) décrivait initialement les `f` plutôt en deux traits. `Pf2` et `Px1` décrivent maintenant une classe différente de celle sur laquelle ils ont été appris.

D'autres méthodes de déplacement comme LVQ1 n'auraient pas permis les changement de classes comme observés dans ces deux exemples (`Px4` pour le scripteur 6 et `Px1` et `Pf2` pour le scripteur 12). En effet elles auraient éloigné les prototypes bien placés mais mal étiquetés même si l'adaptation des conclusions avait essayé de les utiliser. Comme ADAPT prend en compte les poids dans la décision comme une étiquette floue, notre stratégie permet de bien placer les prototypes même s'ils doivent décrire une classe différente de leur classe d'origine.

La figure 7.5 montre les six prototypes du scripteur 12 avant et après l'adaptation dans le but de pouvoir facilement apprécier le déplacement mais aussi la déformation des prototypes flous. Par exemple le prototype `Px1` a changé sa hauteur et sa largeur, mais les prototypes `Pa2` et `Pf2` ont aussi changé leur orientation. Les prototypes se sont dans l'ensemble resserrés sur les caractères du scripteur pour une description plus fine de leur répartition. Nous pouvons aussi constater combien les prototypes initiaux étaient éloignés du style d'écriture de ce scripteur et donc le déplacement nécessaire des centres.

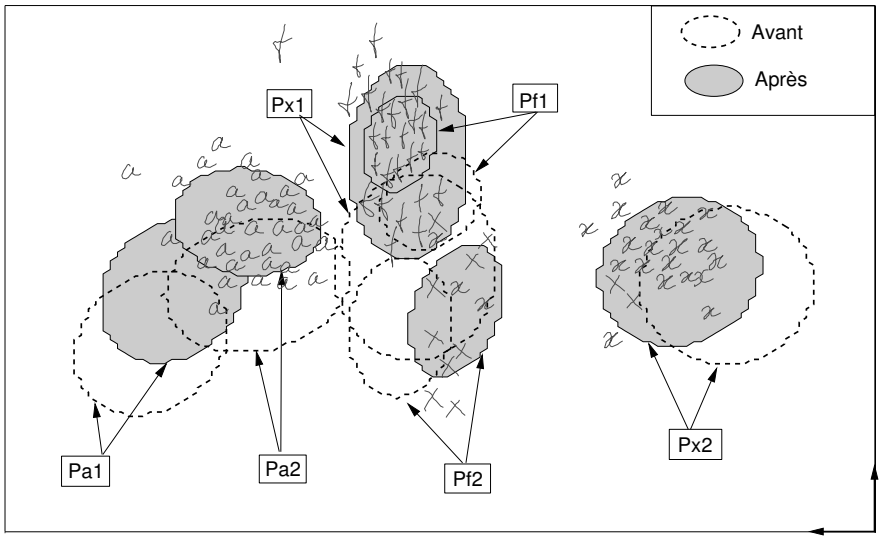


FIG. 7.5 – Positions et formes des prototypes avant et après adaptation au scripteur 12.

Même si les problèmes en plus grande dimension sont beaucoup plus complexes, cet exemple didactique en deux dimensions nous a permis de constater qu'il est important

pour l'adaptation de pourvoir déplacer les centres des prototypes, de prendre en compte dans ce déplacement les conclusions qui peuvent évoluer et de modifier la forme des prototypes représentant initialement plusieurs styles d'écriture.

7.2 Simulation d'une utilisation réelle

Dans les expérimentations des sections précédentes, nous avons montré que notre approche ADAPT est capable d'ajuster un reconnaisseur de caractères à l'écriture du scripteur courant. Mais dans ces expérimentations les données d'adaptation arrivent dans un ordre aléatoire avec une équiprobabilité d'apparition pour chaque classe. Dans un contexte réel d'utilisation, les caractères arrivent dans l'ordre du texte et avec différentes fréquences suivant le langage, le texte... Par exemple la lettre « `e` » apparaît plus souvent que les autres caractères et le trigramme « `the` » est plus probable que le trigramme « `ztw` » en anglais, ce qui n'était pas le cas dans les expériences précédentes de la section 7.1.2. Il est donc intéressant de voir à quel point ADAPT est efficace dans un contexte de saisie d'un texte réel : l'utilisateur saisit les caractères un par un et le système les reconnaît séparément. Nous avons donc simulé la saisie d'un texte en utilisant la même base de caractères que précédemment.

7.2.1 Second protocole expérimental

L'expérimentation que nous proposons ici utilise le protocole suivant. Nous considérons un texte en minuscule écrit dans la table 7.4 coupé en deux parties. Les deux premières phrases (266 caractères) sont utilisées comme séquence d'adaptation et les deux suivantes sont utilisées comme séquence de test. A chaque caractère de ces deux textes est associé un caractère manuscrit en-ligne qui est choisi pour chaque scripteur aléatoirement dans sa base d'adaptation et sa base de test respectivement (les mêmes bases que celles définies dans le premier protocole). Ainsi les caractères de la séquence d'adaptation ne sont pas utilisés dans la séquence de test. La stratégie d'adaptation utilisée est *ADAPT +DG +Déform. +Synth.* Le SIF initial est appris avec un seul prototype par classe sur la base IRonoff comme dans le protocole précédent.

7.2.2 Résultats

La table 7.5 montre le texte de test reconnu par le SIF initial pour l'écriture du scripteur 4 (la correction est signalée sous chaque erreur). La table 7.6 montre le même texte reconnu après adaptation au scripteur 4 sur la première partie du texte. Nous pouvons remarquer que le système commence avec 26 erreurs ce qui représente un taux de reconnaissance de 89% ce qui rend la phrase difficilement lisible. Après adaptation, il ne reste que trois erreurs (soit un taux de reconnaissance de 98.7%) ce qui ne représente qu'une erreur tous les 15 mots. Dans une application réelle, un dictionnaire corrigerait facilement ces erreurs restantes.

Le même test a été effectué sur tous les scripteurs en validation croisée avec cinq tirages de caractères différents, de la même manière que le premier protocole mais en

TAB. 7.4 – Phrases utilisées pour l'adaptation du SIF. Les espaces et la ponctuation sont ignorés pendant le processus.

in this study we present an automatic on-line adaptation mechanism to the writer's handwriting style for the recognition of isolated handwritten characters. the classifier we use here is based on a fuzzy inference system (fis) similar to those we have developed for handwriting recognition but simplified for this study.
doing so, the adaptation mechanisms presented here can be transposed to the original systems. in this fis each premise rule is composed of a fuzzy prototype which represents intrinsic properties of a class. the consequent part of rules associates a score to the prototype for each class.

TAB. 7.5 – Phrases de test reconnues (ainsi que leur correction) avant adaptation au scripteur 4 (26 erreurs soient 11.0% des caractères des phrases).

doing so, tkl adaptation mechanisms presented aese can
he h r
be tsansposed to the original jystlws. in tkis fis eacl
r s em h h
psemise sule is composed of a fuzzy psototype which repsesents
r r r r
intrinjic proplsties ol a class. the consequent past ol rules
s er f r f
associates a scose to the psototype fos lack clajs.
r r r e h s

Tab. 7.6 – Phrases de test reconnues (ainsi que leur correction) après adaptation au scripteur 4 (3 erreurs soient 1.3% des caractères des phrases).

doing so, the adaptatjon mechanisms presented here can i
be transposed to the original systems. in this fis eacl h
premise rule is composed of a fuzzy prototype which represents
jntrinsic properties of a class. the consequent part of rules i
associates a score to the prototype for each class.

respectant le contexte du texte. En moyenne le taux de reconnaissance passe de 88.5% à 96.8% après adaptation sur les 266 caractères du premier texte. Cette évolution est meilleure que celle observée dans les résultats de la section 7.1.3.1 puisque d'après la figure 7.1 la même stratégie d'adaptation permettait de passer de 88.5% à 95.5% pour le même nombre de caractères utilisés en adaptation. Cette différence d'efficacité est due à une bonne adaptation aux lettres les plus fréquentes.

Cette expérimentation montre que notre stratégie d'adaptation n'est pas perturbée par la fréquence et la séquence des caractères d'un texte réel et arrive même à en tirer avantage en optimisant la reconnaissance des caractères les plus fréquents.

7.3 Mise en application réelle

Même si les expérimentations précédentes cherchent à se rapprocher d'un contexte réel d'utilisation en utilisant des caractères saisis sur un périphérique mobile et en se plaçant dans un contexte de saisie de texte, il reste plusieurs points de différence avec une application réelle. Dans cette section nous présentons une plate-forme expérimentale développée dans le but de se rapprocher le plus possible d'une application réelle de reconnaissance de caractères isolés utilisant l'adaptation à la volée. Cette application permet de saisir un texte et, bien qu'il s'agisse toujours de reconnaissance de caractères isolés, le texte est saisi mot par mot.

7.3.1 Impact de l'utilisateur

La principale différence entre une application réelle et nos protocoles de tests utilisés jusqu'ici est l'interaction avec l'utilisateur. Cette interaction peut avoir trois effets sur l'adaptation.

Premièrement, dans toutes les expérimentations précédentes l'utilisateur est considéré comme ne faisant aucune erreur, ce qui est le cas car lors de la saisie l'utilisateur ne doit écrire qu'une seule lettre à la fois. Dans une application réelle l'utilisateur doit écrire un texte et donc réfléchir au sens de ce qu'il écrit. Il y a donc plus de risques qu'il fasse des erreurs de validation de la reconnaissance des caractères, ces erreurs peuvent avoir une influence sur l'adaptation comme le montre [111].

Deuxièmement, les caractères isolés sont placés en contexte de mot donc l'écriture sera moins stable que dans les protocoles précédents. En effet, la forme des lettres pourra changer suivant leur position dans le mot et d'après les caractères précédents. L'interface utilisée dans l'application aura aussi une influence sur la qualité de la saisie de l'utilisateur car par exemple le résultat de la reconnaissance ne sera peut-être pas clairement visible ou la zone d'écriture trop petite.

L'interaction avec l'utilisateur aura un troisième effet sur l'adaptation : l'utilisateur peut lui aussi s'adapter au système de reconnaissance. En effet sur les systèmes existants sans adaptation, l'utilisateur apprend à écrire avec un style bien reconnu par la machine. Cet apprentissage est inévitable puisque chaque erreur demandera une correction et donc un effort de la part de l'utilisateur qui cherchera à minimiser cet effort en s'adaptant au système. Dans les protocoles précédents ce phénomène n'avait pas lieu puisqu'il n'y avait pas de retour de reconnaissance lors de la saisie. Nous pouvions donc être certain que l'amélioration des taux de reconnaissance était due à l'adaptation du système au style d'écriture et non à l'adaptation de l'utilisateur au système.

Une différence moins importante mais dommageable pour l'interprétation des résultats est que les tests en application réelle ne sont disponibles qu'en petite quantité car très coûteux. En effet dans les expérimentations précédentes nous utilisions la validation croisée et nous répétions les tests avec des ordres de caractères différents pour multiplier les résultats et obtenir des valeurs moyennes stables ainsi que des écarts-types. Dans une expérimentation de cas réelle, la saisie d'un scripteur ne peut être utilisée qu'une seule fois et seulement sur le périphérique mobile à la volée. Il n'est pas possible de rejouer l'expérience avec des paramètres différents ou un ordre des caractères différents puisqu'il faut que l'utilisateur soit présent pour l'interaction.

7.3.2 Protocole de test

Nous avons donc développé une application pour PDA pour réaliser un test d'adaptation à la volée dans un contexte réel de saisie de texte. Cette interface nommée IREMA[3] (pour *Interface de Reconnaissance d'Écriture avec Mécanismes d'Adaptation*) est présentée par la figure 7.6.

[3]Merci à Delphine Lamarche pour la réalisation de IREMA pendant son stage de quatrième année.

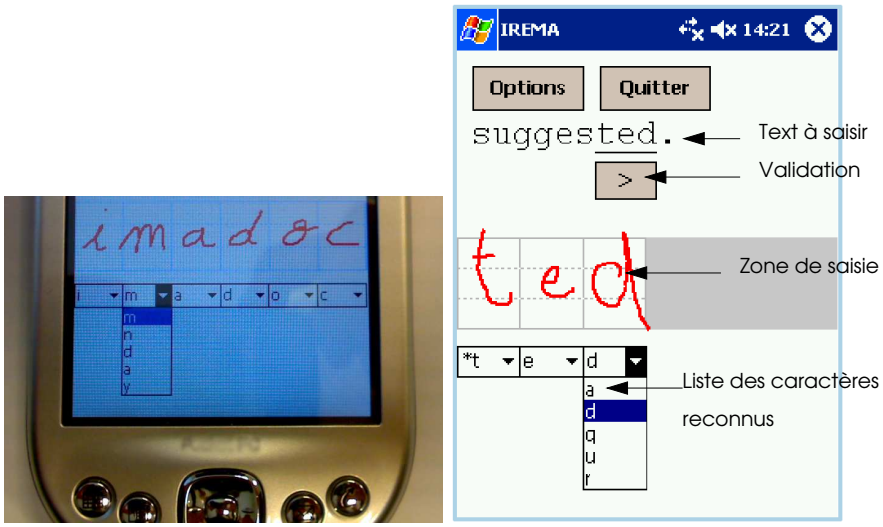


FIG. 7.6 – Interface de l'application IREMA.

