PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

| | |
|---|---|
| **Titre:** Title: | Traitement en-ligne de documents manuscrits structurés : segmentation en mots par algorithmes d'apprentissage |
| **Auteur:** Author: | Mohammad Shahram Moïn |
| **Date:** | 2000 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Moïn, M. S. (2000). Traitement en-ligne de documents manuscrits structurés : segmentation en mots par algorithmes d'apprentissage [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8682/ |

**Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

| | |
|---|---|
| **URL de PolyPublie:** PolyPublie URL: | https://publications.polymtl.ca/8682/ |
| **Directeurs de recherche:** Advisors: | Jean-Jules Brault |
| **Programme:** Program: | Non spécifié |

**Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal**
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

# TRAITEMENT EN-LIGNE DE DOCUMENTS MANUSCRITS STRUCTURÉS : SEGMENTATION EN MOTS PAR ALGORITHMES D'APPRENTISSAGE

MOHAMMAD SHAHRAM MOÏN
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR (Ph.D.)
(GÉNIE ÉLECTRIQUE)
SEPTEMBRE 2000

© Mohammad Shahram MOÏN, 2000

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-57382-6

Canada

**UNIVERSITÉ DE MONTRÉAL**

**ÉCOLE POLYTECHNIQUE DE MONTRÉAL**

**Cette thèse intitulée :**

# TRAITEMENT EN-LIGNE DE DOCUMENTS MANUSCRITS STRUCTURÉS : SEGMENTATION EN MOTS PAR ALGORITHMES D'APPRENTISSAGE

**présentée par : MOÏN Mohammad Shahram**
**en vue de l'obtention du diplôme de : Philosophiae Doctor (Ph.D.)**
**a été dûment acceptée par le jury d'examen constitué de :**

**M. LAURIN Jean-Jacques, Ph. D., président**
**M. BRAULT Jean-Jules, Ph.D., membre et directeur de recherche**
**M. SABOURIN Robert, Ph.D., membre**
**Mme. CÔTÉ Myriam, Ph.D., membre**

*À mon épouse Marjan*
*pour toute sa patience et son amour,*

*À mon père Bahram et ma mère Pari*
*auxquels je dois tout ce que j'ai dans la vie,*

*À mes deux anges, AmirAli et Negar*

*À mes sœurs Shahrzad et Maryam*

*Et à l'esprit de mon oncle Parviz,*
*qui nous a quitté si tôt... .*

# REMERCIEMENTS

Je tiens à exprimer ma reconnaissance et mes remerciements à mon directeur de recherche, *Professeur Jean-Jules Brault*, pour sa disponibilité, ses excellents commentaires et critiques, son soutien moral, ses encouragements valorisants et son intérêt à communiquer le goût de la recherche scientifique.

Je désire également remercier *tous les membres du jury* d'avoir accepté de consacrer un temps précieux à l'évaluation de cette thèse.

Je remercie l'*École Polytechnique de Montréal*, en particulier le *Département de Génie Électrique et de Génie Informatique*, pour avoir mis à ma disposition un cadre de travail stimulant.

Je tiens à remercier *le Centre de recherche sur les télécommunications de l'Iran (ITRC)* pour m'avoir donné l'opportunité d'effectuer cette activité de recherche.

Je remercie tous mes amis et collègues *du laboratoire Scribens,* en particulier *Dr. Wacef Guerfali* et *Dr. Octavian Ureche* pour leurs commentaires pendant la rédaction de la thèse, *Dr. Salim Djeziri, Hong Trang Nguyen, Silvia Sabeva, François Belair* et *Nicolas Leduc* pour m'avoir fourni des échantillons de textes manuscrits.

Je désire également remercier *Madame Diane Blanchet* d'avoir effectué la révision grammaticale de cette thèse et *M. Yves Boudreault,* pour son aide amicale, notamment pendant les dernières années de mes études.

Mes remerciements finals, mais non les moindres, vont à ma chère épouse *Marjan* qui m'a donné le courage et la persévérance nécessaires pour accomplir ce travail et pour cela, je lui serai éternellement reconnaissant.

# RÉSUMÉ

Dans le cadre des travaux de recherche effectués sur l'utilisation de l'écriture manuscrite comme interface homme-système et dans une étape vers la conception et la réalisation d'un environnement de traitement de structures de documents manuscrits, cette thèse présente des systèmes originaux de segmentation en-ligne de textes manuscrits en mots, de reconnaissance et d'édition de la structure physique de documents manuscrits ainsi que des propositions originales pour la conception de cet environnement. Ce dernier permettrait à un scripteur de créer aisément sur ardoise électronique, un document manuscrit structuré et éditable avec un crayon. Le document produit est composé d'un ensemble d'objets formant sa structure physique, comme des blocs, des lignes et des mots. Le balisage de cette dernière, en attribuant des étiquettes (telles que le titre, la date, le paragraphe et la signature) aux objets physiques, permet de construire sa structure logique. En éditant ces structures, l'utilisateur pourrait réorganiser les arborescences physique et logique de son document manuscrit, comme il peut le faire avec des éditeurs de documents typographiques.

Nous présenterons le schéma d'un système complet de traitement de documents manuscrits structurés, lequel inclut les étapes suivantes : l'acquisition des tracés, la détection du type de tracés (donnée ou commande gestuelle), la reconnaissance et l'édition de la structure physique, la reconnaissance et l'édition de la structure logique et la conversion des résultats en langage international de présentation de documents structurés SGML (Standard Generalized Markup Language). Cependant l'emphase a été mise sur la reconnaissance et l'édition de la structure physique. Nous présenterons les spécifications des modules conçus et réalisés, y compris les stratégies adoptées pour la reconnaissance des lignes et des blocs, et les commandes d'édition de la structure physique.

La contribution majeure de ce travail repose sur le développement d'un système original de segmentation en-ligne de textes manuscrits en mots, qui est une étape

cruciale de la reconnaissance de la structure physique. Les mots forment les feuilles de l'arborescence physique et chaque erreur de segmentation peut affecter la performance des étapes de reconnaissance des structures physique et logique. Dans un texte entré au clavier, les mots sont composés de caractères isolés et, grâce à des caractères d'espacement, sont facilement identifiables. Toutefois, dans un texte manuscrit entré à l'aide d'un crayon sur ardoise électronique, les mots peuvent être composés de tracés continus ou discontinus. Le regroupement de ces tracés, sous forme de mots, nécessite donc une interprétation des espacements entre les tracés.

La difficulté de cette tâche repose, d'un côté, sur le choix de caractéristiques discriminantes pour représenter adéquatement les espacements intra-mot et inter-mots et, d'un autre côté, sur la méthode à utiliser pour les classifier efficacement, tout en respectant les contraintes des systèmes en-ligne conviviaux.

Pour atteindre notre objectif, nous avons fait une revue exhaustive des méthodes de segmentation de textes manuscrits en focalisant sur les caractéristiques utilisées pour la segmentation. Les meilleures caractéristiques dans le contexte d'une application en-ligne étaient la distance entre les rectangles englobant des tracés et les distances horizontales minimale et moyenne entre les tracés.

Nous avons amélioré ces caractéristiques dites conventionnelles et proposé également des caractéristiques complémentaires. La première amélioration est obtenue, en effectuant une phase préalable de reconnaissance des signes diacritiques afin de les enlever de l'ensemble de tracés à segmenter. Le module de reconnaissance de ces signes est basé sur les extrema des entités de texte. La deuxième amélioration fut de remplacer la distance entre les rectangles par la distance entre les parallélogrammes. Nous avons conçu un algorithme adaptatif d'estimation de distances entre les parallélogrammes. Quatre caractéristiques utilisant les extrema des entités manuscrits ont été proposées : deux primitives pour représenter la taille de l'écriture et deux autres pour refléter la longueur de tracés et de mots.

Trois différents types de classificateurs ont été utilisés : K plus proches voisins, neuronaux et AdaBoost. Deux scénarios d'apprentissage et de test, commun et personnalisé, ont été appliqués sur ces classificateurs.

Le meilleur taux moyen des classificateurs K plus proches voisins etait de 98.5%, obtenu en appliquant les techniques de « cross-validation » et « validation statistique des résultats » à un classificateur entraîné avec le scénario commun et les caractéristiques proposées.

Les classificateurs neuronaux étaient de type Perceptron à une couche et multicouches. Ils ont été implantés avec les caractéristiques conventionnelles et proposées comme les entrées. Parmi les caractéristiques conventionnelles, le meilleur résultat de classification appartenait à la distance entre rectangles avec un taux moyen de 97.7%. Des classificateurs neuronaux avec une ou deux caractéristiques proposées, entraînés avec le scénario personnalisé ont permis d'améliorer le taux de segmentation. Le meilleur taux moyen obtenu était de près de 98.7%, appartenant à un classificateur avec la distance horizontale minimale entre les tracés à l'entrée, entraîné avec seulement 70 patrons d'apprentissage. En augmentant le nombre de caractéristiques présentées aux classificateurs, on s'attendait à obtenir des taux de segmentation encore plus élevés. Toutefois, selon la théorie statistique de l'apprentissage, pour compenser l'effet d'augmentation de complexité de la structure des réseaux et pour empêcher une augmentation d'erreur de segmentation, il fallait accroître le nombre de patrons d'apprentissage (jusqu'à plus de 900 patrons), en exigeant des scripteurs d'écrire des textes plus longs et de les segmenter; ce qui étaient à l'encontre des contraintes de convivialité de notre système.

Nous avons proposé d'adapter l'algorithme d'AdaBoost pour résoudre les problèmes de classification causés par le compromis entre le nombre de patrons d'entraînement et la complexité de structure des classificateurs neuronaux. AdaBoost est une « comité de machines» permettant de regrouper plusieurs classificateurs (experts) et de combiner leur vote pondéré dans une procédure de classification. Les experts proposés sont des classificateurs neuronaux très simples, nécessitant un nombre très petit

de patrons d'apprentissage (seulement 70 patrons). En plus, chacun est un expert d'une caractéristique; ce qui permet d'obtenir plusieurs experts pour chaque caractéristique. Le nombre d'experts par caractéristique est déterminé dans la phase d'apprentissage.

Les résultats obtenus montrent une augmentation considérable du taux de classification correcte des distances inter-tracés (jusqu'à 99.1%). Ce dernier est le résultat de l'application du scénario personnalisé à un classificateur AdaBoost modifié utilisant les sept caractéristiques proposées.

L'analyse des cas difficiles de segmentation montre que l'utilisation de ce classificateur permet de corriger près de 80% des erreurs du meilleur classificateur conventionnel. Dans un contexte plus spécifique, près de 88% des erreurs causées par les signes diacritiques, 100% des erreurs causées par l'inclinaison d'écriture et près de 73% des autres erreurs ont été corrigées. D'autres résultats obtenus avec des textes uniquement composés de cas difficiles à segmenter ont montré la supériorité systématique de ce classificateur sur le meilleur classificateur conventionnel (avec un écart jusqu'à 35% entre les taux totaux de segmentation).

En somme, voici les points forts de ce classificateur : (1) il possède le meilleur taux moyen de classification; (2) il est le classificateur le plus robuste parmi tous les classificateurs implantés (écart type de 0.8% seulement sur les taux moyens obtenus sur différents textes); (3) il s'adapte aux styles d'écriture des scripteurs; (4) il requiert un petit nombre de patrons d'apprentissage (seulement 70) et (5) il fait la classification des distances inter-tracés avec un nombre d'opérations peu élevé. En conséquence, ce classificateur, rapide, précis et stable, répond parfaitement aux critères présentés précédemment.

# ABSTRACT

Within the context of research on applications of handwriting in man-machine interfaces and as an important step towards the design and implementation of a structured handwritten document processing environment, this thesis presents original systems of on-line word segmentation, layout structure recognition and editing of handwritten documents as well as original proposals for the design and implementation of this environment. The complete system would allow users to create and edit easily a structured handwritten document using an electronic ink pen. Such a document would contain both layout and logical structures: the layout structure being a set of objects such as pages, blocks and lines, and the logical structure a series of logical element such as title, date, paragraph and signature obtained by attributing logical tags to layout objects. A handwritten document structure editor can then be used to reorganize the layout and logical trees of the document.

The diagram of a complete system of structured handwritten document processing is presented, including the following modules: strokes acquisition, stroke type detection (data or gesture), layout structure recognition and editing, logical structure recognition and editing and a converter producing an SGML (Standard Generalized Markup Language) version of the resulting structured document. However, this work is mainly oriented towards the recognition and editing of document layout structure. The features of the designed and implemented modules are presented, including the strategies adopted for the recognition of lines and blocks, and also the editing commands of the layout structure.

The major contribution of this thesis consists of the development of an original system of on-line segmentation of handwriting in words, which is a crucial step in the layout structure recognition procedure. Words, in this context, are the leaves of the layout structure and a single error in the procedure of segmentation can affect the result of the layout and logical structure recognition. When a keyboard is used to create a document, the segmentation in words is done explicitly, since the words are composed of discrete characters and separated by space characters. However, in a handwritten document created in a pen-based computer, words are composed of strokes, which are continuous objects. Thus, in order to regroup strokes in words, one needs to interpret inter-strokes spaces.

The difficulty of this task is related to the following aspects: first, choice of features which can represent adequately the intra and inter-word spacings and second, choice of a classification method to discriminate between intra-word and inter-words spacing, using the extracted features. One must add to this, the complexities caused by the respect of the on-line systems and the freedom of writers to compose their document in the way they like. There is also the necessity of minimizing of the writers tasks for training the system and/or for correcting misclassified patterns.

To reach our goal, we first conducted an exhaustive literature survey of handwritten segmentation methods, with a particular focus on the features used for segmentation. The positive and negative points of the features under review have been highlighted: among these we have identified the best inter-strokes gap metrics for on-line applications, i.e. the bounding box and minimum and average horizontal gaps.

This work also proposes some methods to enhance the performance of existing inter-strokes distance metrics and consequently to produce another category of features. Our proposals consist, in general, of pre-recognizing the diacritical signs and of replacing the bounding box gaps by the gaps between parallelograms. We have also implemented an adaptive algorithm for estimating these gaps and a diacritical sign recognition module based on the extremas of text entities. Seven different features have been proposed, including parallelogram gaps, minimum and average horizontal gaps,

two features to represent the size of the handwriting and two others to reflect the length of strokes and words.

Three different types of classifiers have been utilized: K nearest neighbors, neural and AdaBoost. We have also applied two different strategies for learning and testing these classifiers: generalized and customized.

The highest classification rate of KNN classifiers is 98.5%, by applying the "cross-validation" and "statistical validation" techniques and using the generalized strategy and proposed features,.

We have also implemented one-layer and multi-layer Perceptron neural network classifiers. The use of proposed features in one- and two- input neural classifiers and the application of the customized strategy improved the segmentation rate, compared to the best neural classifier using the conventional features (from 97.7% to almost 98.7%). However, because of the tradeoff between the number of training patterns and the complexity of the classifier's structures, which is based on the "statistical learning theory", the customized strategy could not be applied adequately to classifiers having a higher number of inputs (and consequently more complex structures).

Taking an original approach, we proposed to adapt AdaBoost algorithm to solve the problem of classification due to the tradeoff between the number of training patterns and the classifier structure's complexity. AdaBoost is a "comity machine" regrouping several classifiers (experts) and combining their weighted votes in a classification procedure. In the case of our AdaBoost classifiers, the experts are very simple neural classifiers, requiring a very small number of training patterns. Moreover, each expert is a specialist in one feature, allowing us to obtain several experts per feature in the final classifier structure. The number of experts per feature is determined in the training phase.

The results obtained using the AdaBoost classifiers show a significant improvement in correct classification rates. The highest rate, almost 99.1%, is obtained using the customized strategy and all (seven) extracted features as inputs.

Our analysis of the difficult segmentation patterns shows that close to 80% of the classification errors produced by the best conventional classifier have been corrected. More specifically, 88% of classification mistakes caused by diacritical signs, 100% of errors caused by the slant of handwriting and up to 73% of other types of error have been eliminated. Also, the classification results using texts intentionally constructed with difficult segmentation examples prove the systematic superiority of this classifier over the best conventional classifier (with a gap up to 35% between rates of correct segmentation).

In conclusion, the strong points of this classifier include its average rate of correct classification (99.1%) and its degree of robustness (standard deviation of 0.8% in the average classification rates) which are highest among all implemented classifiers. This classifier adjusts easily to styles of handwriting and requires a small number of training examples and operations to learn and to classify the inter-stroke gaps. Consequently, we have designed and implemented a classifier which is fast, precise and stable, and respects perfectly the previously mentioned criteria concerning the design of a system of on-line handwriting segmentation in words.

# TABLE DES MATIÈRES

# LISTE DES TABLEAUX

# LISTE DES FIGURES

# LISTE DES SIGLES ET ABRÉVIATIONS

On trouve ci-dessous, les sigles et les abréviations les plus utilisés dans cette thèse. En plus de cette liste, les définitions et les représentation graphique des caractéristiques et des mesures utilisées sont présentées dans l'annexe B.

- $\varphi$ : fonction d'activation d'un neurone artificiel.
- $\eta$ : taux d'apprentissage dans l'algorithme de rétro-propagation d'erreur.
- *avgDistBB* : distance moyenne entre les rectangles des tracés précédents pour un tracé donné.
- *avgHeight* : hauteur moyenne des tracés précédents pour un tracé donné.
- *B* : biais d'un neurone artificiel.
- *BackProp* : algorithme d'apprentissage de rétro-propagation d'erreur.
- *BB* : distance entre les rectangles englobant de deux entités manuscrites.
- *CH* : distance entre les coques convexes de deux entités manuscrites.
- *curHghtExtrema* : distance verticale moyenne entre extrema consécutifs du tracé actuel.
- *curNbExtrema* : nombre d'extrema de tracé actuel
- *D* : distribution associée à l'ensemble des patrons d'apprentissage dans l'algorithme d'AdaBoost.
- *Dist_BB* : distance horizontale entre les rectangles de deux entités manuscrites.
- *Dist_Eucl* : distance euclidienne entre deux entités manuscrites.
- *Dist_Paral* ou *D_Prl* : distance horizontale entre les parallélogrammes de deux entités manuscrites.
- *DTD* : définition du type de document.
- *h* : hypothèse simple dans l'algorithme d'AdaBoost.

- *Haut_Moy_Extrema* : distance moyenne verticale entre les paires consécutives de minima et maxima d'un tracé.
- *Minim_Z* : algorithme de minimisation de $Z$ dans l'algorithme d'AdaBoost.
- *prHghtExtrema* ou *pHghtExt* : distance verticale moyenne entre les extrema consécutifs du mot précédent.
- *prNbExtrema* ou *pNbExt* : nombre d'extrema du mot précédent.
- *ReSampel* : distance horizontale moyenne entre deux entités manuscrites.
- *RLavg* : distance horizontale moyenne entre deux entités manuscrites.
- *RLmin* : distance horizontale minimale entre deux entités manuscrites
- *SGML* : Standard Generalized Markup Language, langage de la présentation des structures de documents.
- $T$ : nombre maximal d'itérations dans l'algorithme d'AdaBoost
- $w$, $W$ ou $\omega$ : poids d'un neurone artificiel.
- *WeakLearn* : algorithme d'apprentissage des hypothèses simples dans AdaBoost.
- $y_{max\text{-}moy}$ : valeur moyenne des coordonnées $y$ des maxima d'une entité manuscrite.
- $y_{med\text{-}moy}$ : valeur moyenne des coordonnées $y$ de tous les minima et maxima d'une entité manuscrite.
- $y_{min\text{-}moy}$ : valeur moyenne des coordonnées $y$ des minima d'une entité manuscrite.
- $Z$ : facteur de normalisation de la distribution $D$ dans l'algorithme d'AdaBoost

# LISTE DES ANNEXES

# CHAPITRE 1

# INTRODUCTION

L'écriture manuscrite, dont l'apparition chez les êtres humains se perd dans la nuit des temps, fait partie de notre vie quotidienne. Nous utilisons le crayon pour écrire des notes, des mémos, des lettres, etc. Même le clavier des ordinateurs et les logiciels puissants de traitement de textes ne l'ont pas tout à fait remplacé dans plusieurs tâches simples comme écrire une idée et dessiner. Cette importance de l'écriture manuscrite, et la croissance phénoménale de l'informatique dans les années 80 et 90, ont poussé les chercheurs à concevoir et à développer des interfaces homme-ordinateur basées sur le paradigme papier/crayon. Plusieurs laboratoires de recherche, dont le nôtre, le laboratoire Scribens[1], ont orienté leurs travaux dans ce domaine et chaque année, plusieurs livres, articles et revues traitant des divers aspects de traitements de l'écriture manuscrite sont publiés. Cependant, les groupes de recherche n'ont pas trouvé leur tâche facile et les compagnies qui ont essayé de développer des interfaces crayon ont eu des succès limités[2]. La raison principale de ces échecs prématurés était la concentration exclusive de ces systèmes sur l'aspect de reconnaissance de l'écriture manuscrite. En fait, les algorithmes de reconnaissance implantés n'étaient tout simplement pas assez puissants pour traiter les grandes variations de styles de l'écriture de scripteurs.

Il est toutefois possible d'envisager, dans une interface crayon, des applications informatiques bien intéressantes pour l'écriture manuscrite sans mettre l'accent sur

---

[1] Laboratoire de l'application informatique de l'écriture manuscrite, École Polytechnique de Montréal.
[2] À titre d'exemple, on peut citer Newton de la compagnie Apple.

l'aspect de reconnaissance de caractères. Il s'agit de garder la forme manuscrite du texte et d'essayer d'en extraire des informations structurelles, tout en permettant aux scripteurs de les éditer et d'en construire une nouvelle forme de texte à leur guise.

## 1.1 PROBLÉMATIQUE

L'apparence de l'écriture, incluant la forme et l'inclinaison des caractères et les espacements entre les entités manuscrites, sont des caractéristiques propres à chaque scripteur. Ces informations sont absentes dans un texte typographique. Il arrive également que l'on écrive un texte, tel qu'un message ou une note manuscrite, s'adressant à un être humain et que sa forme originale soit intelligible et suffisante pour le lecteur. En plus, les notes manuscrites prises dans des situations pressantes peuvent être déchiffrables par le scripteur, mais non pas par un autre lecteur ni, bien évidemment, par un système de reconnaissance automatique des caractères. Par conséquent, sans vouloir écarter complètement la possibilité d'intégration des modules de reconnaissance des caractères dans les systèmes en-ligne de traitement de textes manuscrits, ni vouloir nier la nécessité et l'importance de ces modules dans les systèmes hors-ligne, nous croyons que la reconnaissance des caractères ne doit pas devenir la tâche principale des interfaces dynamiques d'acquisition et de traitement des entités manuscrites et que ces interfaces doivent être principalement orientées vers d'autres types de traitement.

Notre proposition à cet égard consiste à mettre l'emphase sur l'aspect de reconnaissance et d'édition des structures de documents manuscrits pour permettre aux scripteurs de garder la forme manuscrite de leurs textes, tout en accédant à des informations plus pertinentes qu'un simple ensemble de tracés[3] et, en plus, d'avoir la possibilité d'éditer ces informations. Nous parlons ainsi des structures physique et logique des documents manuscrits. La structure physique d'un document est l'arborescence de ses entités physiques comme les blocs, les lignes de chaque bloc et les mots de chaque ligne. Ces entités sont souvent communes dans les différents types de

[3] Un tracé est la suite des points acquis par l'interface entre une pause et une levée de crayon.

document. La structure logique est formée de l'ensemble des éléments logiques qui dépendent du modèle de document et qui varient, en conséquence, d'un modèle à l'autre. Les exemples des entités logiques sont des items et des ensembles d'items, dans le cas d'une liste et de la date, l'expéditeur et le destinataire dans le cas d'un mémo[4].

Comme un exemple simple d'utilité des structures de documents manuscrits, on peut citer le cas d'une liste de tâches à effectuer. Dans le paradigme papier/crayon, en général, on commence par écrire les tâches sous formes d'items d'une liste sans tenir compte de leurs priorités. Ensuite on leur attribue des priorités en énumérant les tâches ou en les déplaçant dans la liste, à l'aide, par exemple, de flèches. Il arrive aussi, comme tout autre document, que l'on veuille effacer les mots, les déplacer ou les remplacer par d'autres mots. Le résultat de ces opérations sur le papier rend le texte malpropre. En plus, les opérations possibles sont limitées. Par exemple on ne peut copier des items (des éléments logiques) ou des mots (des entités physiques). Par contre, une interface crayon qui a la capacité de reconnaître les structures de documents manuscrits et de les éditer également, permet aux scripteurs d'effectuer la même tâche avec beaucoup plus de facilités et d'options supplémentaires. Conséquemment cela donne au document produit une apparence propre, en plus d'une représentation interne riche. Parmi les opérations possibles sur une telle liste structurée, on peut nommer l'édition des entités physiques comme les mots, les lignes et les groupes de mots ainsi que l'édition des éléments logiques comme les items de liste ou les ensembles d'items (selon le modèle logique de la liste).

Des exemples comme celui des listes structurées sont nombreux et montrent tous l'utilité des structures de documents manuscrits dans les interfaces crayon, notamment, en ce qui concerne l'interaction avec les scripteurs. En outre et comme on l'expliquera plus loin, les structures de documents permettent d'améliorer d'autres types d'opérations sur le texte, comme le stockage, la transmission et la reconnaissance de caractères, si nécessaire.

---

[4] Des exemples de ces structures sont présentés dans la section suivante.

## 1.2 CONTEXTE DU TRAVAIL

L'objectif général de notre travail consiste à concevoir et à réaliser un système de reconnaissance et d'édition de la structure physique de documents manuscrits. Ce projet s'inscrit dans le cadre d'un système complet de traitement de documents manuscrits lequel comprend d'autres modules dont les modules de reconnaissance et d'édition de la structure logique.

La figure 1.1 montre le diagramme de flux de données du système complet de traitement de documents manuscrits structurés, présenté originalement dans (Moin, 1997). La première étape consiste à acquérir un tracé manuscrit (bloc 1). Étant donné que l'utilisateur peut se servir du même crayon pour entrer son texte et pour l'éditer via les commandes gestuelles, le type du tracé doit être détecté (bloc 2). En plus, s'il s'agit d'une commande, un décodeur de commandes (bloc 4) le classifie en commandes d'édition de la structure physique ou logique.

Si le tracé est une donnée manuscrite, il est traité dans un module dont la tâche consiste à construire la structure physique spécifique du document à partir de la séquence des tracés (bloc 3). L'exemple de la figure 1.2 montre les objets de la structure physique d'un texte manuscrit. Un module d'édition permet ensuite à l'utilisateur d'apporter les modifications nécessaires dans cette structure (bloc 5).

Le résultat de cette phase est l'entrée d'un module de balisage automatique (bloc 6) dont la tâche consiste à reconnaître la structure logique spécifique à partir des informations stockées dans la « définition de type de document » (DTD) (bloc 9). Cette dernière est équivalente à la structure logique générique et contient des renseignements généraux sur le type du document. La DTD utilisée pour le balisage logique du texte de la figure 1.2 et le texte balisé résultant, sont montrés dans les figures 1.3 et 1.4, respectivement. Nous avons présenté, pour la première fois dans le domaine de traitement de documents manuscrits, une DTD conforme à SGML[5]. Ce dernier est une norme de ISO[6], présentée dans (SGML, 1986) et conçue principalement pour la

[5] Standard Generalized Markup Language
[6] International Standard Organization

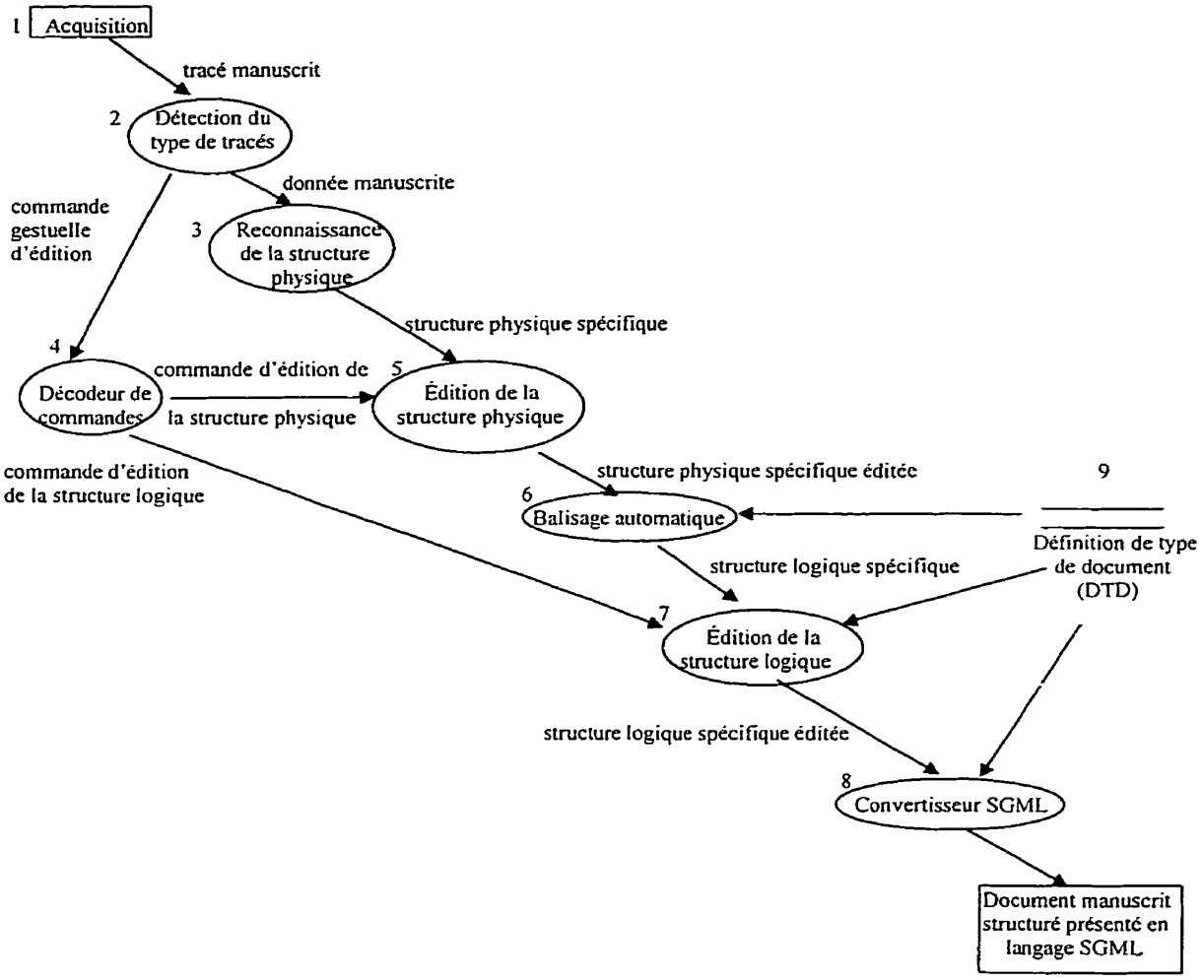



Figure 1.1. Diagramme de flux de données du système complet de traitement de documents manuscrits structurés, présenté dans (Moin, 1997).

transmission de documents structurés typographiques via le réseau Internet. Les détails de ce type de DTD sont expliqués dans (Moin, 1997).

La structure logique spécifique peut être éditée dans un module conçu à cet effet (bloc 7). Pour que le résultat obtenu soit conforme à une norme internationale de présentation de documents, nous avons ajouté un convertisseur SGML (bloc 8). Toujours pour la première fois dans le domaine du traitement de documents structurés, nous avons présenté la version SGML d'un document manuscrit dans (Moin, 1997). La

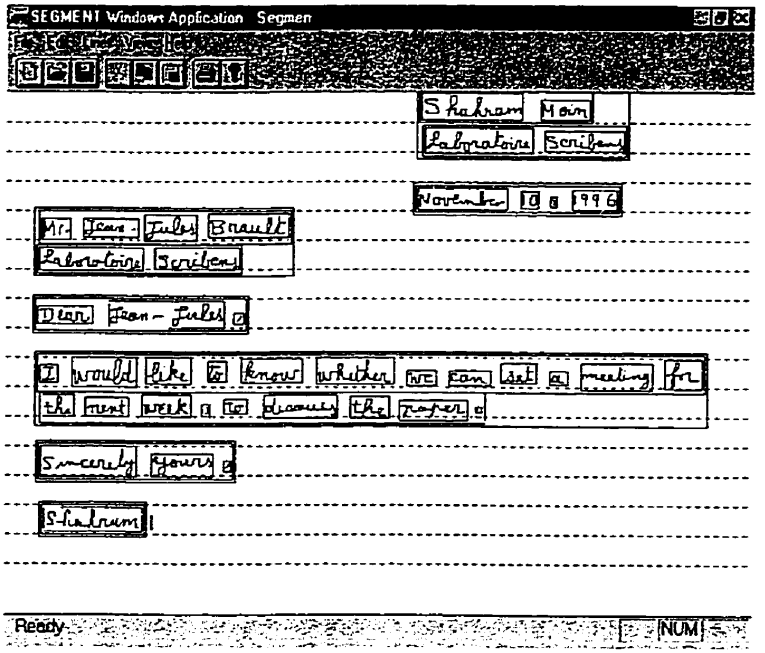



Figure 1.2. Objets physiques d'un texte manuscrit structuré. Les blocs, les lignes et les mots sont identifiés par leurs rectangles englobant (d'après (Moin, 1997)).

```
<!-- DTD for a simple memoranda in the file "memo.dat" -->
<!--          ELEMENTS        MIN CONTENT -->
<!ELEMENT    MEMO            --   (FROM, DATE, TO, SALUT, BODY, FERME, SIGN) >
<!ELEMENT   (FROM|TO)        --   (NAME, ADDRESS) >
<!ELEMENT   (NAME|ADRESSE)   --    EMPTY >
<!ELEMENT   (DATE|SALUT)     --    EMPTY >
<!ELEMENT    BODY       --   PARA+    >
<!ELEMENT    PARA            -O    EMPTY >
<!ELEMENT   (FERME|SIGN)     --    EMPTY >
<!--         ELEMENT     ATTRIBUT-NAME    DECLARED-VALUE    DEFAULT-VALUE -->
<!ATTLIST     FROM        LAY_TYPE           CDATA             #FIXED "bloc" >
                NUM_LINES_LEARN    CDATA              #FIXED "avr1,var1" >
                      WIDTH_LEARN        CDATA             #FIXED "avr2,var2" >
                      POSITION_LEARN     CDATA             #FIXED "avr3, var3" >
<!ATTLIST    NAME          LAY_TYPE           CDATA             #FIXED "line" >
                           NUM_WORDS_LEARN    CDATA            #FIXED "avr4,var4" >
                           FILE                ENTITY          #IMPLIED >
:
<!ATTLIST    BODY         LAY_TYPE     CDATA              #FIXED "blocks" >
                          NUM_BLCKS_LEARN    CDATA                #FIXED "avr5,var5" >
:
:
<!--      ENTITY_NAME  SYS_ID_KEYWORD    SYSTEM_ID     DATA_TYPE     NOTATION_NAME -->
<!ENTITY    NAME1          SYSTEM "name1.hnd"       NDATA       handwritten     >
:
:
<!--         NOTATION_NAME    SYSTEM_ID_KEYWORD      INTERPRETER_PROGRAM  -->
<!NOTATION   handwritten      SYSTEM                "script.exe" >
```

Figure 1.3. Définition de type de document (DTD) utilisée pour le balisage logique du texte de la figure 1.2 (d'après (Moin, 1997)).

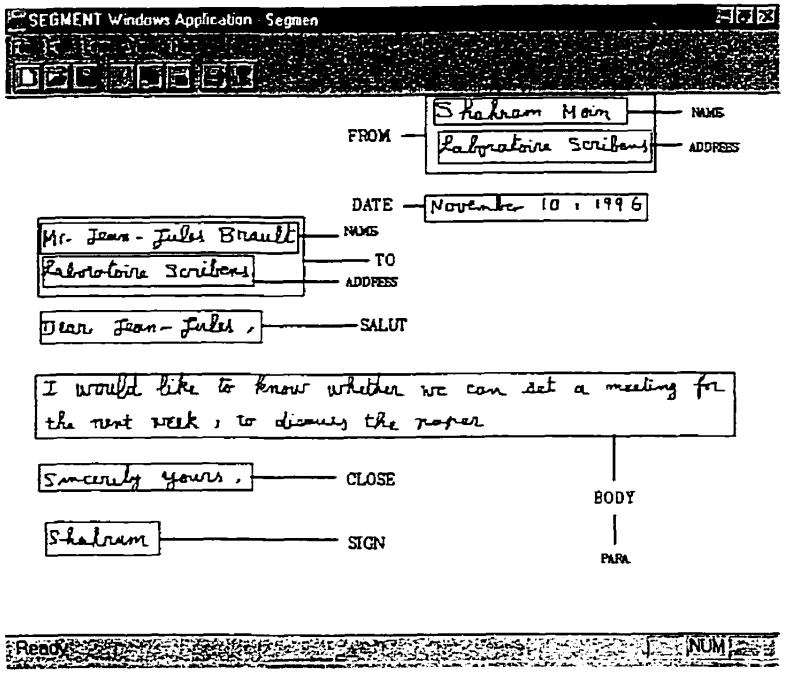



Figure 1.4. Résultat du balisage logique du texte de la figure 1.2 avec la DTD de la figure 1.3. Les étiquettes logiques sont montrées dans cette figure (d'après (Moin, 1997)).

```
<!DOCTYPE    Memo        SYSTEM    "memo.dtd" [
<!-- document type declaration subset including entity -->
<!-- declaration for optional elements of type PARA    -->
<!ENTITY PARA1 SYSTEM "para1.hnd" NDATA handwritten    >]>
<MEMO>
<FROM> <NAME FILE = NAME1> <ADDRESS FILE = ADD1> </FROM>
<DATE FILE = DATA1>
<TO> <NAME FILE = NAME2> <ADDRESS FILE = ADD2> </TO>
<SALUT FILE = SALUT1> <BODY> <PARA FILE = PARA1> </BODY>
<CLOSE FILE = CLOSE1> <SIGN FILE = SIGN1>
</MEMO>
```

Figure 1.5. Instance SGML du document de la figure 1.4 (d'après (Moin, 1997)).

figure 1.5 montre la sortie du convertisseur dans le cas du texte balisé de la figure 1.4.

Les structures de documents sont utiles pour les opérations telles que la transmission, le stockage et la reconnaissance de caractères. En fonction du type logique des éléments d'un document, on peut utiliser les méthodes appropriées de compression et de reproduction des entités manuscrites. En plus, si la reconnaissance de caractères est

nécessaire dans un système donné, les structures de document permettent d'améliorer la performance de la procédure de reconnaissance. Une segmentation précise de texte en mots, lequelle est une partie du module de reconnaissance de la structure physique, réduit le nombre d'erreurs de reconnaissance. En plus, si l'étiquette d'un mot dans la structure logique est reconnue, la procédure de reconnaissance pourra utiliser un dictionnaire plus compact et plus précis pour reconnaître les mots; ce qui permettra d'améliorer la performance du système, en ce qui a trait au temps d'exécution et à la précision de reconnaissance de caractères et de mots.

Nous avons présenté dans (Moin, 1997) nos propositions pour la conception et la réalisation des modules de la figure 1.1. Dans cette thèse, l'emphase a été mise sur les blocs de reconnaissance et de l'édition de la structure physique de documents manuscrits. Nous en parlerons davantage dans les sections suivantes.

## 1.3 RECONNAISSANCE DE LA STRUCTURE PHYSIQUE

### 1.3.1. IMPORTANCE DE RECONNAISSANCE DE LA STRUCTURE PHYSIQUE

La reconnaissance de la structure physique joue un rôle primordial dans le système de la figure 1.1. Une simple erreur dans cette phase pourrait produire de multiples résultats erronés dans la phase du balisage automatique. Bien que l'éditeur de la structure physique permette à l'utilisateur de corriger les erreurs probables de la phase de reconnaissance, il n'est pas souhaitable de toujours compter sur son intervention pour ce faire, puisqu'un nombre élevé d'erreurs peut rendre la tâche d'édition très fastidieuse pour l'utilisateur.

### 1.3.2. DIFFICULTÉS DE RECONNAISSANCE DE LA STRUCTURE PHYSIQUE

La difficulté principale de la reconnaissance de la structure physique réside en un compromis entre la liberté des scripteurs, lors de la composition de documents, et en des règles qu'ils doivent respecter pour produire un document structuré. La solution consiste à établir des contraintes à la fois minimes et fonctionnelles. Nous avons décidé de n'avoir qu'une seule contrainte dans notre système, soit la composition des lignes de texte entre les lignes horizontales pré dessinées. Il s'agit d'une contrainte tolérable pour le scripteur, puisqu'elle lui donne la sensation d'écrire sur une feuille lignée régulière. En plus, il peut ajuster la distance entre les lignes horizontales de son choix. Ces lignes, dont on parlera davantage dans le chapitre 3, sont les guides visuels utilisés dans la procédure de reconnaissance de la structure physique pour distinguer deux entités physiques principales : les lignes et les blocs. Tous les tracés, se trouvant entre deux lignes horizontales, appartiennent à une ligne de document. Les blocs sont, à leur tour, séparés par une ou des lignes vides.

Il reste donc à repérer les mots afin de compléter l'arborescence de la structure physique. Pour ce faire, il faut segmenter chaque ligne de texte en mots, en regroupant les tracés faisant partie de la ligne.

### 1.3.3. SEGMENTATION D'UNE LIGNE MANUSCRITE EN MOTS ET SES DIFFICULTÉS

La segmentation d'une ligne de texte manuscrit en mots constitue, à la fois, la phase la plus importante et la plus compliquée de la procédure de reconnaissance de la structure physique. Nous avons ainsi décidé de concentrer principalement notre travail sur la résolution de ce problème. Dans cette section, nous essayerons de souligner les difficultés reliées à cette opération.

Dans notre système en-ligne de segmentation, on peut envisager deux grands types de difficultés respectivement reliées à la conception du système et aux critères que l'on doit respecter pour que les scripteurs soient à l'aise lorsqu'ils utilisent le système.

Le premier type de difficultés réside, principalement, en un choix de caractéristiques pertinentes et en la conception d'un classificateur puissant. Quant aux critères des scripteurs, on peut citer la liberté de composition, le temps de traitement et la(les) tâche(s) des scripteurs. Nous essayerons d'aborder, dans les sections suivantes, ces deux types de difficultés; lesquelles sont d'ailleurs, sous certains aspects, inter reliées.

### 1.3.3.1. DIFFICULTÉS DANS LA CONCEPTION DU SYSTÈME

La difficulté principale est reliée à *l'interprétation des espacements inter-tracés*. Entre chaque levée de crayon et la pause qui la suit, on doit prendre une décision sur le type d'espacement (transition) afin de déterminer si le tracé commençant par la pause de crayon doit être ajouté à un mot du texte (l'espacement intra-mot) ou si l'on doit en construire un nouveau mot (l'espacement inter-mots). Autrement dit, il faut tenter d'interpréter le geste du scripteur à l'aide d'informations spatiales et temporelles disponibles. Ce n'est pas une tâche facile, puisque cela nécessite d'abord de faire une ou plusieurs traductions fiables du geste sous forme de mesures d'espacement et d'autres caractéristiques extraites du texte et ensuite de trouver une ou des relations entre ces mesures et caractéristiques afin d'établir une stratégie de discrimination des espacements.

Autrement dit, un des problèmes à résoudre consiste à trouver une ou plusieurs mesures appropriées d'espacement inter-tracés. Il faut également penser à introduire l'effet des autres paramètres comme la grandeur de l'écriture et la longueur des mots (en terme de nombre de caractères) dans la décision à prendre et, pour ce faire, il faut d'abord déterminer comment on peut les estimer à partir des informations disponibles.

Un autre problème consiste à *concevoir une méthode de classification* pouvant discriminer précisément les espacements inter-tracés à partir des caractéristiques extraites. Ce problème est, à la base, relié au choix d'un seuil de décision, qui peut être, en général, imposé et fixé, estimé et fixé ou adapté. Le classificateur idéal est celui dont

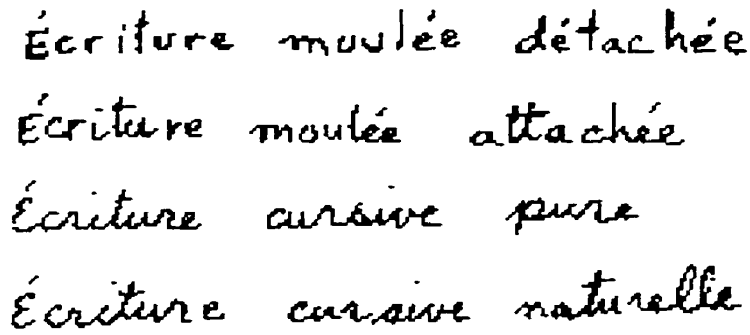



Figure 1.6. Différents types de l'écriture manuscrite (d'après (Tappert, 1984)).

les phases d'apprentissage et de test sont rapides et qui nécessite peu d'exemples pour l'apprentissage.