Le texte à saisir s'affiche en haut de l'écran, comme seuls six caractères peuvent être entrés sur le même écran, les caractères à saisir sont soulignés. L'utilisateur écrit dans les cases dessinées au milieu de l'écran (il s'agit donc bien de caractères isolés). Seul les cases nécessaires à la saisie sont affichées (ici trois cases pour 't','e' et 'd'). La reconnaissance apparaît alors sous le caractère reconnu sous forme de liste ordonnée, la meilleure classe étant la première et celle sélectionnée par défaut. En cas d'échec de la reconnaissance, l'utilisateur peut soit ré-écrire par dessus son caractère, soit choisir dans la liste déroulante la bonne classe. Un joker (signalé par le symbole '*') est mis à disposition si le caractère demandé n'apparaît pas dans la liste. Ce joker serait remplacé par l'accès à un clavier virtuel dans une application réelle. L'utilisateur n'est pas obligé d'attendre qu'un caractère soit reconnu pour commencer à saisir le suivant. Une fois que toutes les cases sont remplies, une flèche apparaît pour passer au mot suivant (ou à la suite du même mot). Il n'y a aucune vérification automatique de la validité de la reconnaissance, c'est l'utilisateur qui doit détecter seul les erreurs[4]. Lorsque l'utilisateur clique sur la flèche, la reconnaissance est considérée comme validée par l'utilisateur et le processus d'adaptation est réalisé sur ces caractères validés. Le système de reconnaissance utilisé pour la reconnaissance des caractères suivants a donc évolué. Le texte à saisir est en anglais et fait 1564 caractères minuscules, la ponctuation est affichée pour la compréhension du texte par l'utilisateur mais n'est pas saisie.

Tout au long de l'utilisation, nous conservons les traces écrites par l'utilisateur, les réponses du classifieur et la classe validée par l'utilisateur. Nous avons donc accès en fin d'utilisation à plusieurs indicateurs intéressants :

[4]Dans une version future, on peut imaginer ici l'utilisation d'un lexique pour corriger une partie des erreurs.

– le nombre de caractères qui ne sont pas reconnus du premier coup *i.e.* qui ont nécessité une correction, noté *erreurs au premier essai*,
– le rang moyen du bon caractère dans la réponse du classifieur parmi tous les essais de l'utilisateur (0 étant la valeur du premier rang), noté *rang de la bonne réponse*,
– le nombre d'erreurs d'étiquetage commises par l'utilisateur, noté *erreurs utilisateur*.

La stratégie d'adaptation que nous avons utilisée pour cette expérimentation est la stratégie *ADAPT +DG* simple. En effet l'utilisation de la déformation des prototypes et de la synthèse de caractères ont aussi été embarquées mais se sont avérées trop coûteuses en temps de calcul, ce qui aurait pu perturber la saisie déjà longue. Ces approches nécessitent une optimisation du code pour être utilisable dans de bonnes conditions. Les paramètres d'adaptation utilisés sont les mêmes que ceux des expérimentations précédentes, notamment la taille de la fenêtre glissante ($F = 20$) qui influence beaucoup le temps de calcul de l'adaptation.

Huit scripteurs ont participé à cette expérimentation. La saisie du texte complet prend environ 1 heure et peut être réalisée en plusieurs fois.

7.3.3 Résultats

La figure 7.7 présente les résultats obtenus pour le test en conditions réelles d'utilisation avec adaptation. Ces courbes représentent la moyenne pour les huit scripteurs des trois indicateurs. Chaque point des courbes correspond à la valeur moyenne de l'indicateur sur les 200 derniers caractères saisis.

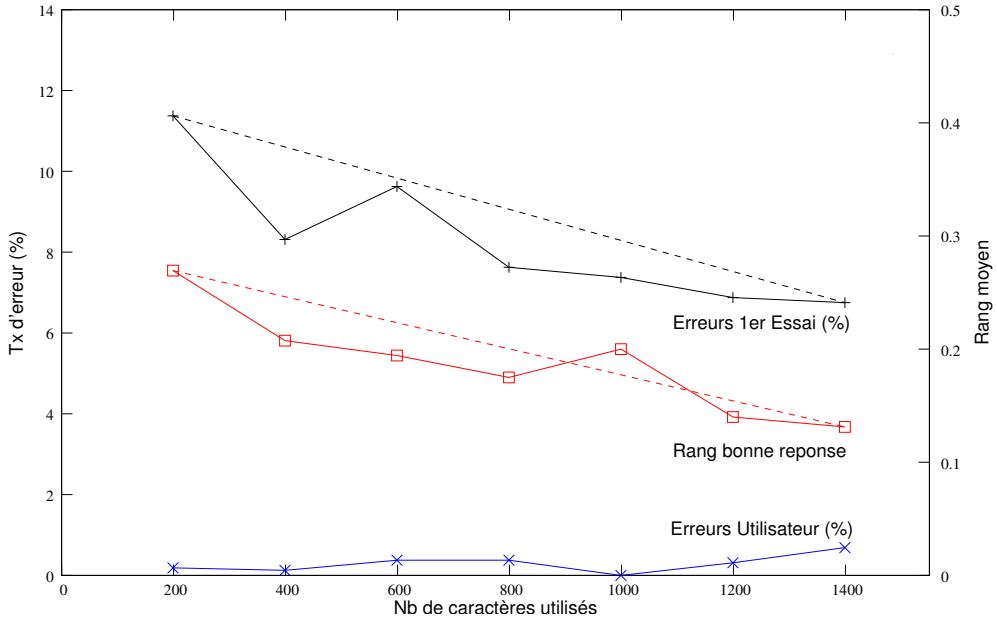


FIG. 7.7 – Résultats obtenus pour le test en conditions réelles.

Nous pouvons d'abord voir que le nombre d'erreurs commises par l'utilisateur est assez faible (inférieur à 1%). Ce petit nombre d'erreurs valide le fait que l'interface est utilisable et n'est pas une source d'erreurs importante de la part de l'utilisateur. Il faut tout de même noter que ces erreurs sont incluses dans les erreurs du type *erreurs au premier essai*, les erreurs réllement commises par le système correspondent à la différence entre ces deux valeurs. Nous négligerons ici cette nuance.

Le nombre d'*erreurs au premier essai* montre que le système s'adapte bien à l'utilisateur puisque la reconnaissance est bonne du premier coup dans 88.6% des cas au début de la saisie puis dans 93.3% des cas à la fin. Nous avons donc une diminution de l'erreur de 41%. Cette adaptation est confirmée par la diminution du *rang de la bonne réponse*, en effet au début de l'utilisation ce rang est en moyenne de 0.27 pour arriver à 0.13 en fin d'utilisation. La diminution du rang est importante car un caractère avec un style vraiment différent des modèles omni-scripteurs n'apparaît pas dans la liste de choix du reconnaisseur (obtient donc un rang maximum de 5) en obligeant le scripteur à utiliser le joker. Le fait que l'adaptation fasse diminuer ce rang signifie donc que non seulement la bonne réponse est plus souvent en première position mais aussi qu'elle est plus souvent dans la liste des réponses proposées.

Les résultats obtenus sur IREMA sont positifs mais cependant moins élevés que ceux obtenus dans les conditions expérimentales précédentes. Cette différence peut s'expliquer déjà par le petit nombre de scripteurs disponibles pour les saisies en contexte réel mais aussi par la plus grande variabilité du style d'écriture du scripteur dans ce contexte. En effet, les caractères sont en contexte de mots et donc, même s'ils restent isolés, leur forme dépend des caractères voisins. De plus la saisie est beaucoup plus longue et l'utilisateur peut se fatiguer, faire des pauses longues (saisies sur deux jours par exemple) ou changer les conditions de la saisie (au bureau, à la maison...) ce qui a pour conséquence de modifier légèrement son style et donc de rendre plus difficile l'adaptation.

Il serait très intéressant de réaliser exactement la même expérience dans les mêmes conditions mais sans utiliser l'adaptation dans le but de mesurer la vitesse d'apprentissage du scripteur. Ainsi, la différence entre les résultats obtenus avec et sans adaptation montrerait l'apport réel de notre approche.

7.4 Conclusion et discussion

Dans le contexte de l'adaptation de classifieurs flous pour la reconnaissance de caractère en-ligne, nous avons présenté dans ce chapitre une nouvelle approche d'adaptation à la volée nommée ADAPT. Cette approche permet d'adapter au fur et à mesure de son utilisation un Système d'Inférence Floue basé sur des prototypes flous avec des conclusions numériques. Cette adaptation incrémentale est effectuée conjointement en déplaçant les centres des prototypes flous, en les déformant et en ré-évaluant les conclusions numériques. L'originalité de cette approche est qu'elle permet une adaptation incrémentale prenant en compte toutes les connaissances contenues dans le SIF combinées grâce à leur interprétation.

Nous proposons aussi d'utiliser la synthèse de caractères pour augmenter la vitesse d'adaptation. La synthèse de caractères permet d'augmenter la quantité de caractères différents disponibles tout en respectant le style d'écriture du scripteur. De plus une stratégie de création de nouvelles règles dans le SIF utilisant le rejet et la synthèse de caractères permet d'améliorer encore les performances du système pour certains scripteurs. Le rejet permet ici de détecter lorsqu'aucune règle ne peut s'appliquer correctement et donc qu'il est nécessaire de créer une nouvelle règle. La synthèse de caractères permet alors d'augmenter la quantité de caractères disponibles pour apprendre les paramètres de cette nouvelle règle.

Ces techniques d'adaptation sont combinées dans des stratégies d'adaptation plus ou moins complètes pour permettre d'embarquer facilement le processus d'adaptation sur des périphériques mobiles. L'utilisation d'une fenêtre glissante des derniers caractères saisis et l'activation des différentes possibilités d'adaptation du SIF permettent de trouver un bon compromis entre performance de l'adaptation et capacité de calcul disponible.

Nous proposons dans nos expérimentations différents protocoles expérimentaux permettant de tester nos approches dans des contextes de plus en plus proches d'un cas réel d'utilisation de l'adaptation à la volée pour la reconnaissance de caractères isolés.

Dans nos premiers tests expérimentaux nous comparons nos différentes stratégies d'adaptation avec des techniques plus classiques comme LVQ ou FLVQ de déplacement de prototypes transposées sur nos SIF. Notre approche complète permet une réduction de 77% de l'erreur en faisant passer de 88.5% à 97.3% le taux de reconnaissance moyen. Une expérimentation en deux dimensions illustre le fonctionnement de notre approche ADAPT en montrant comment le système s'adapte à un nouveau style d'écriture. Le second protocole expérimental simule l'utilisation de l'adaptation en contexte de saisie de texte. Nous avons montré qu'ADAPT permet une meilleure adaptation dans ce contexte en utilisant avantageusement les différentes fréquences de caractères.

Enfin, nous avons porté notre stratégie d'adaptation sur un périphérique mobile pour évaluer les effets de l'adaptation dans un contexte réel de saisie de texte avec une interaction de l'utilisateur. Nous avons mis en valeur les différences, liées à l'utilisateur, entre ce contexte réel et les expérimentations précédentes. Les premiers résultats montrent une diminution de 41% de l'erreur ce qui est positif mais qui confirme aussi les difficultés liées à ce type de tests où le comportement de l'utilisateur peut avoir différents impacts.

Les perspectives de ces travaux sur l'adaptation sont d'étendre les tests en conditions réelles notamment pour mesurer la capacité du scripteur à s'adapter à un système de reconnaissance qui lui ne s'adapte pas. De plus il faut continuer d'étudier la stratégie de création de prototypes pour savoir si elle peut être utile à plus de scripteurs et plus rapidement. Une piste de recherche dans ce sens, mais aussi d'un intérêt plus général, est d'étudier d'autres méthodes de synthèse de caractères pour augmenter la quantité de données représentative du style du scripteur à partir de quelques exemples seulement. Enfin, la suppression des prototypes inutiles ou sources d'erreurs doit être mis au point. En effet ceci permettra non seulement d'optimiser le classifieur mais aussi de l'alléger suivant les besoins de l'utilisateur, ce qui sera très intéressant pour l'embarquement sur

les petits périphériques mobiles.

Conclusion générale et perspectives

Conclusion

L'objectif principal de ces travaux est d'améliorer la reconnaissance en-ligne de l'écriture. Pour cela nous avons proposé deux axes de recherche complémentaires, le *rejet* des erreurs potentielles et l'*adaptation* au style d'écriture de l'utilisateur :

- *le rejet* permet d'éviter de proposer à l'utilisateur des réponses absurdes ;
- *l'adaptation* permet de spécialiser automatiquement le système générique de reconnaissance au style d'écriture du scripteur courant.

Même si ces principes peuvent s'appliquer à beaucoup de domaines plus généraux de la reconnaissance des formes, cette thèse se plaçait dans le contexte de la reconnaissance en-ligne de caractères isolés pour les petits système nomades. Nous avons donc pris en compte un certain nombre de contraintes liées à cet objectif d'embarquement de nos systèmes. C'est pourquoi, parmi les différentes solutions que nous avons explorées, nous avons privilégié celles permettant le meilleur compromis entre les ressources en mémoire, les temps de calcul et les performances. De plus, les systèmes que nous avons proposés sont *transparents*, c'est-à-dire qu'il est possible de comprendre pourquoi telle décision a été prise. Cette propriété nous a permis de proposer des solutions intuitives et efficaces.

Dans la première partie, nous avons étudié avec précision la notion de rejet dans le but d'aider un utilisateur à élaborer une option de rejet en adéquation avec son problème. Dans un premier temps cette étude a considéré le problème général du rejet puis nous avons proposé de nouvelles solutions adaptées à notre contexte applicatif.