La méthode de classification choisie doit, aussi, être adaptative pour tenir compte des *variations de type d'écriture* d'un scripteur à l'autre (approche multi-scripteur) ou pour le même scripteur dans les différentes situations (approche mono-scripteur). Ces divers types d'écriture sont montrés à la figure 1.2. Il s'agit des écritures « moulée ou isolée détachée », « moulée ou isolée attachée », « cursive pure » et « cursive naturelle ».

À tout cela il faut ajouter *le nombre de cas possibles d'espacements* à envisager lorsque l'on parle d'un système en-ligne et pratiquement sans contrainte et d'un texte de longueur variable et composé d'un nombre très élevé de tracés et de mots.

## 1.3.3.2. CRITÈRES À RESPECTER POUR LE SCRIPTEUR

Notre objectif consiste à donner aux scripteurs une *liberté maximale* pendant la composition de leurs textes. Ils peuvent ainsi choisir librement le temps et l'ordre de composition des divers tracés manuscrits, sans avoir à respecter des contraintes de séquence de mots ou d'accents. Le prix à payer pour le concepteur du système d'acquisition et d'édition consiste à traiter les différents cas possibles de composition de

texte lors de la conception de l'algorithme de segmentation. Cet aspect du traitement de l'écriture est plus complexe que le traitement de parole. En fait, pour composer un mot en le prononçant, la séquence des composantes est invariable alors que pour écrire le même mot, il existe plusieurs séquences possibles; par exemple, écrire les accents et les points tout de suite après les caractères ou écrire tous les caractères et ajouter à la fin les accents et les points.

En plus de la liberté pour la composition de son texte, l'utilisateur de notre système de segmentation souhaite avoir une *minimum de tâches à effectuer* (à l'exception de la composition de son texte) et obtenir, en même temps, de bons résultats de segmentation. Toutefois et comme nous allons le démontrer, cela nous amène à un compromis constituant une autre difficulté de notre travail.

En fait, le meilleur scénario à envisager consiste à demander au scripteur de composer son texte, sans effectuer d'autres tâches, et de s'attendre à ce que le système fasse une segmentation parfaite et instantanée; ce qui n'est malheureusement pas toujours le cas, et cela, principalement, en raison de la liberté du scripteur pour composer le texte. Il est donc très probable que, pour améliorer sa performance, l'interface demande au scripteur de l'aider avant ou après la composition de son texte : avant, pour ajuster ses paramètres avec les particularités de l'écriture du scripteur et après pour corriger ses erreurs de segmentation.

D'un côté, pour minimiser la tâche reliée à l'entraînement du système, il faut, autant que possible, réduire la taille de la base d'apprentissage. Cette dernière est composée d'exemples extraites des échantillons de texte écrits et segmentés manuellement par le scripteur.

D'un autre côté, afin de minimiser la tâche reliée à la correction des erreurs, il faut construire un système de segmentation puissant; ce qui nécessite un entraînement bien réussi du système. Pour cela, on doit accéder, en général, à une quantité considérable d'exemples d'apprentissage. Par conséquent, une des difficultés du travail réside dans le compromis entre les qualités d'apprentissage et de segmentation automatique, en d'autres mots, la quantité de travail du scripteur dans ces deux phases.

Le troisième critère est *le temps des opérations* dans un système en-ligne. Dans un contexte interactif, le scripteur souhaite que le système apprenne rapidement les particularités des espacements inter-tracés et qu'il segmente instantanément son texte en mots. Il préfère ainsi un système rapide à un système lent, même si ce denier produit un nombre d'erreurs légèrement inférieur. Cela démontre que les méthodes conçues uniquement pour les systèmes hors-ligne ne sont pas utiles dans notre système.

Dans le chapitre 2, nous présenterons les diverses méthodes de segmentation existantes et évaluerons leurs performances en fonction, entre autres, des critères présentés dans cette section. Nous démontrerons également qu'aucune de ces méthodes n'est capable de répondre à tous les critères et qu'il est, par conséquent, essentiel de concevoir une nouvelle méthode de segmentation en-ligne de texte manuscrits en mots.

## 1.4 APERÇU DU SYSTÈME CONÇU ET IMPLANTÉ

La figure 1.7 montre le schéma général du système que nous avons conçu et implanté pour l'entraînement et le test des différents modules de segmentation de textes manuscrits en mots. Ces modules de segmentation sont des classificateurs dont les paramètres libres sont modifiés dans la phase d'entraînement afin de minimiser l'erreur de segmentation. La phase de test permet d'évaluer le fonctionnement du classificateur obtenu dans la phase d'entraînement. Les détails de ces phases, ainsi que d'autres modules du système seront présentés dans les chapitres suivants. Il faut noter qu'après avoir entraîné et testé un classificateur, on peut l'utiliser dans une autre phase constituant la phase de segmentation automatique (voir la figure 1.8).

Dans la figure 1.7, la tâche des deux premiers modules consiste à acquérir des données manuscrites. Le premier est le matériel d'acquisition permettant au scripteur

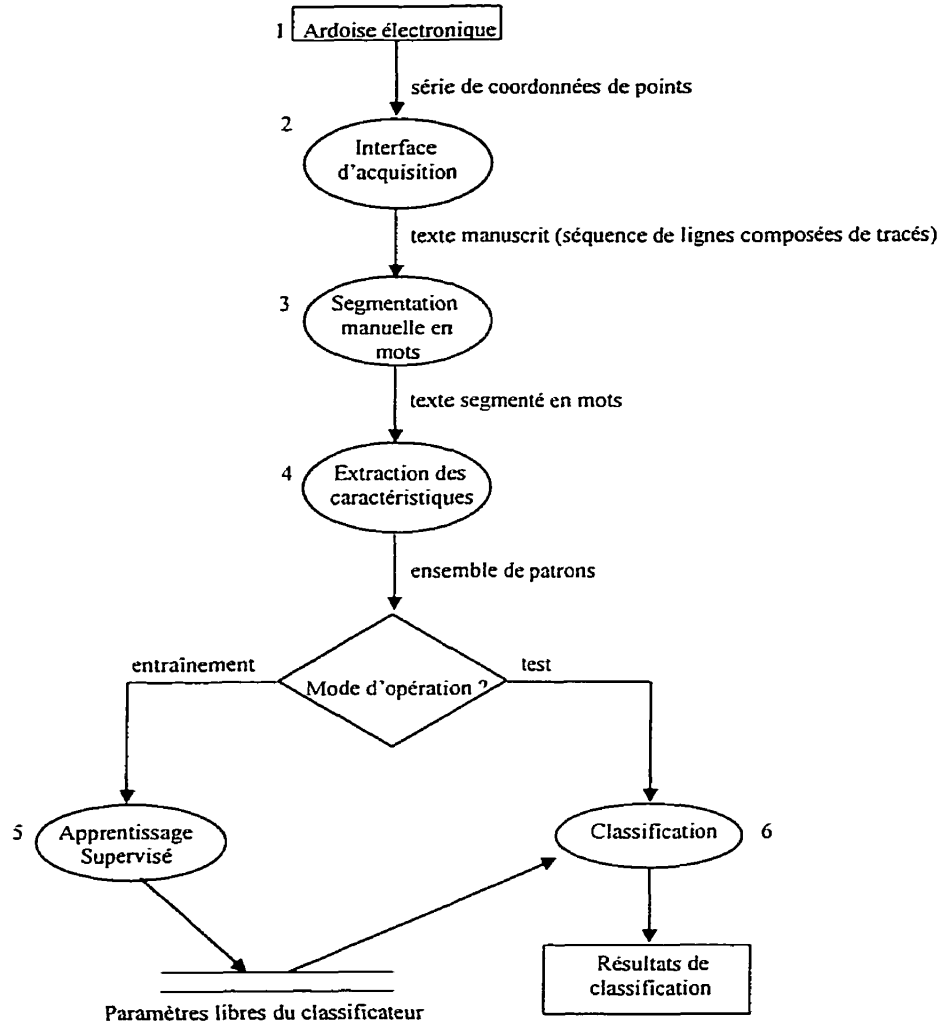



Figure 1.7. Schéma général du système de segmentation de textes manuscrits en mots, modes d'opération d'entraînement et de test.

d'entrer le texte manuscrit via un crayon spécial. La sortie de ce module est une séquence de coordonnées de points traitées par l'interface d'acquisition (module 2). Cette dernière est un logiciel regroupant les points acquis pour en construire les tracés et produire la présentation interne du texte, sous forme de séquence de lignes.

Les modules 3 et 4 permettent de construire l'ensemble de patrons d'apprentissage ou de test. Chaque patron est composé d'une ou de plusieurs

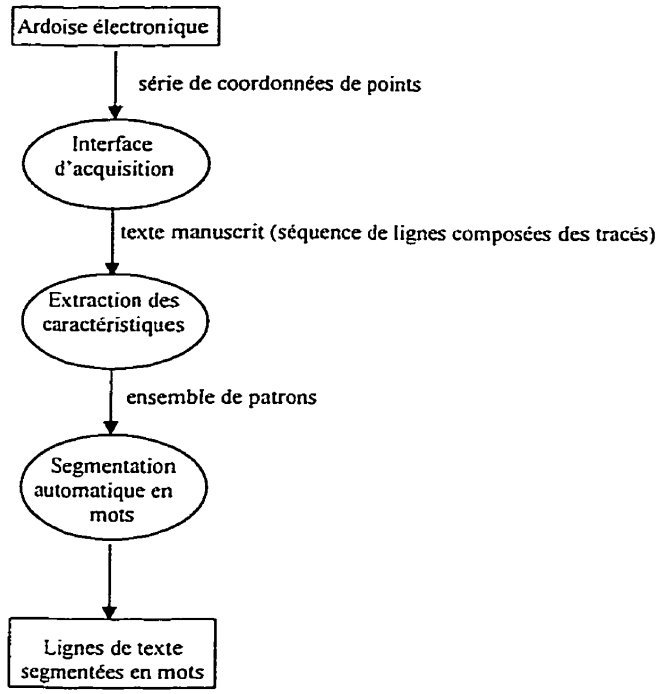



Figure 1.8. Diagramme des différents blocs du mode de segmentation automatique en mots.

caractéristiques et d'une classe. L'interface de segmentation manuelle (module 3) affiche le texte à l'écran et permet à l'utilisateur de le segmenter en mots. Le but de cette opération consiste à déterminer la classe (intra ou inter-mot) des patrons qui seront utilisés dans les phases d'apprentissage (module 5) ou de test (module 6). Dans le module 4, on extrait les caractéristiques nécessaires pour compléter l'ensemble des patrons.

Les patrons extraits peuvent être utilisés pour ajuster les paramètres libres d'un classificateur de distances inter-tracés dans une phase d'apprentissage supervisé (module 5). Le classificateur obtenu (module 6) peut ensuite discriminer les patrons appartenant, soit au texte duquel les patrons d'apprentissage sont extraits, soit à un autre texte. Le but consiste à évaluer la capacité de généralisation du classificateur, c'est-à-

dire calculer le taux de classification correcte des patrons qui ne sont pas utilisés dans la phase d'entraînement.

Dans ce projet, afin de déterminer les meilleures caractéristiques et d'améliorer la performance de segmentation de textes manuscrits en mots, nous avons utilisé différents types de caractéristiques (module 4), d'algorithme d'apprentissage (module 5) et de méthodes de classification (module 6). Les modules 1, 2 et 4 seront détaillés dans le chapitre 3, le module 3 dans le chapitre 4 et les modules 5 et 6 dans les chapitres 4 et 5.

## 1.5 PLAN DE THÈSE

Cette thèse est organisée de la façon suivante. Dans le deuxième chapitre, nous présenterons une étude bibliographique des méthodes de segmentation de textes manuscrits, en mettant l'accent sur les caractéristiques utilisées. Nous analyserons également ces caractéristiques et en soulignerons les points forts et faibles, pour finalement choisir les meilleures caractéristiques à utiliser dans une application en-ligne comme la nôtre.

Le chapitre 3 est consacré à l'extraction des caractéristiques pour la segmentation de textes manuscrits en mots. Nous y présenterons les détails des phases d'acquisition et d'extraction des caractéristiques. Deux groupes de caractéristiques seront étudiées dans ce chapitre : celles choisies dans le chapitre 2 et celles que nous proposons pour résoudre les lacunes de ces dernières.

Le chapitre 4 présente les classificateurs neuronaux implantés pour la segmentation de textes manuscrits en mots. Il comprend principalement la théorie de réseaux neuronaux artificiels, la description des classificateurs neuronaux implantés, les résultats de la classification obtenus avec les caractéristiques conventionnelles et proposées et une analyse des résultats obtenus.

Afin de résoudre les problèmes de segmentation que les classificateurs neuronaux ordinaires du chapitre 4 ne peuvent traiter, nous présenterons, dans le

chapitre 5, la deuxième catégorie des classificateurs implantés pour la segmentation de textes manuscrits en mots. Il s'agit des classificateurs basés sur l'algorithme AdaBoost et composés de plusieurs classificateurs simples. Nous y aborderons la théorie d'AdaBoost, les classificateurs implantés et les résultats obtenus.

Dans le chapitre 6, nous analyserons les résultats obtenus avec les classificateurs AdaBoost et présenterons également une analyse et une discussion générale sur la performance des classificateurs neuronaux et AdaBoost.

Le chapitre 7 est la conclusion générale. L'annexe A présente les textes manuscrits utilisés dans le projet et l'annexe B comprend la définition des caractéristiques et des mesures utilisées. L'annexe C est consacré à notre éditeur de structure physique, servant à acquérir un texte manuscrit, à reconnaître sa structure physique et à permettre également à l'utilisateur de l'éditer via un crayon. Finalement l'annexe D présente les résultats de segmentation obtenus avec les classificateurs K plus proches voisins.

# CHAPITRE 2

# MÉTHODES DE SEGMENTATION DE TEXTES MANUSCRITS EN MOTS

La segmentation de textes en mots est une étape primordiale dans un système de traitement de documents manuscrits. Il s'agit de regrouper les tracés (dans le cas d'un document acquis de façon en-ligne) ou les composantes connexes (dans le cas d'un document acquis de façon hors-ligne) d'un texte manuscrit afin d'en construire des mots. Quand on parle d'un mot, on parle d'une entité sémantique ayant une signification (au moins dans un langue). Toutefois, la plupart des méthodes de segmentation d'écriture manuscrite en mots ne se servent pas de la sémantique pour former les mots. En conséquence, le mot «mot» doit être redéfini dans le contexte du traitement automatique de l'écriture. On peut dire qu'un mot est formé de l'ensemble des tracés manuscrits, ou des composantes connexes, les uns suffisamment proches des autres. Alors, pour décider si deux tracés ou deux composantes connexes appartiennent à un mot, il faut d'abord déterminer leur degré de proximité. Ainsi, deux tracés font partie du même mot, si la distance entre eux est inférieure à un seuil. On peut alors se poser les questions suivantes:

1- Comment peut-on mesurer la distance entre deux tracés?
2- Comment doit-on prendre une décision sur leur proximité? En d'autres mots, comment déterminer la valeur du seuil de décision?

Afin de répondre à ces questions, nous avons effectué une étude bibliographique sur les diverses méthodes de segmentation de textes manuscrits en mots.

Dans ce chapitre, nous proposerons d'abord une taxinomie perceptuelle des méthodes de segmentation de textes manuscrits en mots. Après avoir présenté ces méthodes, nous les analyserons en soulignant leurs points forts et faibles ainsi que leurs cas problématiques. À la lumière des résultats de cette analyse, nous allons faire une discussion sur les méthodes de segmentation et présenterons notre conclusion.

## 2.1 TAXINOMIE DES MÉTHODES DE SEGMENTATION

Afin de segmenter un texte manuscrit en mots, il faut prendre une décision sur le degré de proximité des tracés. Il s'agit donc d'un problème de classification dont deux principales étapes consistent à choisir : 1- les caractéristiques à mesurer et 2- la méthode de classification. Les caractéristiques choisies doivent représenter la distance entre les tracés ou les composantes connexes et la méthode de classification doit utiliser ces caractéristiques afin de discriminer les distances intra-mot des distances inter-mots.

Le choix des caractéristiques joue un rôle très important dans la procédure de classification. La plupart des articles publiés ne se sont concentrés que sur cet aspect du problème. Par conséquent, la taxinomie perceptuelle des méthodes de segmentation présentée dans ce chapitre, est basée principalement sur le choix des caractéristiques, c'est-à-dire la façon dont chaque méthode interprète la distance ou l'espacement entre les tracés. Toutefois, nous discuterons du problème de classification dans les deux dernières sections de ce chapitre.

Nous classifions les méthodes de segmentation de l'écriture manuscrite en mots, en trois grandes catégories. : *spatiale*, *temporelle* et *spatio-temporelle*. La Figure 2.1 illustre cette taxinomie. Les méthodes spatiales sont celles qui interprètent l'espacement spatiale entre les tracés pour mesurer leurs distances. Les méthodes temporelles utilisent le temps entre la fin d'un tracé et le début de l'autre, et les méthodes spatio-temporelles se servent à la fois des distances spatiale et temporelle pour effectuer la segmentation.

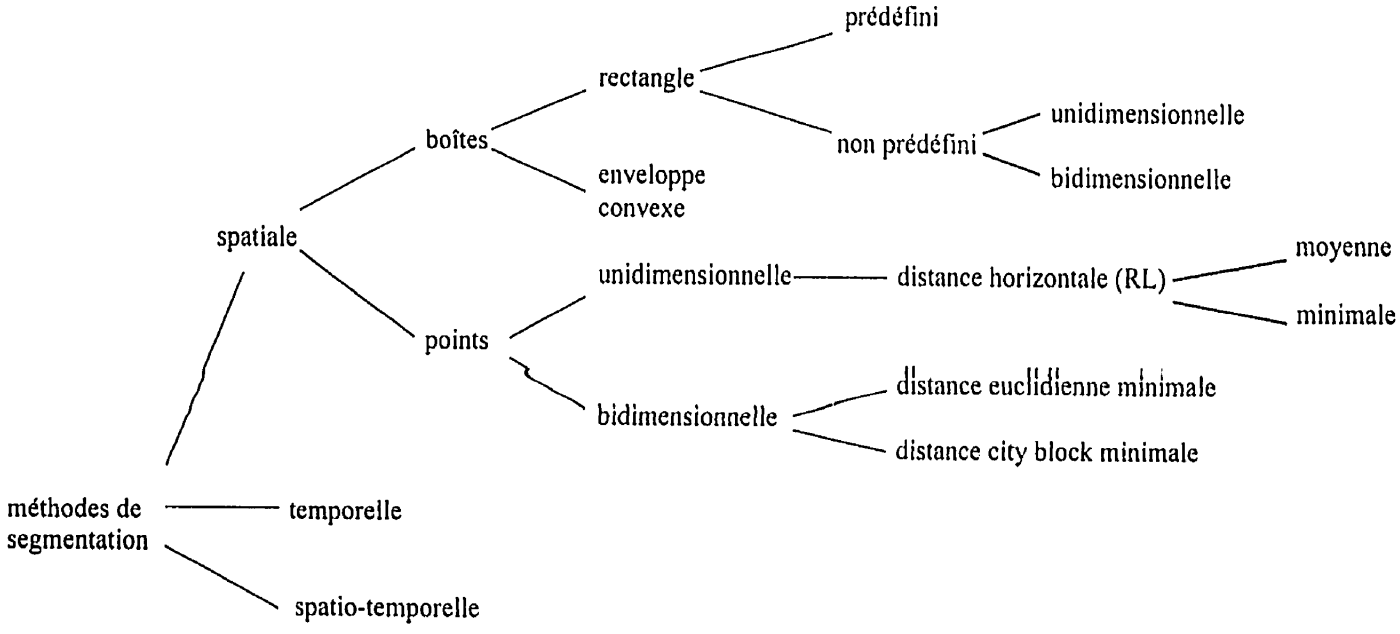



Figure 2.1 Taxinomie perceptuelle des méthodes de segmentation de l'écriture manuscrite en mots.

## 2.1.1 MÉTHODES SPATIALES

Il s'agit des méthodes les plus utilisées dans les systèmes *hors-lignes*[1] et *en-lignes*[2]. Cette catégorie des méthodes de segmentation interprète l'espacement spatial entre deux tracés pour mesurer leur distance. Les méthodes spatiales utilisent, en général, les deux métriques spatiales suivantes: *distance entre boîtes* et *distance entre points*. Dans le premier cas, chaque tracé ou composante connexe est représenté par une forme (géométrique ou non-géométrique) englobant tous ses points. Cette forme est appelée *la boîte* du tracé. La distance entre les boîtes représente donc la distance entre les tracés. En ce qui concerne la deuxième métrique, la (les) distance(s) entre deux ou plusieurs points représentatifs des composantes connexes ou des tracés est (sont) utilisée(s) pour mesurer la distance entre les tracés. A cette étape, nous présentons les différentes méthodes utilisant ces métriques de distance inter-tracés.

### 2.1.1.1 DISTANCE ENTRE BOÎTES

Dans les méthodes de segmentation de textes en mots, basées sur la distance entre boîtes, deux différentes formes de boîtes ont été utilisées: rectangle et polygone.

#### *2.1.1.1.1 DISTANCE ENTRE RECTANGLES*[3]

Les tracés ou composantes connexes sont représentés par des rectangles qui les englobent. Les dimensions des rectangles peuvent être prédéfinies ou calculées à partir des points des tracés.

---

[1] Dans un système hors-ligne ou statique, on ne dispose que des informations spatiales des données (souvent sous forme d'une image bit-map) et on effectue la segmentation, après l'acquisition de tous les tracés ou le balayage de l'image du texte au complet. C'est le cas des systèmes de reconnaissance optique des caractères (OCR).

[2] Dans un système en-ligne ou dynamique, la segmentation est effectuée après l'acquisition de chaque tracé. Outre les informations spatiales, les informations temporelles sont également disponibles. C'est le cas des blocs-notes électroniques.

[3] Bounding-Box(BB) en anglais

***Rectangles prédéfinis***

La Figure 2.2 montre la méthode de segmentation de *caractères en boîtes*. Le scripteur doit écrire les lettres dans les rectangles prédéfinis. Tous les tracés entrés dans un rectangle appartiennent à une lettre, et un rectangle vide est interprété comme un espace inter-mots. L'algorithme utilisé pour regrouper les tracés en lettres et en mots est très simple et si le scripteur respecte la contrainte imposée, l'erreur de segmentation est pratiquement inexistante. Toutefois, la contrainte imposée ralentit la vitesse d'acquisition de l'écriture et risque d'être fastidieuse, puisqu'elle est loin d'être naturelle (Suen, 1979), (Fox, 1987), (Tappert, 1986) et (Tappert, 1990).

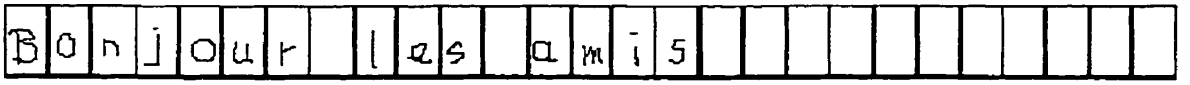


Figure 2.2. Segmentation de caractères moulés en boîtes.

***Rectangles non prédéfinis***

Si les rectangles ne sont pas prédéfinis, il faut d'abord les déterminer à partir des coordonnées des points des tracés. Pour chaque tracé, il s'agit de trouver le rectangle ayant une surface minimale. La deuxième étape consiste à calculer la distance entre les rectangles. Cette distance pourrait être unidimensionnelle (distance horizontale) ou bidimensionnelle (distances horizontale et verticale). Et finalement, en fonction de grandeurs de distance(s), il faut prendre une décision sur le type de distance : intra-mot ou inter-mots.

Dans un premier temps, nous réviserons les méthodes unidirectionnelles et ensuite, nous présenterons les méthodes bidimensionnelles basées sur le calcul de distance entre rectangles.

**Distance unidimensionnelle**

Les méthodes basées sur la distance unidimensionnelle utilisent l'équation suivante pour calculer la distance horizontale entre les rectangles:

$$Dist\_BB = x_{\min D} - x_{\max G} \tag{2.1}$$

où :

*Dist_BB* est la distance horizontale entre les rectangles,

$x_{minD}$ est la coordonnée x de point(s) à l'extrême gauche de tracé situé à droite et

$x_{maxG}$ est la coordonnée x de point(s) à l'extrême droite de tracé situé à gauche.

La Figure 2.3 montre les rectangles et la distance horizontale entre les rectangles pour deux composantes connexes.

Le calcul de distance entre rectangles est suivi d'une décision sur son type en fonction de sa grandeur. Pour ce faire, en général, un seuil $S$ est utilisé de la façon suivante:

Si $Dist\_BB \geq S$ , alors *Dist_BB* est une distance inter-mots (segmentation en mots) ou inter-caractères (segmentation en caractères);

Sinon, *Dist_BB* est une distance intra-mot (segmentation en mots) ou intra-caractères (segmentation en caractères). (2.2)

Dans certaines méthodes, la valeur de $S$ est arbitraire et imposée au scripteur (voir les explications données dans le chapitre 1 sur les différentes façons de détermination du seuil). Il s'agit des méthodes de *segmentation de caractères moulés*

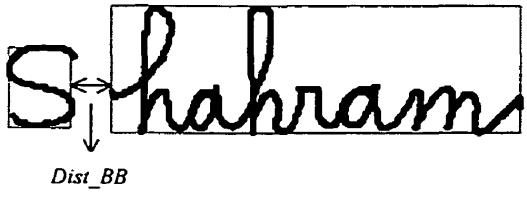



Figure 2.3. Rectangles de deux composantes connexes et leur distance.

*détachés* (Tappert, 1984) et *segmentation de mots purement cursifs* (Tappert, 1982) où le scripteur doit laisser suffisamment d'espace entre les tracés de deux lettres différentes (le cas des caractères moulés) ou deux mots différents (le cas des mots cursifs). Puisque l'utilisateur n'est pas obligé d'écrire dans des rectangles, la segmentation de caractères moulés détachés est à la fois moins précise et moins contraignante que celle de caractères en boîtes (Figure 2.2). Cependant, les résultats des expériences présentés dans (Tappert, 1984) et (Fox, 1987) montrent que les scripteurs trouvent intolérable la contrainte d'écrire les caractères de manière séparée. La contrainte d'écrire les mots cursifs purs espacés a été jugée plus tolérable (Tappert, 1982) et (Fox , 1987).

(Welbourn, 1987), (Welbourn, 1990), (Dimitriadis, 1995) et (Kimura, 1995) déterminent la valeur de $S$ dans la phase d'apprentissage à partir de l'histogramme de distances entre les tracés. Cet histogramme, qui est filtré dans une phase de «lissage»[4], comprend, en général, deux sommets. Le premier correspond à des distances intra-mot et le second à celles d'inter-mots. La valeur du seuil est choisie à un point entre ces deux sommets. Par exemple, dans (Welbourn, 1987) le point situé à égale distance des deux sommets a été retenu.

L'algorithme proposé dans (Leroy, 1994) utilise également l'histogramme de distances entre les rectangles pour déterminer le seuil S. La méthode est basée sur le nombre d'extrema (minima et maxima verticaux) dans un texte. Le seuil $S$ est choisi de sorte qu'après la segmentation en mots, le nombre moyen d'extrema par mot soit immédiatement supérieur au nombre théorique moyen d'extrema par mot . Selon (Leroy, 1994), la longueur moyenne des mots dans des phrases différentes ainsi que le nombre d'extrema dans un texte pour différents scripteurs sont relativement stables. Il est donc possible de déterminer le nombre théorique moyen d'extrema par mot. La première étape de la méthode consiste à construire la liste d'extrema de chaque tracé. Cette dernière

[4] "Smoothing" en anglais.

comprend le premier point, la liste des minima et des maxima (en respectant une alternance entre minima et maxima) et le dernier point du tracé, sous la forme suivante:

$$liste\_extrema = \{[(x_0,y_0)(x_i,y_i)(x_n,y_n)] \mid (\forall i(y_{i-1} < y_i \Rightarrow y_i > y_{i+1}) \vee (y_{i-1} > y_i \Rightarrow y_i < y_{i+1}))\} \quad (2.3)$$

où: $(x_0,y_0)$, $(x_i,y_i)$ et $(x_n,y_n)$ sont ,respectivement, les coordonnées du premier point, d'un extrema et du dernier point. À partir des listes d'extrema des tracés, on construit la liste d'extrema du texte. L'algorithme d'évaluation du seuil $S$ commence par le calcul du nombre d'extrema ainsi que l'histogramme des distances entre les tracés dans un texte donné. Ensuite, on effectue la segmentation en utilisant, comme seuil, différentes valeurs de distances dans l'histogramme et on calcule le nombre de mots et le nombre moyen d'extrema par mot pour chaque valeur du seuil. Finalement, on choisit le seuil pour lequel le nombre moyen d'extrema par mot est immédiatement supérieur au nombre théorique moyen d'extrema par mot. Il faut ajouter que ce dernier, ainsi que le seuil de distance inter-mots, sont propres à chaque scripteur.

La méthode utilisée dans (Oulhadj, 1994) ne se sert pas de l'histogramme des distances entre traits pour trouver la valeur du seuil $S$. Cet algorithme prend comme hypothèse que ce seuil est proportionnel à la largeur des caractères:

$$S \propto (x_{\max-caractere} - x_{\min-caractere}) \quad (2.4)$$

où: $S$ est le seuil des distances inter-mots et $x_{\max-caractere}$ et $x_{\min-caractere}$ sont, respectivement, les valeurs maximale et minimale des coordonnées $x$ des points d'un caractère donné. Pour déterminer les valeurs de $x_{\max-caractere}$ et $x_{\min-caractere}$, il faut d'abord reconnaître les caractères. Si l'étape de segmentation en mots (ou en caractères) précède l'étape de la reconnaissance de caractères, la distance horizontale moyenne entre deux minima ou maxima consécutifs est considérée comme une valeur proportionnelle à la largeur des caractères et donc une estimation pour la valeur du seuil $S$. La Figure 2.4 montre les distances entre les extrema verticaux utilisées dans cette méthode. La

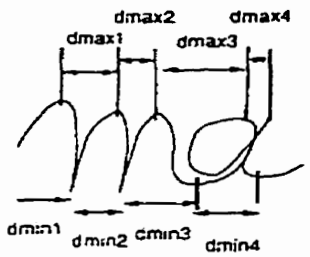



Figure 2.4. Distances entre les extrema verticaux (d'après (Oulhadj, 1994)).

distance moyenne entre les extrema verticaux est calculée selon l'équation suivante:

$$PAS = \frac{\sum d_{\min} + \sum d_{\max}}{n + m} \tag{2.5}$$

où: $d_{\min}$ est la distance entre deux minima consécutifs, $d_{\max}$ est la distance entre deux maxima consécutifs, $n$ est le nombre de minima et $m$ est le nombre de maxima. Le seuil de distance inter-mots est calculé à partir de l'équation suivante:

$$S = PAS + Ecart \tag{2.6}$$

où: $Ecart$ est l'écart type de $PAS$, calculé de la façon suivante :

$$Ecart = \left( \frac{\sum_{i=1}^{n} (PAS - d_{\min,i})^2 + \sum_{j=1}^{m} (PAS - d_{\max,j})^2}{n + m} \right)^{\frac{1}{2}} \tag{2.7}$$

$Ecart$ est considéré comme l'erreur tolérée sur le $PAS$.

**Distance bidimensionnelle**

Le principe de base des méthodes de segmentation bidimensionnelle consiste à calculer les distances horizontale et verticale entre tracés, et à les comparer avec des seuils prédéterminés pour ainsi décider de les segmenter dans une ou deux unités (caractère ou mot).

Dans le projet de «bloc-notes électronique» du laboratoire Scribens, à l'École Polytechnique de Montréal, la méthode de «distance entre rectangle» fut utilisée pour calculer la proximité des tracés (Guerfali, 1990), (Plamandon, 1990), (Plamandon, 1991) et (Nouboud, 1992). Pour classifier un tracé dans un mot, trois cas de proximité sont vérifiés: 1- la proximité à droite, 2- la proximité haute et 3- la proximité «autre». Le premier cas, qui est le cas le plus général d'un mot écrit de gauche à droite, est illustré dans la Figure 2.5 (a). On décide d'inclure le tracé dans le mot si:

$$dy \geq \frac{\min(ym, yc)}{6} \quad \text{et} \quad dx \leq \frac{\max(ym, yc)}{4} \tag{2.8}$$

La proximité haute est le cas des points sur les «i», les barres de «t» et les accents (la Figure 2.5(b)). La composante est incluse dans le mot si :

$$dy \leq \frac{\max(ym, yc)}{2} \quad \text{et} \quad (dx_1 \text{ ou } dx_2 \leq \max(ym, yc)\ ) \tag{2.9}$$

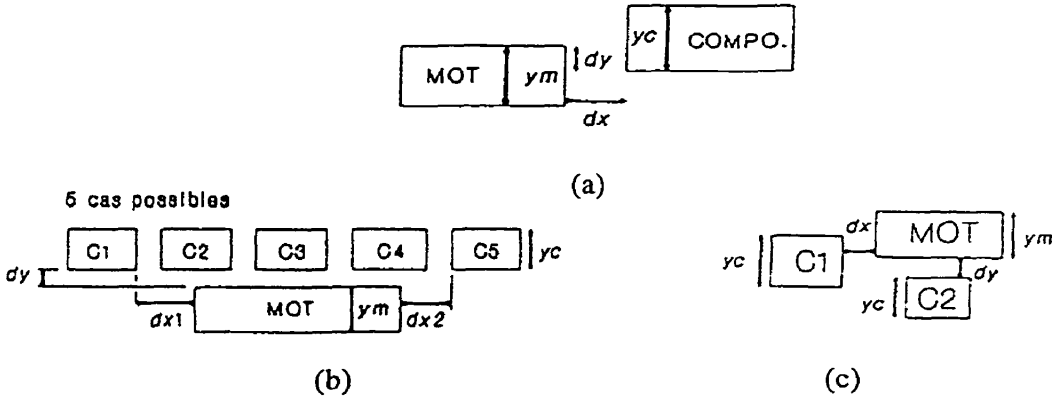



Figure 2.5. Trois cas de proximité: (a) proximité à droite, (b) proximité haute, (c) proximité «autre» (d'après (Guerfali, 1990)).

Les cas autres que les proximités à droite et haute comme les points dans les «? ! : ; » s'appellent la proximité «autre ». C'est également le cas des tracés assez proches dans un mot quelles que soient leurs positions. La proximité «autre» est illustrée dans la Figure 2.5(c). La condition d'appartenance d'un trait à un mot est la suivante:

$$dx \text{ et } dy \leq \frac{\max(yc, ym)}{6} \tag{2.10}$$

### *2.1.1.1.2 DISTANCE ENTRE ENVELOPPES CONVEXES*[5]

Cette méthode, présentée dans (Mahadevan, 1995), est basée sur le calcul de l'enveloppe convexe de composantes connexes. Il s'agit du polygone convexe avec surface minimale[6] englobant la composante. Pour calculer la distance entre les deux composantes, on trouve d'abord leurs enveloppes convexes (Cormen, 1990). Ensuite, on relie les centres de gravité des deux enveloppes. La distance entre les composantes sera alors égale à la distance euclidienne entre les points d'intersection des enveloppes convexes avec la ligne reliant les centres de gravité. La Figure 2.6 illustre les enveloppes convexes des composantes connexes ainsi que leur distance.

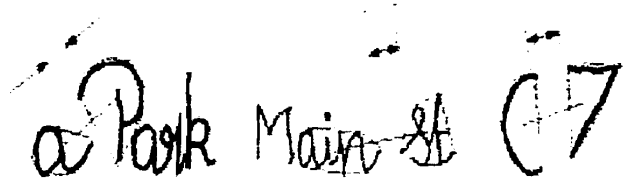



Figure 2.6. Distance entre coques convexes ( d'après (Mahadevan, 1995)).

[5] Convexe Hull (CH) en anglais
[6] En respectant la contrainte suivante : erreur d'approximation de surface < une valeur prédéfinie

## 2.1.1.2 DISTANCE ENTRE POINTS

Dans les méthodes de segmentation en mots, basées sur la distance entre points, la proximité de deux tracés est calculée en fonction de distance entre leurs points. Trois mesures différentes sont utilisées dans cette catégorie des méthodes de segmentation. Il s'agit de: *distance horizontale*[7] *(RL), distance euclidienne et distance city-block.* Ces mesures de distance entre tracés seront présentées dans cette section.

### *2.1.1.2.1 DISTANCE HORIZONTALE*

Afin de trouver la distance entre deux composantes connexes, on peut mesurer les distances horizontales entre les points de portions de deux composantes se chevauchant verticalement. La distance horizontale entre deux points est calculée de la façon suivante:

$$rl_{i,j} = |x_i - x_j| \quad \text{si} \quad y_i = y_j \tag{2.11}$$

où: $rl_{i,j}$ est la distance horizontale entre les points $(x_i, y_i)$ d'une composante et $(x_j, y_j)$ de l'autre. La Figure 2.7 montre les distances horizontales entre une paire de composantes connexes. Deux mesures de distance possibles entre deux composantes

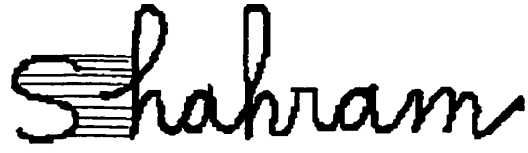


Figure 2.7. Distances horizontales entre une paire de composantes connexes.

[7] Run-length en anglais

sont «la valeur moyenne»[8] et «la valeur minimale»[9] des distances horizontales (Seni, 1994)[10].

### *2.1.1.2.2 DISTANCE EUCLIDIENNE*

La distance entre deux composantes connexes ou deux tracés peut être représentée par leur distance euclidienne minimale. Pour trouver cette distance, il faut d'abord calculer la distance euclidienne, entre toutes les paires de points appartenant à deux composantes différentes de la façon suivante:

$$r_{i,j} = \left((x_i - x_j)^2 + (y_i - y_j)^2\right)^{\frac{1}{2}} \tag{2.12}$$

où: $r_{i,j}$ est la distance euclidienne entre le point $(x_i, y_i)$ d'une composante et le point $(x_j, y_j)$ de l'autre composante. La distance entre les deux composantes est déterminée en trouvant la valeur minimale des distances euclidiennes :

$$Dist\_Eucl = \min(r_{i,j} \mid \forall(x_i, y_i) \,\&\, \forall(x_j, y_j)) \tag{2.13}$$

Cette mesure est montrée à la Figure 2.8. *Dist_Eucl* est une des mesure utilisées dans (Seni, 1994) pour estimer la distance entre les composantes connexes (sans expliquer la façon de calculer le seuil de décision).


Figure 2.8. Distance euclidienne entre deux composantes connexes.

[8] "average run-length"
[9] "minimum run-length"
[10] (Seni, 1994) ne présente pas les détails de calcul de seuil de décision des distances intra-mots et inter-mots.

#### *2.1.1.2.3 DISTANCE CITY BLOCK*

La distance city block minimale entre deux composantes connexes est une autre mesure de distance entre les composantes. Pour trouver cette distance, il faut d'abord calculer la distance city block entre chaque paire de points:

$$rcb_{ij} = |x_i - x_j| + |y_i - y_j| \tag{2.14}$$

où: $rcb_{i,j}$ est la distance city-block entre le point $(x_i, y_i)$ d'une composante et le point $(x_j, y_j)$ de l'autre composante. L'étape suivante consiste à trouver la valeur minimale des distances city-block calculées pour toutes les paires de points. Cette mesure de distance fut utilisée dans (Kutzberg , 1982) et (Fox, 1984)[11].

## 2.1.2 MÉTHODES TEMPORELLES

Les méthodes temporelles se servent des informations dynamiques pour effectuer la segmentation. Elles ne sont donc utilisables que dans les systèmes de segmentation en-lignes. Ces méthodes sont basées sur le calcul du temps écoulé entre la fin d'un tracé (levé de crayon) et le début de tracé suivant (posé de crayon). Si ce temps dépasse un seuil prédéterminé, les tracés seront classifiés dans deux unités différentes de l'écriture (mots ou caractères), sinon dans la même unité (Loy, 1982) .

## 2.1.3 MÉTHODES SPATIO-TEMPORELLES

Les méthodes spatio-temporelles utilisent les informations spatiales et temporelles pour effectuer la segmentation. L'algorithme proposé dans (Fox, 87) en est un exemple.

---

[11] Sans présenter les détails du calcul du seuil de décision des distances intra-mots et inter-mots.

Selon (Fox, 87) la segmentation spatiale est basée sur le fait que, en général, les mots sont séparés par un espace bidimensionnel d'une largeur raisonnable. L'utilisation des informations temporelles pourrait également être avantageuse, en considérant les points suivants: premièrement, les scripteurs prennent une pause entre deux mots et deuxièmement, la position de leur main est fixe lorsqu'ils écrivent un mot, tandis que pour écrire le mot suivant, ils déplacent la main; ce qui cause un délai entre les mots. En outre, les informations linguistiques sont utilisables en se basant sur le fait que les scripteurs ont tendance à compléter un mot avant de commencer le mot suivant. Ils commencent également à compléter un mot une fois que la composition de la partie principale du mot est terminée. Par exemple, le mot «dictée», écrit de façon cursive pure, comprend quatre parties: la partie principale ou le corps du mot, le point du «i», la barre du «t» et l'accent du «é». Or, en général, on complète un mot en y ajoutant le deuxième trait du «x», les points sur le «i» et le «j», la barre du «t» et les accents. Ce type de traits (les points, les barres, etc..) sont facilement détectables puisqu'il sont ajoutés avec un certain délai par rapport à la partie principale du mot. (Fox, 87) propose de combiner les informations spatiales, temporelles et linguistiques pour effectuer la segmentation en-ligne[12].

## 2.2 ANALYSE DES MÉTHODES DE SEGMENTATION DE TEXTES MANUSCRITS EN MOTS

Dans cette section, nous analyserons les différentes méthodes de segmentation présentées précédemment, en mettant l'accent sur leurs avantages et leurs lacunes. Pour ce faire, nous allons d'abord présenter les critères d'analyse et nous évaluerons chacune

[12] (Fox, 87) ne précise pas de quelle manière il faut combiner ces informations.

des méthodes par rapport à ces critères. À la lumière de cette analyse, afin de porter un jugement sur la robustesse des méthodes de segmentation, nous allons ressortir les points forts et faibles des méthodes et soulignerons particulièrement les cas problématiques.

### 2.2.1 CRITÈRES D'ÉVALUATION DES MÉTHODES DE SEGMENTATION

Les critères d'évaluation, proposés dans cette section, sont utilisables dans le contexte de la segmentation en-ligne des tracés en mots dans une ligne de texte manuscrite. Ces critères, présentés dans le tableau 2.1 sous forme de caractéristiques des méthodes de segmentation, sont les suivants :

1- temps de segmentation,

2- contrainte(s) imposée(s) au scripteur,

3- sensibilité aux paramètres de l'écriture.

Le premier critère envisageable pour évaluer la performance d'une méthode de segmentation est probablement le taux de segmentation réussi. Toutefois, la diversité des bases de données utilisées dans les différentes méthodes, nous empêche de l'utiliser. En plus, le taux de segmentation d'une méthode dépend du nombre de cas problématiques dans la base de données et ne pourrait pas être considéré comme une mesure absolue de sa performance. Or, s'il n'est pas impossible, il est très difficile de comparer scientifiquement les résultats numériques des méthodes de segmentation, sans les implanter toutes.

En ce qui concerne *le temps de segmentation*, la situation est plus claire. Ce critère pourrait être présenté par le nombre et l'ordre de complexité des opérations qu'une méthode doit effectuer, pendant la procédure de segmentation, sur les tracés impliquées.

Un autre critère d'analyse consiste à évaluer la présence, le nombre et la sévérité des *contraintes* imposées au scripteur. Ces dernières doivent idéalement être absentes afin d'offrir au scripteur, la possibilité d'une composition naturelle de son texte. Nous allons analyser les méthodes de segmentation en fonction de quatre différents types de contraintes: *encadrement, respect des distances intra/inter mots, temporelle* et *séquence des tracés*. Le critère d'encadrement concerne l'obligation du scripteur d'écrire dans un endroit prédéfini, tel que les rectangles. Quant au deuxième type de contraintes, nous vérifierons si, afin de discriminer les distances intra-mot des distances inter-mots, une méthode impose au scripteur de respecter un seuil prédéfini. Les deux premières contraintes sont applicables dans les systèmes en-ligne et hors-ligne. Les deux dernières ne sont toutefois applicables que dans les systèmes en-ligne, puisqu'elles concernent l'aspect dynamique de composition de textes. En imposant la contrainte temporelle, on force le scripteur à différencier le temps entre deux tracés d'un mot de celui entre deux tracés de deux mots différents. Certaines méthodes imposent au scripteur de respecter la séquence des tracés; par exemple de compléter la composition du mot actuel avant d'en composer un nouveau.

Nous allons utiliser un autre critère portant sur la *sensibilité* des méthodes sur certains paramètres de segmentation comme le décalage vertical, l'inclinaison, la séquence, le temps, les chevauchements horizontal et vertical des tracés et le type d'écriture. Une méthode est considérée sensible au décalage vertical, si sa décision sur le type de distances inter-tracés peut changer, en raison du décalage vertical d'un tracé par rapport à l'autre. En fait, nous rappelons que notre objectif, en ce qui concerne la segmentation de tracés en mots, consiste à évaluer la performance des méthodes de segmentation, en supposant que tous les tracés appartiennent à la même ligne de texte. Par conséquent, la distance verticale entre les tracés, qui est utilisée pour la segmentation de texte en lignes, ne doit pas intervenir dans la procédure de classification et une bonne méthode de segmentation de textes en mots ne doit pas être sensible au décalage vertical des tracés. De la même façon, si les résultats de segmentation d'une méthode donnée

changeraient en fonction de l'inclinaison du texte, de l'ordre de composition des tracés, du temps entre les tracés, des chevauchements horizontal et vertical des tracés et du type d'écriture (isolé, cursif ou mixte), cette méthode sera considérée sensible à ces paramètres.

Le tableau 2.1 montre les différentes caractéristiques des méthodes de segmentation basées sur des critères mentionnés ci-dessus. Un '×' démontre la présence d'une caractéristique. Nous utiliserons ce tableau pour faire une discussion sur les points forts et faibles des méthodes.

Tableau 2.1. Caractéristiques des méthodes de segmentation de textes en mots.

| Concept / Méthode | Temps de segmentation | Contrainte | | | |
|---|---|---|---|---|---|
| | | Spatiale : respect d' | | Temporelle : respect de | |
| | | un encadrement | un seuil spatial | un seuil temporel | séquence des tracés |
| 1. Caractères en boîtes | bas | × | | | |
| 2. Caractères et mots cursifs séparés | bas | | × | | |
| 3. Dist_BB (Welbourne , 1987) | bas | | | | |
| 4. Dist_BB (Leroy,1994) | élevé | | | | |
| 5. Dist_BB (Oulhadj, 1994) | moyen | | | | |
| 6. Dist_BB (Guerfali, 1990) | moyen | | | | |
| 7. Dist_Env_Conv (Mohadevan, 1995) | élevé | | | | |
| 8. Dist_Euclidienne (Seni, 1994) | élevé | | | | |
| 9. RL (Seni, 1994) | moyen | | | | |
| 10. City-block (Fox, 1984) | élevé | | | | |
| 11. Temporelle (Loy, 1982) | bas | | | × | × |
| 12. Spatio-temporelle (Fox, 1987) | moyen | | × | × | × |

Tableau 2.1(suite). Caractéristiques des méthodes de segmentation de textes en mots.

| Concept / Méthode | sensibilité à: | | | | | | |
|---|---|---|---|---|---|---|---|
| | décalage vertical | inclinaison de tracés | séquence de tracés | temps entre tracés | Chevauchement des tracés | | type d'écriture |
| | | | | | Horizontal | Vertical | |
| 1. Caractères en boîtes | | | | | | | × |
| 2. Caractères et mots cursifs séparés | | × | | | × | | × |
| 3. Dist_BB (Welbourne , 1987) | | × | | | × | | |
| 4. Dist_BB (Leroy,1994) | | × | | | × | | |
| 5. Dist_BB (Oulhadj, 1994) | | × | | | × | | |
| 6. Dist_BB (Guerfali, 1990) | × | × | | | × | × | |
| 7. Dist_Env_Conv (Mohadevan, 1995) | × | | | | | | |
| 8. Dist_Euclidienne (Seni, 1994) | × | | | | | | |
| 9. RL (Seni, 1994) | × | | | | | × | |
| 10. City-block (Fox, 1984) | × | | | | | | |
| 11. Temporelle (Loy, 1982) | | | × | × | | | |
| 12. Spatio-temporelle (Fox, 1987) | × | | × | × | | | |

## 2.2.2 POINTS FORTS ET FAIBLES DES MÉTHODES DE SEGMENTATION

Chaque méthode de segmentation a ses points forts et faibles. Dans cette sous-section, nous chercherons à les identifier en utilisant les critères d'évaluation présentés dans 2.2.1 et les informations disponibles dans le tableau 2.1.

Nous croyons qu'une méthode de segmentation aurait des faiblesses importantes, si sa performance dépendait du respect des contraintes par le scripteur. Un des objectifs que les interfaces crayon recherchent consiste à offrir au scripteur, autant que possible, la sensation d'écrire sur du papier ordinaire sans contraintes. D'autre part, une bonne méthode est celle qui est la moins sensible aux paramètres spécifiés dans la section 2.2.1, soit l'inclinaison, le décalage vertical et la séquence des tracés. Un autre avantage d'une méthode de segmentation est *la rapidité* de classification des distances, notamment dans les systèmes en-ligne.

Nous présenterons des exemples portant sur les cas problématiques des méthodes de segmentation. Étant donné la liberté offerte au scripteur pour composer son texte dans les systèmes sans contraintes, aucune méthode n'est, en pratique, sans erreur de segmentation. Il faut également ajouter, qu'établir une mesure précise et quantitative à cet égard, est impossible. Toutefois un algorithme est considéré bon si ses cas problématiques ne sont pas fréquents dans l'écriture typique. Nous présenterons donc des cas typiques de problèmes de segmentation des différentes méthodes, en utilisant des exemples.

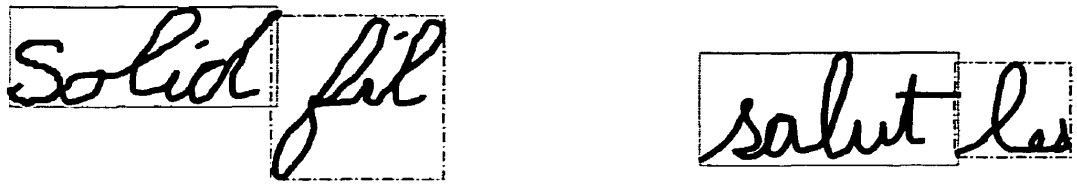
### 2.2.2.1 CARACTÈRES EN BOÎTES ET CARACTÈRES ET MOTS CURSIFS SÉPARÉS

**Points forts:** Les algorithmes de segmentation sont simples et rapides. En outre, si le scripteur respecte parfaitement les contraintes, le taux de succès est pratiquement de 100%.

**Points faibles:** Ces méthodes imposent des *contraintes* au scripteur. La segmentation de «caractères en boîtes» est loin d'être naturelle. En raison de la contrainte imposée, l'écriture est ralentie; ce qui est, en général, intolérable pour l'usager. La segmentation des caractères moulés détachés est à la fois moins contraignante et moins précise que celle des caractères en boîtes. Toutefois, la contrainte d'écrire les caractères de façon séparée n'est pas toujours agréable pour les usagers (Fox, 87). La méthode la moins contraignante et, par conséquent, la plus tolérable est la segmentation de mots cursifs purs et séparés. Bien que, grâce à la contrainte imposée, la précision de cet algorithme de segmentation soit élevée, il n'est pas souhaitable de demander à un usager, dont le type d'écriture habituelle est «cursif naturel» ou «moulé attaché», d'écrire de façon «cursive pure» et de respecter attentivement l'espacement entre les mots. La dépendance des méthodes à un type d'écriture spécifique est un autre point faible de ces méthodes.

### 2.2.2.2 DISTANCE ENTRE RECTANGLES (DIST_BB)

**Points forts:** aucune contrainte n'est imposée au scripteur. L'algorithme n'est pas sensible au décalage vertical, à la séquence, au temps, au chevauchement vertical des tracés ni au type d'écriture. Si la valeur du seuil inter-mots est déterminée pendant une phase d'apprentissage (comme dans (Welbourne, 1987)), la procédure de segmentation est pratiquement aussi simple et rapide que les méthodes ci-dessus mentionnées.



(a) (b)

Figure 2.9. Cas problématiques des méthodes de distance entre rectangles. (a) deux mots inclinés, (b) deux mots qui se chevauchent horizontalement.

**Points faibles:** comme toutes autres méthodes sans contraintes, le taux de segmentation n'est pas toujours parfait. Toutefois, cela ne doit pas être considéré comme la pire lacune de cette méthode, puisqu'elle est le prix à payer pour offrir une liberté au scripteur. Le problème le plus important des méthodes basées sur le calcul de distance entre rectangles est leur *sensibilité à l'inclinaison et au chevauchement horizontal* des tracés. La figure 2.9 montre deux cas problématiques reliés à ces deux points faibles.

Les deux mots de la figure 2.9(a) sont inclinés. En raison de cette inclinaison, les rectangles englobant les tracés se chevauchent sur l'axe des abscisses et la distance entre les rectangles est zéro. Par conséquent, une méthode basée sur la distance horizontale entre rectangles classifie erronément les tracés dans un mot. Les tracés de la figure 2.9(b) ne sont pas inclinés mais se chevauchent horizontalement; ce qui cause une erreur de classification. Les méthodes de segmentation, comme (Guerfali, 1990), utilisant la distance bidimensionnelle entre rectangles, souffrent, en plus, du problème de chevauchement vertical des tracés. La figure 2.10 en montre un cas problématique. Les

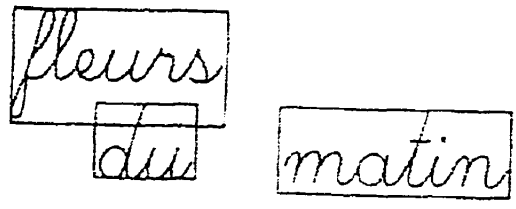



Figure 2.10. Un cas problématique de méthode de segmentation bidimensionnelle (d'après (Guerfali, 1990)).

tracés «fleurs» et «du» sont regroupés formant un mot dont la hauteur $ym$ (figure 2.5) est suffisamment grande pour l'inclusion erronée du tracé «matin» dans le même mot (selon l'équation (2.8)).

Nous allons considérer maintenant le critère de *la complexité du calcul du seuil inter-mots et sa précision.* La méthode présentée dans (Welbourne, 1987) est basée sur le calcul du seuil à partir de l'histogramme des distances intra-tracés. Ce dernier doit d'abord être lissé[13] pour éliminer le bruit et obtenir un histogramme bi-modal. Le seuil sera fixé à un point, à distances égales des deux sommets de l'histogramme. La complexité de ces traitements est moyenne par rapport aux autres méthodes de calcul du seuil. Quant à la précision de la valeur du seuil, puisque les formes des histogrammes (du point de vue de leurs étendus) n'interviennent pas dans le choix du seuil, la segmentation n'est pas optimale.

Les méthodes présentées dans (Leroy, 1994) et (Oulhadj, 1994) sont basées sur le calcul des extremas des tracés. La performance des méthodes dépend donc de la précision du calcul des extremas; ce qui nécessite une sorte de prétraitement, comme le filtrage des tracés pour les lisser et les éliminer (ou du moins les diminuer) la possibilité de détecter des extrema erronés en raison du bruit.

---

[13] "Smoothing" en anglais

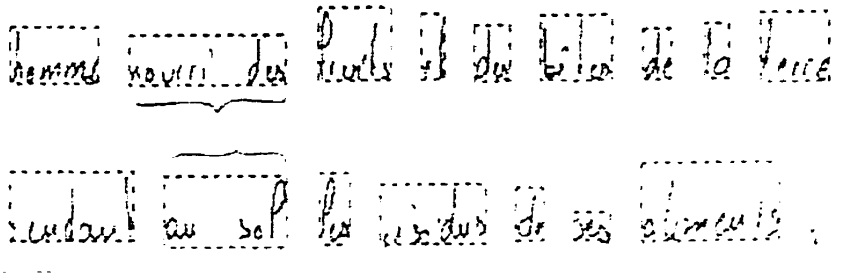


Figure 2.11. Un exemple d'erreurs de segmentation de la méthode de (Leroy, 94) .

Dans la méthode de (Leroy, 1994), le seuil est calculé en fonction du nombre d'extrema par mot; ce qui rend la procédure de segmentation adaptative mais plus compliquée. Plusieurs paramètres interviennent dans ce calcul. Une erreur dans la valeur de chacun de ces paramètres pourrait influencer le résultat de segmentation. Par exemple, la valeur moyenne de la longueur de mots sera différente selon qu'il s'agisse d'un texte littéraire ou scientifique. La précision de cette valeur comme un représentant de la longueur de mots dépend aussi de la valeur de l'écart type. En outre, le nombre théorique moyen d'extrema par mot est une fonction du nombre théorique moyen d'extrema par lettre. Même les mesures effectuées dans (Leroy, 94) montrent que ce dernier peut être différent des valeurs mesurées. Par exemple, le nombre théorique moyen d'extrema choisi pour la lettre «b» est 4, alors que dans un test effectué, 127 scripteurs l'ont écrite avec 4 extrema et 91 scripteurs avec 2. Un exemple de résultats de la segmentation d'un texte est illustré dans (Leroy, 1994). Dans ce texte, la plupart des mots sont écrits d'une façon bien séparés. Toutefois on peut observer des erreurs de segmentation, comme les mots collés «nourri du» et «au sol» dans la figure 2.11. Un autre problème de cette méthode : pour trouver le seuil final, il faut segmenter le texte

dans plusieurs itérations avec différentes valeurs de seuil, et à chaque itération, il faut également calculer le nombre moyen d'extrema par mot. La procédure pourrait donc être longue et inappropriée pour les systèmes en-ligne.

Selon la méthode de (Oulhadj, 1994), le seuil de distance inter-mots est égal à la somme de la distance moyenne entre extrema verticaux et son écart type. La proportionnalité du seuil de distance inter-mots à la largeur des caractères (qui est le principe de base de cette méthode) semble être raisonnable. Toutefois le choix de la valeur du seuil n'est pas clair. Un scripteur peut très simplement dépasser ce seuil lorsqu'il écrit deux traits d'un mot, surtout dans le cas des écritures moulée détachée et cursive naturelle. La valeur de l'écart type (équation (2.10)), qui est considérée comme l'erreur tolérée sur la distance moyenne entre extrema verticaux, pourrait être comparable à cette dernière. La diversité des valeurs des $d_{max}$ dans la figure 2.4 en est une preuve. Les distributions de $d_{max}$ et $d_{min}$ ne sont pas reconnues et ces distances risquent donc de ne pas être convenablement représentées par une moyenne et un écart type. Cette méthode requiert également beaucoup plus de calculs, que pour les deux dernières méthodes, surtout lorsque le tracé est long et que ses extrema sont nombreux.

Quant à la complexité du calcul du seuil inter-mots et sa précision dans le cas de la méthode de (Guerfali, 1990), nous rappelons que ce seuil n'est pas calculé dans une phase d'apprentissage, mais plutôt de façon en-ligne, en fonction du type de proximité et de la hauteur des tracés (équations (2.8), (2.9) et (2.10)). Le calcul du seuil n'est pas compliqué, mais ni l'utilisation des constantes dans les équations pour trouver le seuil, ni les valeurs choisies pour les constantes n'ont été justifiées. L'expression linéaire utilisée pour trouver le seuil, en fonction de la hauteur des tracés, pourrait provoquer des erreurs de segmentation si les tracés de deux mots différents se chevauchent verticalement, produisant ainsi erronément un seuil encore plus grand. Le résultat est donc l'enchaînement des erreurs de segmentation, en raison de l'augmentation itérative de la

valeur des seuils. Un autre problème de cette méthode: si le type de proximité n'est pas déterminé correctement, la décision prise sur le type de distance inter-tracés est très probablement erronée.

### 2.2.2.3 DISTANCE ENTRE ENVELOPPES CONVEXES

**points forts :** Il s'agit d'une méthode sans contrainte. Elle n'est pas sensible ni à la séquence des tracés, ni au type d'écriture. L'enveloppe convexe représente mieux la forme de tracé que le rectangle englobant. La distance entre enveloppes convexes de deux tracés se chevauchant horizontalement n'est pas nécessairement zéro, contrairement à la distance entre rectangles. Par exemple, dans la figure 2.6, la distance entre enveloppes convexes des tracés «a» et «Park» est différente de zéro; ce qui pourrait les classifier correctement dans deux mots différents. Tandis que, en raison du chevauchement horizontal de ces tracés, la distance entre les rectangles est zéro, et elle sera erronément classifiée comme une distance intra-mot.

**Points faibles :** Le plus grand problème de cette méthode est sa *complexité et le temps élevé du calcul* des enveloppes convexes et ensuite la distance entre les enveloppes; ce qui la rend inappropriée pour les applications en-ligne . L'algorithme doit parcourir tous les points de tracé pour trouver les points sommets candidats. À chaque point, plusieurs opérations coûteuses en temps doivent être effectuées, comme le calcul de la distance euclidienne entre le point actuel et un point de référence ainsi que le tri des données. Outre ces opérations qui donnent les points sommets de l'enveloppe convexe sur le tracé, la méthode de (Mahadevan, 1995) requiert une étape de filtrage pour réduire le nombre de sommets qui, selon la forme de tracé, peuvent être très nombreux. L'étape suivante consiste à trouver le centre de gravité de l'enveloppe convexe, suivie du calcul de la distance euclidienne entre les points d'intersection des

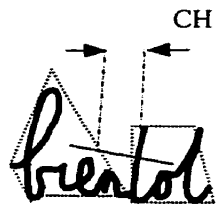



Figure 2.12. Un cas problématique de la méthode de distance entre enveloppes convexes(CH).

enveloppes convexes des deux tracés et les centres de gravité.

Une autre lacune de cette méthode la surestimation de la distance entre les tracés dont un exemple est montré à la figure 2.12. Les deux tracés «bien» et «tot» se chevauchent sur l'axe des abscisses et la distance entre eux devrait normalement être classifiée intra-mot, permettant ainsi de les regrouper dans un mot. Néanmoins, la méthode des enveloppes convexes mesure une distance relativement grande entre les tracés; ce qui risque de conduire la procédure de segmentation, vers une classification erronée inter-mots. La surestimation de la distance inter-tracés est causée par la nature globale de la méthode. En fait, lorsqu'un tracé est situé à droite d'un autre tracé, leur distance doit uniquement être une fonction de l'écart entre les points de l'extrême droite du tracé de gauche et ceux de l'extrême gauche du tracé de droite, et la forme des points dans les autres portions des tracés ne doit pas intervenir dans le calcul de la distance. Dans la méthode des enveloppes convexes, la forme de l'enveloppe et la position de son centre de gravité dépend de la forme globale du tracé. Par exemple, dans la figure 2.12, en raison des parties supérieurs des tracés «bien» et «tot», les centres de gravité sont situés au-dessus de la zone médiane, et la distance CH mesurée est nettement plus grande que les distances calculées par les méthodes de rectangles et de distance horizontale.

Finalement la méthode basée sur la distance entre les enveloppes convexes souffre du problème de la sensibilité au décalage vertical des tracés. Deux tracés écrits sur la même ligne et, en plus, faisant partie de même mot, risquent d'être classifiés dans

deux mots séparés, tout simplement en raison du décalage vertical d'un tracé par rapport à l'autre. Ce décalage pourrait augmenter la distance entre les enveloppes convexes, produisant ainsi une classification erronée des distances inter-tracés (inter-mots dans ce cas).

### 2.2.2.4 DISTANCES EUCLIDIENNE MINIMALE ET CITY BLOCK MINIMALE

Nous examinerons ensemble les mesures des distances euclidienne minimale et city block minimale, puisqu'elles partagent, en général, les mêmes points faibles et forts. La raison principale de cette ressemblance est leur manière d'estimer la distance entre les tracés (équations (2.11) et (2.14)). Les deux méthodes calculent l'ensemble des distances bidimensionnelles entre les paires de points de tracés (un point de chaque tracé) et en choisissent le minimum. Toutefois, la distance euclidienne est plus compliquée mais plus précise que la distance city block.

**Points forts :** aucune contrainte n'est imposée au scripteur. Ces méthodes sont insensibles à la séquence, au temps et aux chevauchements horizontal et vertical des tracés ainsi qu'au type d'écriture manuscrite. Les algorithmes ne sont pas difficiles à implanter.

**Points faibles :** Le problème le plus important de ces méthodes est le nombre de calculs à effectuer pour trouver les distances euclidienne minimale et city block minimale entre deux tracés. Le temps de traitement est donc une fonction du nombre de points des tracés et si les tracés sont longs (comme le cas d'écriture purement cursive), les mesures seront très coûteuses en temps; ce qui les rend non convenables pour les applications en-lignes.

La sensibilité au décalage vertical des tracés est un autre problème de ces

méthodes de segmentation. Comme nous l'avons expliqué à la section précédente, un décalage vertical entre deux tracés du même mot, pourrait produire une augmentation de la valeur de la distance calculée et, par conséquent, une classification erronée.

### 2.2.2.5 DISTANCE HORIZONTALE

**Points forts :** Il s'agit d'une méthode sans contrainte. Elle est également insensible à l'inclinaison, à la séquence, au temps et au chevauchement vertical des tracés ainsi qu'au type d'écriture. Le calcul de la distance horizontale entre deux tracés est moins long en temps que les méthodes «euclidienne», «city block» et «enveloppes convexes». Le grand avantage de la mesure horizontale est sa nature locale; ce qui lui permet de représenter la distance entre les tracés, sans être influencée par des paramètres inappropriés. En d'autres mots, si les tracés se chevauchent verticalement, cette mesure représente la distance entre les portions les plus proches des deux tracés, sans tenir compte des formes des portions éloignées. Comme nous l'avons mentionné à la section 2.2.2.3, dans le cas de mesures globales comme la distance entre les enveloppes convexes, ces parties des tracés pourraient causer des mesures irréalistes de distances entre les tracés (figure 2.12).

**Points faibles :** Cette méthode souffre, en général, des problèmes suivants: la dépendance au chevauchement vertical, les distances horizontales irréalistes pour la phase de segmentation et la sensibilité au décalage vertical des tracés.

La distance horizontale entre deux tracés n'existe que s'ils se chevauchent verticalement. Dans le contexte de la segmentation de tracés d'une ligne de texte manuscrit en mots, normalement les parties principales des mots se chevauchent, et le problème n'est causé que par les accents et les points dans les mots n'ayant pas de partie supérieure (figure 2.13(a)).


(a) (b)

Figure 2.13. Deux cas problématiques de méthode de distance horizontale. (a) distance horizontale inexistante, (b) distance horizontale pouvant causer une classification erronée.

Ces deux entités d'écriture causent un autre problème pour la mesure de distance horizontale. En raison du décalage vertical des accents et des points par rapport à la zone médiane de mots, même si le chevauchement vertical existe entre un point ou un accent et un autre tracé (formant le corps de mot), la distance horizontale calculée risque d'être trompeuse pour la phase de classification. La figure 2.13(b) montre un exemple de ce type de problèmes causés par le décalage verticale des points et des accents.

Pour résoudre les deux premiers problèmes, il faut une étape de prétraitement afin de reconnaître les cas problématiques et les enlever de la procédure de classification.

La troisième catégorie de problèmes est de nature différente et pratiquement sans résolution. Un cas problématique, causé par la sensibilité de mesure de distance horizontale à décalage vertical des tracés, est illustré dans la figure 2.14. Un léger


(a) (b)

Figure 2.14. Augmentation significative de la distance horizontale entre les tracés, causée par un léger décalage vertical.

décalage vertical (de 2.14(a) à 2.14(b)) cause une augmentation significative dans la valeur de la distance horizontale minimale ($RL_{min}$). La valeur moyenne des distances horizontales ($RL_{avg}$) est également grandement affectée par ce décalage minime.

### 2.2.2.6 MÉTHODE TEMPORELLE

**Points forts :** Le plus grand avantage de la méthode temporelle est sa simplicité. Il suffit de mesurer le temps entre deux tracés et de le comparer avec un seuil pour les classifier dans une ou deux unités différentes d'écriture (caractères ou mots). Puisque le seul paramètre du système est le temps, les contraintes spatiales ainsi que la sensibilité de la méthode à des paramètres spatiaux sont absentes.

**Points faibles :** L'étude présentée dans (Hanaki, 1976) montre que le temps entre deux tracés n'est pas un paramètre fiable pour la segmentation. Selon cette étude, les pauses intra-unités sont, en général, plus courtes que les pauses inter-unités. Toutefois, si l'usager écrit vite et ne respecte pas le principe de la séparation temporelle, les temps inter-unités peuvent être aussi courts que les temps intra-unités. La seule façon d'éviter ce genre de chevauchement est de contraindre l'usager d'attendre au moins l'écoulement du seuil de temps inter-unités avant d'entrer une nouvelle unité. Le scripteur est également obligé de terminer la composition du mot actuel avant de commencer le mot suivant; ce qui l'empêche de retourner dans le texte pour compléter un mot. Ces deux contraintes pourrait ralentir la procédure de composition de texte et la rendre difficile pour le scripteur, puisqu'elles sont incompatibles avec ses habitudes naturelles.

### 2.2.2.7 MÉTHODE SPATIO-TEMPORELLE

**Point fort:** On peut imaginer qu'en ajoutant une dimension temporelle à la

dimension spatiale, la procédure de classification pourrait devenir plus performante.

**Points faibles :** le succès de cette méthode (Fox, 87) dépend du respect des seuils spatial et temporel entre les tracés, ainsi que de leur séquence. En fait, les contraintes spatiales sont ajoutées aux contraintes de la méthode temporelle, mentionnées dans la section 2.2.2.5. Il est donc évident, qu'en général, la liberté du scripteur pour garder son propre style de composition de texte n'est pas respectée, notamment du point de vue de ses habitudes temporelles et séquentielles.

## 2.3 DISCUSSION ET CONCLUSION

Dans les deux dernières sections de ce chapitre, nous avons présenté une taxinomie des méthodes de segmentation de textes manuscrits en mots et en avons également fait une analyse. Dans cette section, nous allons utiliser les résultats de cette analyse pour déterminer la (les) méthode(s) les plus robustes pour notre recherche. Nous allons également spécifier les points faibles des méthodes choisies pour introduire la suite des travaux consistant à améliorer leur performance.

Le critère le plus important dans notre choix de méthode porte sur l'absence de contraintes imposées au scripteur (en pire cas, présence d'un nombre minimal de contraintes justifiées). Les méthodes nécessitant l'imposition de contraintes non naturelles, et parfois intolérables pour le scripteur, ne répondent pas à ce critère et seront donc rejetées, même si leur taux de segmentation est parfait (évidemment à condition que le scripteur respecte les contraintes). Dans cette catégorie, on peut citer les méthodes suivantes : «caractères en rectangle», «caractères et mots cursifs séparés», «temporelle» et «spatio-temporelle» .

Vu que notre système est en-ligne, la vitesse de classification des distances est un facteur déterminant dans le choix de la méthode de segmentation. Comme nous l'avons mentionné à la section 2.2.2.4, les méthodes de distances «euclidienne» et «city block»

requérant des opérations simples mais très nombreuses (point par point) sont inappropriées pour la segmentation en-ligne. La méthode basée sur le calcul des distances entre «enveloppes convexes», ayant un nombre élevé d'étapes à suivre et d'opérations à effectuer pour calculer la distance entre les tracés, a le même problème. En plus, cette mesure est globale, résultant parfois de la surestimation des distances inter-tracés (voir la section 2.2.2.3).

Les mesures de «distance entre rectangles» (Dist_BB) et «distance horizontale» (RL) sont, d'après nous, les plus robustes pour la segmentation en-ligne de textes manuscrits en mots. Elles imposent aucune contrainte au scripteur et sont insensibles à la séquence et au temps entre les tracés ainsi qu'au type d'écriture. Leur autre grand avantage est la nature locale des mesures de distance : elles dépendent de la forme et de la position des portions des tracés qui sont en voisinage, plutôt que de celles qui sont loin les unes des autres.

Le calcul de Dist_BB est simple et rapide. Toutefois, cette mesure de distance souffre de problèmes de sensibilité à l'inclinaison et au chevauchement horizontal des tracés (figure 2.9). Nous devons donc trouver une solution pour ce problème, afin d'améliorer sa performance. Un autre point à considérer est l'algorithme de classification. Dans les sections 2.1.1.1 et 2.2.2.2, nous avons présenté les différentes stratégies proposées pour le calcul du seuil de classification des Dist_BB et nous y avons également fait une analyse critique. Cette analyse démontre l'imprécision et la complexité inutile des algorithmes de calcul du seuil proposés dans (Welbourne, 1987), (Oulhadj, 1993) et (Leroy, 1994). La méthode bidimensionnelle de (Guerfali, 1990), est moins compliquée, mais, en même temps, moins précise en ce qui concerne le calcul du seuil, et moins robuste. Nous avons donc à proposer une méthode de classification performante.

Le calcul de la distance horizontale RL nécessite des opérations plus compliquées que celle de la Dist_BB. Par contre, elle n'est pas insensible à l'inclinaison

ni au chevauchement horizontal des tracés. Comme nous l'avons mentionné à la section 2.2.2.5, le point faible principal de la RL est sa sensibilité au décalage et au chevauchement vertical des tracés (figure 2.13 et 2.14). Un autre défi que nous avons donc à relever consiste à minimiser le nombre de cas problématiques causés par cette sensibilité.

Dans le chapitre 3, nous allons présenter nos propositions pour améliorer la performance des méthodes Dist_BB et RL. La manière d'utiliser ces mesures dans la phase de classification, ainsi que les méthodes de classification proposées, seront abordées dans les chapitres 4 et 5.

# CHAPITRE 3

# ACQUISITION DES DONNÉES ET EXTRACTION DES CARACTÉRISTIQUES POUR LA SEGMENTATION DE TEXTES EN MOTS

Dans le chapitre 2, nous avons présenté les différentes méthodes de segmentation de textes manuscrits en mots. Nous avons également effectué une analyse comparative pour identifier les meilleures méthodes en soulignant leurs points forts et faibles. Comme nous l'avons mentionné, notre étude portait plutôt sur les caractéristiques extraites dans chaque méthode. Dans ce chapitre, à la lumière des résultats obtenus, nous allons présenter, parmi les caractéristiques présentées dans la littérature, celles que nous jugeons les plus robustes et nous expliquerons les étapes suivies pour les extraire dans notre projet. Nous allons également présenter nos propositions pour améliorer leur performance et nous expliquerons les détails d'extraction des caractéristiques proposées.

Ce chapitre est organisé de la façon suivante : la première partie du chapitre porte sur l'acquisition des données manuscrites où nous expliquerons les interfaces d'acquisition utilisées, l'échantillonnage des points et la structure interne des documents et des tracés manuscrits dans notre système. La deuxième section comprend les caractéristiques conventionnelles et les stratégies utilisées pour les extraire à partir des tracés manuscrits. Dans la troisième section, nous soulignerons les lacunes des caractéristiques conventionnelles. La quatrième section est consacrée à la présentation de nos propositions pour améliorer la performance de ces caractéristiques. Les étapes d'extraction des caractéristiques proposées seront également détaillées dans cette section. Et finalement la dernière section est la conclusion.

# 3.1 ACQUISITION DES DONNÉES MANUSCRITES

La première étape dans un système de traitement de textes manuscrits est l'acquisition des données manuscrites. Le scripteur utilise un crayon pour entrer le texte sur une surface particulière qui peut détecter la position du crayon. Cette position est capturée par un programme qui affiche le point à l'écran et qui construit un tracé manuscrit sous forme d'une suite de points. Deux parties essentielles pour l'acquisition des données manuscrites dans notre système sont les parties matérielle et logicielle qui seront présentées dans les sections suivantes.

## 3.1.1 MATÉRIEL D'ACQUISITION

Les tracés manuscrits sont les entités de base devant être traitées dans ce travail. Pour l'acquisition d'écriture manuscrite dans un système informatique, les dispositifs d'entrée suivants sont des candidats possibles : le photostyle[1], la tablette à numériser (opaque) et l'ardoise électronique[2] (encre électronique). Dans ce projet nous désirons donner au scripteur la sensation d'écrire avec un crayon ordinaire sur du papier. En d'autres mots, nous voulons lui offrir, dans un environnement informatique, la possibilité d'utiliser une encre électronique. Une tablette opaque ne pourrait être utilisée pour produire l'encre électronique, puisque les surfaces d'acquisition et d'affichage du texte sont différentes. Par contre, le photostyle et l'ardoise électronique simulent bien le paradigme papier/crayon , car la surface d'entrée est également la surface d'affichage et le texte est affiché instantanément. Nous avons donc testé ces deux dispositifs d'entrée des tracés manuscrits que nous allons présenter dans les sections suivantes.

### 3.1.1.1 PHOTOSTYLE

La figure 3.1 montre un photostyle. Ce dispositif peut jouer le rôle d'un crayon ou d'une souris. Il peut être connecté à tout ordinateur ayant un écran cathodique.

---

[1] LightPen en anglais
[2] Pen Computer en anglais

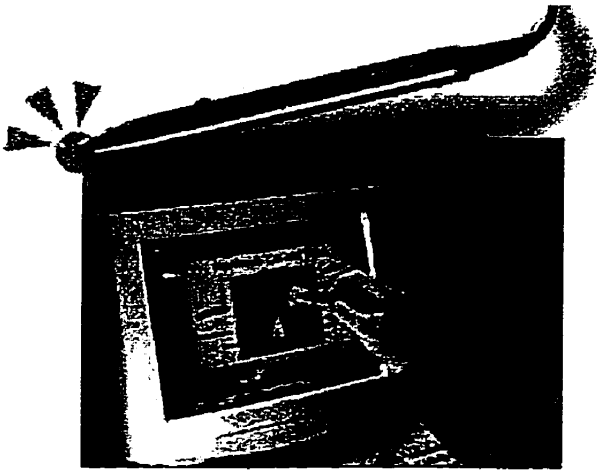


Figure 3.1. Photostyle utilisé comme un dispositif d'entrée

L'interface de connexion peut être interne ou externe. Dans le cas de l'interface externe, la connexion est établie via la porte série ou la porte du clavier. Le principe de base du fonctionnement du photostyle est le suivant : lorsque le crayon est suffisamment proche de l'écran, le rayon lumineux de balayage est capturé par le crayon. Cette lumière active le photodiode interne de crayon qui émet un signal électrique sous forme d'impulsion. La position du crayon sur l'écran est alors calculée à partir du temps d'émission du signal et de la durée d'une période de balayage( (Hatamian, 1985) et (Hatamian, 1987)). Le photostyle génère une coordonnée de point pour chaque cycle de balayage de l'écran qui, selon la configuration de la carte de vidéo du système, peut varier entre 60 et 85 Hz; ce qui représente la fréquence d'échantillonnage. La résolution spatiale d'échantillonnage dépend de la résolution de l'écran. Par exemple, si la résolution horizontale de l'écran est de 1280 points et la largeur de l'écran est de 14", la résolution linéaire sera égale à 91 points/pouce (1280/14).

Il faut ajouter, qu'en raison de sa résolution temporelle et de son taux d'échantillonnage relativement bas, le photostyle n'est pas le meilleur dispositif d'entrée

d'un texte manuscrit[3]; contrairement aux tablettes et aux ardoises électroniques faites spécialement pour cette tâche. Un autre problème du photostyle est que le scripteur doit écrire sur un écran vidéo presque vertical. Cela peut fatiguer le scripteur, notamment si le texte à rédiger est long. Toutefois le photostyle était utile dans notre travail, puisqu'il nous a permis de convertir un écran quelconque à une surface d'acquisition des données manuscrites. En outre, les textes que nous avons utilisés pour nos tests étaient courts; ce qui minimisait l'inconfort des scripteurs lors de la composition d'un texte. Plus encore et contrairement à certaines applications, comme la reconnaissance de caractères manuscrits[4], nous n'avions pas besoin d'une résolution spatiale et d'un taux d'échantillonnage très élevés pour capturer les détails minuscules de tracés. En fait, on s'intéressait à des espacements entre les tracés n'étant pas, en général, sensibles à ces détails.

### 3.1.1.2 ARDOISE ÉLECTRONIQUE

L'ardoise électronique que nous avons utilisée pour l'acquisition des textes manuscrits est la Stylistic 1200 de la compagnie Fujitsu (figure 3.2), dans laquelle le mécanisme de détection de la position du crayon sur l'écran est différent de celui du photostyle. L'écran de Stylistic 1200 joue le rôle d'une tablette électromagnétique. Il contient une grille de capteurs qui reçoivent des champs électromagnétiques émis par le crayon. La position du capteur le plus près du crayon (qui produit évidemment le signal électrique le plus fort) représente les coordonnées du crayon. Ce mécanisme

---

[3] Le photostyle est essentiellement un dispositif d'entée destiné à des tâches autres que dessiner et d'entrer des tracés manuscrits. Les résultats des études effectuées dans (Goodwin, 1975), (Struckman- Johnson, 1984), (Haller, 1984) et (Buxton, 1985) montrent qu'il est le dispositif le plus rapide et le plus précis parmi les dispositifs d'entrée pour les tâches telles que positionner, sélectionner et pointer des objets graphiques sur l'écran.

[4] Toutefois, la compagnie FastPoint (anciennement FTG Data System) a annoncé des résultats intéressants obtenus avec l'utilisation du photostyle dans le produit Caligrapher de Paragraph, qui est un module de reconnaissance d'écriture manuscrite ( pour plus de détails voir http://www.paragraph.com).

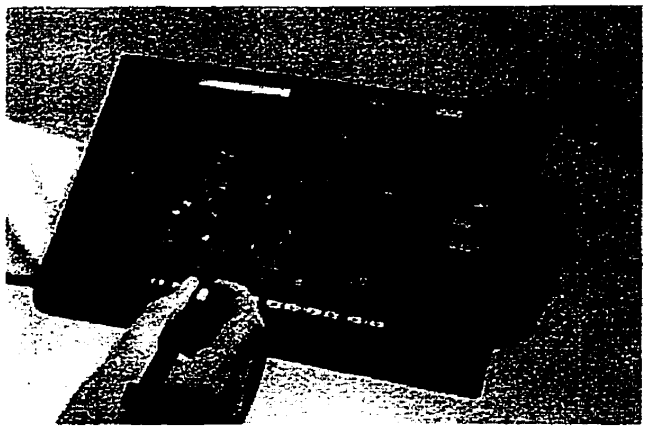


Figure 3.2. Stylistic 1200 de Fujitsu.

d'estimation de la position de crayon permet d'obtenir une résolution spatiale très élevée[5]. Dans le cas de Stylistic 1200, la résolution est de 1016 points/pouce, soit plus de dix fois celle du photostyle. Le taux d'échantillonnage des points est de 33 points/seconde. Les textes utilisés dans les étapes finales d'expérimentation de notre projet, ont été acquis avec Stylistic 1200.

## 3.1.2 LOGICIEL D'ACQUISITION

Une fois que les coordonnées des points sont générées par le matériel d'acquisition, les points doivent être affichés à l'écran. Il faut aussi les regrouper séquentiellement pour construire les tracés. Afin d'effectuer ces tâches, nous avons développé un logiciel d'acquisition qui est une application Windows écrite en langage orienté-objet Visual C++.

La figure 3.3 montre la structure des données dans le logiciel d'acquisition des entités manuscrites. L'entité principale de ce logiciel est un texte qui est un objet de type « document ». Ce dernier est composé d'une liste dynamique d'objet de type « tracé »

[5] Pour obtenir les renseignements supplémentaires sur le fonctionnement des ardoises électronique, vous pouvez vous référer à (Schomaker, 1998).

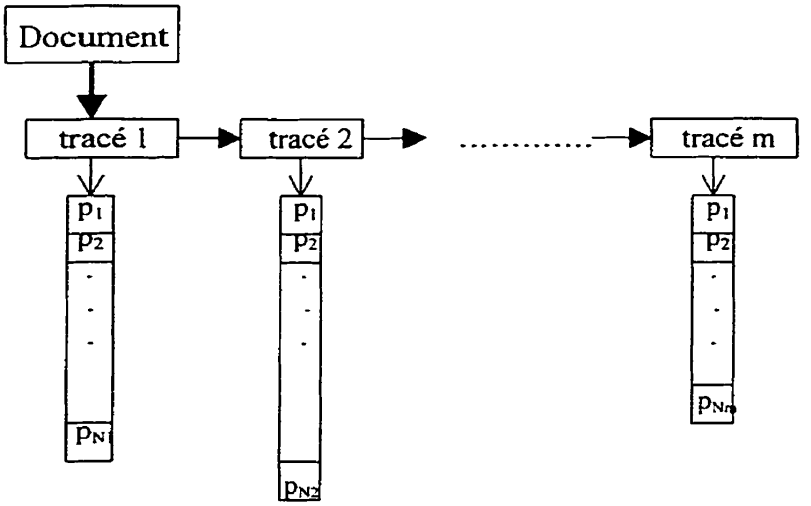



Figure 3.3. Représentation interne d'un texte manuscrit dans notre logiciel d'acquisition.

qui est à son tour construit d'une liste d'objet de type « point » ($p_1$, $p_2$, ...).

Les opérations effectuées dans le logiciel d'acquisition sont les suivantes :

1- Déterminer la distance entre les lignes (par le scripteur);
2- Acquérir les coordonnées des points pour chaque tracé;
3- Calculer les coordonnées relatives des points;
4- Mettre à jour les points extrêmes des tracés;
5- Afficher les tracés à l'écran;
6- Calculer le rectangle englobant les tracés;
7- Déterminer le numéro de la ligne des tracés;
8- Trier les tracés en ordre spatial;
9- Déterminer le dernier tracé de chaque ligne.

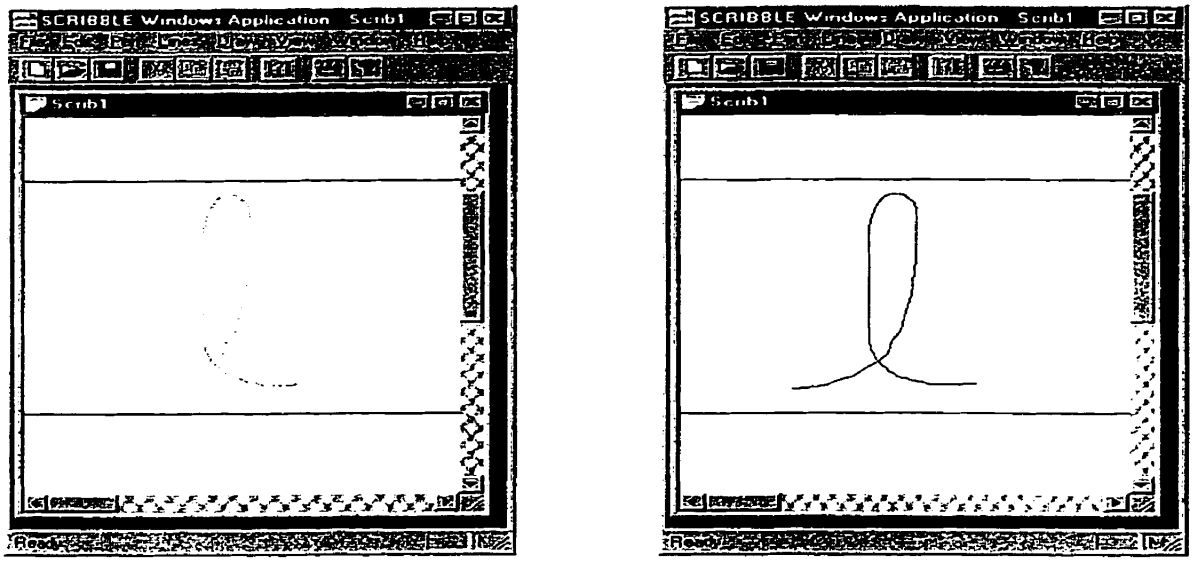
Dans les paragraphes suivants, nous détaillerons ces opérations.

L'interface d'acquisition affiche une fenêtre à l'écran et permet au scripteur d'écrire directement sur la surface d'acquisition. Avant d'entrer un texte, le scripteur doit choisir la distance entre les lignes horizontales divisant la fenêtre en plusieurs

régions. Dans chaque région, le scripteur peut entrer une ligne de texte. Il doit donc écrire dans l'espace entre deux lignes horizontales. Ceci constitue la seule contrainte imposée, qui est à la fois inévitable, utile et tolérable pour l'utilisateur. Inévitable, puisque sans ces lignes guides, l'obtention d'un texte structuré s'avère pratiquement impossible. On ne peut permettre au scripteur d'écrire de n'importe quelle façon, car le résultat pourrait plutôt ressembler à un dessin qu'à un texte. L'utilisation des lignes, qui est tout à fait conforme avec le paradigme crayon/papier et à laquelle les scripteurs sont habitués, aide à écrire proprement et, autant que possible, horizontalement. Nous devons toutefois ajouter que notre système tolère même les lignes penchées de texte, les grandeurs variables d'écriture dans une ligne et un pourcentage important de dépassements des tracés des limites des lignes horizontales (voir l'équation 3.1). En plus, il permet aux scripteurs de choisir la distance entre les lignes en fonction de la grandeur désirée de son texte. Ces lignes peuvent même être effacées et réaffichées, en tout temps, pendant la composition de texte.