Nous avons d'abord dégagé cinq points importants à considérer lorsque l'on souhaite utiliser ou étudier des classifieurs avec une option de rejet :

- définir la nature du rejet considéré : *rejet d'ambiguïté* pour détecter les erreurs de confusion entre deux classes ou le *rejet de distance* pour délimiter les connaissances du classifieur.
- choisir l'*architecture* pour l'option de rejet parmi les quatre principales : une classe de rejet dans le classifieur principal (RC) ; un classifieur spécialisé dans l'espace des caractéristiques (SC) ; un classifieur spécialisé dans l'espace des fonctions de confiance (SCRF) ; des seuils sur les fonctions de confiance (TRF).

- choisir les classifieurs utilisés : suivant la fonction du classifieur dans l'architecture choisie et le type de rejet, il faudra préférer un classifieur avec une modélisation *discriminante* ou *intrinsèque* des connaissances utilisées.
- définir ce qui doit être rejeté : pour le rejet de distance, nous pouvons distinguer trois situations suivant la représentativité des contre-exemples disponibles pour l'apprentissage de l'option de rejet : soit ils sont bien représentatifs, soit ils sont partiellement représentatifs, soit ils ne sont pas représentatifs (voire non disponibles). Suivant les situations, il faudra faire des choix différents d'architecture.
- évaluer l'option de rejet : nous avons décrit et étendu l'utilisation de plusieurs mesures classiques pour qu'elles permettent de comparer entre elles des architectures différentes.

Parmi les différentes architectures, l'architecture TRF répondait le mieux à nos contraintes d'embarquement et de transparence. Nous avons donc proposé pour l'architecture TRF une option de rejet générique utilisant plusieurs seuils sur les fonctions de confiance. La généricité de notre option de rejet permet d'avoir une approche identique du rejet et de son apprentissage quelle que soit la nature du rejet considérée et le classifieur utilisé.

Pour l'apprentissage des seuils de notre option de rejet, nous avons proposé trois algorithmes différents : AMTL1 qui permet de prendre en compte les contre-exemples disponibles lors de l'apprentissage de l'option de rejet ; AMTL2 qui permet d'apprendre un rejet de distance sans avoir de contre-exemples disponibles ; TGD qui est une application de la méthode de la descente du gradient de l'erreur à notre problème.

Les tests exhaustifs comparant les différentes architectures ont montré que l'extension de l'architecture TRF que nous proposons, lorsqu'elle est apprise avec nos algorithmes d'apprentissage, permet d'obtenir des options de rejet de distance et d'ambiguïté efficaces avec un faible coût en ressources.

Ces recherches ont été publiées dans [MAR06, MA06a, MA06b] qui présentent notre approche générique du problème du rejet ainsi que les algorithmes d'apprentissage des seuils AMTL1 et AMTL2.

Dans la seconde partie, nous avons proposé une stratégie d'adaptation rapide à la volée d'un Système d'Inférence Floue (SIF) pour la reconnaissance de l'écriture d'un scripteur, l'utilisateur courant du système. Nous avons aussi présenté plusieurs protocoles de tests d'adaptation du système au style d'écriture du scripteur.

La principale difficulté liée au problème d'adaptation est la faible quantité de données disponibles pour l'apprentissage. Nous avons donc commencé par présenter différentes stratégies pour synthétiser des caractères manuscrits en-ligne à partir d'exemples de caractères. Pour être utile à l'adaptation, cette synthèse doit respecter le style d'écriture du scripteur. Nous avons proposé deux stratégies originales de génération de caractères. La première est basée sur des techniques de déformations des caractères d'origine utilisant des connaissances spécifiques à l'écriture en-ligne. La seconde stratégie utilisant le concept d'analogie entre caractères permet de combiner plusieurs caractères entre eux pour en créer de nouveaux. L'utilisation de l'analogie a été possible grâce à une coopération avec Sabri Bayoudh et Laurent Miclet. Ces recherches sur la synthèse

de caractères ont été publiées dans plusieurs communautés, celle du document [MA06c], de la graphonomie [MBAM07] et de l'apprentissage [BMMA07].

L'adaptation d'un SIF se fait soit en modifiant les règles existantes soit en créant de nouvelles règles. Nous avons présenté la façon de modifier des règles existantes en déplaçant et en déformant les prototypes flous. L'originalité de notre approche, nommée ADAPT, est qu'elle permet une adaptation incrémentale prenant en compte toutes les connaissances contenues dans le SIF, combinées grâce à leur interprétation. Nous avons aussi proposé une stratégie pour créer de nouvelles règles en détectant les caractères nécessitant une création puis en apprenant un nouveau prototype pour les représenter.

Notre stratégie d'adaptation fait le lien entre les trois thèmes abordés : le rejet, l'adaptation et la synthèse de caractères. Par exemple, pour la création de nouvelles règles, le rejet de distance permet de détecter lorsqu'aucune des règles existantes ne s'applique. Ensuite la synthèse de caractères enrichit la variété des données disponibles pour l'apprentissage du nouveau prototype à partir des quelques caractères recueillis. La synthèse de caractères est aussi utilisée dans la modification des règles existantes afin d'accélérer le processus d'adaptation.

Les expérimentations sur l'adaptation ont été effectuées en trois étapes. Un premier protocole expérimental a permis de comparer notre stratégie à des approches existantes que nous avons transposées aux SIF. Le second protocole a validé notre approche dans le contexte simulé de la saisie de textes. Nous avons enfin embarqué notre système de reconnaissance avec notre approche pour l'adaptation sur un petit périphérique mobile. Cette dernière expérimentation a mis en valeur les difficultés nouvelles pour l'adaptation liées à l'interaction du système avec l'utilisateur. En effet, l'utilisateur peut faire des erreurs, faire évoluer son style d'écriture au cours du temps selon le contexte mais il peut aussi adapter son style d'écriture au système de reconnaissance.

Une partie de ces recherches sur l'adaptation ont été publiées en plusieurs étapes. Dans [MAR04, MAR05b] nous avons présenté le principe de l'adaptation à la volée ainsi que notre technique de déplacement de prototypes, puis leur déformation dans [MAR05a]. Enfin, notre approche de modification des prototypes et une partie de nos expérimentations sont présentées dans [MAR07].

Perspectives

Concernant notre étude du rejet, deux perspectives principales peuvent être envisagées : d'abord en appliquant notre approche à d'autres contextes, ensuite en explorant des options de rejet que nous avons jusque là mises de côté pour plusieurs raisons.

Notre expertise sur la notion de rejet nous a permis de commencé à intégrer le rejet à plusieurs niveaux dans les systèmes de reconnaissance d'écriture. Nous avons utilisé notre approche TRF dans le problème de la segmentation de textes en-ligne pour séparer les espaces intra-mot des espaces inter-mots [93], dans la coopération entre segmentation et reconnaissance pour les caractères hors-ligne [95], mais aussi pour la correction d'erreurs dans un système de reconnaissance de phrases [92]. Une étude plus poussée de ces différents domaines d'application du rejet devrait permettre de mieux

définir leur besoins et ainsi d'améliorer leurs performances.

Parmi les options de rejet déjà évoquées que nous n'avons pas explorées, deux architectures en particulier nous semblent intéressantes à étudier. La première améliore l'architecture SC en sélectionnant les meilleures caractéristiques pour chacun des deux classifieurs (classifieur principal et classifieur de rejet), comme ils ne sont pas confrontés aux mêmes problèmes, le jeu de caractéristiques optimal n'est peut-être pas le même. La seconde architecture que nous nous proposons d'explorer consiste à profiter de la généricité de notre formalisme pour combiner dans l'architecture TRF plusieurs types de fonctions de confiance. Cette combinaison peut avoir deux avantages : combiner les informations disponibles de plusieurs ensembles de fonctions de confiance pour un même type de rejet ou bien combiner les deux types de rejet (distance et ambiguïté) dans la même option de rejet. Cette dernière possibilité est très intéressante car beaucoup d'applications peuvent avoir besoin des deux types de rejet en même temps et nous avons vu que le rejet de distance peut avoir une influence sur les performances de classification. Il faudrait donc explorer les possibilités d'utiliser, et pourquoi pas d'apprendre, les deux types de rejet en même temps.

Concernant l'efficacité du rejet, il faudrait effectuer des expérimentations mesurant l'effet du rejet dans les interfaces homme-machine sur le comportement de l'utilisateur. Intuitivement, il est plus agréable de rejeter une forme plutôt que de proposer une erreur, mais une expérimentation mesurant la satisfaction de l'utilisateur, la vitesse de saisie ou la durée de prise en main de l'interface de saisie permettrait de confirmer cette approche.

Notre approche de l'adaptation peut être améliorée par certains aspects mais ouvre aussi de nouveaux champs de recherche connexes.

Tout d'abord, notre stratégie de création de prototypes a montré son efficacité pour certains scripteurs. Pour améliorer cette approche nous pouvons d'abord améliorer la détection de la nécessité d'une création. En effet nous ne créons de nouveaux prototypes qu'après l'apparition d'un certain nombre d'erreurs rejetées, cette double contrainte retarde le moment de la création du prototype qui pourrait avoir lieu plus tôt en relâchant ces prérequis. Une seconde approche, complémentaire de la précédente, est d'enrichir la synthèse de caractères par de nouvelles déformations ou stratégies pour mieux représenter le style du scripteur. Cette amélioration profiterait aussi à la modification des règles existantes.

Un nouveau champ de recherche pour l'adaptation est l'apprentissage d'un système de reconnaissance mono-scripteur à partir d'aucune connaissance *a priori* sur les formes à reconnaître. En effet, dans notre approche le classifieur possède déjà les connaissances omni-scripteurs sur ce qu'il doit reconnaître, mais est-il possible de se passer de ces connaissances ? En partant d'un SIF ne possédant initialement aucune règle et en utilisant notre approche de création de nouvelles règles avec la synthèse de caractères, ce problème pourrait être résolu dans un premier temps. Lorsqu'une nouvelle classe est détectée ou demandée par l'utilisateur, une nouvelle règle est ajoutée, puis l'adaptation ajustera les prototypes et les conclusions pour optimiser les performances. Par cette approche nous pouvons espérer atteindre des résultats au moins équivalents à ceux

présentés dans la section 5 sur la synthèse de caractères. Encore une fois, les résultats de cet apprentissage à partir de très peu de données seront très liés à l'efficacité de nos stratégies de synthèse de caractères pour générer des caractères différents tout en respectant le style du scripteur.

Notre approche pour l'adaptation est basée sur l'interprétation des connaissances d'un SIF, il serait donc très intéressant d'appliquer ADAPT à d'autres types de SIF plus complexes comme ceux utilisés dans le système de reconnaissance *RésifCar* [4].

Enfin, un exemple d'utilisation conjointe de l'adaptation et du rejet a été montré dans l'utilisation du rejet pour la création de nouvelles règles. Un autre cas n'a pas été étudié dans ce manuscrit : comment adapter un classifieur qui possède une option de rejet ? Dans ce contexte il faudra d'un côté adapter l'option de rejet à l'utilisateur et de l'autre côté répercuter les modifications du classifieur principal sur l'option de rejet.

Annexes

Glossaire

AD	: Dissimilarité analogique
ADAPT	: Adaptation par ajustement de prototypes
ADS	: Dissimilarité analogique entre quatre séquences
AP	: Proportion analogique
AMTL	: Algorithme d'apprentissage automatique de seuils multiples
AUROCC	: Aire sous la courbe ROC
AUERC	: Aire sous la courbe ER
CMP	: Apprentissage du rejet par maximisation sous contrainte
DERIV	: Adaptation des prémisses d'un SIF par descente de gradient
DG	: Adaptation des conclusions d'un SIF par descente de gradient
DP	: Programmation dynamique
DTW	: Alignement élastique de séquences, *Dynamic Time Warping*
ER	: Courbe Erreur/Rejet
ERR5	: Taux d'erreur à 5% de rejet
FAR	: Taux de fausse acceptation
FLVQ	: Adaptation des prémisses d'un SIF par *Fuzzy learning vector quantization*
FRR	: Taux de faux rejet
IREMA	: Interface de Reconnaissance d'Écriture avec Mécanismes d'Adaptation
k-PPV	: Classifieur k plus proches voisins
LVQ	: Adaptation des prémisses d'un SIF par *Learning vector quantization*
MLP	: Perceptron multi-couche
NOP	: Nombre de points opérationnels
PDA	: Assistant personnel électronique
PSO	: Optimisation à base de nuée de particules
RBFN	: Réseau de neurones à fonction à base radiale

RC : Option de rejet utilisant une classe de rejet

ROC : Courbe TAR/FAR

RR10/RR30 : Taux de reconnaissances de référence sur 10 ou 30 caractères par classe

SC : Option de rejet utilisant un classifieur spécialisé

SCRF : Option de rejet utilisant un classifieur spécialisé et des fonctions de confiance