Après avoir déterminé la distance entre les lignes horizontales, le scripteur peut entrer des tracés. Chaque tracé est composé d'une suite de points entre une pose et une levée de crayon. Dans le logiciel d'acquisition, les coordonnées de ce point sont calculées, à la pose du crayon dans la fenêtre d'acquisition. Il s'agit des seules coordonnées absolues du tracé. Le programme affiche ce point dans la fenêtre et continue de capturer les coordonnées des points subséquents en effectuant les trois opérations suivantes : calculer les coordonnées relatives de chaque point par rapport au point précédent, mise à jour des quatre points extrêmes de tracé (gauche, droit, haut et bas) et tracer une ligne entre le point précédent et le point actuel. Cette dernière opération permet d'afficher le tracé de façon continue (figure 3.4(b)) au lieu d'une suite de points isolés (figure 3.4(a)).

L'utilisation de coordonnées relatives a deux avantages : (1) compresser la quantité des données à sauvegarder grâce à la diminution de l'étendue des valeurs et (2) calculer les coordonnées des points du tracé en fonction du point initial ; ce qui facilite




(a) (b)

Figure 3.4. Un tracé acquis et affiché dans l'interface d'acquisition de textes manuscrits : (a) les points échantillonnés, (b) après la connexion des points consécutifs.

les opérations de déplacement et copiage du tracé pour éditer le texte. À la levée du crayon, les points extrêmes du tracé sont finalisés, en calculant leurs coordonnées relatives par rapport au point initial du tracé. Ces valeurs sont également utilisées pour calculer les dimensions et la position du rectangle englobant du tracé.

Un autre traitement effectué dans cette fonction consiste à déterminer le numéro de la ligne dans laquelle le tracé est entré. Étant donné que le scripteur est censé écrire entre les lignes horizontales, un tracé fera partie de la ligne numéro $N$, si les conditions suivantes sont satisfaites :

$$y_{\min,trace} \geq (y_{\min,ligneN} - 0.25 * h_{ligne}) \quad \text{et} \quad y_{\max,trace} \leq (y_{\max,ligneN+1} + 0.25 * h_{ligne}) \tag{3.1}$$

où :

$y_{min,tracé}$ et $y_{max,tracé}$ sont les coordonnées $y$ des points extrêmes haut et bas du tracé respectivement;

$y_{min,ligneN}$ et $y_{max,ligneN+1}$ sont respectivement les coordonnées $y$ de $N^{ième}$ et $(N+1)^{ième}$ ligne de texte;

$h_{ligne}$ est la distance entre deux lignes horizontales.

Le coefficient 0.25 est utilisé pour tolérer les débordements éventuels des tracés, causés par leurs parties supérieures et inférieures (comme les lettres 'g' et 'l')[6].

Une fois que les attributs du tracé sont déterminés, il est inséré à la queue de la liste des tracés. Dans cette liste, les tracés sont ainsi triés en fonction de leur temps d'entrée, plutôt qu'en fonction de leur position. Toutefois, pour effectuer la segmentation de textes en mots, nous avons besoin d'une liste des tracés triés en fonction de la position et non pas du temps d'entrée. Il est à noter que la liberté du scripteur concernant le temps et l'ordre d'entrée des tracés est un point fort de notre interface d'acquisition. Cependant, le prix à payer pour offrir cette liberté est la complexité des opérations, en raison du nombre élevé de cas possibles à traiter. Un résultat de cette liberté est le fait que les tracés triés en ordre temporel ne sont pas nécessairement triés en ordre spatial. Par exemple, dans un texte de dix lignes, le scripteur à le choix d'écrire d'abord le dixième ligne et ensuite la première. Il est aussi libre d'entrer la partie principale d'un mot, d'écrire un deuxième mot et de faire un retour pour entrer les accents ou les points du premier mot. En raison de la différence permise entre les ordres temporel et spatial, il est donc nécessaire que l'interface puisse trier les tracés en ordre spatial, malgré la complexité des traitements requis. Pour ce faire, les tracés sont d'abord triés en ordre croissant de numéro de ligne et ensuite, en ordre croissant de la coordonnée $x$ de leur point extrême gauche.

La dernière étape de la phase d'acquisition consiste à déterminer le dernier tracé de chaque ligne de texte. Cette étape est nécessaire pour la phase de l'extraction des caractéristiques, puisque si un tracé est le dernier tracé de la ligne, son prochain tracé fait partie de la ligne suivante et il ne possède aucune distance inter-tracés à être

[6] Si, pour un tracé, les conditions de l'équation (3.1) ne sont pas satisfaites, pour déterminer le numéro de ligne du tracé (qui croise plusieurs lignes), on peut calculer les longueurs des portions du tracé dans les différentes lignes et choisir la ligne qui contient la longueur la plus élevée.

classifiée. Chaque tracé a un attribut qui sert à indiquer s'il est le dernier tracé de la ligne. La tâche de cette étape consiste donc à déterminer la valeur de cet attribut qui sera utilisée dans la phase de l'extraction des caractéristiques.

Les tracés acquis dans cette étape sont prêts pour la phase de l'extraction des caractéristiques qui sera présentée dans les sections suivantes.

## 3.2 EXTRACTION DES CARACTÉRISTIQUES CONVENTIONNELLES

L'extraction des caractéristiques joue un rôle très important dans tous les systèmes de classification. Dans notre système, les caractéristiques extraites doivent être choisies pour permettre à la procédure de segmentation de bien discriminer les distances inter-mots des distances intra-mot. Dans le chapitre 2, nous avons présenté les différentes caractéristiques utilisées dans les systèmes de segmentation de textes manuscrits en mots, et nous avons choisi celles que nous avons trouvées les plus robustes pour la segmentation en-lignes d'une ligne de texte en mots. Il s'agit des caractéristiques suivantes : la distance entre les rectangles (Dist_BB) de la distance horizontale (RL). Toutefois, nous avons souligné les cas problématiques des méthodes de segmentation qui utilisent ces mesures de distance inter-tracés. Dans notre projet, nous avons extrait deux grandes catégories de caractéristiques : (a) les caractéristiques conventionnelles, et (b) celles que nous avons proposées pour améliorer la performance de segmentation.

Dans cette section, nous allons présenter les étapes d'extraction des caractéristiques conventionnelles. L'objectif visé de l'extraction de ces caractéristiques consiste à comparer leurs performances avec celles des caractéristiques proposées. Les mesures de distances inter-tracés choisies sont la distance entre rectangles Dist_BB et les distances horizontales moyenne RLavg et minimale RLmin. Nous avons également utilisé deux autres caractéristiques, la hauteur moyenne des tracés (avgHeight) et la distance inter-tracés moyenne (avgDistBB), lesquelles seront détaillées plus loin.

### 3.2.1 EXTRACTION DE LA DIST_BB

Dist_BB entre deux tracés est la distance entre les rectangles minimaux les englobant (figure 2.3). Il est à noter que cette caractéristique est un attribut et appartient au tracé qui est situé à gauche.

La difficulté principale pour calculer les Dist_BB dans un texte manuscrit, acquis de façon en-ligne, sans contrainte au niveau du temps, de la séquence temporelle, de la forme et de la position des tracés, consiste à couvrir d'un nombre très élevé de séquences, de formes manuscrites et de positionnements relatifs possibles entre les tracés. En fait, pour un texte composé de $n$ tracé, ce nombre s'élève à $n!$. Le tri spatial dans la phase d'acquisition permet d'éliminer les éventuels désordres séquentiels des tracés et de diminuer les cas possibles à traiter. La tâche principale de l'algorithme du calcul de Dist_BB consiste donc à classer adéquatement les tracés manuscrits et leur positionnement relatif, afin de limiter le nombre de cas à traiter. La figure 3.5 montre les trois classes de positionnement relatif des tracés pour le calcul de Dist_BB. Les tracés de la figure 3.5(a) ne se chevauchent pas horizontalement. La Dist_BB est donc positive et attribuée au tracé situé à gauche (« a » dans la figure 3.5(a)). Les figures 3.5(b) et 3.5(c) montrent les cas de chevauchements partiel et complet donnant des Dist_BB égales à zéro. Dans le premier cas, cette valeur est attribuée au tracé situé à gauche (« a » dans la figure 3.5(b)) et dans le deuxième cas, elle est attribuée au tracé situé à droite (« / » dans la figure 3.5(c))[7].

(a) (b) (c)

Figure 3.5. Trois cas possibles de positionnement relatif des tracés pour calculer la Dist_BB.

[7] Dans les trois cas de la figure 3.5, les tracés se chevauchent verticalement. Ce type de chevauchement n'a pas d'importance dans le calcul de la Dist_BB, mais en a plutôt pour calculer les distances horizontales entre les tracés.

La figure 3.6 montre l'algorithme que nous avons conçu pour extraire les Dist_BB, en fonction des cas montrés dans la figure 3.5. Cet algorithme parcourt la liste pour attribuer une Dist_BB à chacun des tracés, sauf pour les derniers tracés des lignes. Toutefois, il faut affecter zéro dans Dist_BB des accents et des points de ces tracés (s'ils en ont évidemment). À chaque tour de la boucle, une paire de tracés est examinée, soit le tracé actuel *TracéActuel* et le tracé suivant *TracéSuivant*. S'ils ne se chevauchent pas horizontalement (comme dans la figure 3.5(a)) la Dist_BB est calculée et affectée dans l'attribut Dist_BB du tracé actuel (c'est le cas du tracé « a » dans l'exemple). Ensuite, il faut mettre à jour le tracé actuel pour passer à la paire suivante (« b » devient le tracé actuel dans l'exemple). Dans le cas de chevauchement partiel (comme dans la figure 3.5(b)), la valeur de zéro est affectée à l'attribut Dist_BB du tracé actuel (le tracé « a » dans l'exemple) et le tracé suivant devient le tracé actuel. Si les deux tracés se chevauchent complètement (comme dans la figure 3.5(c)), la Dist_BB du tracé suivant devient zéro (le tracé '/' dans l'exemple), sans mise à jour des tracés, car $x_{max}$(tracé actuel) > $x_{max}$(tracé suivant).

## 3.2.2 EXTRACTION DES DISTANCES HORIZONTALES

Pour la première fois dans un système en-ligne de segmentation en mots, nous avons utilisées les distances horizontales pour présenter l'espacement entre les entités manuscrites. Contrairement aux systèmes hors-ligne dans lesquels, grâce à la méthode de balayage optique, ces distances sont facilement accessibles à partir de l'image du texte, le calcul des distances horizontales dans un système en-ligne nécessite des efforts supplémentaires et ce, compte tenu de l'aspect dynamique de la composition de textes, notamment lorsque les scripteurs ont les libertés temporelle et séquentielle.

Dans cette section, nous expliquerons les opérations effectuées dans notre interface pour extraire les distances horizontales. L'objectif de cette phase consiste à calculer, pour chaque tracé, les distances horizontales moyenne RLavg et minimale RLmin entre le tracé et le mot le plus près dans le texte. Ces caractéristiques seront

```
CALCUL_DIST_BB
/* Calculer les distances entre les rectangles englobant les tracés dans un texte   */
/* manuscrit. Les tracés dans la liste sont triés en ordre croissant de numéro de   */
/* ligne et de position dans les lignes du texte                                     */

Prendre le premier tracé de la liste → TracéActuel;
Tant que TracéActuel n'est pas le dernier tracé de la liste des tracés
    Si TracéActuel est le dernier tracé de la ligne actuelle, alors
        Prendre le tracé suivant → TracéSuivant;
        /* cas des accents et des points du tracé actuel */
        Tant que TracéSuivant n'est pas le premier tracé de la ligne suivante
            Affecter 0 dans l'attribut Dist_BB de TracéSuivant;
            Prendre le tracé suivant → TracéSuivant;
        Fin de Tant que

        /* mise à jour des tracés pour calculer la prochaine Dist_BB,
           dans la ligne suivante. */
        TracéActuel = TracéSuivant;
    /* Tracé actuel n'est pas le dernier tracé de la ligne */
    Sinon
        Prendre le tracé suivant → TracéSuivant;
        Calculer le chevauchement horizontal des tracés TracéActuel et TracéSuivant
        /* les tracés se chevauchent horizontalement */
        Si ChevauchTracésActuelSuivant est 0, alors
            Dist_BB_Calcul = x_min TracéSuivant − x_max TracéActuel;
            Affecter Dist_BB_Calcul dans l'attribut Dist_BB de TracéActuel
            TracéActuel = TracéSuivant; /* mise à jour des tracés */
        /* les tracés ne se chevauchent pas horizontalement */
        Sinon
            /* chevauchement partiel */
            Si x_max TracéSuivant > x_max TracéActuel, alors
                Affecter 0 dans l'attribut Dist_BB de TracéActuel;
                TracéActuel = TracéSuivant;
            Sinon /* chevauchement complet */
                Affecter 0 dans l'attribut Dist_BB de TracéSuivant;
            Fin de Si-Sinon
        Fin de Si-Sinon
    Fin de Si-Sinon
Fin de Tant que
```

Figure 3.6. Algorithme de calcul en-ligne des Dist_BB pour les tracés dans un texte manuscrit.

utilisées ultérieurement dans la phase de classification pour décider si le tracé doit faire partie du mot actuel ou s'il doit créer un nouveau mot. Étant donné qu'un mot pourrait être composé d'un tracé ou de plusieurs tracés, l'opération consiste à calculer les

distances entre le tracé actuel et le(s) tracé(s) du mot actuel. Pour faciliter la compréhension, nous présenterons les opérations à effectuer pour extraire les distances horizontales moyenne et minimale dans les deux étapes suivantes: (1) le calcul des RLavg et RLmin entre deux tracés (le tracé actuel et un tracé du mot actuel) et (2) le calcul des RLavg et RLmin entre le tracé actuel et le mot actuel.

### 3.2.2.1 CALCUL DES DISTANCES HORIZONTALES ENTRE DEUX TRACÉS

Dans ce cas, les valeurs de RLavg et RLmin sont calculées à partir de l'ensemble des distances horizontales entre les points des deux tracés, lesquelles sont présentées partiellement dans la figure 3.7. Les coordonnées des points montrés dans cette figure ont les significations suivantes :

$(x_{IG}, y_{min})$ : coordonnées du point, appartenant au tracé situé à gauche, utilisé pour calculer la première distance horizontale entre les tracés.

$(x_{NG}, y_{max})$ : coordonnées du point, appartenant au tracé situé à gauche, utilisé pour calculer la dernière ($N^{ième}$) distance horizontale entre les tracés.

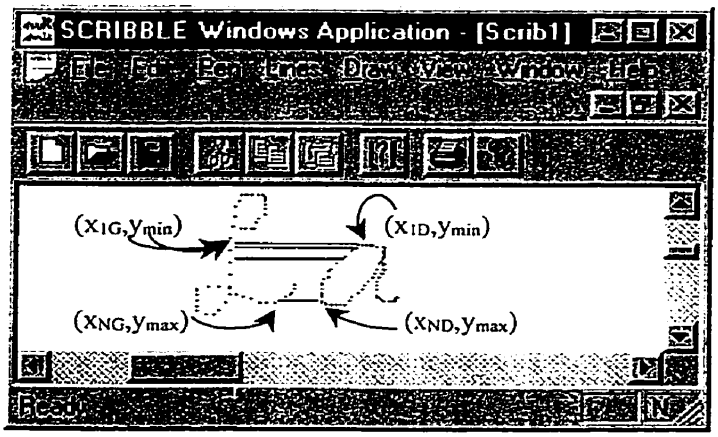



Figure 3.7. Distances horizontales entre deux tracés et les coordonnées de leurs portions se chevauchant verticalement.

($x_{1D}$, $y_{min}$) : coordonnées du point, appartenant au tracé situé à droite, utilisé pour calculer la première distance horizontale entre les tracés.

($x_{ND}$, $y_{max}$) : coordonnées du point, appartenant au tracé situé à droite, utilisé pour calculer la dernière ($N^{ième}$) distance horizontale entre les tracés.

Les étapes à suivre pour calculer les distances horizontales minimale et moyenne sont les suivantes :

1- Interpoler les tracés;
2- Trier les points des tracés en ordre croissant de la coordonnée $y$ et calculer les coordonnées $x_{max}$ et $x_{min}$ pour chaque valeur de $y$;
3- Déterminer si les tracés se chevauchent verticalement et calculer les valeurs de $y_{min}$ et $y_{max}$ des portions qui se chevauchent.
4- Calculer les distances horizontales à partir des valeurs de $x_{max}$ et $x_{min}$ déterminées à l'étape 2;
5- Calculer les valeurs moyenne et minimale des distances horizontales déterminées à l'étape 4.

Les distances horizontales n'existent qu'entre deux tracés qui se chevauchent verticalement. Il est donc nécessaire de déterminer si les tracés se chevauchent verticalement et si, c'est le cas, il faut ensuite calculer les coordonnées $y$ des portions chevauchantes. Il s'agit des $y_{min}$ et $y_{max}$ dans la figure 3.7. Pour déterminer les distances horizontales nous avons besoin de la coordonnée $x$ de tous les points des deux tracés ayant des coordonnées $y$ entre $y_{min}$ et $y_{max}$. Mais, en raison de l'échantillonnage dans le temps, les tracés peuvent ne pas avoir toutes les valeurs de $y$ entre les deux valeurs extrêmes. Comme la figure 3.7 le démontre, le tracé « a » a plus de points échantillonnés entre $y_{min}$ et $y_{max}$ que le tracé 'L'. Il est donc impossible de calculer toutes les distances horizontales. Pour résoudre ce problème, nous effectuons préalablement une étape d'interpolation linéaire des tracés permettant de générer des points ayant tous des

valeurs de $y$ entre $y_{min}$ et $y_{max}$. Le nombre d'octets en mémoire exigeant le stockage d'un tracé interpolé est plus grande que celle de son tracé original. Le pourcentage de cette augmentation dépend de plusieurs paramètres dont la vitesse d'écriture dans les différentes parties d'un tracé et sa longueur. Il peut donc varier significativement d'un tracé à l'autre. Il est toutefois d'une importance minime, puisque l'interpolation est une étape intermédiaire et les tracés interpolés sont générés temporairement pour calculer les distances horizontales, sans qu'ils soient sauvegardés de façon permanente. Cette opération d'interpolation est la première étape de l'extraction des distances horizontales minimale et moyenne.

Dans la deuxième étape, nous trions les points des tracés en ordre croissant de leur coordonnée $y$. Les points dans la liste initiale et la liste interpolée sont triés en ordre temporel, alors que pour un calcul plus rapide des distances horizontales, nous avons besoin de les ordonner en fonction de la coordonnée $y$. Les éléments de la nouvelle liste sont les objets ayant trois attributs : les coordonnées $y$, $x_{min}$ et $x_{max}$. Ces deux derniers représentent respectivement les coordonnées minimale et maximale des points d'intersection d'une ligne horizontale avec le tracé (figure 3.8). Pour une coordonnée $y_i$, le tracé pourraient avoir un ou plusieurs points ayant la même coordonnée. Mais parmi ces points, pour le calcul des distances horizontales entre les tracés, nous nous servons que de ceux qui ont les coordonnées minimale et maximale en $x$. Le résultat du tri et du calcul des $x_{min}$ et $x_{max}$ est la réduction significative du nombre de points à traiter dans la phase du calcul des distances horizontales.

La troisième étape consiste à déterminer si les tracés se chevauchent verticalement ainsi qu'à calculer les valeurs de $y_{min}$ et $y_{max}$ des portions


Figure 3.8. Une ligne horizontale ayant quatre intersections avec le tracé.

chevauchantes. Ces coordonnées sont montrées dans la figure 3.7. Si les tracés ne se chevauchaient pas verticalement, la distance horizontale entre eux n'existerait pas. Dans ce cas, qui est en général le cas des accents et des points, la procédure de calcul des RL, affecte une valeur négative dans les variables RLmin et RLavg, pour permettre à la procédure de segmentation de détecter l'inexistence des distances horizontales. Si les tracés ont un chevauchement vertical, on déterminera les valeurs de $y_{min}$ et $y_{max}$.

L'étape suivante consiste à calculer les distances horizontales pour toutes les valeurs de $y$ entre $y_{min}$ et $y_{max}$. La i-ème distance horizontale entre les tracés est déterminée de la façon suivante :

$$RL_i = x_{\min,iD} - x_{\max,iG} \tag{3.2}$$

où : $RL_i$ est la i-ème distance horizontale entre les tracés;
$x_{min,iD}$ est la coordonnée $x_{min}$ du i-ème point du tracé situé à droite;
$x_{max,iG}$ est la coordonnée $x_{max}$ du i-ème point du tracé situé à gauche.

Si $x_{min,iD} < x_{max,iG}$, les deux tracés se chevauchent horizontalement et la $RL_i$ calculée sera négative. Nous remplaçons cette valeur par zéro, afin de la différencier du cas de l'inexistence des distances horizontales, mentionné ci-dessus. Une fois que toutes les $RL_i$s entre les tracés ont été calculées, il faut en trouver les valeurs moyenne et minimale, c'est-à-dire les RLavg et RLmin.

### 3.2.2.2 CALCUL DES DISTANCES HORIZONTALES ENTRE UN TRACÉ ET UN MOT

Il s'agit du calcul des RLavg et RLmin entre le tracé actuel et l'ensemble des tracés du mot actuel. Nous rappelons que les caractéristiques extraites sont utilisées dans les procédures suivantes : l'apprentissage supervisé des paramètres du module de

segmentation et la segmentation en-ligne basée sur les paramètres ajustés dans la phase d'apprentissage. Le résultat de ces procédures (qui seront expliqués dans le chapitre 4) est un texte composé d'une liste de mots. Chaque mot est composé, à son tour, d'une liste de tracés. Quelle que soit l'opération, l'apprentissage ou la segmentation, à chaque tracé composé, il faut calculer les valeurs de RLmin et RLavg entre le tracé et le(s) tracé(s) du mot le plus près (figure 3.9).

Les étapes à suivre pour calculer ces caractéristiques sont similaires à celles du calcul des distances entre deux tracés. Les points du mot et ceux du tracé actuel doivent être triés en ordre croissant de la coordonnée $y$. Ensuite, on calcule les coordonnées $x_{max}$ et $x_{min}$ pour chaque valeur de $y$ dans le mot et dans le tracé actuel. Les $x_{max}$ et $x_{min}$ peuvent ainsi faire partie de tracés différents du mot. L'étape suivante consiste à déterminer si le tracé et le mot se chevauchent verticalement. S'il n'y a pas de chevauchement vertical, on affecte « -1 » dans les variables RLmin et RLavg. Sinon, il faut calculer les valeurs de $y_{min}$ et $y_{max}$ des portions chevauchantes suivies du calcul des distances horizontales à partir des valeurs des $x_{max}$ et $x_{min}$. Une distance horizontale négative sera remplacée par zéro. La dernière opération à effectuer consiste à calculer les valeurs moyenne et minimale des distances horizontales, RLavg et RLmin respectivement.

Les valeurs de RLavg et RLmin entre le tracé et le mot actuel seront utilisées, soit dans une phase d'apprentissage supervisé pour ajuster les paramètres d'un classificateur de distances, soit dans une phase de segmentation effectuée par le classificateur obtenu de la phase d'apprentissage. Dans tous les cas, le type de la distance, déterminé par l'utilisateur ou par le classificateur, peut être soit intra-mot soit

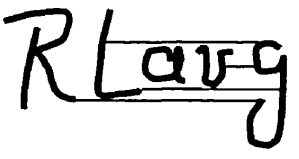



Figure 3.9. Les distances horizontales entre le tracé actuel 'g' et le mot actuel 'RLav'.

inter-mots. S'il est intra- mot, il faut ajouter le tracé dans le mot actuel et mettre à jour la liste triée des points du mot en ordre croissant de la coordonnée *y*, ainsi que les valeurs des $x_{min}$ et $x_{max}$. Autrement (le cas de la distance inter-mots) , un nouveau mot sera créé et le tracé actuel y sera ajouté. Ensuite, on calcule la liste triée des points du mot en ordre croissant de la coordonnée *y*, ainsi que les valeurs des $x_{min}$ et $x_{max}$. Ces opérations de mise à jour et de calcul sont nécessaires pour la poursuite de l'opération de l'extraction des distances horizontales jusqu'à la fin de la composition du texte.

Le résultat de cette phase est donc une série de distances horizontales minimale et moyenne si le tracé et le mot à évaluer se chevauchaient verticalement, sinon une valeur négative pour les deux distances horizontales, afin d'indiquer l'inexistence d'un chevauchement vertical et donc de ces distances.

## 3.2.3 EXTRACTION DE avgHeight ET avgDistBB

Dans cette section nous aborderons le problème d'extraction de la hauteur moyenne des tracés avgHeight et de la distance moyenne entre les rectangles des tracés consécutifs avgDistBB.

Nous utilisons le avgHeight pour évaluer le rôle de la grandeur des tracés dans la procédure de segmentation. Cette caractéristique est calculée de façon en-ligne pour les N derniers tracés :

$$avgHeight_i = \frac{\sum_{j=0}^{N-1} h_{i-j}}{N} \qquad \text{pour } i = 1 : NbTracés \tag{3.3}$$

où : $N$ est la largeur de fenêtre de calcul (en nombre de tracés);

$avgHeight_i$ est la hauteur moyenne de $N$ derniers tracés pour le i-ème tracé;

$h_{i-j}$ est la hauteur de *j-ième* tracé avant le tracé $i$.

*NbTracés* est le nombre de tracés du texte.

La hauteur du *k-ième* tracé dans le texte est calculé selon l'expression suivante :

$$h_k = \left| y_{\max,k} - y_{\min,k} \right| \tag{3.4}$$

où : $h_k$ est la hauteur du *k-ième* tracé,

$y_{max,k}$ et $y_{min,k}$ sont les coordonnées $y$ maximale et minimale du tracé k, respectivement.

La avgDistBB sert à produire à la procédure de la segmentation, une statistique sur la valeur de Dist_BB pour le $N$ derniers tracés. Elle est calculée de la façon suivante :

$$avgDistBB_i = \frac{\sum_{j=0}^{N-1} Dist\_BB_{i-j}}{N} \qquad \text{pour } i = 1 : NbTracés \tag{3.5}$$

où : $N$ est la largeur de fenêtre de calcul (en nombre des tracés);

$avgDistBB_i$ est la valeur moyenne de $N$ derniers *Dist_BB* pour le i-ème tracé;

$Dist\_BB_{i-j}$ est la distance entre les rectangles pour le *j-ième* tracé avant le tracé *i*.

*NbTracés* est le nombre de tracés du texte.

Comme nous allons le voir dans le chapitre 4, le avgHeight et le avgDistBB sont utilisés comme des facteurs de normalisation des caractéristiques dans la procédure de classification.

## 3.3 LACUNES DES CARACTÉRISTIQUES CONVENTIONNELLES

À la lumière de l'analyse présentée dans le chapitre 2, nous avons choisi Dist_BB et RL comme les caractéristiques conventionnelles les plus robustes pour la segmentation en-ligne d'une ligne de texte manuscrit en mots. Nous avons également présenté, dans la section 3.2.1, les étapes à suivre pour les extraire.

Dans cette courte section, nous rappellerons brièvement les lacunes des caractéristiques conventionnelles; ce qui nous permettra de présenter, dans la section suivante, nos propositions pour les améliorer.

En ce qui concerne la distance entre les rectangles, comme nous l'avons mentionné dans la section 2.2.2.2, les problèmes les plus importants des méthodes de segmentation basée sur la Dist_BB, sont les suivants :

(1) la sensibilité à l'inclinaison des tracés,

(2) la sensibilité au chevauchement horizontal des tracés (figure 2.9).

Dans la section 2.2.2.5, nous avons fait une discussion sur les points faibles de la mesure de distance horizontale entre les tracés. Nous rappelons que cette mesure souffre principalement des problèmes suivants :

(1) la dépendance au chevauchement vertical,

(2) les distances horizontales irréalistes et

(3) la sensibilité au décalage vertical des tracés.

Dans la section suivante, nous expliquerons en détails ces problèmes et leurs sources.

## 3.4 EXTRACTION DES CARACTÉRISTIQUES PROPOSÉES

Après avoir présenté les caractéristiques conventionnelles, leurs phases d'extraction ainsi que leurs lacunes dans les sections précédentes, nous introduirons dans cette section, nos propositions pour résoudre les problèmes de ces caractéristiques et présenterons nos caractéristiques proposées. Nous expliquerons également les étapes à suivre pour les extraire. Nous commencerons d'abord par la distance entre les rectangles (Dist_BB) et continuerons avec les distances horizontales (RLmin et RLavg).

## 3.4.1 PROPOSITIONS POUR AMÉLIORER DIST_BB

Dans cette section, nous présenterons nos propositions pour améliorer la performance de la mesure de distance entre les rectangles. Quant aux deux lacunes principales de cette mesure, aux sensibilités à l'inclinaison des tracés et à leur chevauchement horizontal, nous proposons de remplacer la distance entre les rectangles par la distance entre les parallélogrammes et d'effectuer une phase de reconnaissance des accents, points, virgules et de barres de « t » avant la phase d'extraction de la distance entre les parallélogrammes et ce, afin d'éliminer les sources de chevauchements horizontaux indésirables.

Nous nous pencherons d'abord sur le problème causé par les tracés inclinés et continuerons avec la deuxième lacune causée par le chevauchement horizontal.

### 3.4.1.1 PROBLÈME DE SENSIBILITÉ À L'INCLINAISON D'ÉCRITURE

Dans une écriture inclinée, la mesure de Dist_BB a tendance à sous-estimer les distances inter-tracés. Deux tracés du texte peuvent être suffisamment loins l'un de l'autre pour que leur distance soit classifiée par l'utilisateur comme une distance inter-mot. Toutefois, en raison de leur inclinaison et de la sous-estimation de la Dist_BB, la procédure de la segmentation pourrait classifier leur distance comme étant intra-mot. Par exemple, dans la figure 3.10(a) les rectangles englobant les tracés se chevauchent horizontalement et la Dist_BB calculée est inférieure ou égale à zéro, ce qui donnerait un résultat erroné de classification : distance intra-mot.

Pour obtenir une distance inter-tracés plus précise, notamment dans le cas d'écriture inclinée, nous proposons pour la première fois, dans un système en-ligne de segmentation de textes en mots, de remplacer le rectangle englobant par le parallélogramme englobant. Il s'agit du plus petit parallélogramme qui, avec une inclinaison donnée, englobe le tracé. La figure 3.10 montre comment la distance entre les parallélogrammes, que l'on nommera désormais *Dist_Paral*, pourrait aider à

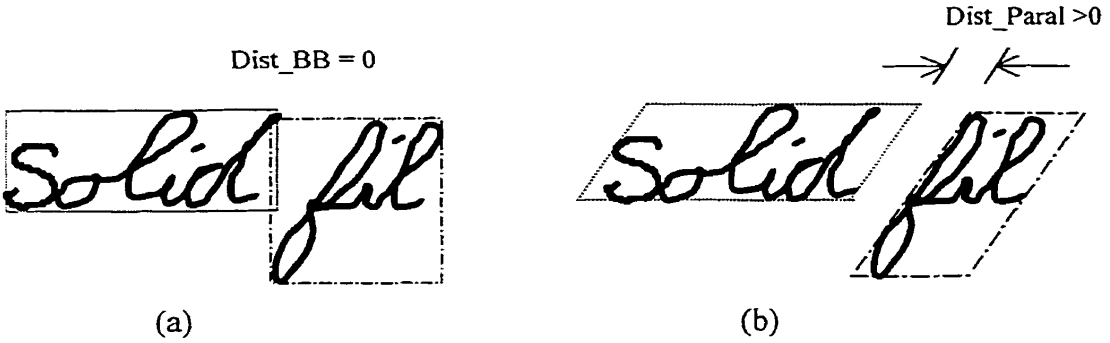



Figure 3.10. Mesures de distances (a) Dist_BB et (b) Dist_Paral entre deux tracés inclinés. La distance inter-tracés est mieux représentée par la Dist_paral.

résoudre le problème de sous-estimation des distances inter-tracés. Les étapes à suivre pour mesurer la Dist_Paral sont les suivantes :

1- Estimer les inclinaisons des tracés;

2- Calculer l'inclinaison moyenne des tracés;

3- Déterminer les parallélogrammes englobant les tracés;

4- Mesurer la distance entre les parallélogrammes.

Nous allons présenter les détails des étapes du calcul de la *Dist_Paral.*

#### *3.4.1.1.1 ESTIMATION DE L'INCLINAISON DES TRACÉS*

L'étape la plus importante pour calculer la *Dist_Paral* consiste à estimer l'inclinaison des tracés. Il faut préciser que la quantification de l'inclinaison d'écriture n'est pas une tâche facile. Premièrement, lorsque les êtres humains observent un texte, il peuvent identifier si l'écriture est penchée, alors que les inclinaisons jugées peuvent être différentes d'un sujet à l'autre. Deuxièmement, l'estimation de l'inclinaison doit être en-ligne et à l'aide de peu de données disponibles, contrairement aux systèmes hors-ligne dans lesquels la rapidité de traitement n'est pas d'une importance primordiale et une quantité plus grande d'information est disponible pour faire ce type d'estimation. Troisièmement, il faut identifier les informations pertinentes, la quantité de l'information et la manière de l'utiliser pour estimer l'inclinaison.

Pour ce faire, nous nous sommes inspirés des résultats présentés dans (Maarse, 1983), qui contient une étude sur l'estimation de l'inclinaison d'écriture manuscrite. Cette étude montre que, puisque dans une écriture régulière, les segments descendants sont bien plus stables que les segments ascendants, la valeur moyenne des inclinaisons jugées par les sujets est nettement mieux estimée par l'inclinaison des segments descendants des tracés que par celle des segments ascendants. Un segment descendant est la ligne reliant deux points consécutifs du tracé, si la direction d'écriture est du haut vers le bas. Pour un segment ascendant la direction d'écriture est du bas vers le haut.

Nous croyons que la stabilité de l'inclinaison des segments descendants par rapport à des segments ascendants peut être expliquée par le phénomène suivant: les segments descendants, en général, font partie des lettres d'alphabet alors que les segments ascendants servent à connecter une lettre à la lettre suivante. Et puisque le début et la fin des différentes lettres peuvent se trouver dans des positions relatives très diversifiées, les segments les reliant ensemble dans un texte donné peuvent aussi avoir des inclinaisons très diversifiées.

La figure 3.11 montre un segment descendant du tracé 'b'. Le tracé commence au point $P_0$ avec une direction descendante. Entre les points $P_i$ et $P_{i+1}$ la direction est toujours descendante et l'angle de descente ($\alpha_i$) est positif. Ce dernier, qui est

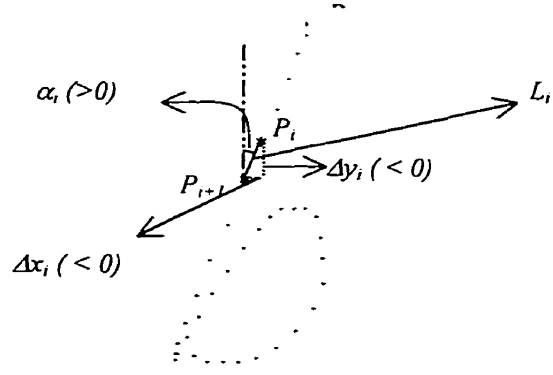



Figure 3.11. Un segment descendant du tracé 'b'et ses paramètres.

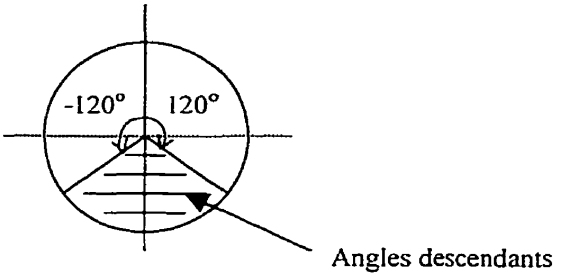



Figure 3.12. Angle des segments descendants des tracés.

l'angle entre l'axe des ordonnées et la ligne reliant les points, serait calculé selon l'expression suivante[8] :

$$\alpha_i = \tan^{-1}(\frac{\Delta x_i}{\Delta y_i}) * (180/\pi) \tag{3.6}$$

où : $\alpha_i$ est l'angle entre l'axe des ordonnées et la ligne qui relie les points;

$\Delta x_i = x_{i+1} - x_i$ et $\Delta y_i = y_{i+1} - y_i$.

La fonction $tan^{-1}$ retourne un angle entre -π et π, selon le quadrant où le point $P_{i+1}$ est situé. Un segment est dit descendant si son angle calculé selon l'équation (3.6) est supérieur à 120° ou inférieur à –120° (figure 3.12). Ces angles seront ensuite transférés dans l'intervalle -60° et 60°. ( En raison de cette conversion, l'angle du segment montré dans la figure 3.11 est positif).

Pour estimer la pente d'un tracé, nous utilisons également la longueur des segments descendants, autrement dit la distance entre les points $P_i$ et $P_{i+1}$ :

$$L_i = \left(\Delta x_i^2 + \Delta y_i^2\right)^{1/2} \tag{3.7}$$

Les valeurs de $\Delta x_i$ et $\Delta y_i$ utilisées dans les équations (3.6) et (3.7) sont les coordonnées relatives des points du tracé et sont déjà calculées dans la phase de l'acquisition (section 3.1.2).

Notre algorithme d'estimation de l'inclinaison d'un tracé est montré dans la figure 3.13. La partie principale de l'algorithme consiste à parcourir la liste de points, afin de calculer la somme des longueurs ainsi que la somme pondérée des angles des segments descendants. La pondération des angles par la longueur des segments descendants permet de donner plus de poids à l'angle des longs segments. L'inclinaison d'un tracé est jugée par la pente des segments longs. Sans pondérer les angles par la longueur, les segments courts peuvent réduire la précision de l'inclinaison estimée. En général, ces segments sont fréquents dans les tracés et peuvent avoir d'une grande variabilité d'inclinaisons; ce qui pourrait réduire ou même neutraliser l'effet des segments longs dans le calcul de la pente estimée. Une fois que le parcours des points du tracé est terminé, on vérifie si le tracé a des segments descendants. L'absence de ces segments montre que le tracé est horizontal ou presque horizontal. Dans la procédure de calcul de la Dist_Paral, ces tracés ne doivent pas être considérés comme des tracés inclinés, puisque leurs fortes pentes pourraient produire des valeurs Dist_Paral irréalistes et causer des erreurs de segmentation. Pour les tracés manquant de segments descendants, l'algorithme attribuerait donc la valeur de zéro à l'angle estimé, à la longueur moyenne des segments descendants ainsi qu'à l'angle pondéré estimé. Ces paramètres seront utilisés pour calculer la Dist_Paral entre les tracés. Si le tracé a des segments descendants, l'algorithme calcule, à partir de la valeur de leurs paramètres cumulatifs correspondants, calculés dans la boucle de parcours, l'angle estimé du tracé, la longueur moyenne des segments descendants et l'angle pondéré estimé. L'algorithme se termine avec le calcul de la pente estimée à partir de la valeur de l'angle estimé. Les résultats de cet algorithme seront utilisés pour déterminer les parallélogrammes englobant, ainsi que la Dist_Paral entre les tracés.

---

[8] Dans un programme écrit en langage C++, il faut remplacer $\Delta y_i$ par $-\Delta y_i$, en raison de la position de l'origine qui est situé en haut au coin gauche de l'écran.

```
ESTIMER_INCLINAISON
/* Estimer l'inclinaison d'un tracé comme la valeur moyenne de l'inclinaison de ses */
/* segments descendants.                                                            */

/* initialisations nécessaires */
Somme_Longueur = 0;
Somme_Longueur_Angle = 0;
Nb_Angle_Des = 0;

Pour tous les points du tracé Pi : i de 1 à NbPoints, faire
    /* les différences entre les coordonnées x et y des points consécutifs */
    Δx_i = x_{i+1} − x_i ;
    Δy_i = y_{i+1} − y_i ;
    Si Δx_i ≠ 0 et Δy_i ≠ 0, alors
        Angle_i = (tan^-1(Δx_i / Δy_i)/π)*180;
        Longueur_i = (Δx_i^2 + Δy_i^2)^1/2 ;
        /* Ne prendre que les segments descendants du tracé */
        Si Angle_i > 120° ou Angle_i < -120°, alors
            Nb_Angle_Des = Nb_Angle_Des + 1;
            Somme_Longueur = Somme_Longueur + Longueur_i ;
            /* Convertir l'angle à l'intervalle [-60, 60]
            Si Angle_i > 120°, alors  /* pente négative*/
                Angle_i = Angle_i − 180;
            Sinon /*Angle_i < -120°,  pente positive*/
                Angle_i = Angle_i + 180;
            Fin de Si-Sinon
            Somme_Longueur_Angle = Somme_Longueur_Angle +
                                   (Longueur_i * Angle_i);
        Fin de Si
    Fin de Si
Fin de Pour

/* tracé sans segment descendant */
Si Nb_Angle_Des = 0, alors
    Longueur_Estimé = 0;
    Longueur_Angle_Estimé = 0;
    Angle_Estimé = 0;
Sinon
    /* valeurs moyennes des longueurs, des angles et des produits angle_longueurs */
    /* des segments descendants du tracé. */
    Longueur_Estimé = Somme_Longueur / Nb_Angle_Des;
    Longueur_Angle_Estimé = Somme_Longueur_Angle / Nb_Angle_Des;
    Angle_Estimé = Somme_Longueur_Angle / Somme_Longueur;
Fin de Si-Sinon

/* pente estimée du tracé */
Pente_Estimée = tan( (Angle_Estimé / 180°) * π);
```

Figure 3.13. Algorithme de l'estimation de la pente d'un tracé.

### *3.4.1.1.2 CALCULE DE L'INCLINAISON MOYENNE DES TRACÉS*

Après avoir calculé l'inclinaison de chaque tracé, il faut en calculer l'inclinaison moyenne. Cette étape est nécessaire pour tenir compte des variations de l'inclinaison dans les différentes parties d'un texte et même d'un tracé à l'autre. En fait, les inclinaisons des deux tracés impliqués dans le calcul de Dist_Paral peuvent être différentes (figure 3.14(a)). Toutefois, pour calculer cette distance, une inclinaison unique est nécessaire et sera utilisée pour construire les parallélogrammes englobant les tracés.

Une façon de procéder consiste à utiliser la valeur moyenne des inclinaisons des tracés. Le résultat de cette opération est présenté dans la figure 3.14(b) démontrant que la distance entre les tracés est encore sous-estimée par la distance entre les parallélogrammes. Ce phénomène est causé par le fait que les deux inclinaisons ont la même importance dans le calcul de l'inclinaison moyenne.

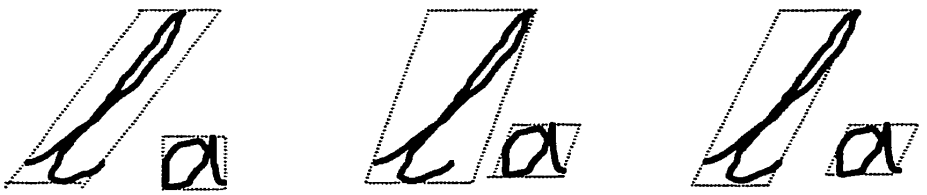
Pour résoudre ce problème, nous proposons d'utiliser l'inclinaison moyenne pondérée, calculée selon l'expression suivante :

$$Pente_{moy} = \frac{Pente_1 L_1 + Pente_2 L_2}{L_1 + L_2} \tag{3.8}$$

où : $Pente_{moy}$ est l'inclinaison moyenne des tracés;

$Pente_1$ et $Pente_2$ sont les inclinaisons estimées des tracés 1 et 2 respectivement;

$L_1$ et $L_2$ sont les longueurs moyennes des segments descendants des tracés 1 et 2, respectivement.



(a) (b) (c)

Figure 3.14. Différentes inclinaisons pour les tracés : (a) inclinaisons indépendantes; (b) inclinaison moyenne simple; (c) inclinaison moyenne pondérée.

Dans l'algorithme de la figure 3.13, $Pente_1$ et $Pente_2$ sont présentées par Pente_Estimée et $L_1$ et $L_2$ par Longueur_Estimé. Cette façon de calculer l'inclinaison moyenne donne plus de poids à l'inclinaison du tracé dans lequel la longueur moyenne des segments est plus grande et permet de calculer une mesure plus réaliste de Dist_Paral. En comparant les figures 3.14(b) et 3.14(c), on constate que la pondération des inclinaisons par la longueur moyenne des segments descendants, n'a pas de grands impacts sur le parallélogramme de la lettre « a », alors qu'elle change significativement le parallélogramme de la lettre 'l'.

Il faut ajouter que si $L_1$ et $L_2$ valent zéro, la valeur de zéro sera affectée dans l'inclinaison moyenne $Pente_{moy}$.

### *3.4.1.1.3 DÉTERMINER LE PARALLÉLOGRAMME ENGLOBANT D'UN TRACÉ*

Une fois que l'inclinaison moyenne des tracés est calculée, il faut déterminer leurs parallélogrammes englobant. Ce parallélogramme, pour chaque tracé, est identifiable à partir des quatre points suivants : les points extrême haut $P_h$, extrême bas $P_b$, extrême gauche $P_g$ et extrême droit $P_d$. Ces points sont montrés dans la figure 3.15. Les deux premiers points sont équivalents aux points extrêmes haut et bas du rectangle englobant du tracé, calculés dans l'étape de l'acquisition. Nous avons donc uniquement besoin de déterminer les points extrême gauche $P_g$ et extrême droit $P_d$ du parallélogramme.

Si l'angle du parallélogramme $\alpha_{moy}$ est positif (comme dans la figure 3.15), un point sera considéré comme le point extrême gauche et ce, si pour tous les points $P_i$ du tracé, les conditions suivantes étaient satisfaites :

$$(\alpha_{Pg \to Pi} \geq \alpha_{moy}) \quad \text{et} \quad (\alpha_{Pg \to Pi} \leq (\alpha_{moy} - 180^{\circ})) \qquad \forall P_i \in tracé \qquad (3.9)$$

où : $\alpha_{Pg \to Pi}$ est l'angle entre l'axe des ordonnées et la ligne reliant le point extrême gauche au point $P_i$.

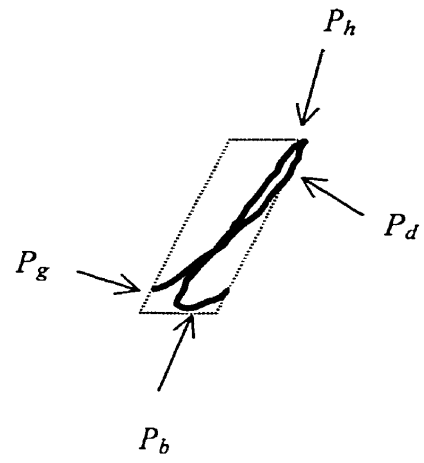



Figure 3.15. Points extrêmes du parallélogramme englobant d'un tracé

Pour un angle négatif du parallélogramme, les conditions à respecter pour qu'un point soit le point extrême gauche sont les suivantes :

$$(\alpha_{Pi \to Pg} \leq \alpha_{moy}) \quad \text{et} \quad (\alpha_{Pi \to Pg} \geq (\alpha_{moy} + 180^{\circ})) \qquad \forall P_i \in tracé \qquad (3.10)$$

où : $\alpha_{Pi \to Pg}$ est l'angle entre l'axe des ordonnées et la ligne reliant le point $P_i$ au point extrême gauche.

Pour qu'un point soit considéré comme le point extrême droit du tracé, si ($\alpha_{moy} \geq 0$), il faut que les conditions suivantes soient satisfaites :

$$(\alpha_{Pi \to Pd} \geq \alpha_{moy}) \quad \text{et} \quad (\alpha_{Pi \to Pd} \leq (\alpha_{moy} - 180^{\circ})) \qquad \forall P_i \in tracé \qquad (3.11)$$

où : $\alpha_{Pi \to Pd}$ est l'angle entre l'axe des ordonnées et la ligne reliant le point $P_i$ au point extrême droit.

Les conditions du point extrême droit, pour le cas de ($\alpha_{moy} < 0$) sont les suivantes:

$$(\alpha_{Pd \to Pi} \leq \alpha_{moy}) \quad \text{et} \quad (\alpha_{Pd \to Pi} \geq (\alpha_{moy} + 180^{\circ})) \qquad \forall P_i \in tracé \qquad (3.12)$$

où : $\alpha_{Pd \rightarrow Pi}$ est l'angle entre l'axe des ordonnées et la ligne reliant le point extrême droit au point $P_i$.

La tâche de l'algorithme déterminant le parallélogramme d'un tracé, consiste donc à parcourir la liste de points du tracé pour trouver ceux qui satisferont les conditions des équations (3.9) à (3.12).

### *3.4.1.1.4 MESURER LA DISTANCE ENTRE LES PARALLÉLOGRAMMES*

La distance entre les parallélogrammes peut être calculée à partir des coordonnées de leurs points extrêmes. Pour ce faire, les deux cas suivants sont identifiables :

1- $P_{g\_tracéD}.y \geq P_{d\_tracéG}.y$

2- $P_{g\_tracéD}.y < P_{d\_tracéG}.y$

où : $P_{g\_tracéD}.y$ est la coordonnée $y$ du point extrême gauche du tracé situé à droite;

$P_{d\_tracéG}.y$ est la coordonnée $y$ du point extrême droit du tracé situé à gauche.

Dans le premier cas, la Dist_Paral est calculée selon l'équation suivante :

$$Dist\_Paral = (P_{g\_tracéD}.x - P_{d\_tracéG}.x) + (\tan \alpha_{moy} * (P_{g\_tracéD}.y - P_{d\_tracéG}.y)) \qquad (3.13)$$

où : $\alpha_{moy}$ est l'angle des parallélogrammes;

$P_{g\_tracéD}.x$ et $P_{g\_tracéD}.y$ sont les coordonnées du point extrême gauche du tracé situé à droite;

$P_{d\_tracéG}.x$ et $P_{d\_tracéG}.y$ sont les coordonnées du point extrême droit du tracé situé à gauche.

Dans le deuxième cas, l'équation suivante est utilisée pour calculer la Dist_Paral:

$$Dist\_Paral = (P_{g\_tracéD}.x - P_{d\_tracéG}.x) - (\tan \alpha_{moy} * (P_{g\_tracéD}.y - P_{d\_tracéG}.y)) \qquad (3.14)$$

Les équations (3.13) et (3.14) donnent la valeur de Dist_Paral, pour les valeurs positive ou négative de $\alpha_{moy}$.

Nous avons proposé d'utiliser la Dist_Paral au lieu de la Dist_BB pour résoudre le problème de la sensibilité de la mesure de distance entre les boîtes, à l'inclinaison

d'écriture. Nous avons également présenté nos méthodes pour l'estimation de pente d'un tracé, le calcul adaptatif de l'inclinaison de texte pour mieux représenter les espacements inter-tracés et le calcul de la Dist_Paral. Ces propositions sont appliquées sur les textes manuscrits de notre base de données pour la segmentation en-ligne de ces textes en mots. La pertinence et la précision des méthodes proposées doivent donc être évaluées à partir de leur impact sur les résultats de segmentation. Ces résultats et l'analyse de cet aspect important de notre travail seront présentés dans les chapitres 4, 5 et 6.

### 3.4.1.2 PROBLÈME DE SENSIBILITÉ AU CHEVAUCHEMENT HORIZONTAL DES TRACÉS

Une deuxième série de cas problématiques de la mesure de la distance entre les rectangles sont causés par la sensibilité de Dist_BB au chevauchement horizontal des tracés. Si deux tracés de deux mots différents se chevauchent horizontalement, leur Dist_BB sera égale à zéro, et le module de la segmentation les classifiera dans un mot. Les tracés ne formant pas la partie principale des mots sont, en général, à l'origine de cette catégorie des erreurs de segmentation. Il s'agit des accents, des points et notamment des barres de « t ». Ces derniers sont des longs tracés pouvant facilement couvrir la distance entre les mots et la transformer en une distance intra-mot. Pour résoudre ces problèmes, il faut donc recourir à une phase de reconnaissance préliminaire des accents, des points et des barres de « t » qui nous permettra de les exclure de la phase de l'extraction des caractéristiques en les ajoutant directement dans un mot du texte. Cette phase de reconnaissance nous aidera également à résoudre une série de problèmes causés, notamment, par les points et les accents, dans la procédure de l'extraction des distances horizontales.

Dans cette section, nous expliquerons les caractéristiques de ces entités d'écriture et nous présenterons ensuite l'algorithme que nous avons utilisé pour les reconnaître à l'aide de ces caractéristiques.

### *3.4.1.2.1 CARACTÉRISTIQUES DES ACCENTS, POINTS ET BARRES DE « T »*

Notre but consiste à reconnaître les accents, les points et les barres de 't' dans un texte manuscrit. Nous allons donc commencer par analyser leurs caractéristiques. Les caractéristiques suivantes sont identifiables pour ces signes diacritiques :

1- Elles sont petites.
2- Elles possèdent une structure très simple;
3- Excepté les ponctuations, elles sont situées au-dessus de la partie principale des mots.

Quant à la projection sur l'axe des ordonnées de ces entités (que nous appellerons désormais Accent_Point_TBars), on peut facilement constater qu'elles sont, en général, plus petites que les tracés de la partie principale des mots.

En plus, les Accent_Point_TBars sont des tracés simples. La direction du tracé entre la pose et la levée du crayon est, en général, unique. Par conséquent, le nombre des minima et maxima est limité dans ce type de tracés.

Et, finalement, elles ne se placent pas dans la zone médiane du texte. Les ponctuations sont situées dans la zone inférieure d'une ligne de texte et les accents, les points et les barres de « t », se trouvent dans la zone supérieure, au-dessus de la partie principale des mots.

En somme, pour reconnaître les Accent_Point_TBars, il faut déterminer la grandeur relative, la complexité de la structure et la position relative des tracés. La difficulté principale consiste à encoder ces caractéristiques pour l'écriture manuscrite qui est d'une nature très variable. L'être humain est facilement capable de reconnaître ce type de tracés en évaluant la grandeur, la complexité et la position des tracés. En outre, pour ce faire, il peut se servir de la sémantique. Alors que dans notre système nous n'utilisons pas de sémantique, et en raison de la variabilité des paramètres d'écriture manuscrite, l'évaluation de la grandeur, de la structure et de la position relative des tracés ne constituent pas des tâches faciles.

En ce qui concerne la grandeur relative, il faut d'abord trouver une caractéristique pouvant représenter la grandeur d'écriture. Cette caractéristique doit avoir, autant que possible, une relation linéaire avec la grandeur d'écriture. En d'autres mots, dans une écriture ordinaire ayant une grandeur relativement stable, sa valeur ne doit pas subir de grandes variations. Les premiers candidats, pour ce faire, sont la largeur et la hauteur des tracés. L'avantage de ces paramètres est qu'ils sont facilement calculables à partir des coordonnées des points extrêmes des tracés. Toutefois, ils ne sont pas des candidats idéals pour représenter la grandeur d'écriture. Quant à l'étendue horizontale ou la largeur des tracés, elle est très variable d'un tracé à l'autre, notamment dans les écritures purement cursive et cursive mixte. L'étendue verticale ou la hauteur des tracés est plus stable que leur largeur. Cependant, même dans une écriture de grandeur stable, la hauteur des lettres majuscules et de celles qui ont des parties supérieures ou inférieures (comme les lettres « b », « l », « p » et « g ») peut être deux fois la hauteur des autres lettres ou même plus grande (par exemple, comparer la hauteur de la lettre « b » avec celle de la lettre « o » dans figure 3.16(a)). La différence est encore plus grande pour les lettres ayant les parties supérieure et inférieure, comme la lettre « f » dans l'écriture purement cursive.

La caractéristique que nous proposons pour représenter la grandeur d'écriture est la moyenne des distances verticales entre les minima et les maxima consécutifs des tracés :

$$Haut\_Moy\_extrema = \frac{\sum_{i=1}^{N_{min\_max}} h\_ext_i}{N_{min\_max}} \tag{3.15}$$

où : *Haut_Moy_extrema* est la distance moyenne verticale entre les paires de minima et maxima consécutifs;

$h\_ext_i$ est la distance verticale entre la i-ème paire de minima et maxima;

$N_{min\_max}$ est le nombre de paires de minima et maxima consécutives.

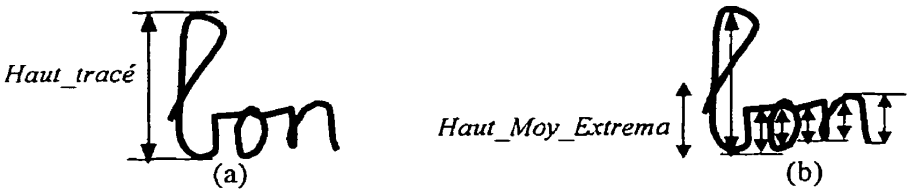



Figure 3.16. Deux mesures de grandeur des tracés : (a) la hauteur absolue du tracé, (b) la hauteur moyenne des minima et maxima consécutifs. La *Haut_Moy_Extrema* est la valeur moyenne des distances montrées dans la figure.

La figure 3.16(b) montre les distances entre les minima et maxima consécutifs ainsi que leur valeur moyenne *Haut_Moy_Extrema*. En comparant les figures 3.16(a) et 3.16(b), on constate que la hauteur absolue du tracé *Haut_Tracé* est plus grande que la *Haut_Moy_Extrema*, et que la valeur de cette dernière est plus près de la hauteur de la zone médiane. Grâce à l'opération du filtrage de l'équation (3.15) et la présence des lettres sans parties supérieure et inférieure dans les tracés, les lettres ayant ces parties ont un effet beaucoup moins important dans la valeur de *Haut_Moy_Extrema* que dans celle de *Haut_Tracé*. Cela montre que *Haut_Moy_Extrema* est plus stable que *Haut_Tracé* et donc représente mieux la grandeur des tracés. Ainsi, pour représenter la grandeur des mots, la moyenne des *Haut_Moy_Extrema* de leurs tracés peut être utilisée.

Quant à l'évaluation de la complexité de la structure des tracés, les extrema peuvent encore être utiles. Pour ce faire, nous proposons d'utiliser le nombre d'extrema dans un tracé. Les Accent_Point_TBars sont des tracés simples ayant un nombre limité d'extremas. Ce nombre ne dépasse pas 3 pour les Accent_Point_TBars, dans les langues française et anglaise, alors que les lettres de l'alphabet ont, en général, plus de 3 extrema (Leroy, 1994).


(a) (b)

Figure 3.17. Position relative des tracés : (a) le tracé point en dessous du point extrême haut du tracé principal; (b) le tracé point au-dessus de la moyenne des $y$ des maxima.

La position relative des tracés est le dernier critère que nous avons proposé pour reconnaître les Accent_Points_TBars. Toujours en raison de leurs parties supérieure et inférieure, les points extrêmes haut et bas des tracés ne peuvent être utilisées pour déterminer la position relative des tracés dans une ligne de texte. Prenons l'exemple de la figure 3.17(a). Pour les humains, le « point » sur le mot « Bonjour » est situé au-dessus du tracé formant la partie principale du mot. Tandis que sa coordonnée $y$ est plus grande que celle du point extrême haut du tracé principal et avec une simple comparaison de ces coordonnées, la position du « point » ne sera pas jugé au-dessus du mot. Toutefois, comme la figure 3.17(b) le montre, le résultat de comparaison de la coordonnée $y$ du tracé « point » avec la moyenne des coordonnées $y$ des maxima $y_{max\_moy}$ est que le point est situé au-dessus du mot. Nous proposons donc d'utiliser, dans le calcul de la position relative des tracés, la moyenne des $y$ des maxima $y_{max\_moy}$, la moyenne des $y$ des minima $y_{min\_moy}$ ainsi que la moyenne des $y$ de tous les minima et maxima $y_{med\_moy}$ des tracés. $y_{max\_moy}$ et $y_{min\_moy}$ représentent respectivement les bornes supérieure et inférieure de la zone médiane d'un tracé et $y_{med\_moy}$ est la coordonnée $y$ de la ligne médiane de cette zone (figure 3.18). En excluant les Accent_Point_TBars, il est possible d'extraire les caractéristiques de la zone médiane d'un mot à partir de celles de ses tracés, avec un simple calcul de la moyenne.

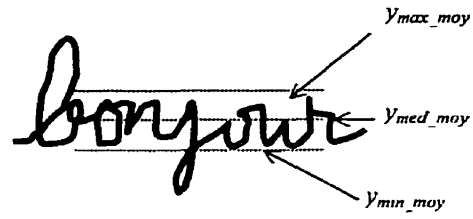



Figure 3.18. Coordonnées $y$ des bornes supérieure et inférieure ($y_{max\_moy}$ et $y_{min\_moy}$) et la ligne médiane $y_{med\_moy}$, de la zone médiane d'un tracé, extraites des coordonnées $y$ de ses minima et maxima.

Nous avons expliqué l'avantage de l'utilisation des extrema dans l'extraction des caractéristiques des tracés pour la reconnaissance des Accents_Point_TBars. Nous allons continuer avec la méthode que nous avons appliquée pour calculer la position des extrema et d'autres caractéristiques relatives à ces entités.

### *3.4.1.2.2 CALCUL DES EXTREMA D'UN TRACÉ*

La liste des extrema d'un tracé comprend le premier point du tracé, la liste des minima et des maxima, en alternance, et le dernier point du tracé. Les minima et les maxima sont les sommets du tracé où la dérivée est zéro. Toutefois, en raison des imperfections dans l'écriture, on ne doit pas interpréter tous les sommets du tracé comme un minimum ou un maximum. Le tremblement de la main du scripteur, la résolution spatiale et le taux d'échantillonnage de la tablette ou de l'écran se combinent pour produire des imperfections sous forme de faux sommets minuscules. Ce phénomène se produit fréquemment à la pose et à la levée du crayon, ce que l'on appelle les crochets. La caractéristique principale de ces imperfections est leur grandeur, en général, de l'ordre de quelques points. Un autre cas à considérer pour le calcul des minima et maxima est la possibilité de plateaux lesquels sont des parties horizontales des tracés (la pente est zéro). Dans ce cas, il faut prendre le point au milieu du plateau comme un minimum ou un maximum.

La procédure de l'extraction des extrema d'un tracé comprend les étapes suivantes :

1- Extraire la liste de tous les sommets du tracé;

2- Extraire les vrais minima et maxima à partir de la liste des sommets.

La première étape consiste à parcourir la liste des points du tracé pour trouver les points où la dérivée est zéro ou négative. Ces points seront insérés ensuite dans une liste de sommets.

Dans la deuxième étape, les sommets de l'étape 1 seront évalués pour construire la liste de vrais extrema du tracé. L'algorithme commence par l'insertion du premier point du tracé dans la liste des extrema. Ensuite, on parcourt la liste des sommets pour enlever les crochets et les faux sommets causés par l'imperfection du tracé. Il faut également identifier les plateaux pour en prendre le point au milieu. L'autre aspect de l'algorithme est le respect de l'alternance entre les minima et les maxima. L'algorithme se termine en insérant le dernier point du tracé dans la liste des extrema tout en vérifiant les crochets probables à cet endroit pour les éliminer.

La figure 3.19 montre le résultat de l'exécution du programme que nous avons développé pour l'extraction des extrema et leurs caractéristiques, pour un texte manuscrit. Dans cette figure, les extrema sont présentés à l'aide du caractère '×'.

En plus de la liste des extrema, le programme extrait deux listes séparées pour les minima et les maxima du texte. Ces trois listes nous permettent de calculer les caractéristiques suivantes : la distance moyenne verticale entre les minima et maxima consécutifs *Haut_extrema* selon l'équation (3.15); les bornes supérieure et inférieure de la zone médiane des tracés $y_{max\_moy}$ et $y_{min\_moy}$ respectivement ainsi que la coordonnée $y$ de la ligne médiane de cette zone $y_{med\_moy}$ (figure 3.18). Ces informations ainsi que le nombre d'extrema des tracés seront utilisés dans l'algorithme de la reconnaissance des Accent_Point_TBars. Il faut ajouter que le programme offre également les options suivantes à l'utilisateur: afficher le texte en couleur ou en noir et blanc, effacer le texte,

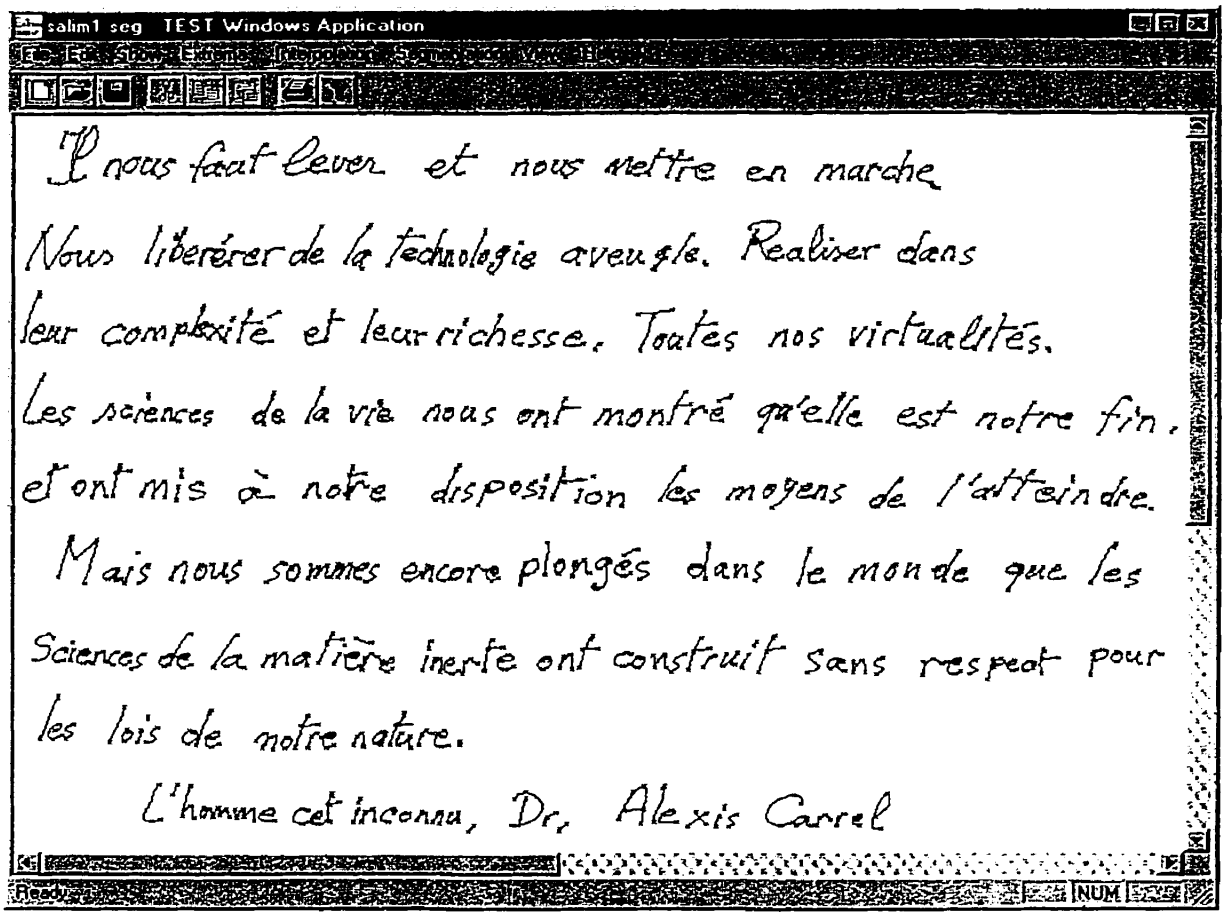



Figure 3.19. Un texte manuscrit et les extrema de ses tracés.

afficher et effacer tous les sommets, les extrema, les minima et les maxima, *Haut_extrema*, $y_{max\_moy}$, $y_{min\_moy}$ et $y_{med\_moy}$ des tracés.

### *3.4.1.2.3. RECONNAISSANCE DES ACCENTS, DES POINTS, DES VIRGULES ET DES BARRES DE « T »*

**Algorithme de la reconnaissance**

À la lumière des explications mentionnées précédemment sur les caractéristiques des Accent_Point_TBars, nous avons développé un algorithme à la fois simple et performant pour la reconnaissance de ces signes diacritiques. Cet algorithme est présenté dans la figure 3.20.

```
RECOG_ACCENT_POINT_TBAR
/* Déterminer si le tracé actuel est un accent, un point, une barre de 't' ou une   */
/* ponctuation. Cet algorithme est basé sur les caractéristiques tirées             */
/* des extrema du tracé et du mot actuel .                                          */
/* L'algorithme retourne VRAI si le tracé est classifié dans la catégorie des       */
/* Accent_Point_TBars, sinon il retourne FAUX.                                      */

Si NbExtremaTracéActuel ≤ 3 , alors /* Le tracé à une structure simple */
    /* Structure très simple, sans minima ni maxima */
    Si NbExtremaTracéActuel = 2 , alors
        Si HauteurTracéActuel ≤ 3, alors   /* Tracé est absolument petit */
            Retourner VRAI;
        /* Tracé est très petit par rapport aux autres tracés du mot actuel */
        Sinon, Si HauteurTracéActuel < (0.25* HautExtremaMot), alors
            Retourner VRAI;
        /* Tracé situé en dessous du mot actuel, le cas des ponctuations */
        Sinon, Si y_min TracéActuel > (y_med_moy Mot + 0.5* HautExtremaMot), alors
            Retourner VRAI;
    Sinon  /* Tracé avec 3 extrema, c'est-à-dire un minima ou un maxima */
        /* Tracé est plus petit que les autres tracés du mot actuel */
        Si HauteurTracé < HautExtremaMot, alors
            /* Tracé actuel est complètement situé au-dessus du tracé */
            /* suivant ou du tracé précédent.                          */
            Si y_max TracéActuel < y_min TracéPrécéd ou
               y_max TracéActuel < y_min TracéSuiv, alors
                /* Tracé situé au dessus du mot actuel également */
                Si y_med_moy TracéActuel < y_med_moy Mot , alors
                    Retourner VRAI;
            /* Tracé bien situé au dessus du mot actuel */
            Sinon, Si y_med_moy TracéActuel < y_med_moy Mot et
                    y_max TracéActuel < y_max_moy Mot, alors
                Retourner VRAI;
            /* Le tracé actuel se chevauche avec le tracé précédent ou */
            /* le tracé suivant.                                        */
            Sinon, Si ChevauchTracéActuelPrécé = VRAI ou
                    ChevauchTracéActuelSuiv  = VRAI, alors
                /* Tracé au dessus du mot actuel */
                Si y_max TracéActuel < y_max_moy Mot, alors
                    Retourner VRAI;
                /* Tracé est très petit par rapport aux autres tracés du
                   mot actuel */
                Sinon, Si HauteurTracéActuel < 0.25* HautExtremaMot, alors
                    Retourner VRAI;

Sinon Retourner FAUX;
```

Figure 3.20. Algorithme de la reconnaissance des accents, des points, des barres de 't' et des ponctuations.

L'algorithme commence par la vérification du nombre d'extrema du tracé actuel. Si ce nombre est supérieur à 3, on décide immédiatement que le tracé n'est pas un Accent_Point_TBar (il est à noter que le bruit causé par les crochets et les faux sommets, qui peut augmenter erronément le nombre d'extrema et peut être la source d'erreur à cette étape, est déjà éliminé dans la phase d'extraction des extrema). Autrement, si le nombre d'extrema est égal à 2, le tracé est probablement un Accent_Point_TBar, mais pour prendre la décision finale, il faut effectuer d'autres tests. Si la hauteur du tracé ($y_{max}TracéActuel - y_{min}TracéActuel$) est extrêmement petite, il est un Accent_Point_TBar. Autrement, s'il est très petit par rapport à la distance moyenne entre les extrema des tracés du mot actuel, il est également considéré comme un Accent_Point_TBar. Parfois, le point et la virgule de la fin d'une phrase ne sont pas petits, mais on peut les reconnaître à partir de leur position dans la ligne. Pour ce faire, on se sert de la coordonnée de la ligne médiane des extrema du mot actuel et de la distance moyenne entre les extrema de ses tracés. Ce test nous permet de distinguer les ponctuations des tracés comme 'i' pouvant être écrits avec 2 extrema, mais normalement au-dessus de la ligne médiane.

Si le nombre d'extrema du tracé actuel est 3, il pourrait être un Accent_Point_TBar comme tout autre tracé simple. En conséquence, il faut distinguer ces cas, à l'aide de tests supplémentaires sur la grandeur, la position et le chevauchement du tracé avec les tracés voisins. Comme l'algorithme de la figure 3.20 le montre, pour effectuer ces tests, nous avons principalement utilisé les caractéristiques extraites des extrema des tracés.

**Résultats de la reconnaissance**

La figure 3.21 illustre les Accents_Point_TBars et les ponctuations d'un texte manuscrit, détectés par l'algorithme de reconnaissance présenté à la figure 3.20. On peut constater la bonne performance de l'algorithme, notamment dans le cas de ce type d'écriture, c'est-à-dire l'écriture isolée. En fait, distinguer les Accent_Point_TBars des autres tracés simples est plus compliqué dans l'écriture isolée que dans les écritures

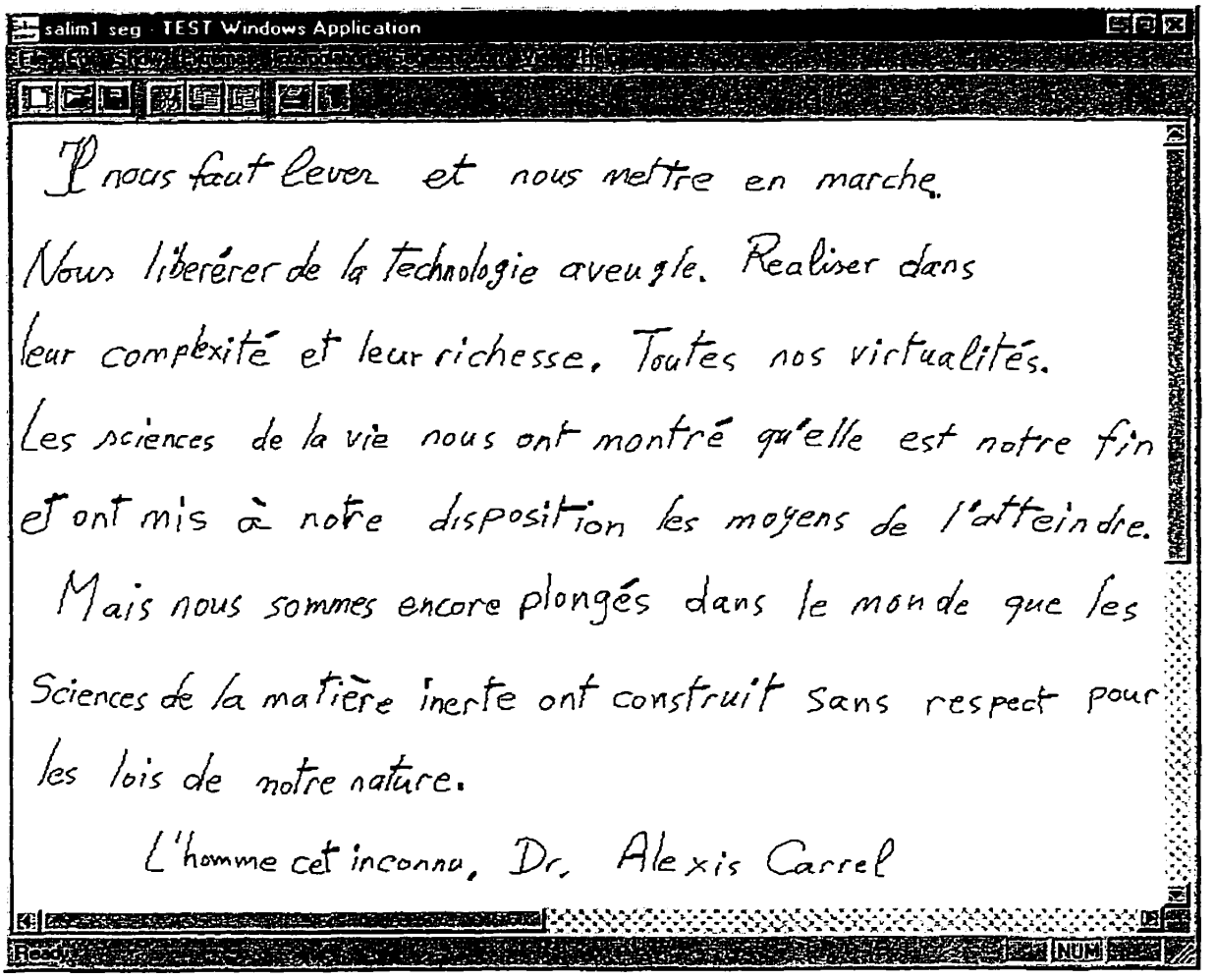



Figure 3.21. Résultat de l'application de l'algorithme de reconnaissance des Accent_Point_TBar sur un texte manuscrit. Les Accent_Point_TBars sont montrés plus épais que les autres tracés.

purement cursive et cursive mixte. La raison de cette complexité est la suivante : l'algorithme de reconnaissance est basé sur le fait que, dans un texte, les Accents_Points_TBar ont, en général, moins d'extrema que d'autres tracés. Dans l'écriture cursive, les tracés sont composés de plusieurs lettres et ont donc un grand nombre d'extrema. Tandis que dans une écriture isolée, les tracés composés d'une seule lettre sont nombreux. Par conséquent, le nombre de tracés simples (comme les parties principales des lettres 'i' et 't' et la lettre 'l') dans l'écriture isolée, est plus élevé que

dans l'écriture cursive. Toutefois, la figure 3.21 montre que notre algorithme à bien réussi à reconnaître les Accent_Point_TBars dans une écriture isolée.

En somme, sur tous les signes diacritiques présents dans notre base de données, nous avons obtenu un taux de reconnaissance correct de plus de 93%; ce qui est, encore une fois, une preuve numérique de performance de notre algorithme.

Nous avons démontré, qu'en estimant l'inclinaison d'écriture et en excluant les Accent_Point_TBars de la procédure de l'extraction de la distance entre les boîtes de tracés, on peut améliorer la performance de cette mesure de distance. Nous avons également expliqué comment nous avons procédé pour estimer l'inclinaison d'écriture et pour reconnaître les accents, les points, les barres de 't' et les ponctuations dans un texte manuscrit. Dans les chapitres 4 et 5, nous soulignerons l'importance de ces modifications dans les résultats de segmentation en-ligne de textes manuscrits en mots, en comparant ces résultats sans et avec l'application de nos propositions.

Nous nous pencherons maintenant sur les problèmes de la deuxième mesure choisie de distances inter-tracés, c'est-à-dire la distance horizontale RL.

### 3.4.2 PROPOSITIONS POUR AMÉLIORER LES DISTANCES HORIZONTALES INTER-TRACÉS

Dans la section 2.2.2.5, nous avons présenté une discussion sur les points faibles de la mesure de distance horizontale entre les tracés. Nous rappelons que cette mesure souffre principalement des trois problèmes suivants : la dépendance au chevauchement vertical, les distances horizontales irréalistes et la sensibilité au décalage vertical des tracés. Les accents et les points des mots sont la source principale des deux premiers problèmes. Pour les résoudre, il faut alors une étape de reconnaissance préliminaire afin d'éliminer les accents et les points du calcul des distances horizontales. Dans la section 3.2.2.1.2, nous avons montré l'utilité d'une phase de reconnaissance des Accent_Point_TBars pour résoudre le problème de la sensibilité de la mesure de distance entre les boîtes (Dist_BB ou Dist_Paral) au chevauchement horizontal des tracés. Heureusement, on peut se servir des résultats de la même procédure pour

résoudre les deux premiers problèmes de la mesure de distance horizontal RL et améliorer sa performance.

Quant à la troisième catégorie de problèmes, nous ne connaissons aucun moyen direct pour la résoudre de façon définitive. La mesure RL est, par nature, sensible au décalage vertical des tracés. Toutefois, il faut mentionner qu'un décalage vertical minime des tracés ne produit un changement significatif, dans la valeur de RL, que si le tracé à gauche se termine avec une partie horizontale ou presque horizontale et le tracé à droite commence avec la même forme (figure 2.14). En plus, le respect d'écrire horizontalement entre les lignes directrices et ne pas changer la position horizontale des mots dans la ligne aiderait à diminuer le nombre de cas problématiques causés par la sensibilité de la mesure horizontale au décalage vertical des tracés.

## 3.5 CONCLUSION

Dans ce chapitre, nous avons d'abord expliqué les aspects matériel et logiciel des interfaces implantées pour l'acquisition de textes manuscrits. Nous avons souligné les points forts de cette interface, notamment la liberté offerte aux scripteurs pour composer leur texte sans aucune contrainte temporelle ni séquentielle. Le seul point à respecter par les scripteurs consiste à écrire entre les lignes horizontales. Nous avons toutefois expliqué pourquoi cela ne constitue pas une contrainte intolérable pour les scripteurs. Nous avons également présenté les opérations effectuées pour avoir une acquisition performante et fiable de données, qui seront utilisées dans un module de segmentation en-ligne, tout en respectant la liberté de composition des scripteurs.

Deux différentes catégories de caractéristiques sont traitées dans ce travail : conventionnelles et proposées. Les caractéristiques conventionnelles sont basées sur les mesures sélectionnées parmi les mesures classiques d'espacement entre les entités manuscrites, présentées dans le chapitre 2. La deuxième catégorie constitue celle que nous avons proposée pour la première fois afin de résoudre les problèmes de caractéristiques conventionnelles. Dans ce chapitre, nous avons présenté ces catégories

de caractéristiques ainsi que leurs étapes d'extraction à partir de données manuscrites acquises.

Dist_BB et RL sont les principales caractéristiques conventionnelles utilisées dans ce travail. La difficulté principale pour calculer les Dist_BB, dans un système en-ligne sans contrainte au niveau du temps, de la séquence temporelle, de la forme et de la position des tracés, consiste à tenir compte d'un nombre très élevé de séquences, de formes manuscrites et de positionnements relatifs possibles entre les tracés. Notre algorithme du calcul de Dist_BB réduit considérablement le nombre de cas à traiter pour ensuite produire des mesures précises de distances entre les rectangles.

Pour la première fois dans un système en-ligne de segmentation en mots, nous avons utilisé les distances horizontales RL pour présenter l'espacement entre les entités manuscrites. Dans les systèmes hors-ligne, grâce au balayage optique horizontal, ces distances sont facilement accessibles dans l'image des textes. Toutefois, dans un système en-ligne, en raison de l'aspect dynamique de composition de texte, le calcul des distances horizontales nécessite des opérations supplémentaires, surtout lorsque le scripteur dispose les libertés temporelle et séquentielle; ce qui est le cas de notre système de segmentation. Dans ce chapitre, nous avons présenté les opérations effectuées par notre interface pour extraire les distances horizontales.

Nous avons également expliqué les étapes de calcul de deux autres caractéristiques que nous avons ajouté aux caractéristiques ci-dessus mentionnées. Il s'agit de avgHeight et avgDistBB représentant respectivement la hauteur moyenne et la distance moyenne entre les rectangles de $N$ tracés précédents. Ces caractéristiques seront utilisées dans le chapitre suivant comme des facteurs de normalisation des caractéristiques principales.

Tout en soulignant les lacunes des méthodes conventionnelles, nous avons présenté nos propositions pour les améliorer. Nous avons démontré que, pour résoudre le problème de la sensibilité de la mesure de Dist_BB à l'inclinaison d'écriture, il faut estimer l'inclinaison relative des tracés et remplacer la distance entre les rectangles par la distance entre les parallélogrammes Dist_Paral. À cette égard, les difficultés de

l'estimation de l'inclinaison dans un système en-ligne ont été énumérées, notamment la nécessité de réduire le temps de calcul, la quantité limitée de données disponibles pour ce faire comparativement aux systèmes hors-ligne et les questions de choix des données pertinentes et de la meilleure façon de les combiner. Nous avons ensuite présenté l'algorithme d'estimation de l'inclinaison d'un tracé, ainsi qu'une méthode de calcul adaptatif de la pente moyenne des tracés pour tenir compte des variations de l'inclinaison de texte et pour permettre d'obtenir une mesure plus précise de distance entre les parallélogrammes. Les opérations de calcul des parallélogrammes englobant des tracés et leur distance ont également été expliquées.

Nous avons mentionné que, dans le cas de Dist_BB et Dist_Paral, le chevauchement horizontal des tracés faisant partie de deux mots différents, cause une erreur de segmentation. Ce type de chevauchement est souvent produit par des accents, des points et surtout des barres de « t ». Pour résoudre ce problème, nous avons proposé d'ajouter une étape préliminaire de la reconnaissance des signes diacritiques et nous avons analysé les caractéristiques de ces entités manuscrites à partir de leur grandeur, leur structure et leur position par rapport aux autres tracés. La difficulté d'estimation de ces caractéristiques, en raison des variations intrinsèques d'écriture, a également été démontrée. Nous avons proposé d'utiliser les mesures extraites des extrema des tracés pour estimer ces caractéristiques. Les détails du calcul des coordonnées des extrema et de l'extraction des caractéristiques à partir de ces coordonnées ont été présentés. Nous avons également expliqué notre algorithme original de la reconnaissance de accents, des points, des barres de « t » et des ponctuations, basé sur les extrema des tracés. Le pourcentage élevé (93%) de reconnaissance des signes diacritiques est une preuve de performance de cet algorithme. Quant à la mesure de la distance horizontale RL, nous avons justifié que, la suppression des accents et des points après la phase de la reconnaissance, contribue à améliorer la performance de cette mesure en éliminant les cas problématiques de segmentation causés par ces entités manuscrites.

Les caractéristiques conventionnelles et proposées dans ce chapitre sont utilisées dans les phases d'apprentissage et de test des modules de segmentation en-ligne de texte

manuscrit en mots. Dans les chapitres 4 et 5, nous présenterons les méthodes de classification utilisées pour la segmentation ainsi que les résultats obtenus avec ces méthodes.

# CHAPITRE 4

# CLASSIFICATEURS NEURONAUX POUR LA SEGMENTATION DE TEXTES MANUSCRITS EN MOTS

La segmentation en-ligne d'une ligne de texte manuscrit en mots est une procédure de classification dans laquelle les entités à classifier sont les distances inter tracés. L'objectif de cette procédure consiste donc à décider, pour chaque tracé du texte, soit de l'ajouter à un mot du texte, soit de lui créer un nouveau mot. La décision est prise principalement selon la grandeur de l'espacement entre le tracé actuel et le mot le plus proche situé dans la même ligne. Comme tout autre système de classification, les parties principales de notre système sont le module de l'extraction des caractéristiques et le classificateur. Le premier module et les caractéristiques extraites ont été présentés dans les chapitres 2 et 3. Dans ce chapitre et le chapitre suivant, nous nous pencherons sur le problème de la classification des distances inter-tracés en utilisant les caractéristiques extraites. Notre objectif dans les chapitres 4 et 5 consiste à concevoir le ou les classificateur qui permettent d'exploiter au maximum les caractéristiques discriminatives extraites, relativement aux espacements inter-tracés, afin de produire les meilleurs résultats de segmentation en mots, tout en respectant les contraintes des systèmes interactifs et en-ligne.

Le chapitre 4 est consacré aux classificateurs neuronaux. Toutefois, nous le commencerons avec une discussion sur le choix du type de classificateurs dans notre système, portant sur les classificateurs classiques bayésien et K plus proches voisins. Nous nous pencherons ensuite sur les aspects théoriques des réseaux neuronaux artificiels en général, et les réseaux multicouches en particulier. Nous continuerons ce chapitre avec la présentation des textes manuscrits utilisés dans les phases d'apprentissage et de test des réseaux neuronaux[1]. Nous aborderons également l'interface de segmentation manuelle que nous avons conçue pour préparer les données à présenter aux classificateurs. Cela nous amènera à la présentation des deux groupes de classificateurs neuronaux implantés pour la segmentation de textes manuscrits en mots, incluant leurs structures, leurs entrées, leurs phases d'apprentissage et de test ainsi que les résultats obtenus et l'analyse de ces résultats. Nous terminerons ce chapitre avec la présentation des conclusions obtenues.

# 4.1. CHOIX DU TYPE DE CLASSIFICATEURS

Dans un problème de classification statistique comme celui de notre système de segmentation, différents types de classificateurs peuvent être envisagés, tels que les classificateurs classiques de types bayésiens et K plus proches voisins et les classificateurs neuronaux. Parmi ces choix possibles, nous avons opté pour les classificateurs neuronaux qui seront expliqués en détail, plus loin dans ce chapitre. Les caractéristiques de ces classificateurs sont présentées dans les sections suivantes ainsi que les raisons pour lesquelles nous les avons choisis pour résoudre notre problème de segmentation. Dans cette section, nous essayons donc de nous pencher sur d'autres types de classificateurs, bayésiens et K plus proches voisins, et d'expliquer pourquoi ils n'ont pas été utilisés dans notre système de segmentation en-ligne de textes manuscrits en mots.

---

[1] Il faut mentionner que les mêmes textes sont utilisés pour l'entraînement et le test des classificateurs présentés dans le chapitre 5.