SVM : Système à vaste marge

SIF : Système d'inférence floue

TAR : Taux de vraie acceptation

TGD : Apprentissage du rejet par descente du gradient de l'erreur

TRR : Taux de vrai rejet

TRF : Option de rejet utilisant des seuils sur des fonctions de confiance

Annexe A

Résultats complets du rejet

A.1 Résultats du rejet d'ambiguïté sur la base artificielle

TAB. A.1 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les RBFN avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.248±0.035	1.45± 0.23	103
TRF (AMTL1, ψ_{1}^{Amb})	0.242±0.023	1.38± 0.29	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	0.236±0.032	1.37± 0.23	103
TRF (AMTL1, ψ_{i}^{AmbCl})	0.240±0.031	1.35± 0.27	103
TRF (AMTL1, $\psi_{i}^{DistRBF}$)	0.652± 0.10	2.69± 0.40	103
TRF (AMTL1, ψ_{i}^{Dist})	0.697± 0.15	2.67± 0.41	103
TRF (AMTL1, ψ_{C1}^{Dist})	0.597± 0.10	2.65± 0.42	103
TRF (AMTL1, $\psi_{i}^{DistNorm}$)	0.264±0.038	1.48± 0.35	103
TRF (AMTL1, $\psi^{DistRbfW}$)	0.618± 0.11	2.69± 0.43	103
TRF (AMTL1, $\psi_{i}^{DistNRBF}$)	0.600± 0.10	2.63± 0.38	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.588± 0.10	2.62± 0.41	103
TRF (CMP, $\psi_{i,j}^{Amb}$)	0.433±0.045	1.58± 0.21	103
TRF (CMP, ψ_{1}^{Amb})	0.232± 0.03	1.35± 0.29	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	0.305±0.029	1.48± 0.26	103
TRF (CMP, ψ_{i}^{AmbCl})	0.361±0.098	1.51± 0.36	103
TRF (CMP, $\psi_{i}^{DistRBF}$)	1.617± 0.21	3.07± 0.4	103
TRF (CMP, ψ_{i}^{Dist})	1.617± 0.21	3.07± 0.4	103
TRF (CMP, ψ_{i}^{Fumera})	1.030± 0.35	2.67± 0.5	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	0.36 ± 0.09	1.34± 0.26	41
TRF (PSO, ψ_{1}^{Amb})	0.276±0.047	1.35± 0.3	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	0.294±0.043	1.33± 0.26	41
TRF (PSO, ψ_{i}^{AmbCl})	0.355±0.072	1.3 ± 0.26	41
TRF (TGD, $\psi_{i,j}^{Amb}$)	0.266±0.046	1.31± 0.26	83
TRF (TGD, ψ_{1}^{Amb})	0.274±0.053	1.37± 0.30	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	0.266±0.041	1.35± 0.28	83
TRF (TGD, ψ_{i}^{AmbCl})	0.286±0.061	1.34± 0.29	83

TAB. A.2 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les MLP avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.225±0.044	1.41± 0.32	103
TRF (AMTL1, ψ_{1}^{Amb})	0.217±0.027	1.31± 0.25	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	0.218±0.033	1.38± 0.21	103
TRF (AMTL1, ψ_{i}^{Dist})	0.224±0.038	1.46± 0.26	103
TRF (AMTL1, ψ_{C1}^{Dist})	0.239±0.038	1.39± 0.25	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.214±0.030	1.46± 0.29	103
TRF (TGD, $\psi_{i,j}^{Amb}$)	0.269±0.055	1.34± 0.2	83
TRF (TGD, ψ_{1}^{Amb})	0.262±0.049	1.34± 0.22	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	0.259±0.041	1.37± 0.22	83
TRF (CMP, $\psi_{i,j}^{Amb}$)	0.579± 0.12	2.01± 0.23	103
TRF (CMP, ψ_{1}^{Amb})	0.225±0.026	1.31± 0.24	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	0.293±0.034	1.62± 0.16	103
TRF (CMP, ψ_{i}^{Fumera})	1.61 ± 0.19	3.05± 0.36	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	0.383± 0.13	1.31± 0.25	41
TRF (PSO, ψ_{1}^{Amb})	0.293±0.059	1.32± 0.23	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	0.358±0.081	1.29± 0.24	41

TAB. A.3 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les SVM avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.638± 0.1	1.53± 0.27	103
TRF (AMTL1, ψ_{1}^{Amb})	0.643± 0.13	1.41± 0.35	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	0.629± 0.13	1.60± 0.29	103
TRF (AMTL1, ψ_{i}^{Dist})	0.683± 0.19	2.03± 0.63	103
TRF (AMTL1, ψ_{C1}^{Dist})	0.623± 0.18	2.00± 0.61	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.653± 0.18	1.86± 0.38	103
TRF (TGD, $\psi_{i,j}^{Amb}$)	0.498± 0.11	1.39± 0.32	83
TRF (TGD, ψ_{1}^{Amb})	0.678± 0.14	1.40± 0.35	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	0.512± 0.093	1.40± 0.32	83
TRF (CMP, $\psi_{i,j}^{Amb}$)	0.738± 0.18	2.11± 0.36	103
TRF (CMP, ψ_{1}^{Amb})	0.667± 0.12	1.42± 0.35	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	0.602± 0.11	1.61± 0.24	103
TRF (CMP, ψ_{i}^{Fumera})	1.65 ± 0.27	3.14± 0.51	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	0.627± 0.13	1.38± 0.29	41
TRF (PSO, ψ_{1}^{Amb})	0.681± 0.14	1.4 ± 0.35	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	0.59 ± 0.15	1.36± 0.34	41

TAB. A.4 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture RC

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
RC (RBFN)	0.225±0.033	1.57± 0.19	103
RC (MLP)	0.202±0.032	1.4 ± 0.25	103
RC (SVM)	0.225±0.046	1.6 ± 0.29	103

TAB. A.5 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture SC sur un RBFN

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
SC (RBFN)	0.264±0.058	2.13± 0.39	103
SC (MLP)	0.257±0.046	1.73± 0.2	103
SC (SVM)	0.234±0.038	1.79± 0.28	103

TAB. A.6 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture SCRF sur un RBFN avec les fonctions de confiance $\psi_{i,j}^{Amb}$.

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
SCRF (RBFN)	0.216±0.028	1.29± 0.27	103
SCRF (MLP)	0.279±0.051	1.32± 0.19	103
SCRF (SVM)	0.222±0.038	1.30± 0.26	103

A.2 Résultats du rejet de distance sur la base artificielle

TAB. A.7 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture TRF sur un RBFN.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, $\psi_i^{DistRBF}$	90.5±0.49	92.3±0.39	91.3±0.17	103
AMTL1, ψ_i^{Dist}	91.0±0.48	91.8±0.53	91.1±0.15	103
AMTL1, ψ_{C1}^{Dist}	89.3±0.77	95.1±0.21	92.2±0.28	103
AMTL1, $\psi_i^{DistNorm}$	94.5±0.36	73.6± 2.1	83.9± 1	103
AMTL1, $\psi^{DistRbfW}$	89.2±0.72	95.1±0.25	92.2±0.25	103
AMTL1, $\psi_i^{DistNRBF}$	90.8±0.63	92.2± 1.1	91.2±0.59	103
AMTL1, ψ_i^{Fumera}	91 ±0.56	92.7±0.79	91.7±0.34	103
AMTL1, $\psi_{i,j}^{Amb}$	94.3±0.27	69.5± 1.9	81.7± 1	103
AMTL1, $\psi_{\{i,j\}}^{AmbSym}$	93.4±0.52	81.1±0.66	87.1±0.48	103
AMTL2, $\psi_i^{DistRBF}$	89.4±0.70	95.0±0.23	92.2±0.23	103
AMTL2, ψ_i^{Dist}	89.5±0.79	94.9± 0.2	92.2±0.27	103
AMTL2, ψ_{C1}^{Dist}	89.3±0.77	95.1±0.21	92.2±0.28	103
AMTL2, ψ_i^{Fumera})	89.2±0.86	95.1±0.28	92.1±0.34	103
TGD, $\psi_i^{DistRBF}$	90.6±0.55	91.5±0.54	91.1±0.22	103
TGD, ψ_i^{Dist}	91.0±0.57	90.5±0.69	90.8±0.25	103
TGD, ψ_{C1}^{Dist}	89.0±0.90	95.1±0.22	92.1±0.28	103
TGD, $\psi_i^{DistNRBF}$	91.0±0.43	88.3±0.95	89.4±0.39	103
TGD, ψ_i^{Fumera}	91.2±0.53	88.4±0.87	89.6±0.41	103
CMP, ψ_{C1}^{Dist}	89.3±0.78	95 ±0.28	92.2±0.32	103
CMP, $\psi^{DistRbfW}$	89.2±0.74	95.1±0.26	92.2±0.26	103
CMP, $\psi_i^{DistNRBF}$	91 ±0.24	85.8±0.28	87.4±0.17	103
CMP, ψ_i^{Fumera}	90.3±0.39	72.5± 0.3	78.8±0.35	103
PSO, $\psi_i^{DistRBF}$	90.1±0.46	94.2± 1	91.3±0.25	43
PSO, ψ_i^{Dist}	89.7±0.97	94.1±0.55	90.9±0.93	43
PSO, ψ_{C1}^{Dist}	88.8± 1	94.9±0.36	91.9±0.43	43
PSO, ψ_i^{Fumera}	89.9±0.46	87.9±0.91	86.8± 2.2	43

TAB. A.8 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture TRF sur un MLP.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, ψ_i^{Dist}	81.9±0.78	51.6±0.93	65.4±0.69	103
AMTL1, ψ_{C1}^{Dist}	68.8± 1.6	62.4±0.67	64.8±0.57	103
AMTL1, ψ_i^{Fumera}	82.2±0.31	58.9± 1.4	67.6±0.51	103
AMTL2, ψ_i^{Dist}	77 ±0.93	61.9± 0.7	69 ±0.69	103
AMTL2, ψ_{C1}^{Dist}	68.8± 1.5	62.4±0.65	64.8±0.55	103
AMTL2, ψ_i^{Fumera}	74.9± 1.2	61.9± 1	68.1±0.87	103
TGD, ψ_i^{Dist}	81.2±0.55	54.3±0.27	66.2±0.62	103
TGD, ψ_{C1}^{Dist}	67.4± 1.7	58.8±0.68	63.2±0.69	103
TGD, ψ_i^{Fumera}	80.1±0.51	55.2± 0.5	66.2±0.25	103
CMP, ψ_{C1}^{Dist}	50 ± 0	50 ± 0	50 ± 0	103
CMP, ψ_i^{Fumera}	50 ± 0	50 ± 0	50 ± 0	103
PSO, ψ_i^{Dist}	84.9±0.46	57 ± 1	68.7±0.58	43
PSO, ψ_{C1}^{Dist}	68.6± 1.6	59.5±0.55	63.9±0.47	43
PSO, ψ_i^{Fumera}	82.4±0.28	58.4±0.19	68.6±0.33	43

TAB. A.9 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture TRF sur un SVM (la mention *SM* signifie que *SoftMax* est utilisée).

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, ψ_i^{Dist}	80.0± 1.9	64.4± 0.5	70.2± 0.8	103
AMTL1, ψ_{C1}^{Dist}	69.4± 2.6	77.8± 2.8	72.7± 1.3	103
AMTL1, ψ_i^{Fumera}	79.8± 1.9	64.7±0.55	70.2±0.86	103
(SM) AMTL1, ψ_i^{Dist}	78 ± 1.7	60.6± 1	67.6± 1.6	103
(SM) AMTL1, ψ_{C1}^{Dist}	68.8± 1.3	81.8± 1.6	74.5± 1.1	103
(SM) AMTL1, ψ_i^{Fumera}	78.2± 1.0	65.6± 1.4	69.9± 1.2	103
AMTL2, ψ_i^{Dist}	69.5± 2.4	77.4± 2.8	72.5± 1.4	103
AMTL2, ψ_{C1}^{Dist}	69.4± 2.6	77.8± 2.9	72.7± 1.4	103
AMTL2, ψ_i^{Fumera}	67.9± 1.6	76.1± 2.6	70.7± 1.7	103
(SM) AMTL2, ψ_i^{Dist}	69.2± 1.2	81.4± 1.4	74.3± 1.1	103
(SM) AMTL2, ψ_{C1}^{Dist}	68.8± 1.3	81.8± 1.6	74.5± 1.1	103
(SM) AMTL2, ψ_i^{Fumera}	67.8±0.71	80.3± 1.2	72.3± 1.4	103
TGD, ψ_i^{Dist}	81.5± 1.2	63.5± 2.8	70.6± 1.4	103
TGD, ψ_{C1}^{Dist}	69.2± 2.5	74.3± 1.4	71.4± 1.3	103
TGD, ψ_i^{Fumera}	81.2± 1.1	69.3± 2.3	73.4± 1.1	103
(SM) TGD, ψ_i^{Dist}	82.2± 2.9	52.2±0.67	64.9± 1.4	103
(SM) TGD, ψ_{C1}^{Dist}	72.1± 5.7	53.6± 1.8	61.4± 3.4	103
(SM) TGD, ψ_i^{Fumera}	81.2± 4	53.3±0.47	64.9± 1.4	103
CMP, ψ_i^{Dist}	80.7± 1.5	63.6± 2.4	69.7± 2.5	103
CMP, ψ_{C1}^{Dist}	69.4± 2.6	77.8± 2.9	72.7± 1.3	103
CMP, ψ_i^{Fumera}	80.7± 1.5	63.8± 2.1	69.7± 2.4	103
(SM) CMP, ψ_{C1}^{Dist}	50 ± 0	50 ± 0	50 ± 0	103
(SM) CMP, ψ_i^{Fumera}	50 ± 0	50 ± 0	50 ± 0	103
PSO, ψ_i^{Dist}	67.7± 1.3	65 ± 1.1	66.1±0.62	43
PSO, ψ_{C1}^{Dist}	66.7± 1.3	70.3± 1.8	68.3± 1.4	43
PSO, ψ_i^{Fumera}	67.7± 1.3	65.6± 1.3	66 ±0.67	43
(SM) PSO, ψ_i^{Dist}	81.3±0.56	65.6± 2	71 ±0.98	43
(SM) PSO, ψ_{C1}^{Dist}	68.7± 1.1	80.4± 1.5	73.7± 1.4	43
(SM) PSO, ψ_i^{Fumera}	81.1±0.61	67.6± 1.8	70.8± 0.9	43