Les classificateurs bayésiens sont probablement les classificateurs les plus abordés dans la littérature sur la reconnaissance de formes (Fukunaga, 1972) (Chen, 1973) (Chen, 1999). Ils sont basés sur la théorie de Bayes et calculent la frontière de décision en fonction des probabilités a priori des classes et des fonctions de densité conditionnelle. Si pour un problème donné, ces probabilités et fonctions de densité sont connues, la frontière de décision pourra facilement être calculée et le classificateur obtenu sera le classificateur optimal avec l'erreur minimale de classification. Sinon, lorsque les probabilités a priori des classes et surtout la forme et/ou les paramètres des fonctions de densité conditionnelle sont inconnus, il faut avoir recours à des méthodes paramétriques et non paramétriques pour les estimer. Cette procédure d'estimation, en plus de produire des résultats sous-optimaux, est coûteuse en terme de temps d'exécution et de complexité des opérations, notamment lorsque les fonctions de densité ne sont pas Gaussiennes.

Dans le cas de notre système de segmentation, l'analyse de formes des distributions de densité conditionnelle des caractéristiques utilisées a démontré que ces distributions n'ont pas une forme Gaussienne (des exemples de ces distributions sont présentés dans les figures 4.7, 4.13, 4.14 et 4.15). En plus, les probabilités a priori des classes, c'est-à-dire les espacements intra-mot et inter-mots, sont très variables d'un texte à l'autre et surtout d'un style d'écriture à l'autre. En somme, à cause de l'impossibilité de l'application directe de la règle de décision bayésienne, de la diversité de formes des distributions de densité, du fait qu'elles ne sont pas Gaussiennes et la grande variabilité des probabilités a priori des classes en question, nous avons décidé de ne pas utiliser des classificateurs bayésiens classiques dans notre système interactif de segmentation en-ligne dans lequel la réduction de temps de traitements et de complexité des opérations est d'une importance primordiale. Cependant, comme nous allons le démontrer dans la section 5.1.1.2, les classificateurs présentés dans le chapitre 5 sont interprétables selon une approche bayésienne, pouvant atteindre une précision élevée de classification, sans toutefois souffrir des problèmes des classificateurs bayésiens classiques dans les systèmes en-ligne.

Les classificateurs K plus proches voisins[2] sont également parmi les classificateurs classiques les plus abordés dans les références sur la reconnaissance de formes (Fukunaga, 1972) (Chen, 1973) (Chen, 1999). Selon l'algorithme utilisé dans ces classificateurs, un patron donné est classifié dans une classe possédant le plus grand nombre de patrons parmi ses K plus proches voisins. L'avantage principal de ce type de classificateurs est la simplicité de son algorithme de classification. Toutefois, ils souffrent des problèmes du nombre élevé d'opérations à effectuer et de la quantité importante d'information à stocker. Pour résoudre ces problèmes, certaines solutions ont été proposées consistant, en principe, à extraire un sous-ensemble de patrons et à effectuer l'opération de K plus proches voisins uniquement sur ces patrons. Cependant, l'opération d'extraction d'un sous-ensemble pertinent à partir de l'ensemble de points de référence est, en général, une opération coûteuse en temps produisant, en plus, des résultats sous-optimaux par rapport à l'algorithme original.

En raison des problèmes mentionnés ci-dessus, les classificateurs K plus proches voisins ne peuvent être considérés, en principe, comme de bons classificateurs dans notre système de segmentation, lequel est une application en-ligne. Toutefois, afin d'effectuer une comparaison expérimentale de leurs performances avec celles de nos classificateurs proposés, nous les avons conçus et implantés. Les résultats obtenus avec les classificateurs K plus proches voisins, présentés dans l'annexe D, montrent l'infériorité des classificateurs KNN par rapport à nos classificateurs proposés, en ce qui a trait au temps et au taux de classification correcte des distances inter-tracés.

# 4.2. RÉSEAUX NEURONAUX ARTIFICIELS

## 4.2.1 APERÇU GÉNÉRAL

« Les réseaux neuronaux sont une métaphore des structures cérébrales: des assemblages de constituants élémentaires, qui réalisent chacun un traitement simple voire simpliste, mais dont l'ensemble fait émerger des propriétés globales dignes

[2] K-Nearest Neighbor ou KNN en anglais

d'intérêt. Chaque constituant fonctionne indépendamment des autres, de telle sorte que l'ensemble est un système parallèle, fortement interconnecté»[3]. Des définitions semblables se trouvent dans les différentes références sur les réseaux neuronaux dont entre autres (Haykin, 1999). Selon (Haykin, 1999) : « un réseau neuronal est un processeur massivement parallèle et distribué, construit des unités de traitements simples. Il a la capacité de stocker des informations expérimentales et de les mettre en disponibilité. Il ressemble au cerveau dans deux aspects suivants : (1) l'information est acquise de l'environnement par une procédure d'apprentissage; (2) les poids synaptiques entre les neurones sont utilisés pour stocker les informations». Les termes d'unités de traitement simples ou des constituants élémentaires utilisés dans les définitions ci-dessus réfèrent aux «neurones artificiels». La figure 4.1 montre le modèle d'un neurone proposé pour la première fois par W. McCulloch et W. Pitts (McCulloch, 1943). Dans cette figure, les $x_i$ et les $w_i$ forment les vecteurs des entrées et des poids synaptiques du neurone, respectivement. Le $\theta$ est le biais, $v$ est la somme pondérée des entrées, $\varphi$ est la fonction d'activation et $y$ est la sortie du neurone. Les valeurs de $v$ et de $y$ sont calculées selon les expressions suivantes :

$$v = \sum_{i=1}^{N} w_i x_i + \theta \tag{4.1}$$

$$y = \varphi(v) \tag{4.2}$$

Les stimulations sont reçues par le vecteur des entrées $X = \{x_1, x_2, \ldots, x_N\}$. Chaque entrée $x_i$ est ensuite pondérée par le poids synaptique correspondant $w_i$ qui pourrait être positif ou négatif. Les poids positifs et négatifs sont respectivement appelés les synapses *excitatrices* et *inhibitrices*. Le neurone calcule ainsi une somme pondérée de ses entrées (équation 4.1). Selon l'équation (4.2), La fonction d'activation $\varphi$ est ensuite appliquée à cette somme pondérée pour produire la sortie du neurone $y$. Dans le modèle de

[3] Nous avons emprunté cette définition de (Belayo, 1996).

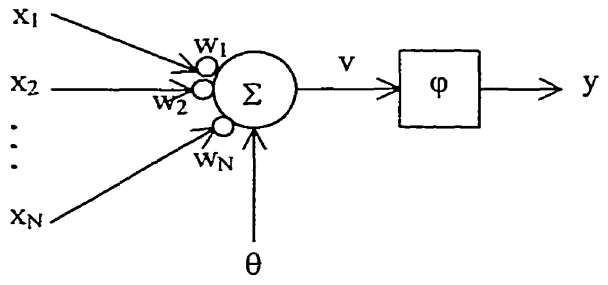



Figure 4.1. Modèle d'un neurone artificiel.

McCulloch et Pitts, la fonction d'activation est une fonction à échelon permettant de prendre une décision en fonction de $v$ et de $\theta$:

$$\text{si} \quad v > \theta, \qquad \text{alors} \quad y = +1$$

$$\text{sinon} \quad y = -1 \tag{4.3}$$

Dans l'équation (4.3), le biais $\theta$ joue le rôle d'un seuil de décision et la sortie $y$ est binaire. La fonction en échelon n'étant pas dérivable, elle deviendra donc inutilisable si dans une application (comme les Perceptron multicouches) la dérivé de la fonction d'activation est nécessaire. En fait, la fonction d'activation est un élément du modèle qui dépend en partie du problème. Parmi les fonctions dérivables les plus utilisées, on peut nommer les fonctions d'allure *sigmoïde*. Ce sont des équations dont la forme générale est celle d'une tangente hyperbolique avec des valeurs comprises entre −1 et +1 (fonction *tansig*) ou avec des valeurs comprises entre 0 et +1 (fonction *logsig*). Les fonctions *logsig* et *tansig* sont montrées dans les équations (4.4) et (4.5) respectivement.

$$logsig(x) = \frac{1}{1+e^{-x}} \tag{4.4}$$

$$tansig(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{4.5}$$

Un réseau neuronal est un assemblage de neurones. En interconnectant les neurones de différentes façons, on peut construire diverses structures neuronales[4]. Un réseau neuronal peut être entraîné sur des données acquises. Pendant la phase d'entraînement, un ensemble de patrons d'entraînement avec une distribution inconnue est présenté au réseau neuronal et les paramètres libres du réseau (les poids et les biais de ses neurones) sont ajustés afin de minimiser une fonction de coût ou maximiser une probabilité. On souhaite ainsi que le réseau entraîné puisse bien effectuer la *généralisation*, c'est-à-dire classifier, estimer ou reconnaître (selon l'application) les patrons faisant partie d'un ensemble autre que celui utilisé pour l'entraînement, mais ayant la même distribution que les exemples d'apprentissage. Parmi les tâches bien adaptées au traitement par réseaux neuronaux, on trouve l'association, la discrimination, l'estimation et la classification. Toutes ces tâches sont très différentes de celles traitées par l'informatique traditionnelle telles que le calcul scientifique et la gestion de bases de données. En fait, lorsqu'on s'éloigne des tâches facilement formulables sous forme d'expressions logiques, l'informatique traditionnelle et l'intelligence artificielle, qui se basent sur une logique préalablement définie, échouent.

Dans ce projet, notre objectif consiste à classifier les distances inter-tracés, et pour ce type de tâches complexes, non linéaires et mal connues, les réseaux neuronaux et en particulier les réseaux de type Perceptron à une couche ou à couches multiples (Perceptron multicouche) sont d'excellents candidats. Étant donné que nous avons utilisé les deux types de réseaux pour implanter nos classificateurs, nous nous pencherons, dans la section suivante, sur les aspects théoriques des Perceptrons à une couche et à couches multiples.

---

[4] Pour en connaître davantage sur les différents types de réseaux neuronaux, le lecteur intéressé est invité à consulter divers ouvrages sur ce sujet, dont entre autres (Haykin, 1999).

## 4.2.2 PERCEPTRON À UNE COUCHE ET À COUCHES MULTIPLES

### 4.2.2.1 PERCEPTRON À UNE COUCHE

La figure 4.2 montre la structure d'un réseau de type Perceptron à une couche. Cette dernière est une structure composée de plusieurs neurones ayant en commun les mêmes entrées. Au sein d'une couche, chaque neurone agit indépendamment des autres sans recevoir, en particulier, aucune connexion en provenance des autres neurones de cette couche. La règle du Perceptron (Rosenblatt, 1958) est destinée, dans une phase d'apprentissage, à modifier les poids de chacun des neurones dans la bonne direction lorsqu'une mauvaise classification survient. L'adaptation des poids est effectuée selon l'équation suivante :

$$w_{ij}(n+1) = w_{ij}(n) + \eta[d_j(n) - y_j(n)]\varphi_j'(v_j(n))x_i(n) \tag{4.6}$$

où: $w_{ij}(n+1)$ est la nouvelle valeur du *i-ème* poids du *j-ème* neurone;

$w_{ij}(n)$ est l'ancienne valeur du *i-ème* poids du *j-ème* neurone;

$x_i(n)$ est la *i-ème* entrée à l'itération *n*;

$y_j(n)$ est la *j-ème* sortie à l'itération *n*;

$d_j(n)$ est la valeur désirée de la *j-ème* sortie à l'itération *n*;

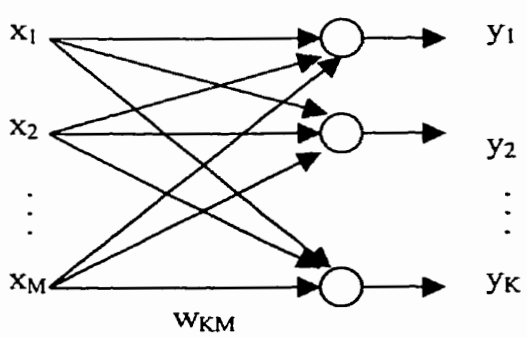



Figure 4.2. Structure d'un Perceptron à une couche.

$\varphi_j'$ est la dérivée de la fonction d'activation du *j-ème* neurone;

$v_j(n)$ est la somme pondérée des entrées du *j-ème* neuone à l'itération *n*;

$\eta$ est le paramètre du taux d'apprentissage $(0 < \eta \leq 1)$.

Le Perceptron à une couche est une collection de séparateurs linéaires. En d'autres mots, il peut être utilisé pour classifier des données linéairement séparables. Pour traiter des problèmes qui ne sont pas linéairement séparables, il faut permettre la composition des couches; ce qui nous amène vers le Perceptron multicouche.

### 4.2.2.2 PERCEPTRON MULTICOUCHE

Un Perceptron multicouche comprend plusieurs couches de neurones : une couche d'entrée, une couche de sortie et une ou plusieurs couches intermédiaires ou cachées (figure 4.3). Les fonctions d'activation des neurones sont toutes dérivables et sont particulièrement des fonctions d'allure *sigmoïde*. Les neurones sont donc tous non linéaires et une combinaison de ces neurones permet de classifier les problèmes non linéaires. Les poids des neurones qui constituent la mémoire long terme du réseau, sont obtenus par un processus d'entraînement supervisé, basé sur l'algorithme de *rétro-*

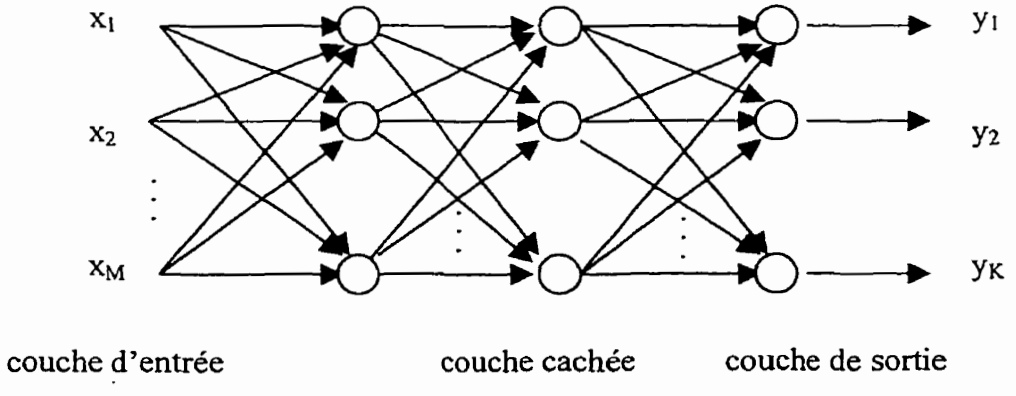



Figure 4.3. Structure d'un Perceptron multicouche

*propagation*[5] d'erreur. Cet algorithme, qui s'apparente à la règle de l'équation (4.6), comprend deux phases principales: une propagation avant et une propagation arrière. Dans la première phase, un patron est appliqué à l'entrée du réseau et son effet est propagé couche par couche, jusqu'à ce que la sortie correspondante soit produite. Pendant cette phase, les poids synaptiques du réseau sont fixes. Dans la deuxième phase, qui est la propagation arrière, les poids sont ajustés dans le but d'approcher la valeurs des sorties actuelles à leur valeur désirée. Pour ce faire, le signal d'erreur, qui est égal à la différence entre les sorties actuelle et désirée, est d'abord calculé pour la couche de sortie; ce qui permet d'ajuster les poids de cette couche. L'erreur est ensuite propagée vers les couches précédentes afin de modifier les poids de la (les) couche(s) cachée(s).

#### *4.2.2.2.1 APPRENTISSAGE SELON L'ALGORITHME DE RÉTRO-PROPAGATION*

Le réseau est entraîné avec un ensemble de $N$ patrons présentés, pendant une *époque* d'entraînement, l'un après l'autre, à l'entrée du réseau. Le processus d'entraînement peut ainsi être composé de plusieurs époques. L'objectif du processus consiste, en général, à minimiser l'erreur quadratique moyenne de la couche de sortie pendant une époque d'entraînement. Cette erreur est calculée de la façon suivante :

$$E_{moy} = \frac{1}{N}\sum_{n=1}^{N} E(n) \tag{4.7}$$

où : $E_{moy}$ est l'erreur quadratique moyenne de l'époque avec $N$ patrons,

$E(n)$ est l'erreur quadratique de la couche de sortie pour le n-ième patron d'entraînement, calculé selon l'équation suivante :

$$E(n) = \frac{1}{2}\sum_{k=1}^{K} e_k^2(n) \tag{4.8}$$

[5] Back-propagation

où : $K$ est le nombre de neurones de la couche de sortie;

$e_k(n)$ est l'erreur de neurone $k$ de la couche de sortie à la présentation de n-ième patron d'entraînement. $e_k(n)$ est déterminée de la façon suivante :

$$e_k(n) = d_k(n) - y_k(n) \tag{4.9}$$

$d_k(n)$ et $y_k(n)$ sont la sortie désirée et la valeur obtenue à la sortie du neurone $k$ de la couche de sortie pour le n-ième patron d'entraînement. Selon l'algorithme de rétro-propagation d'erreur et la technique de la *descente de gradient*, la valeur de correction à appliquer au poids synaptique qui relie le neurone $i$ au neurone $j$, peut être calculée de la façon suivante (Haykin, 1999) :

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{4.10}$$

où : $\eta$ est le paramètre du taux d'apprentissage et $0 < \eta \leq 1$,

$\delta_j(n)$ est le gradient local pour le neurone $j$,

$y_i(n)$ est le signal d'entrée du neurone $j$ en provenance du neurone $i$.

$\delta_j(n)$ dépend de la position du neurone. Si le neurone $j$ est situé dans la couche de sortie, sa valeur est calculée selon l'équation suivante :

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)) \tag{4.11}$$

où $\varphi'_j(n)$ est la dérivé de la fonction d'activation :

$$\varphi'_j(v_j(n)) = \frac{\partial y_j(n)}{\partial v_j(n)} \tag{4.12}$$

$y_j(n)$ et $v_j(n)$ sont la sortie du neurone $j$ pour le n-ième patron, après et avant la fonction d'activation, respectivement.

Pour un neurone $j$ situé dans une couche cachée, $\delta_j(n)$ est déterminé de la façon suivante :

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_{l=1}^{L} \delta_l(n) w_{lj}(n) \tag{4.13}$$

où : $L$ est le nombre de neurones de la couche suivante qui sont connectés au neurone $j$;

$\delta_l(n)$ est le gradient local de neurone $l$ de la couche suivante pour le patron $n$;

$w_{lj}(n)$ est le poids synaptique qui relie le neurone $j$ au neurone $l$.

Pour arrêter le processus d'apprentissage, on peut se baser sur les critères suivants: (1) la valeur de l'erreur d'apprentissage; (2) le nombre d'époques d'entraînement ou (3) la technique de « validation croisée »[6].

Dans le cas du premier critère, la valeur de $E_{moy}$ (équation (4.7)) est calculée après chaque époque d'entraînement et l'entraînement est arrêté si cette valeur descend en dessous d'un seuil représentant l'erreur désirée. Selon le deuxième critère, l'entraînement est arrêté, si l'erreur d'apprentissage n'atteint pas sa valeur désirée, mais le nombre d'époques dépasse un seuil correspondant au nombre maximal d'époques d'entraînement. En pratique, ces deux conditions sont combinées pour arrêter l'entraînement du réseau, lorsque l'une ou l'autre survient.

Comme nous l'avons mentionné précédemment, l'objectif de la phase d'apprentissage est d'ajuster les paramètres libres du réseau pour minimiser l'erreur de généralisation. Cette dernière est calculé en testant le réseau sur un autre ensemble ayant la même distribution statistique que les patrons d'apprentissage. Si le nombre de patrons d'apprentissage n'est pas suffisamment élevé, après un certain nombre d'époques, l'erreur de généralisation se mettra à augmenter, alors que l'erreur d'entraînement continuera à diminuer. Dans ce cas, la complexité du réseau augmente inutilement et le réseau se met à mémoriser les particularités des patrons d'apprentissage; ce que l'on appelle le phénomène de « sur apprentissage ». Si les patrons d'apprentissage sont bruités, le sur apprentissage correspondant à la mémorisation de ces patrons sera la cause d'augmentation d'erreur de généralisation. Pour minimiser la probabilité de sur apprentissage du réseau, on peut utiliser le troisième critère d'arrêt basé sur la technique de la «validation croisée».

---

[6] « cross-validation » en anglais.

Selon cette technique, l'ensemble d'entraînement est divisé en deux sous-ensembles d'estimation et de validation. Le sous-ensemble d'estimation est utilisé pour entraîner le réseau et après un certain nombre d'époques, l'entraînement est arrêté pour estimer l'erreur de généralisation à partir de l'erreur de validation, calculée sur le sous-ensemble de validation. Ce processus d'estimation suivi de validation, continue tant que l'erreur de généralisation est en voie de diminution[7]. En général, si l'entraînement est poursuivi, après avoir atteint sa valeur minimale, l'erreur de généralisation peut se mettre à augmenter (le sur apprentissage du réseau), se plafonner ou diminuer très légèrement. Dans tous les cas, l'arrêt du processus empêche le sur apprentissage ou l'entraînement inutile du réseau[8].

L'algorithme de rétro-propagation d'erreur souffre, en principe, de deux problèmes majeurs : les minimums locaux et le temps d'apprentissage élevé. Nous continuerons donc avec la présentation des techniques proposées pour pallier ces problèmes.

#### *4.2.2.2.2 AMÉLIORATION DE L'ALGORITHME DE RÉTRO-PROPAGATION*

Le premier problème de l'algorithme de rétro-propagation est le risque d'arrêt des itérations causé par les minimums locaux de la surface d'erreur de l'apprentissage. Pour réduire ce risque, on peut ajouter un terme de *moment* à l'algorithme (Haykin, 1999). Le moment joue le rôle d'un filtre passe-bas permettant à l'algorithme de ne pas tenir compte des petits détails et notamment des minimums locaux peu profonds sur la surface d'erreur. En ajoutant un moment à l'équation (4.10), on obtient l'équation suivante pour la correction des poids:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + (1-\alpha)\eta \delta_j(n) y_j(n) \qquad (4.14)$$

où : $\alpha$ est le moment. Pour que l'algorithme converge, la valeur de $\alpha$ doit satisfaire la condition suivante : $0 \leq |\alpha| < 1$. Lorsque $\alpha = 0$, l'adaptation des poids ne sera basée que

[7] Cette méthode d'arrêt des itérations s'appelle « early-stopping » en anglais.

sur le gradient. Si $\alpha = 1$, l'effet du gradient est simplement éliminé et la nouvelle valeur de changement du poids sera égale à son ancienne valeur.

En plus, le moment aide à accélérer la descente de gradient dans les portions continuellement descendante sur la surface d'erreur. Ce phénomène contribue grandement à la diminution du temps d'apprentissage et donc à pallier le deuxième problème de l'algorithme de rétro-propagation. Le moment a également un effet stabilisateur lorsque la direction de gradient sur la surface d'erreur change de signe.

Une autre technique pour accélérer l'apprentissage consiste à utiliser un taux d'apprentissage adaptatif. Ce taux peut être ajusté en fonction du ratio de la nouvelle erreur sur l'ancienne erreur. Si ce ratio dépasse un seuil prédéfini *Err_ratio*, les nouveaux poids, les nouveaux biais, les sorties et l'erreur obtenues seront tous écartés. En plus, le taux d'apprentissage sera diminué. Pour ce faire, on multiplie ce dernier par une constante inférieure à 1:

$$\eta_{n+1} = \eta_n C_{dim} \tag{4.15}$$

où : $\eta_n$ et $\eta_{n+1}$ sont l'ancien et le nouveau taux d'apprentissage, respectivement;

$C_{dim}$ est le coefficient de diminution de taux d'apprentissage.

Si la nouvelle erreur est inférieure à l'ancienne erreur, on augmente le taux d'apprentissage en le multipliant par une constante supérieure à 1.

$$\eta_{n+1} = \eta_n C_{aug} \tag{4.16}$$

où : $C_{aug}$ est le coefficient d'augmentation de taux d'apprentissage.

Nous avons utilisé ces techniques dans notre projet pour améliorer la performance de l'algorithme de rétro-propagation d'erreur. Nous en donnerons plus de détails à la section 4.4.

---

[8] Toutefois, selon (Amari, 1996), l'amélioration produite sur l'erreur de généralisation est négligeable si le nombre de patrons d'entraînement est suffisamment élevé ($N > 30W$, où $N$ est le nombre de patrons d'entraînement et $W$ est le nombre total de paramètres libres du réseau).

Afin de mieux détailler les phases de l'entraînement et du test des classificateurs neuronaux implantés, nous présenterons les textes manuscrits à partir desquels les caractéristiques utilisées dans ces phases ont été extraites. Nous aborderons également une étape essentielle pour la préparation des patrons d'apprentissage et de test, soit la segmentation manuelle en mots. Ces deux sujets seront présentés dans les deux sections suivantes.

## 4.3. TEXTES MANUSCRITS UTILISÉS POUR L'ENTRAÎNEMENT ET LE TEST DES CLASSIFICATEURS

Afin d'évaluer la performance des méthodes conventionnelles et celles que nous avons proposées pour la segmentation en-ligne de textes manuscrits en mots, et dans le but de les comparer pour choisir la ou les meilleures méthodes de segmentation, nous avons créé une base de données sur laquelle nous avons appliqué les différentes méthodes de segmentation. Cette base de données a été utilisée pour l'entraînement et le test des classificateurs conçus pour la segmentation de textes manuscrits. La base de données comprend 13 textes manuscrits acquis par l'interface d'acquisition expliquée dans la section 3.1. Dans ces textes, présentés dans l'annexe A, il existe 3726 distances inter-tracés, donc 3726 patrons.

Pour construire cette base de données, nous avons demandé à huit sujets, soit sept hommes et une femme d'écrire deux textes, un en français et l'autre en anglais, sur l'ardoise électronique Stylistic 1200 de Fujitsu (figure 3.2). Les scripteurs ont été choisi de façon aléatoire, sans effectuer aucun pré-test pour évaluer la qualité de leur écriture. Nous avons choisi des documents dans deux langues différentes pour pouvoir accéder à deux différentes structures de documents, notamment en ce qui concerne la distribution du nombre de caractères dans un mot. Les sujets devaient écrire entre les lignes horizontales. Toutefois, ils avaient la liberté de choisir la distance entre les lignes.

Aucune contrainte spatiale ni temporelle n'a été imposée aux scripteurs, notamment en ce qui concerne le respect des distances intra-mot et inter-mots.

Quant à la qualité des textes dans notre base de données, nous devons mentionner que ce sont des textes manuscrits composés naturellement et librement par les scripteurs, contenant une diversité de cas simples et difficiles pour les différentes méthodes de segmentation abordées dans ce travail. En plus, comme on peut le constater, les textes sont de différents types d'écriture manuscrite: moulée détachée, moulée attachée, cursive pure et cursive naturelle (voir la figure 1.6 pour la définition de ces termes). Nous devons souligner un point important concernant l'évaluation de la qualité des textes de la base de données : le jugement sur cet aspect ne doit pas être fait de façon visuelle et subjective. En fait, un texte peut nous sembler visuellement de bonne qualité, autrement dit facile à segmenter, alors qu'il ne l'est pas pour les méthodes de segmentation (les exemples de ces cas problèmes sont montrés dans le chapitre 2). Ce constat est également vrai pour les algorithmes de reconnaissance de caractères. Un caractère ou un mot, qui est facilement lisible pour l'être humain, peut même être la source d'erreur dans une méthode de reconnaissance. En somme, notre base de données sert uniquement à évaluer la performance des différentes méthodes de segmentation et à les comparer ensemble, plutôt qu'avec la qualité de segmentation des êtres humains.

Lors de la présentation de nos classificateurs, dans ce chapitre et le chapitre suivant, nous référerons à ces textes en spécifiant ceux qui étaient choisis pour l'entraînement et ceux utilisés dans la phase de test.

## 4.4. INTERFACE DE SEGMENTATION MANUELLE DE TEXTES MANUSCRITS

Pendant la phase d'apprentissage des classificateurs neuronaux de distances inter-tracés, les exemples d'apprentissage sont présentés aux classificateurs et les poids des neurones sont ajustés pour minimiser l'erreur de la classification. Selon les équations (4.7) à (4.9), pour calculer cette erreur, nous avons besoin de connaître la valeur de la

sortie désirée, c'est-à-dire le type de la distance inter-tracés. Par conséquent, l'entraînement des classificateurs requiert des textes segmentés de façon manuelle. Afin d'effectuer la segmentation manuelle de textes manuscrits en mots, nous avons implanté des interfaces crayon à l'aide de l'environnement de programmation de Visual C++. Ces interfaces permettent à l'utilisateur de se servir d'un crayon (le même qui est utilisé pour entrer le texte manuscrit) pour désigner les points de segmentation directement sur un texte. La figure 4.4 montre l'aspect visuel de l'interface de la segmentation manuelle de textes manuscrits en mots (pour un texte relativement facile à segmenter manuellement) .

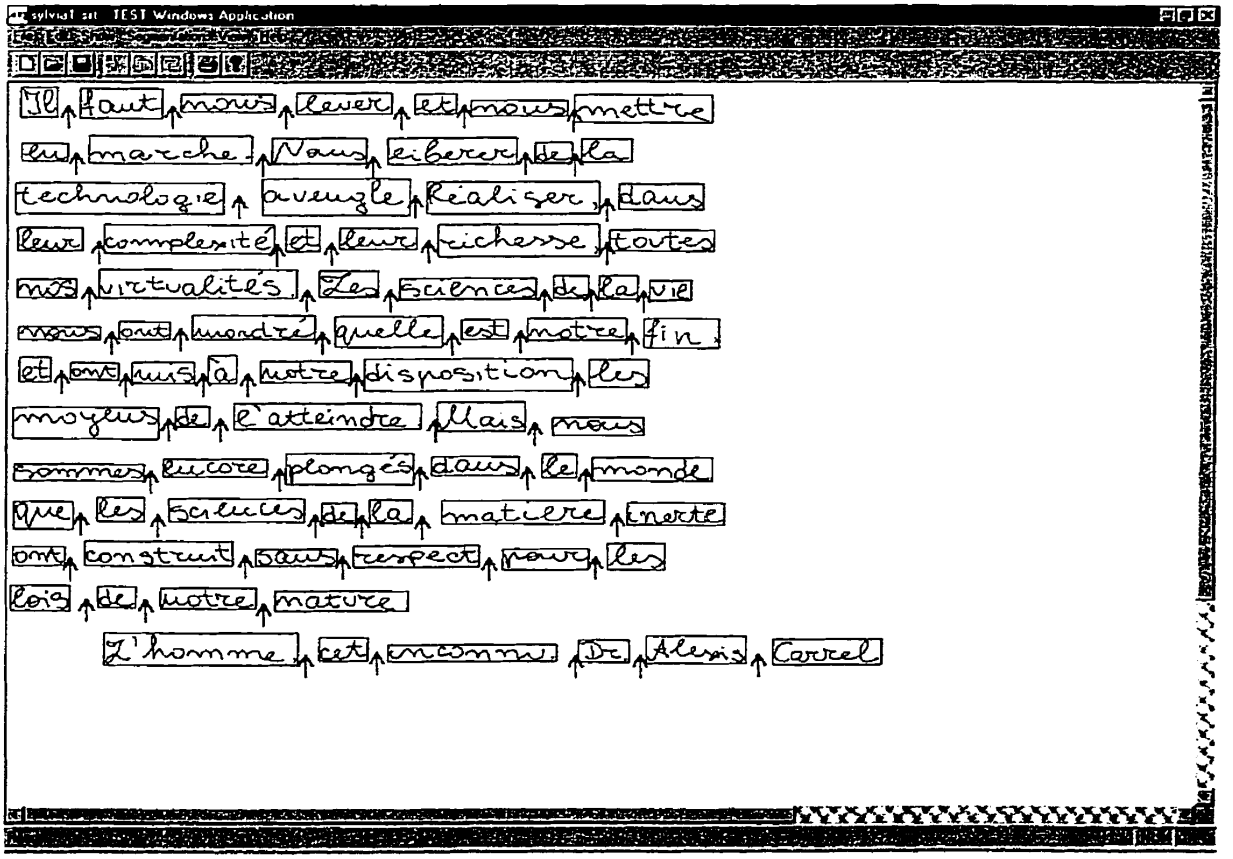



Figure 4.4. Interface de segmentation manuelle de texte manuscrit. Les flèches et les rectangles montrent les points de segmentation désignés par l'utilisateur et le résultat de la segmentation manuelle respectivement.

Dans cet interface, les étapes principales à suivre sont les suivantes :

1- Désigner et éditer (si nécessaire) les points de segmentation par l'utilisateur;
2- Déterminer automatiquement le type de toutes les distances inter-tracés du texte;
3- Construire la liste des mots.

L'utilisateur doit d'abord ouvrir un fichier contenant un texte manuscrit déjà acquis et sauvegardé par le logiciel d'acquisition (voir la section 3.1.2). Il peut débuter la segmentation en choisissant une option de menu. La segmentation manuelle se poursuit en désignant les points de segmentation par l'utilisateur. Les coordonnées de ces points seront capturées par l'interface et ajoutées dans une liste de points de segmentation. La position de chaque point de segmentation est affichée à l'écran à l'aide d'une flèche. L'utilisateur peut éditer la liste des points, en effaçant le dernier point ou tous les points de segmentation de la liste à n'importe quel moment durant la segmentation manuelle. Il a également la possibilité de faire apparaître ou de faire disparaître les flèches à l'écran. La liste des points de segmentation est finalisée lorsque l'utilisateur choisit l'option de la fin de segmentation.

Le traitement se poursuit en parcourant la liste des points de segmentation afin de déterminer le type de distance inter-tracés pour tous les tracés du texte. Cette distance est d'abord initialisée à intra-mot. Si pour un tracé donné du texte, un point de segmentation désigne sa distance inter-tracés, le type de distance sera alors modifié à inter-mots. Chaque point de segmentation correspond donc à une distance (espacement) inter-mots et la tâche de cette phase de traitement consiste à trouver le tracé dont l'espacement avec le tracé suivant est pointé. En plus des points de segmentation choisis par l'utilisateur, le programme ajoute automatiquement un point de segmentation après le dernier tracé de chaque ligne du texte. La valeur de la distance inter-tracés n'est pas mesurable pour ces tracés et les caractéristiques extraites ne sont pas utilisables dans les phases

d'apprentissage et de test des classificateurs. Toutefois, l'ajout des points de segmentation, après ces tracés, est nécessaire pour identifier tous les mots du texte.

La dernière étape consiste à parcourir la liste des tracés afin d'en construire la liste de mots en utilisant le type de distances inter-tracés déterminées à l'étape précédente. Pour le premier tracé du texte, le programme crée le premier mot et y rajoute le tracé. Les tracés suivants seront également insérés dans la liste des tracés du mot si le type de leur distance inter-tracés est intra-mot. Le premier tracé dont la distance inter-tracés est inter-mots sera ainsi le dernier tracé du nouveau mot et le tracé qui le suit sera le premier tracé du deuxième mot du texte. Cette procédure se poursuit jusqu'au dernier tracé du texte, permettant ainsi la construction de la liste des mots du texte dans laquelle chaque mot est composé, à son tour, d'une liste de tracés.

Afin de vérifier l'exactitude des résultats de la segmentation, les rectangles englobant les mots seront ensuite affichés à l'écran (figure 4.4). L'interface permet également à l'utilisateur de faire apparaître et disparaître ces rectangles.

Finalement, pour l'utilisation ultérieure des résultats de la segmentation manuelle, le document segmenté en mots est sauvegardé dans un fichier. Les modules de l'extraction des caractéristiques (décrits dans le chapitre 3) se servent de ces résultats pour construire les bases d'apprentissage et de test des classificateurs implantés.

## 4.5. DESCRIPTION DES CLASSIFICATEURS NEURONAUX

Comme nous l'avons mentionné dans le chapitre 3, deux grandes catégories de caractéristiques ont été extraites dans notre projet pour la segmentation de textes manuscrits en mots: les caractéristiques conventionnelles et les caractéristiques proposées. Dans cette section, nous présenterons les différents réseaux qui utilisent ces caractéristiques comme les entrées pour classifier les distances inter-tracés . Il s'agit alors de la structure des réseaux, des phases d'apprentissage et de test, des résultats obtenus de la classification, de la comparaison entre la performance des deux catégories

de caractéristiques utilisées et des discussions sur les différents aspects des résultats obtenus.

## 4.5.1. CLASSIFICATEURS NEURONAUX AVEC LES CARACTÉRISTIQUES CONVENTIONNELLES À L'ENTRÉE

Dans cette section, nous nous pencherons sur les réseaux neuronaux implantés pour classifier les distances inter-tracés et ce, en utilisant les caractéristiques conventionnelles. Au début, nous présenterons les entrées des réseaux. Ensuite, nous expliquerons la structure des réseaux et leurs phases d'apprentissage et de test et présenterons les résultats de la classification. Nous terminerons cette section avec une analyse et une discussion sur les résultats obtenus.

### 4.5.1.1. ENTRÉES DES CLASSIFICATEURS

Au chapitre 3, nous avons désigné, parmi les mesures de distances inter-tracés présentées dans le revue des méthodes du chapitre 2, celles que nous avons trouvées comme étant les mesures les plus robustes pour la segmentation en-ligne d'une ligne de texte manuscrit en mots. Il s'agit des mesures de distance entre les rectangles des tracés ou « Dist_BB » et les distances horizontales moyenne « RLavg »et minimale « RLmin ». Nous avons implanté des réseaux neuronaux qui reçoivent principalement ces caractéristiques à l'entrée pour classifier les distances inter-tracés. En ce qui concerne Dist_BB, toutes les valeurs existantes dans les fichiers de caractéristiques ont été utilisées pour l'entraînement ou le test des classificateurs. Toutefois, ce n'est pas le cas pour RLavg et RLmin. Comme nous l'avons mentionné dans la section 3.2.2.1, si un tracé et son mot correspondant ne se chevauchaient pas verticalement, la distance horizontale entre eux ne serait pas calculable. Par conséquent, pour cette paire de tracé et mot, contrairement à Dist_BB, RLmin et RLavg ne peuvent être utilisées comme les

entrées des classificateurs neuronaux. Le résultat est donc la diminution du nombre de patrons utilisables pour l'entraînement ainsi que le taux de succès des classificateurs dans la phase de test.

En plus de ces caractéristiques, nous avons extrait deux autres caractéristiques servant de facteurs de normalisation des mesures de distances. Il s'agit de « avgHeight » et « avgDistBB » (voir la section 3.2.3). AvgHeight est la hauteur moyenne de $N$ tracés précédents et avgDistBB est la distance moyenne entre les rectangles des tracés consécutifs pour $N$ dernières paires de tracés. Nous avons effectué des tests avec différentes valeurs de $N$ = 3, 5, 7 et 9. Étant donnée que les meilleurs résultats de la classification ont été obtenus avec $N$ = 9, les résultats présentés dans ce chapitre et le chapitre suivant correspondent à cette valeur de $N$[9].

### 4.5.1.2. STRUCTURE DES CLASSIFICATEURS

Les classificateurs neuronaux utilisés sont des réseaux de type Perceptron à couches multiples. En somme, neuf classificateurs ont été implantés avec structures identiques mais différentes entrées. Les classificateurs ont tous une entrée, une couche d'entrée, une couche cachée avec deux neurones et une couche de sortie avec un neurone. La fonction d'activation est la fonction *logsig* dont la valeur varie entre (0, +1). Les entrées sont les trois mesures de distances Dist_BB, RLmin et RLavg ainsi que leurs valeurs normalisées sur avgHeight et avgDistBB. La sortie du neurone de la dernière couche des réseaux détermine le type des distances inter-tracés. Étant donnée que la valeur de cette sortie peut varier entre 0 et +1. Un seuil égal à 0.5 a été utilisé pour déterminer le type des distances inter-tracés : les sorties supérieures et inférieures à ce seuil sont alors considérées « inter-mot » et « intra-mot » respectivement.

[9] À cause du compromis entre le nombre de tracés précédents impliqués dans le calcul des valeurs moyennes et la sensibilité des calculs à des changements rapides survenus dans la grandeur des tracés et les distances inter-tracés, nous avons décidé de ne pas utiliser des $N$ plus grand que 9.

### 4.5.1.3. PHASE D'APPRENTISSAGE

La technique de la « validation croisée », présentée à la section 4.2.2.2.1, fut utilisée pour entraîner les classificateurs. Cette technique permet de minimiser la probabilité de sur apprentissage des classificateurs. Pour ce faire, nous avons d'abord divisé notre base de données en deux ensembles d'apprentissage et de test. L'ensemble d'apprentissage est formé des patrons du septième texte manuscrit (figure A.7 de l'annexe A). L'ensemble de test contient les patrons extraits des autres textes manuscrits[10].

Nous avons ensuite divisé notre ensemble d'apprentissage en deux sous-ensembles, 80% des patrons dans le sous-ensemble d'estimation et 20% dans le sous-ensemble de validation. Les exemples d'estimation servaient à entraîner les classificateurs que l'on testait à l'aide des exemples de validation. Pour s'assurer que les classificateurs n'avaient pas mémorisé les exemples d'estimation, on arrêtait, à chaque 500 itérations, l'entraînement des classificateurs pour les tester sur les exemples de validation. L'entraînement se poursuivait tant que l'erreur de validation diminuait. Dans nos expérimentations, le nombre maximum d'itérations était de 4000.

Pour l'apprentissage des réseaux, nous avons utilisé la version améliorée de l'algorithme de rétro-propagation d'erreur, présentée à la section 4.2.2.2.2. Le moment $\alpha$ dans l'équation (4.14) a pris 0.95 comme valeur. Pour l'apprentissage du réseau avec le taux adaptatif, le ratio de la nouvelle erreur sur l'ancienne erreur *Err_ratio*, le taux d'apprentissage initial $\eta_0$ et les coefficients d'augmentation $C_{aug}$ et de diminution $C_{dim}$ de taux d'apprentissage ont été choisis égals à 1.04, 0.01, 1.05 et 0.7, respectivement (équations (4.15) et (4.16))[11]. Afin d'accélérer la procédure d'apprentissage en empêchant des paramètres libres du réseau de tendre vers l'infini, les valeurs des sorties désirées ont été choisies égales à 0.1 et 0.9 plutôt que des valeurs limites de la fonction d'activation, c'est-à-dire 0 et +1 (Haykin, 1999).

---

[10] Les nombres de patrons dans les ensembles d'apprentissage et de test sont présentés dans le tableau 4.1.
[11] Les valeurs de paramètres sont celles que le « Neural Network Toolbox » de Matlab a suggérées. Nous devons ajouter, qu'avec ces valeurs, les réseaux ont atteint leur erreur désirée. En plus, les mêmes valeurs ont été utilisées pour tester différents réseaux avec différentes entrées.

### 4.5.1.4. PHASE DE TEST

Après avoir entraîné les réseaux avec les patrons d'apprentissage, d'autres patrons ont été présentés aux réseaux et les résultats de la classification des distances inter-tracés ont été comparés avec les sorties désirées. L'objectif de cette phase, appelée la phase de test, était d'évaluer la capacité de généralisation des réseaux, c'est-à-dire classifier correctement les patrons autres que ceux utilisés dans la phase d'entraînement. Puisque le septième texte de notre base de données servait à entraîner les classificateurs, nous avons effectué la phase de test sur les patrons de douze autres textes de la base de données (annexe A).

Le pourcentage de la classification correcte des distances inter-tracés a été choisi comme la mesure de la performance des classificateurs. Pour ce faire, nous avons présenté à l'entrée des classificateurs, les caractéristiques des patrons de test et nous avons calculé le taux de classification correcte pour chacun des textes. Pour avoir une idée globale de la performance d'une méthode de classification, nous avons calculé la moyenne des taux sur tous les textes ainsi que l'intervalle de confiance de cette mesure. Les résultats numériques de cette phase seront présentés dans la section suivante.

### 4.5.1.5. RÉSULTATS DE CLASSIFICATION

Pour obtenir des résultats de classification, nous avons utilisé des caractéristiques extraites des textes manuscrits numéros 1 à 6 et 8 à 13 (annexe A). Ces textes contiennent, en somme, 3276 distances inter-tracés. Quant aux caractéristiques Dist_BB, avgHeight et avgDistBB tous les patrons sont utilisables, tandis que le nombre de patrons valides pour les caractéristiques RLavg et RLmin est, en général inférieur au nombre total de patrons. Le tableau 4.1 montre, pour chacun des textes utilisés dans les phases d'entraînement et de test des classificateurs neuronaux, le nombre total de patrons et le nombre de patrons dont les distances horizontales sont invalides.

Tableau 4.1. Nombre total des patrons et le nombre de patrons invalides pour les distances horizontales dans les textes manuscrits.

| Numéro de Texte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nombre total de patrons | 169 | 149 | 438 | 440 | 158 | 509 | 452 | 198 | 194 | 350 | 221 | 282 | 166 | 3726 |
| Nombre de patrons invalides | 9 | 7 | 27 | 26 | 7 | 30 | 14 | 10 | 10 | 20 | 15 | 16 | 5 | 196 |

Si un réseau neuronal n'utilisait que la (les) distance(s) horizontale(s) pour classifier les patrons, son taux de classification serait affecté par les patrons invalides.

Nous avons donc tenu compte de l'effet négatif de ces patrons dans le calcul des taux de classification obtenus avec les caractéristiques RLmin et RLavg. Pour ce faire, le nombre de patrons classifiés correctement a été divisé sur le nombre total de patrons et non pas sur le nombre de patrons valides, puisque les patrons sans distances horizontales valides ne pouvaient être classifiés et étaient considérés comme des cas d'erreur de classification. Les pourcentages de ces patrons, par rapport au nombre total des patrons, sont illustrés dans le tableau 4.2. Ce tableau montre que les patrons invalides en distances horizontales forment entre 3% et 6,8% et en moyenne 5,15% du nombre total des patrons.

Les résultats obtenus de la phase de test sont présentés dans le tableau 4.3. Ce tableau montre les pourcentages de la classification correcte des distances inter-tracés pour toutes les caractéristiques présentées à l'entrée des réseaux neuronaux et pour tous les textes utilisés dans la phase de test. Les cases de la dernière colonne du tableau contiennent les valeurs moyennes des taux de classification de tous les fichiers pour une caractéristique donnée. Pour déterminer la degré de signification et de confiance de ces

Tableau 4.2. Pourcentage des patrons invalides pour les distances horizontales sur le nombre total des patrons.

| Numéro de Texte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | Moy. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| % de patrons invalides | 5.3 | 4.7 | 6.2 | 5.9 | 4.4 | 5.9 | 3.1 | 5.1 | 5.2 | 5.7 | 6.8 | 5.7 | 3.0 | 5.15 |

Tableau 4.3. Pourcentage de la classification correcte des distances inter-tracés pour les caractéristiques conventionnelles.

**Numéro de Texte**

| Entrée | *1* | *2* | *3* | *4* | *5* | *6* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Dist_BB*** | 97 | 99.3 | 99.8 | 99.8 | 98.7 | 99.6 | 97 | 97.4 | 97.7 | 92.8 | 96.1 | 97 | **97.68** |
| ***Dist_BB/avgHeight*** | 92.9 | 98.7 | 98.9 | 98.9 | 100 | 99 | 96 | 96.4 | 94.9 | 81.4 | 94.7 | 96.4 | **95.68** |
| ***RLmin*** | 81.1 | 90.6 | 90.2 | 89.5 | 88 | 90.2 | 83.8 | 85.1 | 84.6 | 77.4 | 84.8 | 87.3 | **86.05** |
| ***Dist_BB/avgDistBB*** | 78.1 | 67.8 | 96.3 | 97 | 71.5 | 94.9 | 86.4 | 81.4 | 92.6 | 84.6 | 92.6 | 70.5 | **84.48** |
| ***RLmin/avgHeight*** | 76.3 | 89.3 | 87.4 | 86.8 | 89.2 | 88.6 | 80.8 | 86.1 | 79.7 | 75.6 | 80.5 | 88.6 | **84.08** |
| ***RLavg*** | 85.2 | 88.6 | 86.1 | 88 | 86.1 | 85.5 | 79.3 | 83 | 73.4 | 63.8 | 73.4 | 86.1 | **81.54** |
| ***RLavg/avgHeight*** | 85.8 | 85.9 | 83.6 | 84.5 | 88 | 84.7 | 78.8 | 83.5 | 68.3 | 72.4 | 66.7 | 86.7 | **80.74** |
| ***RLmin/avgDistBB*** | 65.1 | 58.4 | 85.8 | 85.9 | 59.5 | 80 | 71.7 | 70.6 | 80.3 | 71 | 80.5 | 59.6 | **72.37** |
| ***RLavg/avgDistBB*** | 64.5 | 51 | 81.1 | 82.3 | 52.5 | 77.2 | 66.2 | 67 | 76.6 | 66.1 | 73 | 56 | **67.79** |

valeurs moyennes, nous avons effectué une analyse statistique sur ces résultats, laquelle sera présentée dans la section suivante.

### *4.5.1.5.1. STABILITÉ STATISTIQUE DES RÉSULTATS*

Afin d'évaluer le degré de stabilité des résultats obtenus avec différents échantillonnages (textes) de base de données, nous avons calculé l'intervalle de confiance des valeurs moyennes des taux de classification de nos classificateurs. Cette intervalle est un indicateur statistique de la stabilité des résultats et donc de la robustesse des classificateurs relativement aux variations des échantillons dans la base de données. Plus cet intervalle est petit, plus le classificateur est robuste.

L'expression suivante donne l'intervalle de confiance de $100(1-\alpha)\%$ pour une moyenne calculée sur différents échantillonnages:

$$(\bar{y} - t_{\alpha/2,n-1}\frac{s}{\sqrt{n}}, \bar{y} + t_{\alpha/2,n-1}\frac{s}{\sqrt{n}}) \qquad (4.17)$$

où : $\bar{y}$ et $s$ sont les estimés pour la valeur moyenne et l'écart type respectivement, $n$ est le nombre d'échantillons (textes). et $t_{\alpha/2,n-1}$ est le quantile $100\,\alpha/2$ de la distribution du $t$

de Student à n-1 degrés de liberté. Les valeurs de $\bar{y}$ et $s^2$ sont calculées à partir des équations suivantes :

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{4.18}$$

$$s^2 = \frac{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}{n(n-1)} \tag{4.19}$$

Le tableau 4.3 montre, pour tous les échantillonnages (textes) de notre base de test, les pourcentages de classification correcte obtenus avec différentes méthodes ainsi que les moyennes de ces valeurs $\bar{y}$. $n$ est donc le nombre de textes sur lesquels une méthode est testée. Dans le cas du tableau 4.3, $n$ est égal à 12[12]. La valeur choisie pour $\alpha$ est de 0.1 afin d'obtenir un degré de confiance de 90%.

Le tableau 4.4 montre les résultats d'application de cette analyse statistique sur les pourcentages de classifications du tableau 4.3. Dans le tableau 4.4, les colonnes $s$ et $\bar{y}_{med}$ représentent respectivement les estimés pour la valeur moyenne et l'écart type et la

Tableau 4.4. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 4.3.

| | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| ***Dist_BB*** | **2.00** | **97.68** | **(96.65, 98.72)** |
| ***Dist_BB/avgHeight*** | **4.99** | **95.68** | **(93.10, 98.27)** |
| ***RLmin*** | **4.06** | **86.05** | **(83.94, 88.16)** |
| ***Dist_BB/avgDistBB*** | **10.59** | **84.48** | **(78.99, 89.96)** |
| ***RLmin/avgHeight*** | **5.15** | **84.08** | **(81.40, 86.75)** |
| ***RLavg*** | **7.61** | **81.54** | **(77.60, 85.49)** |
| ***RLavg/avgHeight*** | **7.45** | **80.74** | **(76.88, 84.61)** |
| ***RLmin/avgDistBB*** | **10.15** | **72.37** | **(67.11, 77.63)** |
| ***RLavg/avgDistBB*** | **10.67** | **67.79** | **(62.26, 73.32)** |

[12] Dans les différentes expérimentations présentées dans la thèse, la valeur de $n$ dépend du scénario d'apprentissage appliqué. Pour les deux scénarios commun et personnalisé (voir la section 4.4.2.3.1) $n$ vaut 12 et 13, respectivement.

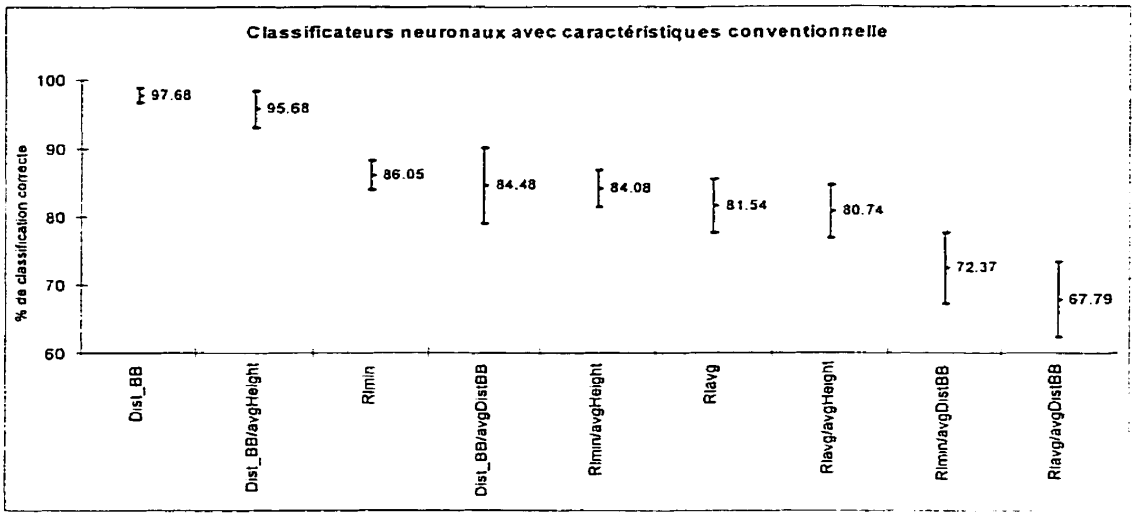



Figure 4.5. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs neuronaux avec les caractéristiques conventionnelles.

dernière colonne contient les intervalles de confiance des moyennes. Les résultats présentés dans le tableau 4.4 sont montrés dans la figure 4.5. Dans la section 4.5.1.7, nous présenterons une analyse et une discussion sur ces résultats.

## 4.5.1.6 ANALYSE DES ERREURS DE SEGMENTATION

Dans cette section, nous présenterons les résultats d'une étude quantitative effectuée sur les différents types d'erreurs de segmentation afin d'identifier les sources d'erreurs et leurs poids dans la procédure de segmentation.

Pour ce faire, nous avons d'abord identifié les patrons mal classifiés dans la base de test, pour le classificateur ayant produit le meilleur taux de classification avec les caractéristiques conventionnelles. Il s'agit d'un classificateur neuronal avec la distance entre rectangles à l'entrée (le classificateur avec Dist_BB à l'entrée dans le tableau 4.4).

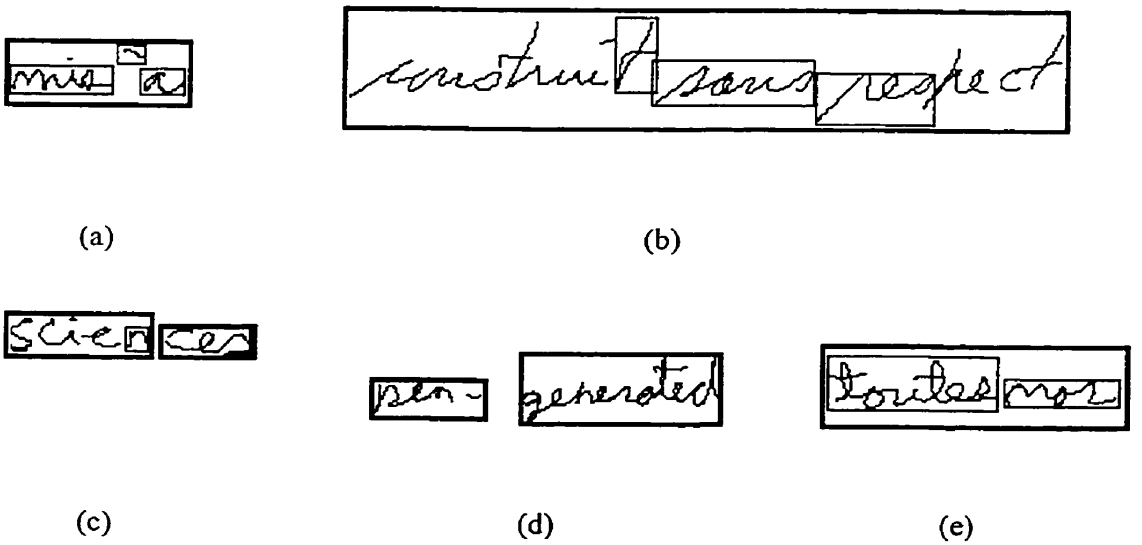
Nous avons ensuite calculé les pourcentages de trois types d'erreurs de segmentation présentés dans le tableau 4.5. La figure 4.6 montre les exemples de ces types d'erreurs dans notre base de données.

Tableau 4.5. Pourcentages de trois types d'erreurs de segmentation.

| *Type d'erreur* | *Pourcentage d'erreur* |
|---|---|
| **Signes diacritiques** | **50** |
| **Inclinaison** | **16.7** |
| **Composition-Classification** | **33.3** |

Le premier type d'erreurs correspond aux erreurs causées par des signes diacritiques (Accent_Point_TBars). Autrement dit, une erreur de ce type devrait, en principe, être corrigée si le signe diacritique était correctement reconnu. Un exemple de ces erreurs est présenté dans la figure 4.6(a). Dans cet exemple, l'accent de 'à' réduit la distance entre 'mis' et 'à'. En conséquence le classificateur reconnaît un espacement intra-mot et les regroupe dans un mot.

Les erreurs de deuxième type sont causées par l'inclinaison de l'écriture. Par exemple, dans la figure 4.6(b), les rectangles englobant des tracés et des mots voisins se chevauchent et tous les tracés impliqués se trouvent dans un mot.



(a) (b)

(c) (d) (e)

Figure 4.6. Exemples de trois types d'erreurs de segmentation de la meilleure méthode conventionnelle (Dist_BB du tableau 4.4) dans la base de test: (a) erreur de signes diacritiques; (b) erreur d'inclinaison (c), (d) et (e) erreurs de composition-classification.

Toutes les erreurs dont la source n'est ni la présence de signes diacritiques ni l'inclinaison de l'écriture sont classées dans la troisième catégorie. Ces erreurs peuvent être causées par, soit l'imprécision dans la composition, soit l'erreur de classification du classificateur de distances inter-tracés. Toutefois, le calcul précis de pourcentages de ces deux sources d'erreurs est impossible. Les figures 4.6(c) à 4.6(e) montrent trois exemples de cette catégorie d'erreurs. Dans la figure 4.6(c), l'erreur peut plutôt être attribuée à l'imprécision du classificateur, alors que celles des figures 4.6(d) et 4.6(e) sont attribuables à l'imprécision dans l'écriture du scripteur lors de la composition du texte.

Dans ce chapitre et le chapitre 5, nous expliquerons nos démarches pour la conception et réalisation des mécanismes efficaces de correction de ces types d'erreurs et dans le chapitre 5 (section 5.3.2.2), nous présenterons des statistiques sur les pourcentages des erreurs corrigées, de chaque catégorie grâce à nos propositions.

## 4.5.1.7 ANALYSE ET DISCUSSION

Dans cette section, à la lumière des résultats présentés aux sections précédentes, nous ferons une analyse portant principalement sur les points suivants :

1- La performance des différentes caractéristiques utilisées pour représenter les distances inter-tracés sans normalisation.

2- L'effet de la normalisation des caractéristiques sur les résultats.

Notre objectif consiste à utiliser les résultats numériques et les distributions des différentes mesures de distances pour souligner les points forts et faibles de ces mesures.

### *4.5.1.7.1 ANALYSE DE PERFORMANCE DES MESURES DE DISTANCES SANS NORMALISATION*

Les tableaux 4.3 et 4.4 et la figure 4.5 montrent que parmi les trois mesures conventionnelles de distances inter-tracés, Dist_BB, RLavg et RLmin, la première produit les meilleurs résultats de classification. Autrement dit, Dist_BB représente

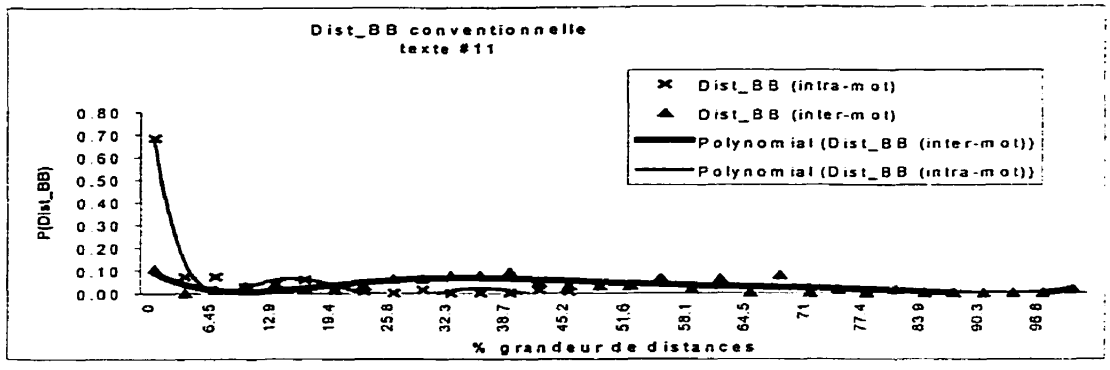
mieux les distances inter-tracés que les deux autres mesures et cela non seulement pour l'ensemble des textes manuscrits en moyenne, mais aussi pour chacun d'eux. Pour appuyer cette conclusion et aussi pour souligner les points forts et faibles de ces mesures de distances, nous utilisons les distributions des distances intra et inter-mots.

La figure 4.7 montre les distributions des distances intra et inter-mots des trois mesures conventionnelles de distances inter-tracés pour le onzième texte de notre base de données[13]. Nous avons également ajouté, à chaque distribution, une courbe de tendance. Les valeurs sur l'axe des abscisses sont normalisées sur la grandeur maximale des distances inter-tracés. Il faut ajouter que les distances horizontales invalides n'ont pas été introduites dans ces distributions, puisque la valeur correcte de ces distances est indéterminable.

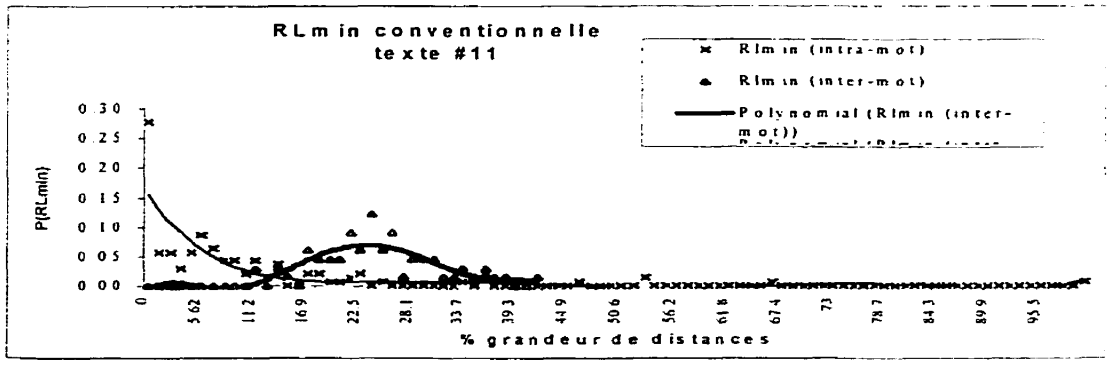
En principe, une mesure de distances inter-tracés doit, autant que possible, permettre de différencier les distances intra-mot des distances inter-mots et donc de minimiser le chevauchement entre leurs distributions. Comme on peut l'observer dans les distributions de la figure 4.7, les spectres intra-mot et inter-mots des trois mesures de distances se chevauchent. Toutefois, le pourcentage du chevauchement est moins élevé pour Dist_BB que pour les RLmin et RLavg; ce qui est la raison principale du taux de classification plus élevé de Dist_BB par rapport à deux autres mesures de distances.

Nous analysons d'abord la distribution des distances intra-mot. 85% des Dist_BB intra-mot ont une valeur inférieure à 10% et la valeur maximale des distances intra-mot pour l'histogramme des Dist_BB est 45%. Cela montre que les distances intra-mot sont bien présentées par la mesure de Dist_BB. Par contre, les surfaces sous les histogrammes intra-mots pour les valeurs inférieures à 10% de la distance maximale, sont de 72% et de 45%, respectivement pour RLmin et Rlavg, et de 4% des RLmin et RLavg intra-mot sont au-dessus de la barre de 45%. En plus, contre toutes attentes, la valeur maximale des distances inter-tracés (100%) pour ces deux dernières mesures,

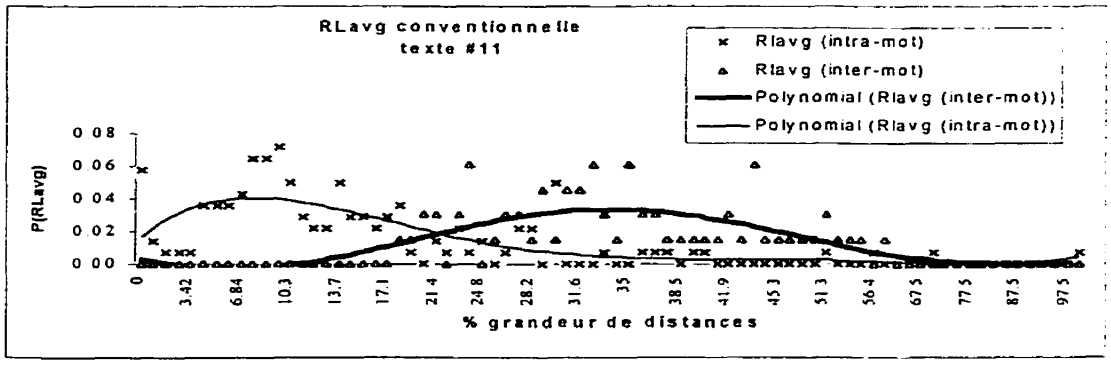
13 Nous avons choisi ce texte à titre d'exemple parmi les textes disponibles. Cela peut être considéré comme un choix aléatoire des patrons disponibles, puisque les autres textes ont approximativement les mêmes formes de distributions.




(a)




(b)




(c)

Figure 4.7. Distribution des trois mesures conventionnelles de distances inter-tracés et leurs courbes de tendance pour le texte #11.

appartient à des entités intra-mot et non pas inter-mots. Ces valeurs irréalistes de distances intra-mot sont principalement causées par les accents et les points situés au-dessus ou en dessous des parties principales des mots. RLavg et RLmin sont touchées pareillement par ce problème, puisque les accents et les points sont, en général, des tracés de petite taille produisant des distances horizontales moyenne et minimale semblables.

Contrairement à Dist_BB et RLmin, RLavg souffre du problème de la surestimation des distances intra-mot. Le sommet de la distribution intra-mot, pour Dist_BB et RLmin, est situé à zéro, alors qu'il est à 10% pour RLavg. En plus, 24% des RLavg intra-mot dépassent 20% de la valeur maximale des distances, tandis que ce pourcentage est de 11% et de 4% pour RLmin et Dist_BB, respectivement. Cette surestimation des distances est causée par la nature de l'opération de calcul de la moyenne qui est un filtrage passe-bas neutralisant l'effet des petites distances horizontales par des grandes distances. En somme, toutes ces observations montrent que les distances intra-mot sont mieux représentées par Dist_BB, suivie de RLmin et, de loin, de RLavg.

Malgré la capacité de Dist_BB pour représenter les distances intra-mot, celle-ci souffre du problème de la sous-estimation des distances inter-mots. On peut observer cette lacune dans la distribution des distances inter-mots de la figure 4.7(a). Dans cette distribution, 11% des distances ont une longueur de zéro et 66% d'entre elles ont moins de 50% de la valeur maximale des distances inter-mots. Ces phénomènes sont causés, en général, par les tracés inclinés et les barres de « t ». En plus, la distribution est presque uniforme, distribuée tout le long de l'axe des abscisses (de 0 jusqu'à 100%). RLmin et RLavg ont des distributions inter-mots moins uniformes commençant avec des valeurs plus élevées que celles de Dist_BB (11% et 19% respectivement). La meilleure forme de distribution inter-mots semble appartenir à RLavg, et ce, grâce à la surestimation des distances inter-tracés. Toutefois, comme les résultats numériques de la classification le montrent et nous l'avons également mentionné, cette surestimation est, en général, plutôt problématique pour la mesure de RLavg. En plus, la distribution inter-mots pour RLavg

est plus étendue que celle de RLmin; ce qui augmente le chevauchement avec la distribution intra-mot et donc le taux d'erreur de classification. La distribution RLmin inter-mots n'occupe que 30% de l'étendue de l'axe des abscisses, tandis que cette valeur est de 40% pour RLavg.

En somme, à la lumière de la discussion précédente et des résultats présentés dans les tableaux 4.3 et 4.4 et la figure 4.5, on peut conclure que les distances intra-mot sont bien présentées par Dist_BB, tandis que cette dernière souffre du problème de la sous-estimation des distances inter-mots causée principalement par l'inclinaison des tracés et les barres de « t ». Le point faible de RLmin et RLavg est les valeurs excessives de distances intra-mot attribuables aux accents et aux points. RLavg souffre, en plus, du problème de surestimation des distances intra-mot. Nous avons présenté dans le chapitre 3, nos propositions pour résoudre ces problèmes. Ces dernières seront appuyées par les résultats empiriques, présentés à la section 4.5.

#### *4.5.1.7.2 EFFET DE LA NORMALISATION DES MESURES DE DISTANCE*

Dans le but d'analyser l'effet de grandeur des tracés et les variations de distances inter-tracés précédentes dans la décision du classificateur de distances, nous avons utilisé deux caractéristiques extraites des textes manuscrits pour normaliser les mesures de distances inter-tracés. Il s'agit de avgHeight et de avgDistBB qui représentent respectivement la hauteur moyenne de N derniers tracés et la moyenne de N dernières distances inter-tracés. Les mesures de distances conventionnelles sont normalisées sur ces caractéristiques. Les résultats des tableaux 4.3 et 4.4 et de la figure 4.5 (obtenus avec N = 9) montrent que non seulement ces normalisations n'ont pas amélioré la performance des classificateurs, mais ont plutôt réduit le taux de la classification correcte. Dans cette section, nous essayons d'expliquer les causes de ce phénomène.

**Effet de la normalisation sur avgHeight**

Nous commençons par l'effet de avgHeight sur le taux de la classification. En général, on s'attend à observer une relation directe entre la grandeur des distances inter-

tracés et la grandeur de l'écriture. Autrement dit, dans une procédure de classification des distances inter-tracés, la normalisation des distances sur la grandeur de l'écriture devrait compenser l'effet de changement de la grandeur de l'écriture dans la décision du classificateur. Par conséquent, dans le cas d'une écriture de grandeur constante, les résultats de la classification ne doivent pas être influencés par la normalisation des distances sur la grandeur de l'écrit. Par contre, si le classificateur est entraîné par un texte dont la grandeur est différente de celle des textes utilisés dans la phase de test, on doit s'attendre à une amélioration dans les résultats.

En somme, si la grandeur de l'écriture est bien estimée, la normalisation doit avoir un effet positif sur le taux de classification de l'ensemble des patrons. Or, la diminution des taux de classification dans les tableaux 4.3 et 4.4 et la figure 4.5, causée par la normalisation des distances inter-tracés sur avgHeight, montre que la grandeur de l'écriture n'est probablement pas représentée adéquatement par ce facteur de normalisation.

Pour appuyer cette conclusion, nous utiliserons la courbe de la figure 4.8 qui montre les variations de avgHeight pour les patrons du onzième texte de notre base de données (figure A.11). Dans ce texte, l'écriture semble avoir une grandeur uniforme. Nous devrions donc nous attendre à une avgHeight constante ou presque constante, notamment en raison du phénomène de filtrage passe-bas des hauteurs des tracés précédents. Toutefois, la courbe de la figure 4.8 montre que les variations de cette caractéristique sont importantes (entre 12 et 43). Les valeurs irrégulières de avgHeight au début du texte (avant le $9^{ième}$ patron) sont naturellement attribuables au manque de tracés pour le calcul de la moyenne. Cependant, même à la présence d'un nombre suffisant de tracés pour effectuer cette opération, on constate de grands changements dans la valeur de avgHeight (entre 12 et 25). Ce phénomène peut être attribué en partie au nombre pas assez élevé de tracés utilisés pour calculer avgHeight. Toutefois, nous devons souligner le compromis entre la longueur du filtre (le nombre de tracés) et sa sensibilité aux changements rapides de la grandeur de l'écriture. Pour augmenter la vitesse d'adaptation du filtre à un changement produit dans l'écriture, il faut diminuer

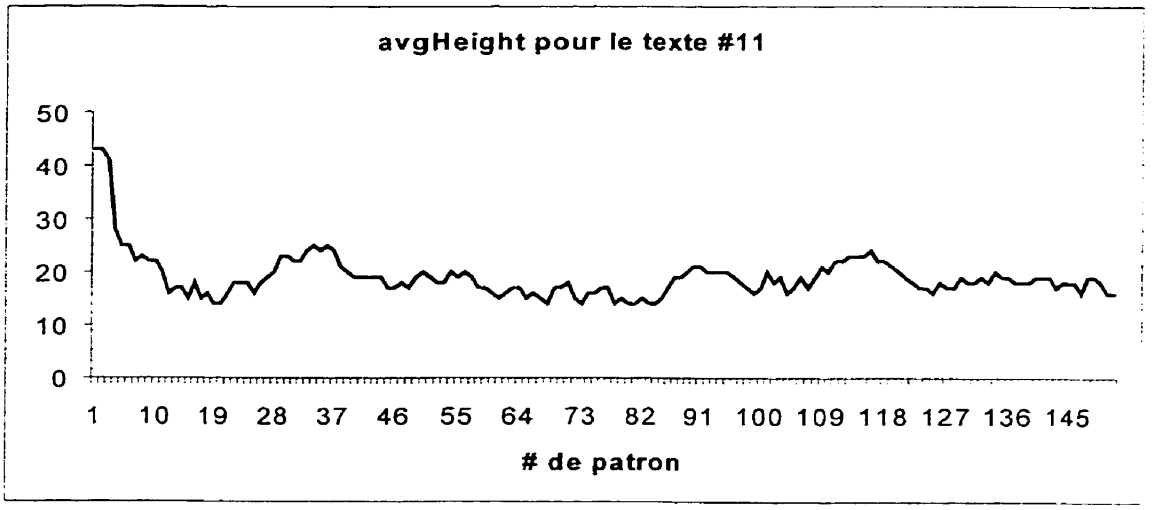



Figure 4.8. Variations d'avgHeight pour le texte #11 (N = 9).

le nombre de tracés impliqués dans le calcul de avgHeight. La raison principale de ces variations inattendues repose non pas sur l'opération de calcul de la moyenne elle-même, mais plutôt sur les valeurs instantanées des hauteurs qui sont calculées selon l'équation (3.4) et qui sont utilisées dans le calcul de la moyenne. Comme nous l'avons expliqué théoriquement dans la section 3.4.1.2, la hauteur des tracés ne représente pas précisément la grandeur de l'écriture et les résultats présentés ci-dessus prouvent également ce constat.

Le fait de diviser une distance sur avgHeight, avant de la présenter à un classificateur neuronal, pourrait aussi être la source de certaines erreurs de classification. Prenons comme exemple, dans un texte de grandeur constante, des patrons de deux tracés à classifier avec des distances inter-tracés semblables, mais avec des avgHeight bien écartées. Cet écart pourrait tout simplement être causé par : (1) le nombre élevé de tracés avec des parties inférieures et/ou supérieures situées dans la fenêtre de calcul de avgHeight pour un tracé et (2) le manque de ce type de tracés ou le grand nombre d'accents ou de points situés avant l'autre tracé. En principe, ces paramètres ne devraient pas intervenir dans la décision finale, et les deux patrons en question devraient être

classifiés dans la même classe, alors qu'à cause de la normalisation, ils risquent de se trouver dans deux classes différentes.

**Effet de la normalisation sur avgDistBB**

L'objectif de la normalisation sur avgDistBB consiste à présenter aux classificateurs des informations sur la grandeur des distances inter-tracés précédentes, pour ainsi leur permettre de prendre des décisions plus précises sur le type des patrons à classifier. Par exemple, si, lors de la composition d'un texte, le scripteur décidait d'écrire moins compressé en laissant plus d'espace entre les tracés, une distance inter-tracés qui devait se classifier précédemment comme inter-mots, devrait maintenant être classifiée intra-mot.

Pour que les classificateurs soient capables de tenir compte de ce changement dans leur décision, nous avons décidé de leur fournir la valeur moyenne de N dernières Dist_BB, en normalisant les distances à classifier sur cette valeur. Nous nous attendons donc à ce que, dans une écriture uniforme en terme des distances inter-tracés, la normalisation n'ait aucun effet sur les décisions des classificateurs. En plus, si les classificateurs sont entraînés sur une série de textes où les distances inter-tracés sont différentes de celles des textes utilisés pour le test, cette normalisation devra, en principe, améliorer les résultats. Toutefois, les résultats présentés dans le tableau 4.3 et la figure 4.5 vont à l'encontre de cette conclusion. La raison de cette contre-performance est l'instabilité des Dist_BB, même dans un texte où les distances inter-tracés semblent être uniformes.

La figure 4.9 montre que, pour le onzième texte de notre base de données (figure A.11), les avgDistBB varient entre 1 et 11. En raison de cette grande plage de variations, les patrons avec des distances inter-tracés identiques, mais avec des avgDistBB différentes, peuvent être erronément classifiées dans deux classes. En ce qui concerne les variations de avgDistBB et celles de avgHeight, puisuqe le ratio de $avgDistBB_{max}$ / $avgDistBB_{min}$ est nettement plus grand que celui de $avgHeight_{max}$ / $avgHeight_{min}$ (11 contre 3.58), la normalisation des distances inter-tracés sur avgDistBB produit des

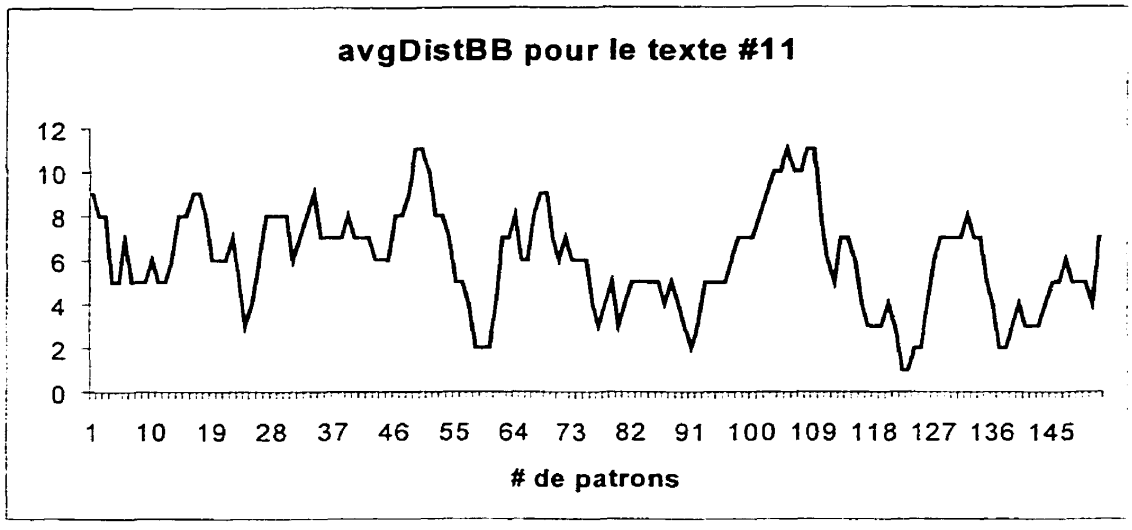



Figure 4.9. Variations d'avgGapBB pour le texte #11 (N = 9).

résultats encore pires que ceux produits par la normalisation sur avgHeight.

Le tableau 4.4 montre également que la normalisation des distances sur avgDistBB produit des effets nettement plus négatifs sur les taux de segmentation des textes cursifs ou cursifs mixtes que sur ceux des textes isolés. En moyenne, pour les textes 3, 4, 6, 10 et 12 qui sont des écritures isolées (ou en grande partie isolées), le taux de classification diminue de 4.3% après la normalisation sur avgDistBB. Tandis que cette diminution est de 21.4% pour les textes 1, 2, 5, 8, 9, 11 et 13 qui sont des écritures cursives ou cursives mixtes. Cet effet négatif chez les textes cursifs est causé par le nombre élevé des distances inter-mots comparativement à celui des distances intra-mot. Dans ce type de textes, avgDistBB est grande et sa valeur est proche de la moyenne des distances inter-mots. Par conséquent, en normalisant les distances sur avgDistBB, les valeurs des entrées du classificateur neuronal diminuent et indépendamment de leurs vrais types de distance, les patrons seront majoritairement classifiés intra-mot. La conclusion tirée est que les résultats de la normalisation des distances sur avgDistBB dépend du type d'écriture; ce qui peut être considéré comme une lacune de cette opération de normalisation.

## 4.5.2. CLASSIFICATEURS NEURONAUX AVEC LES CARACTÉRISTIQUES PROPOSÉES À L'ENTRÉE

Après avoir présenté les classificateurs neuronaux des distances inter-tracés, basés sur les caractéristiques conventionnelles, nous présenterons dans cette section, les classificateurs neuronaux utilisant les caractéristiques proposées dans le chapitre 3. Tout comme la section 4.5.1, nous présenterons au début, les entrées des réseaux. Ensuite, nous expliquerons la structure et les phases d'apprentissage et de test des réseaux et présenterons les résultats de la classification. Nous terminerons cette section avec une analyse et une discussion sur les résultats obtenus.

### 4.5.2.1 ENTRÉES DES CLASSIFICATEURS

Dans cette section, nous présenterons les caractéristiques utilisées comme les entrées des classificateurs suivies de la stratégie adoptée pour choisir les combinaisons de ces caractéristiques.

#### *4.5.2.1.1. PRÉSENTATION DES CARACTÉRISTIQUES*

Pour classifier la distance entre un tracé et son mot précédent dans une des deux classes intra ou inter-mots, nous avons utilisé au total sept différentes caractéristiques. Il s'agit de Dist_Paral, RLmin, RLavg, prHghtExtrema, curHghtExtrema, prNbExtrema et curNbExtrema. Dist_Paral est la distance horizontale entre le parallélogramme du tracé actuel et celui du mot précédent (voir la section 3.4.1.1 pour obtenir plus de détails). RLmin et RLavg sont, respectivement, les distances horizontales minimum et moyenne entre le tracé actuel et le mot précédent. PrHghtExtrema et curHghtExtrema sont les distances verticales moyennes entre les extrema consécutifs pour le mot précédent et le tracé actuel, respectivement. Et finalement, prNbExtrema est le nombre d'extrema du mot précédent et curNbExtrema est le nombre d'extrema

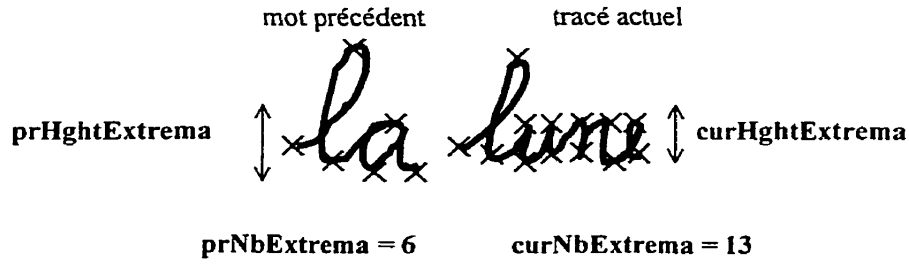



Figure 4.10. Représentation graphique de quatre caractéristiques proposées pour classifier les distances inter-tracés. Les '×' montrent l'emplacement des extrema.

du tracé actuel. La figure 4.10 montre, dans un exemple, les quatre dernières caractéristiques.

Parmi les sept caractéristiques ci-dessus, les trois premières sont les mesures de distances inter-tracés. Selon les explications présentées dans le chapitre 3, pour résoudre les problèmes des caractéristiques conventionnelles de distances inter-tracés, Dist_Paral tient compte des inclinaisons du tracé actuel et du mot précédent. En plus, toutes les mesures de distances sont calculées après une phase de reconnaissance des Accent_Points_TBars permettant d'éliminer ces sources d'erreur de classification.

Comme nous l'avons mentionné en détail dans la section 3.4.1.2.1, la grandeur d'un tracé est représentée plus précisément par la distance verticale moyenne entre ses extrema que par sa hauteur absolue. En utilisant prHghtExtrema et curHghtExtremas dans le processus de classification, nous désirons introduire l'effet de grandeur de l'écriture dans la décision des classificateurs de distances. Comme nous allons voir dans la section suivante, nous avons appliqué prHghtExtrema et curHghtExtrema comme les entrées des classificateurs ou les facteurs de normalisation des autres entrées.

En général, dans un texte rédigé dans une langue vivante, les mots longs en terme de nombre de caractères sont moins fréquents que les mots courts ou moyens. Dans le cas de notre système, on s'attend donc à une augmentation de probabilité de la classification d'un espacement inter-tracés en distance inter-mots, lorsque le nombre de

caractères de mot précédent et de tracé actuel augmentent. Toutefois, la relation entre le seuil de décision et le nombre de caractères n'est pas évidente. Dans notre système, puisque l'on ne reconnaît pas les caractères et que les informations précises sur leur nombre dans les mots ne sont pas disponibles, les caractéristiques prNbExtrema et curNbExtrema servent à fournir à nos classificateurs, des renseignements approximatifs sur la longueur du mot précédent et du tracé actuel, en terme de nombre d'extrema. Ce dernier est une fonction du nombre de caractères dans un mot ou dans un tracé.

#### *4.5.2.1.2. STRATÉGIE ADOPTÉE POUR CHOISIR LES ENTRÉES*

Les entrées des classificateurs neuronaux ont été choisi parmi les caractéristiques présentées dans la section 4.5.2.1.1. Dans cette section, nous essayons d'expliquer la stratégie que nous avons adoptée pour sélectionner les entrées des classificateurs et leurs combinaisons.

Afin de les comparer directement avec les caractéristiques conventionnelles, les trois mesures principales d'espacement inter-tracé proposées, c'est-à-dire Dist_Paral, RLmin et Rlavg, sont appliquées séparément aux classificateurs. Nous avons également utilisé ces caractéristiques après la normalisation sur prevHeightExtrema, pour observer l'effet de normalisation des espacements inter-tracés sur un facteur plus précis que avgHeight (utilisé dans le cas des caractéristiques conventionnelles). Toutefois, à la lumière des résultats obtenus avec la normalisation des caractéristiques conventionnelles sur avgDistBB (la distance moyenne entre les rectangles) et la discussion présentée sur ce sujet dans la section 4.5.1, avgDistBB n'a pas été utilisée pour la normalisation des caractéristiques proposées. Aucune des quatre autres caractéristiques n'a été appliquée de façon indépendante aux classificateurs, puisque nous les avons proposées uniquement pour améliorer la performance de classification des trois mesures principales.

Ces trois caractéristiques ont également été appliquées aux classificateurs sous forme de combinaisons de deux et de trois et cela pour observer l'effet de leurs combinaisons sur les résultats et trouver la meilleure combinaison possible.

Et finalement pour évaluer l'impact d'application de toutes les caractéristiques extraites à un classificateur, un des réseaux implantés les utilisent pour la classification de distances.

### 4.5.2.2 STRUCTURES DES CLASSIFICATEURS

Afin de segmenter les textes manuscrits à l'aide de diverses combinaisons de sept caractéristiques présentées dans la section précédente, nous avons construit des classificateurs neuronaux de type Perceptron à couches multiples avec cinq différentes structures : (1) un seul neurone avec une entrée et une sortie; (2) une entrée, une couche d'entrée, une couche cachée avec deux neurones et une couche de sortie avec un neurone; (3) deux entrées, une couche d'entrée, une couche cachée avec deux neurones et une couche de sortie avec un neurone; (4) trois entrées, une couche d'entrée, une couche cachée avec six neurones et une couche de sortie avec un neurone; (5) sept entrées, une couche d'entrée, une couche cachée avec dix neurones et une couche de sortie avec un neurone. La fonction d'activation de tous les neurones est la fonction *logsig* dont la valeur varie entre (0, +1). La sortie du neurone de la dernière couche des réseaux détermine le type de distances inter-tracés. Puisque la valeur de cette sortie peut varier entre 0 et +1, si elle dépasse 0.5, le type de distance sera inter-mots sinon intra-mot.

### 4.5.2.3 APPRENTISSAGE ET TEST

Deux scénarios différents ont été appliqués pour l'apprentissage des classificateurs: (1) scénario commun et (2) scénario personnalisé. Dans cette section, nous expliquerons d'abord ces scénarios et présenterons ensuite les paramètres et les patrons utilisés dans les phases d'apprentissage et de test.

#### *4.5.2.3.1 SCÉNARIOS D'APPRENTISSAGE*

Dans le scénario « commun », l'apprentissage d'un classificateur est effectué à l'aide de patrons extraits d'un texte manuscrit et les autres textes de la base de données sont utilisés dans la phase de test. Dans le cas du scénario « personnalisé », l'apprentissage est effectué sur une partie des patrons d'un texte donné et le test sur le reste des patrons du même texte.