TAB. A.10 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture RC.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
RC (RBFN)	97.1±0.17	50.0± 0	70.0±0.15	103
RC (MLP)	97.1±0.21	59.9±4.8	77.8± 2.3	103
RC (SVM)	97.0±0.21	59.3±7.5	76.8± 3.2	103

TAB. A.11 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture SC.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
SC (RBFN)	97.1±0.29	51.8±3.1	71.4± 2	103
SC (MLP)	96.8± 0.3	60.3±4.5	76.2±2.4	103
SC (SVM)	97.1±0.23	55.1±5.1	73.7±3.5	103

TAB. A.12 – AUROCC et NOP pour le rejet de distance sur les données artificielles avec l'architecture SCRF sur un RBFN avec $\psi_i^{DistRBF}$.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
SCRF (RBFN)	94.1±0.38	81 ±3.4	86.9± 1.2	53
SCRF (MLP)	95.7±0.22	72.6±3.6	83.3± 1.4	53
SCRF (SVM)	96.1±0.18	82.1±1.3	88.4±0.62	53

A.3 Résultats du rejet d'ambiguïté sur la base réelle

TAB. A.13 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les RBFN avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.643±0.069	3.84± 0.47	103
TRF (AMTL1, ψ_{1}^{Amb})	0.682± 0.06	4.08± 0.4	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	0.671±0.078	4.03± 0.47	103
TRF (AMTL1, ψ_{i}^{AmbCl})	0.662±0.065	4.06± 0.42	103
TRF (AMTL1, ψ_{i}^{Dist})	1.07 ± 0.12	4.43± 0.52	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.931±0.069	4.22± 0.33	103
TRF (TGD, $\psi_{i,j}^{Amb}$)	0.738±0.046	4.46± 0.41	83
TRF (TGD, ψ_{1}^{Amb})	0.766± 0.06	4.07± 0.35	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	0.727±0.048	4.37± 0.46	83
TRF (CMP, $\psi_{i,j}^{Amb}$)	1.75 ± 0.21	4.76± 0.37	103
TRF (CMP, ψ_{1}^{Amb})	0.68 ±0.058	4.05± 0.36	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	1.69 ± 0.24	4.76± 0.35	103
TRF (CMP, ψ_{i}^{AmbCl})	1.05 ±0.083	4.21± 0.45	103
TRF (CMP, ψ_{i}^{Fumera})	1.01 ±0.072	4.47± 0.32	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	0.942±0.058	4.14± 0.58	41
TRF (PSO, ψ_{1}^{Amb})	0.746±0.057	4.09± 0.38	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	0.872±0.044	4.1 ± 0.41	41
TRF (PSO, ψ_{i}^{AmbCl})	0.794±0.059	4.01± 0.42	41

TAB. A.14 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les MLP avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	1.15 ± 0.18	4.81± 0.52	103
TRF (AMTL1, ψ_{1}^{Amb})	1.14 ± 0.19	4.86± 0.51	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	1.13 ± 0.13	4.78± 0.46	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.655±0.053	3.66± 0.45	103
TRF (TGD, $\psi_{i,j}^{Amb}$)	1.25 ± 0.14	5.42± 0.58	83
TRF (TGD, ψ_{1}^{Amb})	1.23 ± 0.21	4.88± 0.52	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	1.14 ± 0.13	4.72± 0.51	83
TRF (CMP, $\psi_{i,j}^{Amb}$)	2.89 ± 0.34	6.12± 0.62	103
TRF (CMP, ψ_{1}^{Amb})	1.24 ± 0.21	4.81± 0.52	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	2.62 ± 0.38	6.05± 0.67	103
TRF (CMP, ψ_{i}^{Fumera})	0.909± 0.1	3.66± 0.37	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	1.51 ± 0.12	4.72± 0.51	41
TRF (PSO, ψ_{1}^{Amb})	1.27 ± 0.21	4.91± 0.51	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	1.35 ± 0.15	4.75± 0.55	41

TAB. A.15 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté sur les SVM (avec normalisation *soft max*) avec l'architecture TRF

Option de Rejet	100*AUERC	100*ERR5(%)	NOP
TRF (AMTL1, $\psi_{i,j}^{Amb}$)	0.507±0.026	2.47± 0.13	103
TRF (AMTL1, ψ_{1}^{Amb})	0.517±0.057	2.37± 0.073	103
TRF (AMTL1, $\psi_{\{i,j\}}^{AmbSym}$)	0.501±0.030	2.53± 0.19	103
TRF (AMTL1, ψ_{i}^{Dist})	0.733±0.089	2.7 ± 0.18	103
TRF (AMTL1, ψ_{i}^{Fumera})	0.617±0.065	2.64± 0.055	103
TRF (TGD, $\psi_{i,j}^{Amb}$)	0.6 ± 0.05	2.49± 0.17	83
TRF (TGD, ψ_{1}^{Amb})	0.555±0.075	2.4 ± 0.11	83
TRF (TGD, $\psi_{\{i,j\}}^{AmbSym}$)	0.534±0.045	2.33± 0.1	83
TRF (CMP, $\psi_{i,j}^{Amb}$)	1.55 ± 0.13	3.39± 0.17	103
TRF (CMP, ψ_{1}^{Amb})	0.522±0.057	2.35± 0.08	103
TRF (CMP, $\psi_{\{i,j\}}^{AmbSym}$)	1.32 ± 0.18	3.22± 0.21	103
TRF (CMP, ψ_{i}^{Fumera})	1.82 ± 0.18	3.43± 0.34	103
TRF (PSO, $\psi_{i,j}^{Amb}$)	0.927±0.025	2.4 ± 0.16	41
TRF (PSO, ψ_{1}^{Amb})	0.568±0.067	2.3 ± 0.078	41
TRF (PSO, $\psi_{\{i,j\}}^{AmbSym}$)	0.865±0.039	2.37± 0.13	41

TAB. A.16 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture SC

Principal	Rejet	100*AUERC	100*ERR5(%)	NOP
RBFN	RBFN	1.26 ± 0.12	4.92± 0.34	103
RBFN	MLP	1.27 ± 0.13	4.93± 0.36	103
RBFN	SVM	0.969± 0.087	4.19± 0.31	103
MLP	RBFN	1.26 ± 0.1	4.59± 0.43	49
MLP	MLP	1.18 ± 0.11	4.38± 0.49	53
MLP	SVM	0.966± 0.12	3.77± 0.43	103
SVM	RBFN	1.54 ± 0.1	3.76± 0.25	67.75
SVM	MLP	1.36 ±0.079	3.78± 0.23	53
SVM	SVM	1.29 ± 0.056	3.46± 0.14	103

TAB. A.17 – AUERC, ERR5 et NOP pour le rejet d'ambiguïté avec l'architecture SCRF.

Principal	Option de Rejet	100*AUERC	100*ERR5(%)	NOP
RBFN	SCRF (RBFN sur ψ_i^{AmbCl})	0.831±0.076	4.02± 0.38	92.5
RBFN	SCRF (RBFN sur ψ_1^{Amb})	0.679±0.059	4.03± 0.36	103
RBFN	SCRF (MLP sur $\psi_{\{i,j\}}^{AmbSym}$)	1.63 ± 0.33	5.81± 0.48	53
RBFN	SCRF (MLP sur ψ_i^{AmbCl})	0.719± 0.07	4.52± 0.32	103
RBFN	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	0.598±0.046	3.71± 0.41	103
RBFN	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	0.638±0.046	3.91± 0.42	53
RBFN	SCRF (SVM sur ψ_i^{AmbCl})	0.624±0.065	3.94± 0.4	103
SVM	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	0.579±0.026	2.57± 0.049	53
SVM	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	0.572±0.035	2.59± 0.25	53
MLP	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	1.25 ±0.096	4.57± 0.44	53
MLP	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	1.25 ± 0.09	4.71± 0.47	53

A.4 Résultats du rejet de distance sur la base réelle

TAB. A.18 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture TRF sur un RBFN.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, $\psi_i^{DistRBF}$	92.6±0.069	95 ± 0.28	92.1±0.19	103
AMTL1, ψ_i^{Dist}	90.6±0.068	94.3± 0.15	90 ±0.29	103
AMTL1, ψ_{C1}^{Dist}	88.1± 0.40	94.6± 0.27	88.3±0.32	103
AMTL1, $\psi_i^{DistNorm}$	89.4± 0.50	91.6± 0.39	89.4±0.35	103
AMTL1, $\psi^{DistRbfW}$	85.3± 0.61	94.2± 0.32	84.9±0.33	103
AMTL1, $\psi_i^{DistNRBF}$	92.3±0.081	94.5± 0.50	91.6±0.30	103
AMTL1, ψ_i^{Fumera}	90.6± 0.11	94.3± 0.26	90 ±0.38	103
AMTL1, $\psi_{i,j}^{Amb}$	89.1± 0.68	89.9± 0.78	88.5±0.68	103
AMTL1, $\psi_{\{i,j\}}^{AmbSym}$	87.7± 0.52	90.2± 0.76	87.7±0.64	103
AMTL2, $\psi_i^{DistRBF}$	85.2± 0.63	94.1± 0.31	84.8±0.35	103
AMTL2, ψ_i^{Dist}	88.1± 0.38	94.6± 0.27	88.3±0.31	103
AMTL2, ψ_{C1}^{Dist}	88.1± 0.41	94.6± 0.27	88.3±0.32	103
AMTL2, $\psi_i^{DistNorm}$	86.3± 0.53	92.3± 0.50	86.8±0.48	103
AMTL2, $\psi^{DistRbfW}$	85.3± 0.60	94.2± 0.31	84.9±0.32	103
AMTL2, $\psi_i^{DistNRBF}$	85.8± 0.32	94.3± 0.34	85.5±0.32	103
AMTL2, ψ_i^{Fumera})	87.9± 0.36	94.9± 0.26	88.1±0.31	103
TGD, $\psi_i^{DistRBF}$	84.3± 1	87.4± 1.2	83.6±0.88	103
TGD, ψ_i^{Dist}	90.5±0.064	94.2±0.082	89.8±0.33	103
TGD, ψ_{C1}^{Dist}	87.9± 0.39	94.3± 0.22	88 ±0.27	103
TGD, ψ_i^{Fumera}	90.4± 0.16	94.1± 0.35	89.6±0.45	53
CMP, ψ_i^{Fumera}	89.6± 0.11	93.3± 0.38	89.1±0.32	53
PSO, $\psi_i^{DistRBF}$	85.2± 0.37	89.7± 0.52	85.3±0.35	43
PSO, ψ_i^{Dist}	88.5± 0.17	93.6± 0.28	87.7±0.24	43
PSO, ψ_{C1}^{Dist}	87.9± 0.42	94.4± 0.23	88 ± 0.3	43
PSO, ψ_i^{Fumera}	88.5± 0.15	93.3± 0.41	87.7±0.15	43

TAB. A.19 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture TRF sur un MLP.

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, ψ_i^{Dist}	80.9±0.14	82.3±2.5	81.1±0.87	103
AMTL1, ψ_{C1}^{Dist}	75.5±0.73	74.2±3.2	75.6± 1.2	103
AMTL1, ψ_i^{Fumera}	80.3± 0.4	82.2±1.9	80.8±0.48	103
AMTL1, $\psi_{i,j}^{Amb}$	82.1± 1.5	83.7±2.1	81.7± 1.8	103
AMTL1, $\psi_{\{i,j\}}^{AmbSym}$	79.5±0.53	82.6±1.4	80.5± 0.4	103
AMTL2, ψ_i^{Dist}	75.4±0.78	74.3±3.0	75.6± 1.0	103
AMTL2, ψ_{C1}^{Dist}	75.5±0.73	74.2±3.2	75.6± 1.2	103
AMTL2, ψ_i^{Fumera}	76.1±0.62	74.8±2.7	76.1± 0.5	103
AMTL2, $\psi_{i,j}^{Amb}$	73.6±0.32	78 ±1.3	74.1±0.42	103
TGD, ψ_i^{Dist}	79.1±0.69	82.8± 2	80.3±0.46	103
TGD, ψ_{C1}^{Dist}	74 ± 1.2	72.5±2.6	74.3±0.61	103
TGD, ψ_i^{Fumera}	79.1±0.81	82.6±2.1	80.2±0.42	103
TGD, $\psi_{i,j}^{Amb}$	79.4± 1	81.4±1.6	79.1± 1.5	53
TGD, $\psi_{\{i,j\}}^{AmbSym}$	77.9±0.38	81.4±2.3	78.2± 1.2	53
CMP, ψ_i^{Fumera}	73.5± 2.3	77.8±3.9	74.7± 2.7	53
PSO, ψ_i^{Dist}	81.2±0.61	83.7± 2	81.5±0.55	43
PSO, ψ_{C1}^{Dist}	75.2±0.58	74 ± 3	75.4± 1	43
PSO, ψ_i^{Fumera}	80.3±0.75	83.4±1.6	81 ±0.58	43
PSO, $\psi_{i,j}^{Amb}$	78.3± 1	82 ±1.7	77.8±0.73	43
PSO, $\psi_{\{i,j\}}^{AmbSym}$	77.8±0.63	82.9±2.4	78 ± 1	43

TAB. A.20 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture TRF sur un SVM (la normalisation *SoftMax* n'est pas utilisée).

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
AMTL1, ψ_i^{Dist}	83.9±0.82	86 ± 2.0	84.2± 1.4	103
AMTL1, ψ_{C1}^{Dist}	81 ± 1.0	84.8± 2.5	82.4± 1.4	103
AMTL1, ψ_i^{Fumera}	83.8± 1.1	85.8± 2.1	84 ± 1.8	103
AMTL1, $\psi_{\{i,j\}}^{AmbSym}$	77.8±0.55	87 ± 1.4	79 ±0.89	103
AMTL2, ψ_i^{Dist}	80.9± 0.8	85.7± 2.5	82.4± 1.3	103
AMTL2, ψ_{C1}^{Dist}	81 ± 1	84.8± 2.5	82.4± 1.4	103
AMTL2, ψ_i^{Fumera}	80.8±0.79	85.4± 2.4	82.4± 1.1	103
AMTL2, $\psi_{i,j}^{Amb}$	72.9±0.44	77.3± 1	75.4±0.64	103
AMTL2, $\psi_{\{i,j\}}^{AmbSym}$	71.9±0.38	76.3± 1.1	74 ±0.75	103
TGD, ψ_i^{Dist}	83.8±0.52	88.7±0.93	84.3± 0.9	53
TGD, ψ_{C1}^{Dist}	80.9± 1	84.7± 2.5	82.3± 1.4	53
TGD, ψ_i^{Fumera}	83.5±0.36	87.5±0.84	83.8± 0.79	53
TGD, $\psi_{i,j}^{Amb}$	64.4± 2	70 ± 3.6	66 ± 1.8	53
TGD, $\psi_{\{i,j\}}^{AmbSym}$	66.2± 2	70.9± 2.5	68.1± 2.1	53
CMP, ψ_i^{Fumera}	83.5± 0.5	87.7± 1.2	83.7± 0.92	53
PSO, ψ_i^{Dist}	78.9±0.85	83.6± 2.5	80.7± 1.3	43
PSO, ψ_{C1}^{Dist}	78.6±0.78	83.1± 2.5	80.6± 1.3	43
PSO, ψ_i^{Fumera}	78.9±0.83	83.6± 2.5	80.8± 1.3	43
PSO, $\psi_{i,j}^{Amb}$	69.5±0.55	73.9± 2.4	72.1± 1.2	43
PSO, $\psi_{\{i,j\}}^{AmbSym}$	69.5±0.57	73.9± 2.4	72.1± 1.2	43

TAB. A.21 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture TRF sur un SVM avec la normalisation *SoftMax* (SM).

Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
(SM) AMTL1, ψ_i^{Dist}	85.1±0.46	86.7±0.85	85.3±0.83	103
(SM) AMTL1, ψ_{C1}^{Dist}	81.6± 0.6	86.3± 2	82.8±0.92	103
(SM) AMTL1, ψ_i^{Fumera}	84.9±0.49	86.5±0.97	85.2±0.79	103
(SM) AMTL1, $\psi_{i,j}^{Amb}$	87.9±0.24	89.3±0.45	88.1±0.25	103
(SM) AMTL1, $\psi_{\{i,j\}}^{AmbSym}$	87.7±0.52	90.2±0.76	87.7±0.64	103
(SM) AMTL2, ψ_i^{Dist}	81.5±0.55	86.5± 2	82.7±0.96	103
(SM) AMTL2, ψ_{C1}^{Dist}	81.6± 0.6	86.3± 2	82.8±0.91	103
(SM) AMTL2, ψ_i^{Fumera}	81.6±0.46	86.3± 2.1	82.7±0.81	103
(SM) AMTL2, $\psi_{i,j}^{Amb}$	83.5±0.25	86.6± 1.1	84.5±0.59	103
(SM) TGD, ψ_i^{Dist}	85.1±0.16	88.6± 0.4	85.1±0.67	53
(SM) TGD, ψ_{C1}^{Dist}	81.3±0.63	85.8± 2.1	82.4±0.99	53
(SM) TGD, ψ_i^{Fumera}	85.4±0.16	88.6±0.69	85.5±0.67	53
(SM) TGD, $\psi_{i,j}^{Amb}$	87.3±0.29	89.7±0.37	87.7±0.17	53
(SM) TGD, $\psi_{\{i,j\}}^{AmbSym}$	86.7±0.19	89.9±0.55	87.2±0.38	53
(SM) CMP, ψ_i^{Fumera}	72.3± 1.2	85.6± 1.9	76.9±0.62	53
(SM) PSO, ψ_i^{Dist}	84.2±0.37	89.5±0.75	84.7±0.91	43
(SM) PSO, ψ_{C1}^{Dist}	81.3±0.62	86.2± 2.4	82.6±0.98	43
(SM) PSO, ψ_i^{Fumera}	84.4±0.074	89.5± 1	84.9±0.97	43
(SM) PSO, $\psi_{i,j}^{Amb}$	84.5±0.21	89.1±0.61	85.5±0.23	43
(SM) PSO, $\psi_{\{i,j\}}^{AmbSym}$	84.9±0.22	90.2±0.93	86 ±0.66	43

TAB. A.22 – AUROCC et NOP pour le rejet de distance sur les données réelles avec l'architecture SCRF.

Principal	Option de rejet	$A \rightarrow A$	$A \rightarrow B$	$A \rightarrow AB$	NOP
RBFN	SCRF (RBFN sur $\psi_i^{DistRBF}$)	88.5± 0.47	90.8±0.32	87.9±0.29	53
RBFN	SCRF (RBFN sur ψ_i^{Dist})	87.6± 0.14	92.2±0.51	86.7±0.51	53
RBFN	SCRF (MLP sur $\psi_i^{DistRBF}$)	67.7± 4.2	75 ± 3.2	69.7± 3.4	53
RBFN	SCRF (MLP sur ψ_i^{Dist})	61.2± 1.2	60.9± 1.2	61 ± 1.1	53
RBFN	SCRF (SVM sur $\psi_i^{DistRBF}$)	92.1± 0.18	94.6±0.18	91.5± 0.3	53
RBFN	SCRF (SVM sur ψ_i^{Dist})	95 ± 0.15	95.6±0.094	93.3±0.22	53
MLP	SCRF (SVM sur ψ_i^{Dist})	84.7± 1.1	85.9± 2	84.6± 1.6	53
MLP	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	81.2± 0.99	83.3± 1.4	80.8± 1.5	53
MLP	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	78.6± 0.38	82.3± 2	78.8± 1.1	53
SVM	SCRF (SVM sur ψ_i^{Dist})	93.5± 0.17	94.1±0.15	91.9±0.22	53
SVM	SCRF (SVM sur $\psi_{i,j}^{Amb}$)	86.8±0.075	90.1±0.28	87.3±0.25	53
SVM	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	85.6±0.032	89.7±0.65	86.3±0.35	53
SVM (SM)	SCRF (SVM sur $\psi_{\{i,j\}}^{AmbSym}$)	87 ± 0.27	90 ±0.32	87.4±0.41	53

Annexe B

Démonstrations pour l'adaptation

B.1 Descente du gradient de l'erreur quadratique

Soient :

N	nombre de classes
R	nombre de règles
$\vec{X}$	position de l'exemple
$\vec{\mu_i}$	centre du prototype i
$\vec{w_i}$	$= \vec{X} - \vec{\mu_i}$
Q_i	matrice de covariance du prototype i
$d_i = (\vec{X} - \vec{\mu_i})^T Q_i^{-1} (\vec{X} - \vec{\mu_i})$	distance de Mahalanobis de l'exemple X au prototype i
$\beta_i = \frac{1}{1+d_i}$	activation du prototype i par l'exemple X
s_c^i	score du prototype i pour la classe c
$s_c = \frac{\sum_r^R \beta_r s_c^r}{\sum_r^R \beta_r}$	score de la classe c
$b_c = 1$ ou 0	score objectif pour la classe c
E	Erreur quadratique

Pour le déplacement des centres on a :

$$E = \frac{1}{2}\sum_{c=1}^{N}(b_c - s_c)^2 \quad (B.1)$$

$$= \frac{1}{2}\sum_{c=1}^{N} E_c \quad (B.2)$$

$$\frac{\partial E}{\partial \vec{\mu_k}} = \frac{1}{2}\sum_{c=1}^{N}\frac{\partial E_c}{\partial \vec{\mu_k}} \quad (B.3)$$

$$\frac{\partial E_c}{\partial \vec{\mu_k}} = 2(b_c - s_c)\frac{\partial (b_c - s_c)}{\partial \mu_k} \quad (B.4)$$

$$= -2(b_c - s_c)\frac{\partial s_c}{\partial \mu_k} \quad (B.5)$$

$$\frac{\partial s_c}{\partial \vec{\mu_k}} = \frac{\partial\left(\frac{\sum_r^R \beta_r s_c^r}{\sum_r^R \beta_r}\right)}{\partial \vec{\mu_k}} \quad (B.6)$$

$$= \frac{\frac{\partial \sum_r^R \beta_r s_c^r}{\partial \vec{\mu_k}}(\sum_r^R \beta_r) - (\sum_r^R \beta_r s_c^r)\frac{\partial \sum_r^R \beta_r}{\partial \vec{\mu_k}}}{(\sum_r^R \beta_r)^2} \quad (B.7)$$

$$\frac{\partial \sum_r^R \beta_r s_c^r}{\partial \vec{\mu_k}} = \sum_r^R \frac{\partial \beta_r s_c^r}{\partial \vec{\mu_k}} \quad (B.8)$$

$$= s_c^k \frac{\partial \beta_k}{\partial \vec{\mu_k}} \quad (B.9)$$

$$= s_c^k \frac{\partial \frac{1}{1+d_k}}{\partial \vec{\mu_k}} \quad (B.10)$$

$$= -s_c^k \left(\frac{1}{1+d_k}\right)^2 \frac{\partial d_k}{\partial \vec{\mu_k}} \quad (B.11)$$

$$= -s_c^k \beta_k^2 \frac{\partial d_k}{\partial \vec{\mu_k}} \quad (B.12)$$

$$\frac{\partial d_k}{\partial \vec{\mu_k}} = \frac{\partial d_k}{\partial \vec{w_k}}\frac{\partial \vec{w_k}}{\partial \vec{\mu_k}} \text{ avec } \vec{w_k} = \vec{X} - \vec{\mu_k} \quad (B.13)$$

$$= -\frac{\partial (\vec{w_k}^T Q_k^{-1} \vec{w_k})}{\partial \vec{w_k}} \quad (B.14)$$

$$= -2\vec{w_k}^T Q_k^{-1} \quad (B.15)$$

Donc

$$\frac{\partial \sum_r^R \beta_r s_c^r}{\partial \vec{\mu_k}} = 2 s_c^k \beta_k^2 \vec{w_k}^T Q_k^{-1} \quad (B.16)$$

De même

$$\frac{\partial \sum_r^R \beta_r}{\partial \vec{\mu_k}} = 2\beta_k^2 \vec{w_k}^T Q_k^{-1} \quad \text{(B.17)}$$

Ensuite

$$\frac{\partial s_c}{\partial \vec{\mu_k}} = 2\beta_k^2 \left(\frac{s_c^k \sum_r^R \beta_r - \sum_r^R \beta_r s_c^r}{(\sum_r \beta_r)^2} \right) \vec{w_k}^T Q_k^{-1} \quad \text{(B.18)}$$

$$= 2\beta_k^2 \left(\frac{s_c^k - s_c}{\sum_r^R \beta_r} \right) \vec{w_k}^T Q_k^{-1} \quad \text{(B.19)}$$

$$\frac{\partial E_c}{\partial \vec{\mu_k}} = -4\beta_k^2 (b_c - s_c) \frac{s_c^k - s_c}{\sum_r^R \beta_r} \vec{w_k}^T Q_k^{-1} \quad \text{(B.20)}$$

$$\frac{\partial E}{\partial \vec{\mu_k}} = \frac{1}{2} \sum_{c=1}^{N} -4\beta_k^2 (b_c - s_c) \frac{s_c^k - s_c}{\sum_r^R \beta_r} \vec{w_k}^T Q_k^{-1} \quad \text{(B.21)}$$

$$\frac{\partial E}{\partial \vec{\mu_k}} = -2 \frac{\beta_k^2}{\sum_r^R \beta_r} \left(\sum_{c=1}^{N} (b_c - s_c)(s_c^k - s_c) \right) \vec{w_k}^T Q_k^{-1} \quad \text{(B.22)}$$

Donc

$$\Delta \vec{\mu_k} = -\lambda \frac{\partial E}{\partial \vec{\mu_k}} \quad \text{(B.23)}$$

$$\Delta \vec{\mu_k} = 2\lambda \frac{\beta_k^2}{\sum_r^R \beta_r} \left(\sum_{c=1}^{N} (b_c - s_c)(s_c^k - s_c) \right) \vec{w_k}^T Q_k^{-1} \quad \text{(B.24)}$$

Pour la modification des matrices de covariance, on obtient la même démonstration avec :

$$\frac{\partial d_k}{\partial Q_k^{-1}} = \frac{\partial \vec{w_k}^T Q_k^{-1} \vec{w_k}}{\partial Q_k^{-1}} \quad \text{(B.25)}$$

$$= \vec{w_k} \vec{w_k}^T \quad \text{(B.26)}$$

$$\text{donc } \frac{\partial E}{\partial Q_k^{-1}} = -2 \frac{\beta_k^2}{\sum_r^R \beta_r} \left(\sum_{c=1}^{N} (b_c - s_c)(s_c^k - s_c) \right) \vec{w_k} \vec{w_k}^T \quad \text{(B.27)}$$

Bibliographie

Références de l'auteur

[BMMA07] Sabri Bayoudh, Laurent Miclet, Harold Mouchère, and Eric Anquetil. Learning a classifier with very few examples : knowledg based and analogy generation of new exemples for character recognition. In *ECML*, pages 527–534, 2007.

[MA06a] Harold Mouchère and Éric Anquetil. A unified strategy to deal with different natures of reject. In *18th International Conference on Pattern Recognition (ICPR'06)*, pages 792–795, 2006.

[MA06b] Harold Mouchère and Eric Anquetil. Generalization capacity of handwritten outlier symbols rejection with neural network. In *Proceedings of the 10th International Workshop on Frontier in Handwriting Recognition (IWFHR'06), to be published*, pages 187–192, La Baule, France, October 2006.

[MA06c] Harold Mouchère and Eric Anquetil. Synthèse de caractères manuscrits en-ligne pour la reconnaissance de lŠécriture. In *Actes du Colloque International Francophone sur l'Ecrit et le Document (CIFED'06)*, pages 187–192, 2006.

[MAR04] Harold Mouchère, Éric Anquetil, and Nicolas Ragot. Étude des mécanismes d'adaptation pour l'optimisation de classifieurs flous dans le cadre de la reconnaissance d'écriture manuscrite. In *12es rencontres francophones sur la Logique Floue et ses Applications (LFA'04)*, Nantes, France, novembre 2004.

[MAR05a] Harold Mouchère, Éric Anquetil, and Nicolas Ragot. On-line writer adaptation for handwriting recognition using fuzzy inference systems. In Bob Werner, editor, *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 1075–1079, Seoul, Korea, August 2005. IEEE Computer Society.

[MAR05b] Harold Mouchère, Éric Anquetil, and Nicolas Ragot. Writer style adaptation of on-line handwriting recognizers : A fuzzy mechanism approach. In A. Marcelli and C. De Stefano, editors, *Proceedings of the 12th Conference of the International Graphonomics Society (IGS)*, pages 193–197, Salerno, Italy, June 2005.

[MAR06] Harold Mouchère, Éric Anquetil, and Nicolas Ragot. Etude et gestion des types de rejet pour l'optimisation de classifieurs. In *RFIA*, 2006.

[MAR07] Harold Mouchère, Eric Anquetil, and Nicolas Ragot. Writer style adaptation in on-line handwriting recognizers by a fuzzy mechanism approach : The adapt method. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 21(1) :99–116, 2007.

[MBAM07] Harold Mouchère, Sabri Bayoudh, Eric Anquetil, and Laurent Miclet. Synthetic on-line handwriting generation by distortions and analogy. In *13th Conference of the International Graphonomics Society (IGS)*, pages 10–13, 2007.

Références de l'état de l'art

[1] M. Aksela, J. Laaksonen, E. Oja, and J. Kangas. Rejection methods for an adaptive committee classifier. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 982–986, 10-13 Sept. 2001.

[2] E. Alpaydin, C. Kaynak, and F. Alimoglu. Cascading multiple classifiers and representations for optical and pen-based handwritten digit recognition. In *Proc. of the 7th International Workshop on Frontiers in Handwriting Recognition*, 2000.

[3] E. Anquetil and G. Lorette. Perceptual model of handwriting drawing application to the handwriting segmentation problem. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition (ICDAR'97)*, pages 112–117, Ulm, Allemagne, 1997.

[4] Éric Anquetil. *Modélisation et reconnaissance par la logique floue : application à la lecture automatique de l'écriture manuscrite omni-scripteur*. PhD thesis, Université de Rennes I, 1997.

[5] Éric Anquetil and H. Bouchereau. Integration of an on-line handwriting recognition system in a smart phone device. In *Sixteenth ICPR International Conference on Pattern Recognition (ICPR'2002)*, volume 3, pages 192–195, 2002.

[6] Éric Anquetil, Bertrand Couasnon, and Frédéric Dambreville. A symbol classifier able to reject wrong shapes for document recognition systems. In *Proceedings of 3rd IAPR International Workshop on Graphics Recognition (GREC'99)*, pages 195–202. Irisa, sep 1999.

[7] S. Bayoudh, L. Miclet, and A. Delhay. Learning by analogy : a classification rule for binary and nominal data. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, volume 20, pages 678–683, 2007.

[8] Sabri Bayoudh. *Apprentissage par proportion analogique*. PhD thesis, IRISA Ű Université de Rennes 1, 2007.

[9] Abdelouahab Bellili, M Gilloux, and Patrick Gallinari. An MLP-SVM combination architecture for offline handwritten digit recognition. *International Journal of Document Analysis and Recognition*, 5(4) :244–252, 2003.

[10] H. Bersini, J. Nordvik, and A. Bonarini. Comparing rbf and fuzzy inference systems on theoretical and practical basis. In *International Conference on Artificial Neural Networks (ICANN95)*, pages 169–174, Paris, France, 1995.

[11] Christoper M. Bishop. *Neural Network for Pattern Recognition*. Oxford University Press, 1995.

[12] A. Boatas, B. Dubuisson, and M.A. Dillies-Peltier. A new statistical pattern recognition distance rejection model : application to the monitoring of car catalytic converters. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2839–2844, 2000.

[13] Bernadette Bouchon-Meunier. *La logique floue et ses applications*. Addison-Wesley, 1995.

[14] Bernadette Bouchon-Meunier and Christophe Marsala, editors. *Logique Floue, Principes, Aide à la Décision*. Hermès-Lavoisier, 2003.

[15] François Bouteruche and Eric Anquetil. Fuzzy point of view combination for contextual shape recognition : Application to on-line graphic gesture recognition. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)*, pages 1088–1091, 2006.

[16] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7) :1145–1159, 1997.

[17] Anja Brakensiek, Andreas Kosmala, and Gerhard Rigoll. Comparing adaptation techniques for on-line handwriting recognition. In *6th ICDAR*, pages 486–490, 2001.

[18] Jane Bromley and John S. Denker. Improving rejection performance on handwritten digits by training with rubbish. *Neural Computation*, 5(3) :367–370, 1993.

[19] Javier Cano, Juan-Carlos Pérez-Cortes, Joaquim Arlandis, and Rafael Llobet. Training set expansion in handwritten character recognition. In *Proceedings of the 9th International Workshop on Structural and Syntactic Pattern Recognition (SSPR) and 4th Statistical Pattern Recognition (SPR)*, pages 548–556, 2002.

[20] Clément Chatelain, Laurent Heutte, and Thierry Paquet. Segmentation-driven recognition applied to numerical field extraction from handwritten incoming mail documents. In *Document Analysis Systems*, pages 564–575, 2006.

[21] Clément Chatelain, Laurent Heutte, and Thierry Paquet. A two-stage outlier rejection strategy for numerical field extraction in handwritten documents. In *18th International Conference on Pattern Recognition (ICPR'06)*, 2006.

[22] Yi-Kai Chen and Jhing-Fa Wang. Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11) :1304–1317, 2000.

[23] C.K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transaction on Inforamtion Theory*, 16 :41–46, 1970.

[24] Fu Lai Chung and Tong Lee. Fuzzy competitive learning. *IEEE Transaction on Neural Network*, 7(3) :539–551, 1994.

[25] R. Collobert, S. Bengio, and J. Mariéthoz. Torch : a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[26] Scott D. Connell and Anil K. Jain. Learning prototype for on-line handwritten digits. In *Processing 14th International Conference Pattern Recognition*, pages 182–184, Departement of Computer Science, aug 1998. Michigan State University.

[27] Scott D. Connell and Anil K. Jain. Writer adaptation for online handwriting recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(3) :329–346, 2002.

[28] L.P. Cordella, C. Sansone, F. Tortorella, M. Vento, and C. De Stefano. Neural network classification reliability : Problems and applications. In Academic Press, editor, *Neural Network Systems Techniques and Applications*, volume 5, pages 161–200, San Diego (California), 1998.

[29] Antoine Cornuéjols, Laurent Miclet, and Yves Kodratoff. *Apprentissage artificiel, concepts et algorithmes*. Eyrolles, 2002.

[30] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3) :273–297, 1995.

[31] S. De Backer and P. Scheunders. Texture segmentation by frequency-sensitive elliptical competitive learning. *Image and Vision Computing*, 19(9–10) :639–648, 2001.

[32] Claudio De Stefano, Carlo Sansone, and Mario Vento. To reject or not to reject : That is the question - an answer in case of neural classifiers. *IEEE Transaction on Systems, Man and Cybernetics*, 30(1) :84–94, feb 2000.

[33] A. Delhay and L. Miclet. Analogical equations in sequences : Definition and resolution. In *Proc. of International Colloquium on Grammatical Inference*, pages 127–138, 2004.

[34] B. Dubuisson and M. Masson. A statistical decision rule with incomplete knowledge about classes. *Pattern Recognition*, 26(1) :155–165, January 1993.

[35] Richard O. Duda, David G. Stork, and Peter E. Hart. *Pattern Classification, Second Edition*. Wiley Interscience, 2001.

[36] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8) :861–874, 2006.

[37] C. Frélicot, M.H. Masson, and B. Dubuisson. Reject options in fuzzy pattern classification rules. In *Proc. of 3rd European Congress on Intelligent Techniques and Soft Computing*, 1995.

[38] Giorgio Fumera and Fabio Roli. Support vector machines with embedded reject option. In *Proc. of Int. workshop on Pattern Recognition with Support Vector Machines (SVM2002)*, pages 68–82, Niagara Falls, Canada, August 2002. Springer, LNCS Vol. 2388.

[39] Giorgio Fumera, Fabio Roli, and Giorgio Giacinto. Multiple reject thresholds for improving classification reliability. In *Proceedings of Advances in Pattern Recognition : Joint IAPR International Workshops, SSPR 2000 and SPR 2000*, volume LNCS 1876/2000, pages 863–871, Alicante, Spain, 2000.

[40] Giorgio Fumera, Fabio Roli, and Giorgio Giacinto. Reject option with multiple thresholds. *Pattern Recognition*, 33(12) :2099–2101, dec 2000.

[41] D. Gentner, K. J. Holyoak, and B. Kokinov. *The analogical mind : Perspectives from cognitive science*. MIT Press, 2001.

[42] Marco Gori and Franco Scarselli. Are multilayer perceptrons adequate for pattern recognition and verification ? *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20(11) :1121–1132, 1998.

[43] N. Gorski. Optimizing error-reject trade off in recognition systems. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 1092–1096vol.2, 18-20 Aug. 1997.

[44] N. Gorski, V. Anisimov, E. Augustin, O. Baret, D. Price, and J.-C. Simon. A2ia check reader : a family of bank check recognition systems. In *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, pages 523–526, 20-22 Sept. 1999.

[45] N. Gorski, V. Anisimov, E. Augustin, S. Mysko, and J.-C. Simon. A new a2ia bankcheck recognition system. In *Handwriting Analysis and Recognition (Ref. No. 1998/440), IEE Third European Workshop on*, pages 24/1–24/6, 14-15 July 1998.

[46] I. Guyon, L. Schomaker, R. Palmondon, M. Liberman, and S. Janet. Unipen project of on-line data exchange and recognizer benchmarks. In *Proc. of the 12th Int. Conf. on Pattern Recognition*, pages 29–33, 1994.

[47] Isabelle Guyon. Handwriting synthesis from handwritten glyphs. In *Porc. the Fifth International Workshop on Frontiers of Handwriting Recognition (IWFHR'96)*, pages 309–312, Colchester, England, 1996.

[48] Thien M. Ha and Horst Bunke. Off-line, handwritten numeral recognition by perturbation method. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 19(5) :535–539, may 1997.

[49] Lars Kai Hansen, Christian Liisberg, and Peter Salamon. The error-reject tradeoff. *Open Systems & Information Dynamics*, 4 :159–184, Apr 1997.

[50] Muriel Helmer and Horst Bunke. Generation and use of synthetic training data in cursive handwriting recognition. In *Proceedings of 1st Iberian Conference Pattern Recognition and Image Analysis (IbPRIA)*, pages 336–345, 2003.

[51] M. Hesse. Aristotle's logic of analogy. *The Philosophical Quarterly*, 15(61) :328–340, 1965.

[52] S. Impedovo, P. S. P. Wang, and Horst Bunke, editors. *Automatic Bankcheck Processing*. World Scientific Pub Co Inc, 1997.

[53] H. Ishibuchi and T. Nakshima. Fuzzy classification with reject options by fuzzy if-then rules. In *Proc. of the 7th IEEE International Conference on Fuzzy Systems, 1998*, volume 2, pages 1452–1457, 1998.

[54] J.-S. Roger Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transaction on Neural Network*, 4(1) :156–159, Janvier 1993.

[55] Guido Kaufmann and Horst Bunke. Automated reading of cheque amounts. *Pattern Anal. Appl.*, 3(2) :132–141, 2000.

[56] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[57] S. Knerr, V. Anisimov, O. Baret, N. Gorski, D. Price, and J. C. Simon. The a2ia intercheque system : Courtesy amount and legal amount recognition for french checks. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(4) :505–548, 1997.

[58] Teuvo Kohonen. The self-organizing map. *Proceedings of IEEE*, 78(9) :1464–1480, sep 1990.

[59] Myoung-Wan Koo, Chin-Hui Lee, and Biing-Hwang Juang. Speech recognition and utterance verification based on a generalized confidence score. *Speech and Audio Processing, IEEE Transactions on*, 9(8) :821–832, Nov. 2001.

[60] R. Krishnapuram and J.M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2) :98–110, 1993.

[61] R. Krishnapuram and J.M. Keller. The possibilistic c-means algorithm : insights and recommendations. *Fuzzy Systems, IEEE Transactions on*, 4(3) :385–393, Aug. 1996.

[62] Louisa Lam. Classifier combinations : Implementations and theoretical issues. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 77–86. Springer, 2000.

[63] T.C.W. Landgrebe, D.M.J. Tax, P. Paclik, and R.P.W. Duin. The interaction between classification and reject performance for distance- based reject-option classifiers. *Pattern Recognition Letters*, 27 :908–917, 2006.

[64] Y. Lepage. Solving analogies on words : an algorithm. In *Proc. of COLING-ACL'98*, volume 1, pages 728–735, 1998.

[65] Edouard Lethelier, Manuel Leroux, and Michel Gilloux. An automatic reading system for handwritten numeral amounts on french checks. In *ICDAR*, pages 92–97, 1995.

[66] Cheng-Lin Liu, H. Sako, and H. Fujisawa. Effects of classifier structures and training regimes on integrated segmentation and recognition of handwritten numeral strings. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(11) :1395–1407, Nov. 2004.

[67] Cheng-Lin Liu, Hiroshi Sako, and Hiromichi Fujisawa. Performance evaluation of pattern classifiers for handwritten character recognition. *IJDAR*, 4(3) :191–204, 2002.

[68] Jinhui Liu and Paul Gader. Neural networks with enhanced outlier rejection ability for on-line handwritten word recognition. *Pattern Recognition*, 35 :2061–2071, 2002.

[69] Markos Markou and Sameer Singh. Novelty detection : a review - part 1 : statistical approaches. *Signal Processing*, 83(12) :2481–2497, 2003.

[70] Markos Markou and Sameer Singh. Novelty detection : a review - part 2 : neural network based approaches. *Signal Processing*, 83(12) :2499–2521, 2003.

[71] Alvin Martin, George Doddington, Terri Kamm, Mark Ordowski, and Mark Przybocki. The DET curve in assessment of detection task performance. In *Proc. Eurospeech '97*, pages 1895–1898, Rhodes, Greece, 1997.

[72] S. Marukatat, T. Artieres, P. Gallinari, and B. Dorizzi. Rejection measures for handwriting sentence recognition. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*, pages 24–29, 6-8 Aug. 2002.

[73] N. Matić, I. Guyon, J. Denker, and V. Vapnik. Writer-adaptation for on-line handwritten character recognition. In IEEE Computer Society Press, editor, *ICDAR93*, pages 187–191, 1993.

[74] Jonathan Milgram, Mohamed Cheriet, and Robert Sabourin. Two-stage classification system combining model-based and discriminative approaches. In *17th International Conference on Pattern Recognition (ICPR'04)*, pages 152–155, 2004.

[75] Jonathan Milgram, Mohamed Cheriet, and Robert Sabourin. Estimating accurate multi-class probabilities with support vector machines. In *Int. Joint Conf. on Neural Networks*, pages 1906–1911, 2005.

[76] B. Mitaim, S. Kosko. The shape of fuzzy sets in adaptive function approximation. *IEEE Transactions on Fuzzy Systems*, 9(4) :637–656, Aug 2001.

[77] Akira Nakamura. A method to accelerate adaptation for on-line handwriting recognition of large character set. In *IWFHR-9 2004*, pages 426–431, Tokyo, Japan, 2004.

[78] A. Nosary, L. Heutte, and T. Paquet. Reconnaissance de mots manuscrits par segmentation-reconnaissance : apports d'une reconnaissance lettres par niveau avec rejet. In *Colloque International Francophone sur l'Ecrit et le Document, CIFED'02*, 2002.

[79] A. Nosary, T. Paquet, and L. Heutte. Reconnaissance de textes manuscrits par adaptation au scripteur. In *Colloque International Francophone sur l'Ecrit et le Document, CIFED'02*, 2002.

[80] A. Nosary, T. Paquet, L. Heutte, and A. Bensefia. Handwritten text recognition through writer adaptation. In *8th International Workshop on Frontiers in Handwriting Recognition*, 2002.

[81] Luiz S. Oliveira, Alceu S. Britto Jr., and Robert Sabourin. Improving cascading classifiers with particle swarm optimization. In *Proc. of 8th ICDAR*, pages 570–574, Seoul, South Korea, August 2005. IEEE CS Press.

[82] Luiz S. Oliveira, Robert Sabourin, Flávio Bortolozzi, and Ching Y. Suen. Automatic recognition of handwritten numerical strings : A recognition and verification strategy. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(11) :1438–1454, 2002.

[83] Loïc Oudot, Lionel Prevost, Alvaro Moises, and Maurice Milgram. Self-supervised writer adaptation using perceptive concepts : Application to on-line text recognition. In *17th International Conference on Pattern Recognition (ICPR'04)*, volume 2, pages 598–601, 2004.

[84] Chitre Panchapakesan and Marimuthu Palaniswami. Effects of moving the centers in an rbf network. *IEEE Transaction on Neural Network*, 13(6) :1299–1307, nov 2002.

[85] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. In *SAC '02 : Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607, New York, NY, USA, 2002. ACM Press.

[86] John F. Pitrelli and Micael P. Perrone. Confidence-scoring post-processing for off-line handwritten-character recognition verification. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, 2003.

[87] John F. Pitrelli, Jayashree Subrahmonia, and Micael P. Perrone. Confidence modeling for handwriting recognition : algorithms and applications. *International Journal of Document Analysis*, 8(1) :35–46, 2005.

[88] R. Plamondon and F.J. Maarse. An evaluation of motor models of handwriting. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(5) :1060–1072, 1989.

[89] Réjean Plamondon and Wacef Guerfali. The generation of handwriting with delta-lognormal synergies. *Biological Cybernetics*, 78 :119–132, 1998.

[90] John C. Platt and Nada P. Matić. A constructive RBF network for writer adaptation. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 765. The MIT Press, 1997.

[91] L. Prevost, C. Michel-Sendis, A. Moises, L. Oudot, and M. Milgram. Combining model-based and discriminative classifiers : application to handwritten character recognition. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 31–35vol.1, 3-6 Aug. 2003.

[92] S. Quiniou and E. Anquetil. Use of a Confusion Network to Detect and Correct Errors in an On-line Handwritten Sentence Recognition System. In *9th International Conference on Document Analysis and Recognition (ICDAR'07)*, pages 382–386, Curitiba, Brazil, September 2007.

[93] S. Quiniou, F. Bouteruche, and E. Anquetil. Word Extraction for the Recognition of On-Line Handwritten Sentences. In *13th Conference of the International Graphonomics Society (IGS'07)*, pages 52–55, Melbourne, Australia, November 2007.

[94] N. Ragot and E. Anquetil. Melidis : Pattern recognition by intrinsic/discriminant dual modeling based on a hierarchical organization of fuzzy inference systems. In *10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU'04)*, pages 2069–2076, Perugia, Italy, 2004.

[95] Christophe Renaudin, Yann Ricquebourg, and Jean Camillerapp. A general method of segmentation-recognition collaboration applied to pairs of touching and overlapping symbols. In *ICDAR 2007*, pages 659–663, Curitiba, Brazil, 2007.

[96] Douglas A. Reynolds, Thomas F. Quatieri, and Robert B. dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10 :19–41, 2000.

[97] Christophe Saint-Jean and Carl Frélicot. Une méthode paramétrique et robuste de classification semi-supervisée avec rejet. In *Plateforme AFIA, 2ème Conférence sur l'Apprentissage (CAp)*, pages 85–100, 2001.

[98] L. Schomaker, E. Helsper, H. Teulings, and G. Abbink. Adaptive recognition of online, cursive handwriting. In *6th International Conference on Handwriting and Drawing (ICOHD'93), Paris, France, July 4-7, 1993*, pages 19–21, 1993.

[99] Jürgen Schürmann. *Pattern Classification : a unified view of statistical and neural approaches*, chapter Recursive parameter estimation. Wiley-Interscience, 1996.

[100] Holger Schwenk and Maurice Milgram. Constraint tangent distance for on-line character recognition. In *Proceedings of the 13th Internationnal Conference on Patern Recognition (ICPR)*, 1996.

[101] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practice for convolutional neural network applied to visual analysis. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

[102] S.N. Srihari, Sung-Hyuk Cha, and Sangjik Lee. Establishing handwriting individuality using pattern recognition techniques. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 1195–1204, 10-13 Sept. 2001.

[103] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, 15(1) :116–132, 1985.

[104] Katsuhiko Takahashi and Daisuke Nishiwaki. A class-modular glvq ensemble with outlier learning for handwritten digit recognition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 268, Washington, DC, USA, 2003. IEEE Computer Society.

[105] David M. J. Tax and Robert P. W. Duin. Support vector data description. *Machine Learning*, 54(1) :45–66, 2004.

[106] David M.J. Tax and Robert P.W. Duin. Combining one-class classifiers. In *Multiple Classifier Systems*, 2001.

[107] Tamás Varga and Horst Bunke. Generation of synthetic data for an hmm-based handwriting recognition system. In *Proceedings of 7th International Conference on Document Analysis and Recognition (ICDAR)*, pages 618–622, 2003.

[108] Tamás Varga, Daniel Kilchhofer, and Horst Bunke. Template-based synthetic handwriting generation for the training of recognition systems. In *Proceedings of 12th Conference of the International Graphonomics Society (IGS)*, pages 206–211, 2005.

[109] G. C. Vasconcelos, M. C. Fairhurst, and D. L. Bisset. Investigating feedforward neural networks with respect to the rejection of spurious patterns. *Pattern Recognition Letters*, 16(2) :207–212, 1995.

[110] C. Viard-Gaudin, P. M. Lallican, S. Knerr, and P Binter. The irest on/off dual handwrittinf database. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR'99)*, pages 455–458, Bangalore, India, 1999.

[111] Vuokko Vuori, Jorma Laaksonen, and Jari Kangas. Influence of erroneous learning samples on adaptation in on-line handwriting recognition. *Pattern Recognition*, 35 :915–925, 2002.

[112] Vuokko Vuori, Jorma Laaksonen, and Erkki Oja. On-line adaptation in recognition of handwritten alphanumeric characters. In *5th ICDAR*, pages 792–795, Bangalore, India, 1999.

[113] Jue Wang, Chenyu Wu, Ying-Qing Xu, Heung-Yeung Shum, and Liang Ji. Learning-based cursive handwriting synthesis. In *Proceedings of 8th International Workshop on Frontiers in Handwriting Recognition (IWFHR)*, 2002.

[114] J.G. Wilpon, L.R. Rabiner, C.-H. Lee, and E.R. Goldman. Automatic recognition of keywords in unconstrained speech using hidden markov models. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 38(11) :1870–1878, Nov. 1990.

[115] Hui Zhu, L. Tang, and Peng Liu. An mlp-orthogonal gaussian mixture model hybrid model for chinese bank check printed numeral recognition. *Int. Journal of Document Analysis and Recognition*, 8(1) :27–34, 2006.

[116] Matthias Zimmermann, Roman Bertolami, and Horst Bunke. Rejection strategies for offline handwritten sentence recognition. In *17th International Conference on Pattern Recognition (ICPR'04)*, volume 02, pages 550–553, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

Résumé

L'émergence de l'informatique nomade a rendu indispensable l'utilisation des interfaces orientées stylo sur des petits périphériques mobiles disposant de ressources limitées : l'utilisateur écrit et dessine directement des textes, des croquis, des commandes, etc. sur l'écran tactile du système. Pour permettre une mise en oeuvre conviviale et efficace de ces nouvelles modalités de communication, il est nécessaire de concevoir des moteurs de reconnaissance robustes et performants pour interpréter l'écriture manuscrite et les tracés graphiques. L'objectif de ces travaux est donc d'améliorer les performances des systèmes de reconnaissance existants en étudiant deux axes de recherches, le rejet et l'adaptation.

Le premier axe de recherche, le rejet, permet de décider si la réponse du classifieur peut être considérée comme pertinente ou non. Pour cela il s'agit de délimiter le domaine de validité des connaissances du classifieur. Nous définissons différentes notions de rejet qui représentent différents cas d'utilisation du rejet :

- le rejet de distance : la forme à reconnaître ne correspond pas du tout à une forme que le classifieur a appris à reconnaître, la réponse du classifieur ne peut donc être pertinente et il faut la rejeter ;
- le rejet d'ambiguïté : la forme peut appartenir à deux classes distinctes, le classifieur ne peut pas prendre de décision sûre, il faut donc rejeter la forme.

Pour mettre en oeuvre ces rejets, nous définissons une option générique de rejet utilisant la notion de fonctions de confiance qui permet, grâce à des seuils, de décider du rejet. Nous proposons un algorithme générique nommé AMTL pour fixer ces seuils avec ou sans contre-exemples disponibles. Nous démontrons les capacités de généralisation de notre approche en la comparant avec les solutions plus classiques. Notre approche est particulièrement efficace dans un contexte de ressources limitées.

Le principe du second axe de recherche, l'adaptation, est de spécialiser automatiquement un système de reconnaissance de caractères conçu pour reconnaître l'écriture de n'importe quel utilisateur (système omni-scripteur) en un système spécialisé dans la reconnaissance de l'écriture d'une seule personne (système mono-scripteur), l'utilisateur principal du périphérique mobile. Cette adaptation se fait à la volée, c'est-à-dire au fur et à mesure de l'utilisation du système par le scripteur.

Nous proposons une approche nommée ADAPT permettant de réaliser cette adaptation sur les systèmes d'inférence floue. Les prototypes flous du système sont déplacés, déformés et crées à la volée en tenant compte de tous les paramètres du classifieur. Pour améliorer encore l'efficacité de l'adaptation, nous proposons d'augmenter la quantité de données disponibles lors de l'adaptation pour éviter à l'utilisateur de saisir plusieurs dizaines de fois chaque caractère. Pour cela nous synthétisons de nouveaux caractères à partir de ceux déjà entrés par l'utilisateur en considérant les propriétés particulières de l'écriture manuscrite. Pour valider nos approches, nous proposons une série d'expérimentations dans différents contextes, depuis des conditions expérimentales idéales jusqu'à une utilisation en conditions réelles sur un périphérique mobile.