Le premier scénario est celui que nous avons utilisé dans les phases de l'apprentissage et du test des classificateurs avec les caractéristiques conventionnelles à l'entrée (voir les sections 4.5.1.3 et 4.5.1.4). Un avantage de cette méthode est que l'apprentissage d'un classificateur est effectué une seule fois à l'aide d'un sous-ensemble de la base de données. Les classificateurs ne sont pas entraînés de nouveau avec l'ajout de nouveaux patrons à la base de données. En plus, si le nombre total de patrons disponibles est élevé, on disposera d'un nombre suffisant de patrons utilisables pour l'apprentissage et le test. Toutefois, le succès de ce scénario dépend principalement de la représentativité des patrons d'apprentissage sur l'ensemble des patrons. Autrement dit, pour que le réseau puisse bien généraliser, les patrons utilisés dans les phases d'apprentissage et de test doivent posséder des distributions statistiques semblables.

Le deuxième scénario s'avère utile lorsque la base de données est composée de sous-ensembles de patrons dont les distributions sont différentes en terme de forme, de moyenne et/ou d'étendue. Cette méthode est donc plus adaptative et plus flexible que la première et pour ce type de bases de données, les résultats de la classification doivent être plus satisfaisants. Toutefois, ce scénario nécessite la partition préalable des patrons. En plus, il faut entraîner un classificateur pour chaque sous-ensemble de la base de données. Un autre problème est la réduction du nombre de patrons disponibles pour l'apprentissage et le test, causée par la répartition de la base de donnée en plusieurs sous-ensembles. Le résultat pourrait être le manque de patrons suffisants pour l'entraînement du réseau, notamment dans le cas de réseaux ayant des structures complexes.

L'écriture manuscrite, en général, et la base de données que nous disposons, en particulier, favorisent l'application de l'apprentissage personnalisé. Les êtres humains ont tous leurs propres styles d'écriture dont les espacements inter-tracés et les relations de ces espacements avec d'autres paramètres de l'écriture. Étant donné que notre base de données est composée de textes écrits par différents sujets, elle est automatiquement subdivisée en sous-ensembles statistiquement homogènes. Nous pouvons donc diviser l'ensemble des patrons d'un texte en deux parties : une partie pour entraîner nos classificateurs et l'autre pour les tester. Le nombre de patrons à utiliser pour l'apprentissage dépend de la complexité du réseau neuronal et de la précision souhaitable pour la généralisation présentée sous forme de l'erreur désirée.

Dans les paragraphes suivants, nous expliquerons comment nous avons déterminé la taille des sous-ensembles de chaque fichier de texte utilisé pour l'apprentissage personnalisé. Ce type d'apprentissage permet, à chaque personne, d'avoir son propre classificateur de distances inter-tracés. Dans notre cas, puisque certains sujets avaient composé des textes en différents styles isolé, cursif ou cursif mixte, nous avons attribué un classificateur à chacun des textes et non pas à chacun des sujets.

#### *4.5.2.3.2 PARAMÈTRES ET PATRONS D'APPRENTISSAGE ET DE TEST*

Pour les deux scénarios (commun et personnalisé) appliqués aux classificateurs neuronaux avec les caractéristiques proposées, nous avons utilisé les mêmes valeurs de paramètres d'apprentissage des classificateurs neuronaux utilisées pour les caractéristiques conventionnelles (section 4.5.1.3). Il s'agit donc des valeurs suivantes :

- Moment $\alpha$ : 0.95;
- Ratio de la nouvelle erreur sur l'ancienne erreur $Err\_ratio$ : 1.04;
- Taux d'apprentissage initial $\eta_0$ : 0.01;
- Coefficient d'augmentation de taux d'apprentissage $C_{aug}$ : 1.05;
- Coefficients de diminution de taux d'apprentissage $C_{dim}$ : 0.75.
- Sorties désirées : 0.1 pour intra-mot et 0.9 pour inter-mots.

La technique de la « validation croisée » a été utilisée pour entraîner les classificateurs et pour déterminer le point d'arrêt de l'entraînement (voir la section 4.2.2.2.1). Pour ce faire, 80% des patrons de l'ensemble d'apprentissage ont été utilisés dans le sous-ensemble d'estimation et 20% dans le sous-ensemble de validation. Le nombre maximal d'itérations était de 4000 et à chaque 500 itérations, on testait la performance des classificateurs sur l'ensemble de validation pour arrêter les itérations dès que l'erreur de validation cessait de diminuer.

Quant aux patrons d'apprentissage, tout comme les classificateurs des caractéristiques conventionnelles, les patrons extraits du texte no. 7 (figure A.7) ont été utilisés pour l'apprentissage commun. Toutefois, à cause d'une phase préalable de la reconnaissance des Accents_Points_TBars, le nombre de patrons d'apprentissage est inférieur à celui des distances inter-tracés dans le texte original. Pour déterminer le nombre de patrons nécessaires pour l'apprentissage de nos classificateurs (qui sont des Perceptron multicouches), nous avons employé l'équation suivante, basée sur la théorie statistique d'apprentissage de Vapnik (Vapnik, 1982) et présentée dans (Haykin, 1999) :

$$N_{app} = O(\frac{W}{\varepsilon}) \tag{4.20}$$

où : $N_{app}$ est le nombre de patrons d'apprentissage;
$W$ est le nombre total des paramètres libres du réseau;
$\varepsilon$ est l'erreur de généralisation permise;
$O(.)$ représente l'ordre de grandeur.

Les résultats de l'application de l'équation (4.20) sur les différents classificateurs neuronaux sont montrés dans le tableau 4.6. Pour calculer $N_{app}$, nous avons choisi une valeur relativement élevée de $\varepsilon$. Le tableau 4.6 montre que dans le cas du scénario commun des quatre premiers classificateurs, on dispose d'un nombre suffisant de patrons d'apprentissage, sauf, probablement, pour le cinquième type de classificateurs. Toutefois, afin de comparer les résultats de la classification, nous avons utilisé le même texte pour le scénario commun de tous les types de classificateurs.

Tableau 4.6. Nombre de partons d'apprentissage ($N_{app}$) en fonction du nombre de paramètres libres des classificateurs neuronaux ($W$) et de l'erreur de généralisation ($\varepsilon$).

| Structure de réseau | $W$ | $\varepsilon$ | $N_{app}$ |
|---|---|---|---|
| 1-1 | 2 | 0.1 | 20 |
| 1-2-1 | 7 | 0.1 | 70 |
| 2-2-1 | 9 | 0.1 | 90 |
| 3-6-1 | 31 | 0.1 | 310 |
| 7-10-1 | 91 | 0.1 | 910 |

En ce qui concerne le scénario personnalisé, le nombre de patrons utilisés pour l'apprentissage doit, autant que possible, être limité, puisque les patrons d'un texte donné servent à entraîner et aussi à tester les classificateurs. Cette contrainte peut empêcher l'utilisation de réseaux complexes pour la segmentation de courts textes (en terme de nombre d'espacements inter-tracés). Dans le cas de notre base de données, en comparant le nombre de patrons des différents textes (tableau A.1) avec le nombre nécessaire pour l'apprentissage des classificateurs neuronaux, on constate que seulement les trois premiers types de réseaux dans le tableau 4.6 peuvent être utilisés adéquatement pour classifier tous les textes disponibles[14]. Dans la section suivante, nous présenterons les classificateurs utilisés pour l'apprentissage personnalisé, le nombre de patrons d'entraînement et les résultats de la classification.

## 4.5.2.4 RÉSULTATS DE LA CLASSIFICATION

Pour les deux scénarios de l'apprentissage commun et personnalisé, les taux de classification correcte des distances inter-tracés sont présentés dans cette section. Les résultats de la reconnaissance des Accent_Point_TBars sont inclus dans les pourcentages de classification correcte.

[14] Nous aborderons ce sujet en détail dans la section 4.5.2.5.3 et nous démonterons que cette limitation n'est pas causée par la particularité de la base de données, mais plutôt par celle des systèmes interactifs ergonomiques.

Tableau 4.7. Pourcentage de la classification correcte des distances inter-tracés pour les classificateurs neuronaux avec caractéristiques proposées (scénario commun).

**Numéro de Texte**

| | *1* | *2* | *3* | *4* | *5* | *6* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **D_Prl, RLavg** | 97.3 | 98.7 | 99.8 | 100 | 99.4 | 99.2 | 99.1 | 99 | 97.5 | 97.1 | 98.6 | 97.8 | **98.63** |
| **D_Prl, RLmin** | 93.4 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.6 | 98.1 | 98.1 | 97.9 | 98.6 | 98.4 | **98.4** |
| **D_Prl, RLavg, pNbExt** | 97.3 | 98.7 | 99.3 | 99.3 | 99.4 | 98.8 | 98.6 | 99 | 96.4 | 97.5 | 98.6 | 97.3 | **98.35** |
| **D_Prl** | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| **D_Prl*** | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| **D_Prl, RLmin, pNbExt** | 90.1 | 100 | 99.8 | 100 | 98.8 | 99.4 | 98.6 | 98.1 | 98.1 | 97.9 | 99 | 97.8 | **98.13** |
| **Rlmin, Rlavg** | 94 | 99.4 | 99.8 | 99.8 | 98.8 | 99.8 | 98.1 | 97.6 | 96.4 | 97.1 | 97.6 | 97.3 | **97.98** |
| **Rlmin** | 92.3 | 99.4 | 99.8 | 99.8 | 98.8 | 99.8 | 98.1 | 97.1 | 96.4 | 97.1 | 98 | 97.3 | **97.83** |
| **Rlmin*** | 86.8 | 99.4 | 99.8 | 100 | 98.8 | 99.8 | 96.7 | 98.1 | 98.1 | 96.7 | 98.3 | 97.3 | **97.48** |
| **D_Prl / pHghtExt** | 94.5 | 99.4 | 99.3 | 99.6 | 97.6 | 99.2 | 98.1 | 98.1 | 95.1 | 92.1 | 98 | 97.3 | **97.36** |
| **RLmin / pHghtExt** | 92.9 | 98.7 | 99.1 | 99.1 | 97.1 | 98.7 | 96.7 | 96.2 | 91.2 | 95.8 | 94.9 | 96.2 | **96.38** |
| **D_Prl, RLavg, RLmin** | 79.7 | 99.4 | 99.8 | 100 | 97.6 | 99.4 | 91.9 | 93.8 | 96.4 | 94.1 | 99 | 95.1 | **95.52** |
| **Rlavg*** | 94 | 98.7 | 97.6 | 98.4 | 97.1 | 96 | 92.9 | 94.3 | 93.2 | 85.8 | 90.2 | 95.6 | **94.48** |
| **Rlavg** | 95.6 | 98.1 | 97.1 | 98 | 97.1 | 96.2 | 93.4 | 94.3 | 90.7 | 84.5 | 88.8 | 96.2 | **94.17** |
| **RLavg / pHghtExt** | 92.9 | 98.1 | 95.1 | 95.3 | 94.7 | 95 | 92.9 | 94.3 | 86.3 | 86.6 | 85.8 | 95.1 | **92.68** |
| **7 entrées** | 77.5 | 98.1 | 99.6 | 99.1 | 91.2 | 99.6 | 86.7 | 87.6 | 92.1 | 93.3 | 96.3 | 84.7 | **92.15** |

#### *4.5.2.4.1 RÉSULTATS DU SCÉNARIO COMMUN*

Le tableau 4.7 montre les résultats obtenus avec le scénario de l'apprentissage commun des classificateurs. Dans ce tableau nous utilisons les symboles suivants : D_Prl pour Dist_Paral, RLavg et RLmin pour les distances horizontales moyenne et minimale entre le tracé actuel et le mot précédent et pNbExt et pHghtExt pour le nombre d'extrema et la distance verticale moyenne entre les extrema du mot précédent, respectivement. Dans le cas des classificateurs avec une entrée, deux structures de réseau ont été utilisées : 1-2-1 (une entrée, 2 neurones dans la couche cachée et une sortie) et 1-1 (un neurone à une entrée et une sortie) et pour les distinguer dans le tableau 4.7, nous avons ajouté un '*' à ceux ayant la deuxième structure (1-1). Un des

Tableau 4.8. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 4.7.

| | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *D_Prl, RLavg* | 0.98 | 98.63 | (98.12, 99.13) |
| *D_Prl, RLmin* | 1.73 | 98.40 | (97.50, 99.30) |
| *D_Prl, RLavg, pNbExt* | 0.98 | 98.35 | (97.84, 98.86) |
| *D_Prl* | 2.62 | 98.14 | (96.78, 99.50) |
| *D_Prl** | 2.62 | 98.14 | (96.78, 99.50) |
| *D_Prl, RLmin, pNbExt* | 2.65 | 98.13 | (96.76, 99.51) |
| *RLmin, Rlavg* | 1.72 | 97.98 | (97.08, 98.87) |
| *RLmin* | 2.11 | 97.83 | (96.73, 98.92) |
| *RLmin** | 3.56 | 97.48 | (95.64, 99.33) |
| *D_Prl / pHghtExt* | 2.31 | 97.36 | (96.16, 98.56) |
| *RLmin / pHghtExt* | 2.48 | 96.38 | (95.10, 97.67) |
| *D_Prl, RLavg, RLmin* | 5.67 | 95.52 | (92.57, 98.46) |
| *RLavg** | 3.71 | 94.48 | (92.56, 96.41) |
| *RLavg* | 4.18 | 94.17 | (92.00, 96.34) |
| *RLavg / pHghtExt* | 4.10 | 92.68 | (90.55, 94.80) |
| *7 entrées* | 6.97 | 92.15 | (88.54, 95.76) |

classificateurs possède 7 entrées qui correspondant à toutes les caractéristiques proposées (voir la section 4.5.2.1).

Dans le tableau 4.8, nous présentons le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 4.7, incluant les estimés pour l'écart type et la moyenne ainsi que l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

La figure 4.11 est une représentation graphique des intervalles de confiance montrés dans le tableau 4.8.

#### *4.5.2.4.2 RÉSULTATS DU SCÉNARIO PERSONNALISÉ*

La deuxième série de résultats correspond à l'application du scénario de l'apprentissage personnalisé aux classificateurs neuronaux utilisant les caractéristiques proposées. Ces résultats sont présentés dans le tableau 4.9. Comme nous l'avons

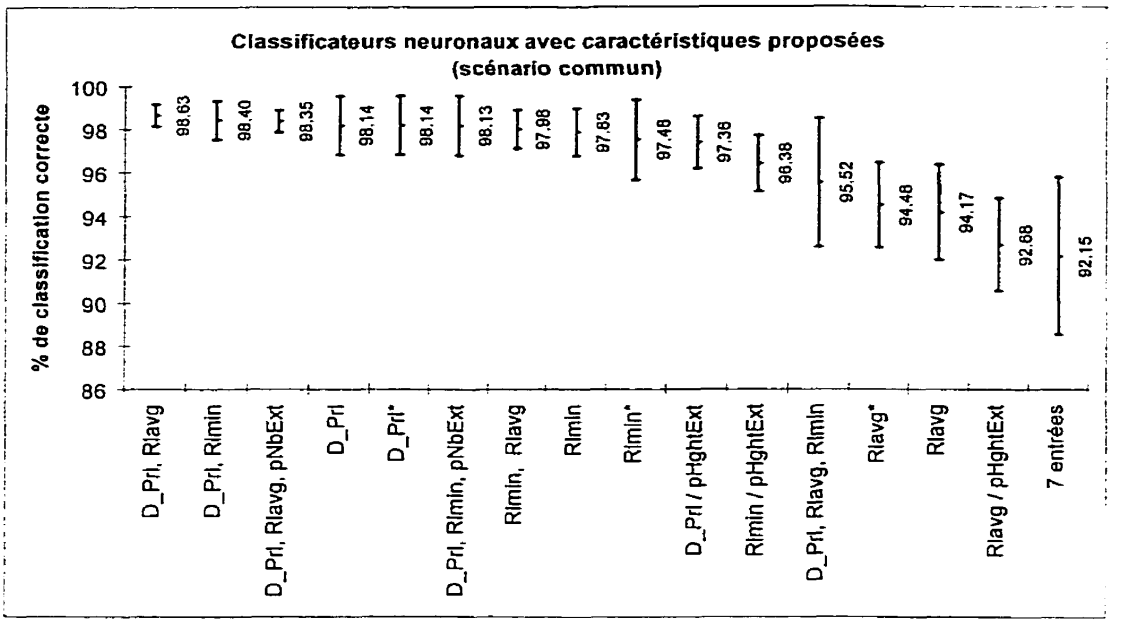



Figure 4.11. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs neuronaux avec les caractéristiques proposées (scénario commun).

mentionné dans la section précédente, la contrainte du nombre de patrons nécessaires pour l'entraînement, nous limite à appliquer cette méthode uniquement sur les

Tableau 4.9. Pourcentage de la classification correcte des distances inter-tracés pour les classificateurs neuronaux avec caractéristiques proposées (scénario personnalisé).

| | **Numéro de Texte** | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** | ***11*** | ***12*** | ***13*** | ***Moy.*** |
| ***RLmin (N = 70)*** | 98.3 | 100 | 99.7 | 100 | 100 | 99.8 | 98.9 | 96.9 | 98.9 | 98.6 | 95.7 | 100 | 95.8 | **98.66** |
| ***D_Prl (N = 40)*** | 98.2 | 97.5 | 99.8 | 100 | 98.8 | 99.8 | 98.6 | 98.6 | 99.3 | 98.7 | 95.6 | 98.3 | 95.3 | **98.35** |
| ***D_Prl (N = 70)*** | 100 | 95.3 | 99.7 | 100 | 100 | 99.8 | 98.7 | 99 | 100 | 99.3 | 95 | 100 | 91.7 | **98.35** |
| ***RLmin (N = 40)*** | 95.5 | 100 | 99.8 | 100 | 97.7 | 99.8 | 98.6 | 96.6 | 98.6 | 97.8 | 96.2 | 100 | 97.2 | **98.29** |
| ***D_Prl, RLmin (N = 70)*** | 100 | 100 | 99.7 | 100 | 100 | 99.8 | 98.7 | 97.9 | 100 | 98.9 | 96.4 | 98 | 81.3 | **97.75** |
| ***D_Prl, RLavg (N = 70)*** | 93.1 | 100 | 99.7 | 100 | 100 | 99.8 | 97.9 | 97.9 | 100 | 98.6 | 94.3 | 99 | 87.5 | **97.52** |
| ***RLmin, RLavg (N =70)*** | 94.9 | 90.7 | 99.7 | 99.7 | 91.3 | 99.1 | 98.2 | 95.8 | 98.9 | 97.1 | 92.9 | 100 | 91.7 | **96.15** |
| ***RLavg (N = 70)*** | 96.6 | 100 | 97.2 | 98.6 | 100 | 95.6 | 94.5 | 92.7 | 95.7 | 95.7 | 88.6 | 94.4 | 93.8 | **95.65** |
| ***RLavg (N = 40)*** | 92.8 | 98.7 | 96.3 | 98.5 | 94.2 | 95.7 | 94.5 | 92.4 | 94.4 | 94.6 | 87.4 | 94.5 | 95.3 | **94.56** |

Tableau 4.10. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 4.9.

| | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| ***RLmin (N = 70)*** | **1.58** | **98.66** | **(97.88, 99.44)** |
| ***D_Prl (N = 40)*** | **1.47** | **98.35** | **(97.62, 99.07)** |
| ***D_Prl (N = 70)*** | **2.64** | **98.35** | **(97.04, 99.65)** |
| ***RLmin (N = 40)*** | **1.59** | **98.29** | **(97.51, 99.08)** |
| ***D_Prl, RLmin (N = 70)*** | **5.07** | **97.75** | **(95.24, 100)** |
| ***D_Prl, RLavg (N = 70)*** | **3.75** | **97.52** | **(95.67, 99.38)** |
| ***Rlmin, RLavg (N = 70)*** | **3.50** | **96.15** | **(94.43, 97.88)** |
| ***RLavg (N = 70)*** | **3.09** | **95.65** | **(94.12, 97.17)** |
| ***RLavg (N = 40)*** | **2.84** | **94.56** | **(93.16, 95.97)** |

classificateurs avec des structures simples. Le tableau 4.9 ne contient donc que des réseaux neuronaux avec une ou deux entrées. Nous avons précisé, pour chacun des classificateurs, le nombre de patrons utilisés *(N)* dans la phase de l'apprentissage.

Le tableau 4.10 présente le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 4.9, incluant les estimés pour l'écart type et la moyenne ainsi que l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

La figure 4.12 est une représentation graphique des intervalles de confiance montrés dans le tableau 4.10.

## 4.5.2.5 ANALYSE ET DISCUSSION

Dans cette section, nous jetterons un coup d'œil plus profond sur les résultats présentés dans la section précédente, et ce, en mettant l'accent sur les aspects suivants :

1- Impacts de nos propositions sur les résultats;

2- Effets de l'utilisation de plus d'une caractéristique pour la segmentation en-ligne de textes manuscrits en mots;

3- Comparaison entre les scénarios commun et personnalisé.

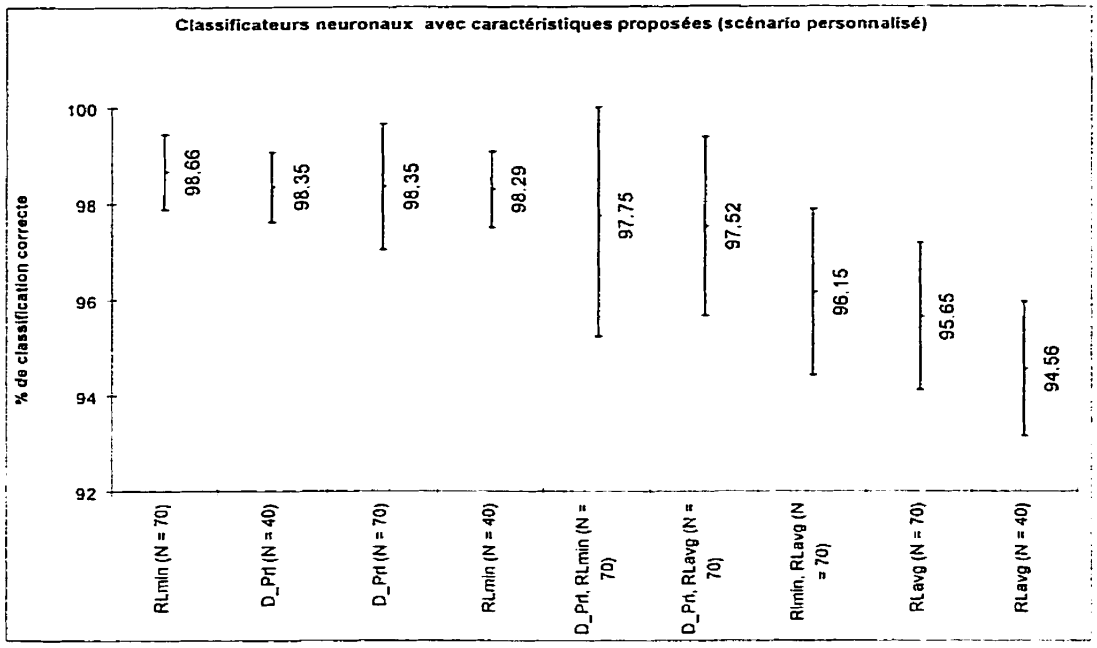



Figure 4.12. Les pourcentages moyens de la classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs neuronaux avec les caractéristiques proposées (scénario personnalisé).

#### *4.5.2.5.1 AMÉLIORATION DE LA PERFORMANCE DES CLASSIFICATEURS GRÂCE À NOS PROPOSITIONS*

Les tableaux 4.3 et 4.7 contiennent respectivement les résultats de la classification des distances inter-tracés avec les caractéristiques conventionnelles et proposées. Ces résultats montrent une amélioration considérable de la performance de la classification grâce à l'application de nos propositions pour résoudre les problèmes des caractéristiques conventionnelles. Rappelons ces propositions : (1) ajouter une phase de la reconnaissance des Accent_Point_TBars avant l'extraction des caractéristiques et (2) remplacer la distance entre les rectangles (Dist_BB) par la distance entre les

parallélogrammes (Dist_Paral ou D_Prl dans le tableau 4.7). La première a des effets positifs sur toutes les caractéristiques, alors que la deuxième n'améliore que Dist_BB.

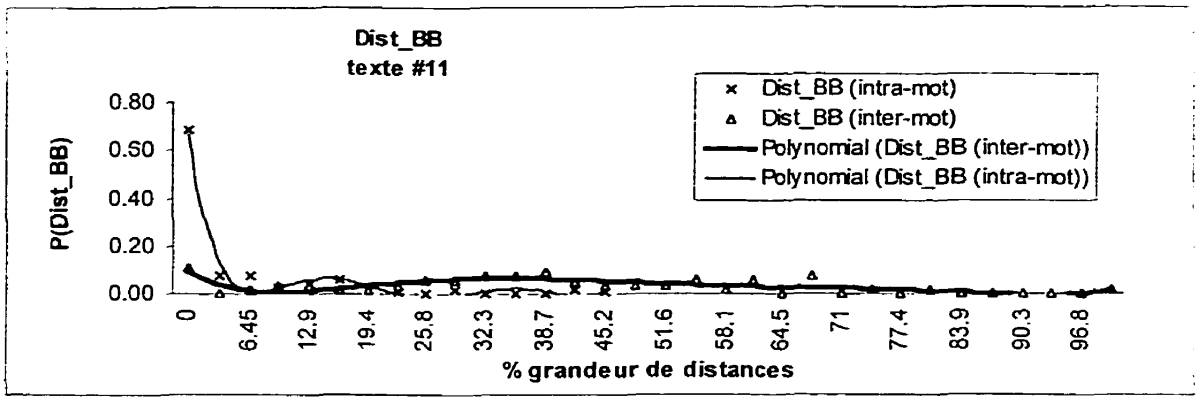
Selon les résultats du tableau 4.7, (et toujours à cause des mêmes arguments présentés dans la section 4.5.1.7 pour comparer les performances des mesures conventionnelles), parmi les trois mesures de distances proposées, c'est Dist_Paral qui connaît la meilleure performance suivie de RLmin et RLavg. Nous allons ainsi nous concentrer sur une comparaison entre les mesures de distances avant et après l'application de nos propositions, sans comparer, entre elles, les performances de Dist_Paral, RLmin et RLavg.

**De Dist_BB à Dist_Paral**

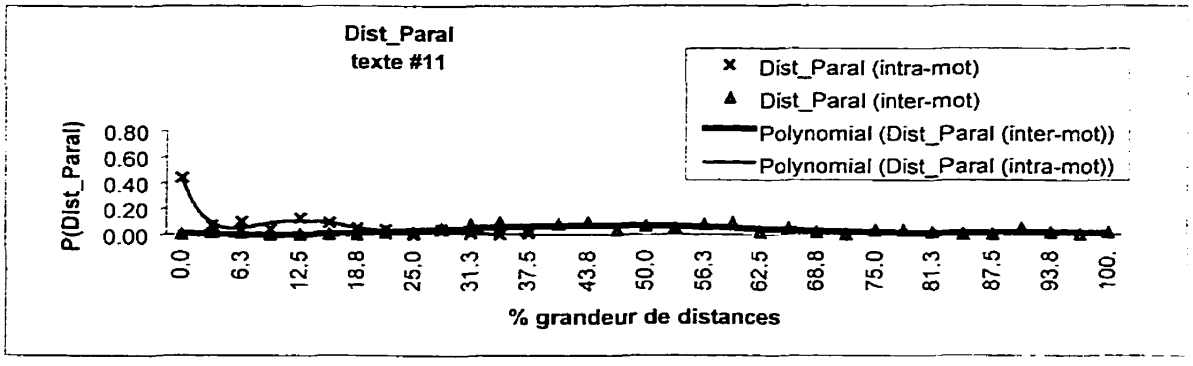
Nous commençons par comparer Dist_BB avec Dist_Paral. Les résultats des tableaux 4.3 et 4.7 montrent une augmentation du taux de classification correcte pour la Dist_Paral par rapport à la Dist_BB (98.14% et 97.7% respectivement). Nous nous servons des graphiques de la figure 4.13 pour expliquer les causes de cette amélioration de la performance.

La figure 4.13(a) montre les distributions de Dist_BB intra et inter-mots et leurs courbes de tendance (régression polynomiale de sixième degré) pour le texte #11. Nous avions déjà utilisé ce graphique dans la figure 4.7(a) pour analyser les résultats de la classification des distances en utilisant les caractéristiques conventionnelles. Nous le présenterons de nouveau, à ce point, afin de le comparer avec les distributions de Dist_Paral intra et inter-mots présentées dans la figure 4.13(b).

Comparons d'abord les distributions inter-mots. Pour ces distributions, une des améliorations les plus importantes est observable à l'origine (au point de 0%). Les distances inter-mots à ce point, qui sont les sources d'erreurs de classification pour Dist_BB, sont disparues dans la distribution de Dist_Paral. À la section 4.5.1.7.1, nous avons mentionné que ces valeurs inattendues de Dist_BB inter-mots sont, en général, causées par le chevauchement horizontal des tracés qui est, à son tour, produit




(a)




(b)

Figure 4.13. Distributions de (a) Dist_BB et (b) Dist_Paral intra et inter-mots et leurs courbes de tendance pour le texte #11.

par les tracés inclinés et les barres de 't'. Pour le calcul des Dist_Paral, nous tenons compte des inclinaisons de l'écriture; ce qui nous permet de mesurer les distances inter-tracés plus précisément et donc d'éliminer les chevauchements erronés (voir l'exemple de la figure 3.10). En plus, grâce à la phase de la reconnaissance des Accents_Points_TBars, les autres sources de chevauchements erronés sont supprimées de l'ensemble des patrons à classifier.

Une autre cause de l'amélioration de la performance de Dist_Paral est l'estimation plus précise des distances inter-tracés par rapport à Dist_BB. Cette dernière a tendance à sous-estimer les distances. Par exemple, dans la figure 4.13, 66% des

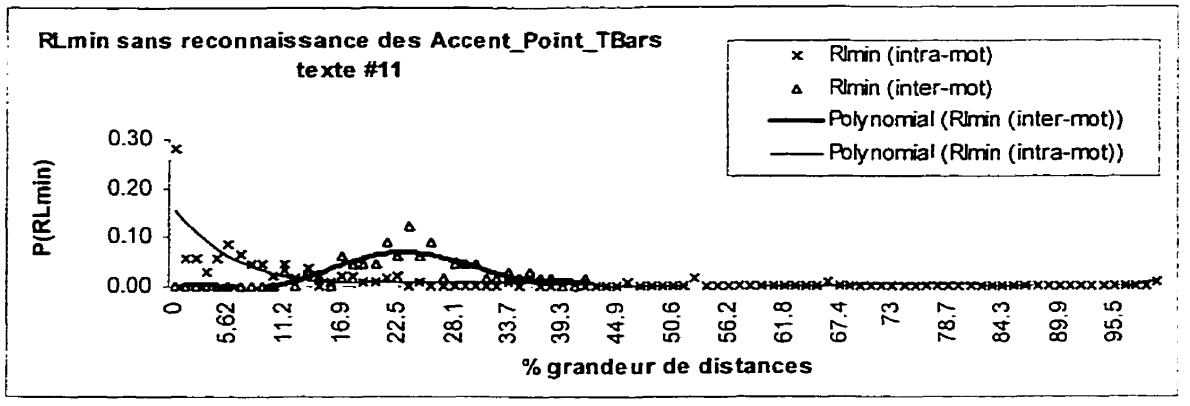
Dist_BB inter-mots sont inférieures à 50% de la distance maximale, tandis que seulement 53% des Dist_Paral sont en dessous de cette valeur. En plus, le sommet des Dist_BB inter-mots est situé au point du 33%, alors que ce sommet pour les Dist_Paral inter-mots se trouve au point du 44%. Ce déplacement des Dist_Paral inter-mots diminue le chevauchement entre les courbes intra et inter-mots et contribue à l'augmentation du taux de classification correcte.

En ce qui concerne les distances intra-mot, toujours en raison de la réduction du nombre de chevauchements horizontaux, le pourcentage des distances à 0% a diminué pour Dist_Paral par rapport à Dist_BB (de 68% à 44%). Toutefois, cette diminution est accompagnée par une augmentation (de même grandeur) du nombre de distances intra-mot entre 1 et 30%. En plus, la distance intra-mot maximale a été diminuée (de 45% pour Dist_BB à 37% pour Dist_Paral), puisque les accents et les points qui sont les causes des grandes distances intra-mot sont éliminés. En somme les Dist_Paral intra-mot sont déplacées vers la gauche; ce qui fait diminuer le chevauchement entre les distributions intra et inter-mots, notamment en tenant compte du déplacement de la distribution des Dist_Paral inter-mots vers la droite.

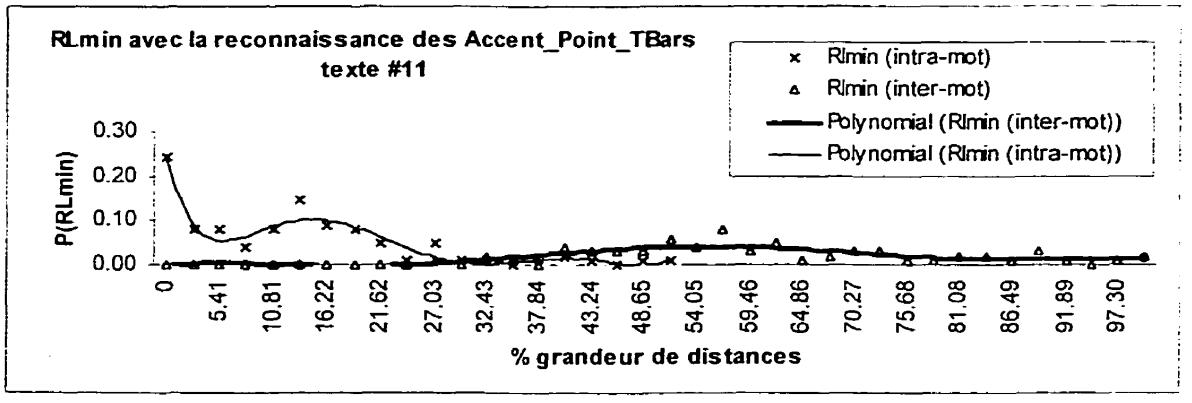
**Distances horizontales et la reconnaissance des Accent_Point_TBars**

Nous évaluons maintenant l'effet de la reconnaissance des Accent_Point_TBars sur les distributions des distances horizontales. Les résultats de la classification des distances inter-tracés avec RLmin et RLavg (tableaux 4.3 et 4.7) montrent que l'élimination des accents et des points améliore la performance des classificateurs. Les distributions des distances intra et inter-mots sont également en accord avec cette conclusion. Dans les figures 4.14 et 4.15, respectivement, nous montrons les distributions intra et inter-mots sans et avec la reconnaissance des Accent_Point_TBars, pour les mesures RLmin et RLavg extraites du texte #11 de notre base de données.

En ce qui concerne les distances intra-mot, on peut voir dans les distributions des figures 4.14(b) et 4.15(b), que les valeurs extrêmes et irréalistes (100%), causées par les accents et points, sont éliminées par rapport aux figures 4.14(a) et 4.15(a).



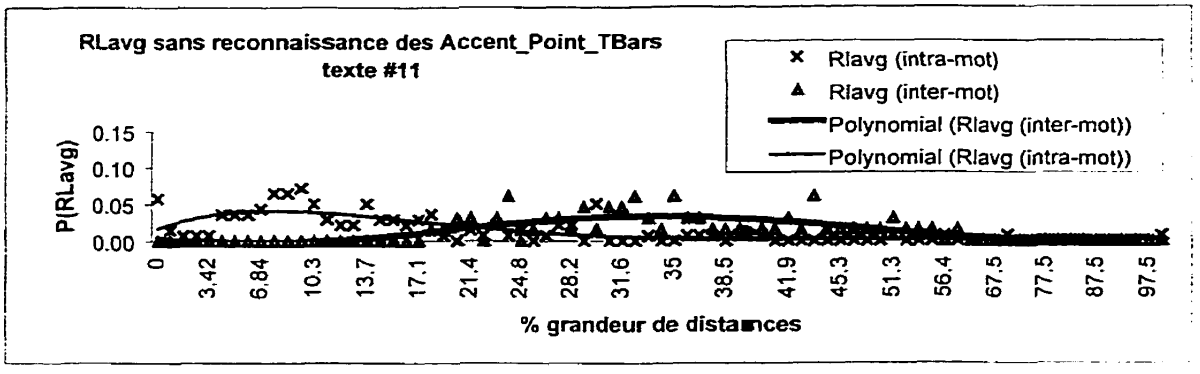

(a)



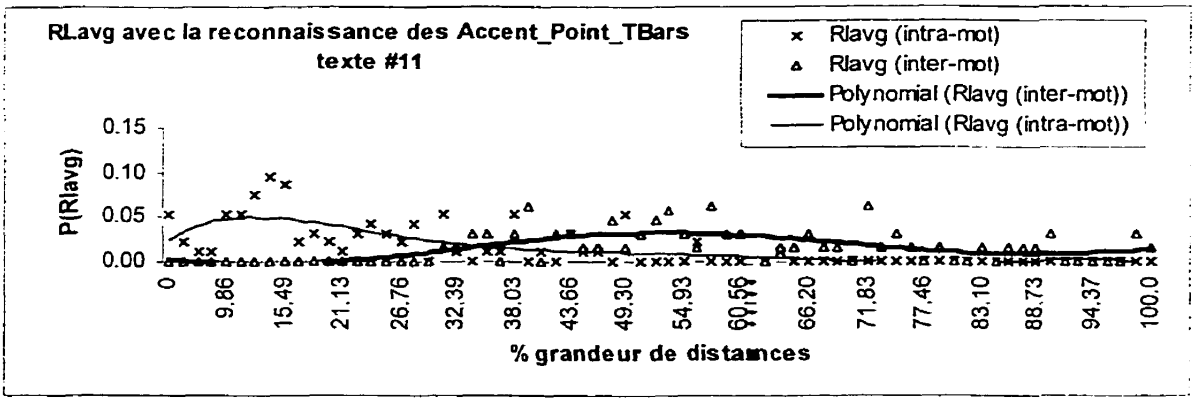

(b)

Figure 4.14. Distributions des RLmin intra et inter-mots et leurs courbes de tendance pour le texte #11 : (a) avant et (b) après la reconnaissance des signes diacritiques.

Les valeurs minimales des RLmin et RLavg inter-mots sans la reconnaissance des Accent_Point_TBars sont de 11% et de 18.8% respectivement, tandis que les mêmes valeurs pour les RLmin et RLavg inter-mots après la reconnaissance des Accent_Point_TBars sont de 27% et de 31%. Cette augmentation est principalement due à l'élimination des points et des virgules entre les phrases. En plus, les sommets des distributions inter-mots sont déplacés de 23% à 56% pour RLmin, et de 33% à 53% pour RLavg. En somme, ce déplacement des distributions inter-mots vers la




(a)




(b)

Figure 4.15. Distributions des RLavg intra et inter-mots et leurs courbes de tendance pour le texte #11 : (a) avant et (b) après la reconnaissance des signes diacritiques.

droite aide à séparer les distributions intra-mot de celles des inter-mots, à réduire leur chevauchement et donc à diminuer l'erreur de classification des distances.

### *4.5.2.5.2 EFFET DE COMBINER LES CARACTÉRISTIQUES PRÉSENTÉES AUX CLASSIFICATEURS*

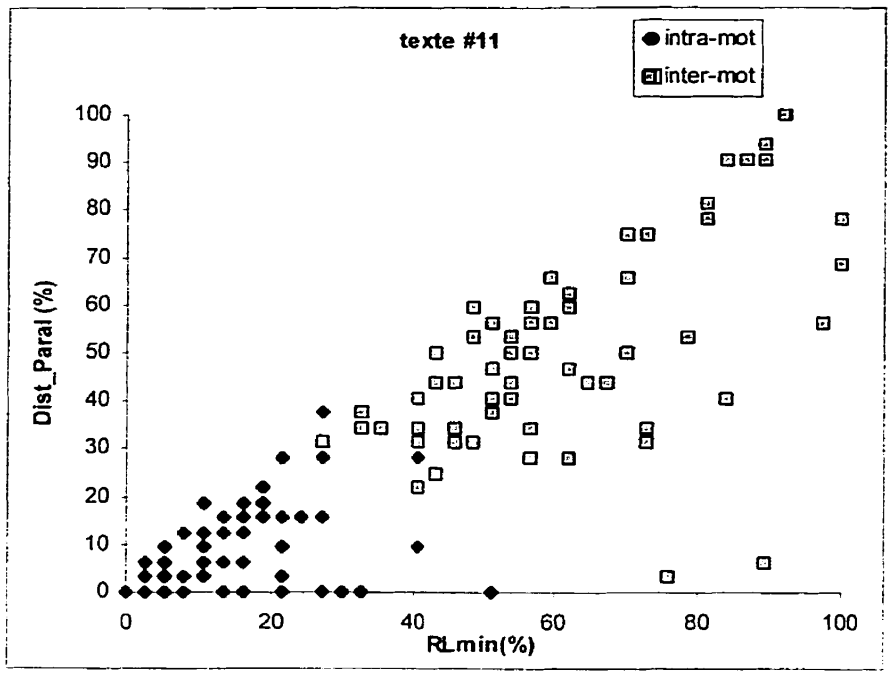
Les taux de classification correcte présentés dans le tableau 4.3 montrent que parmi les caractéristiques conventionnelles, Dist_BB donne le meilleur résultat. Selon

le tableau 4.7 également, parmi les trois caractéristiques proposées, Dist_Paral, étant la version améliorée de Dist_BB, produit la meilleure performance. Toutefois, les taux de classification les plus élevés appartiennent à des classificateurs auxquels nous avons présenté deux combinaisons de caractéristiques. Il s'agit de Dist_paral et RLavg avec la meilleure performance (98.63%) suivie de Dist_Paral et RLmin (98.4%). Grâce à cette augmentation du nombre de dimensions dans l'espace des caractéristiques, nous avons réussi à augmenter le degré de la séparation des patrons intra et inter-mots; ce qui permet d'obtenir de meilleurs résultats par rapport à ceux obtenus avec une seule entrée. La figure 4.16 montre, à titre d'exemple (pour le texte #11), les espaces de caractéristiques pour les deux meilleures combinaisons des entrées du tableau 4.5.

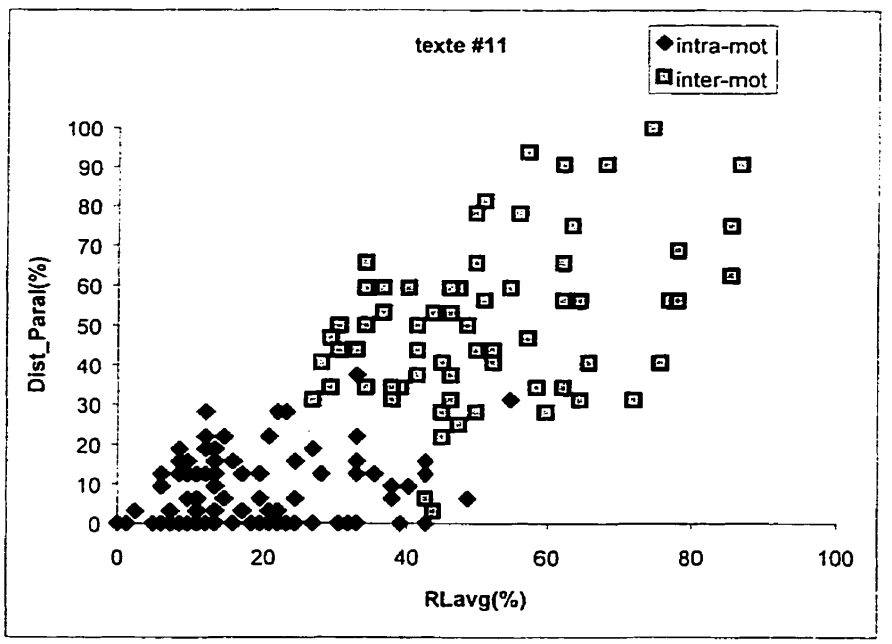
Selon le tableau 4.6 et la discussion présentée dans la section 4.5.2.3.2, le nombre de patrons utilisés pour l'apprentissage des classificateurs est suffisant pour les réseaux dont le nombre d'entrées est inférieur ou égal à trois. Nous pouvons donc déduire, à partir du tableau 4.7, que l'ajout d'une troisième caractéristique n'aide pas à améliorer la performance de la classification. Toutefois, toujours selon le tableau 4.6, en raison du nombre de patrons utilisés pour l'entraînement, le taux de classification obtenu avec les sept caractéristiques extraites à l'entrée (tableau 4.7), est probablement inférieur au taux de classification maximum que l'on peut obtenir avec la même structure de classificateur. Nous expliquerons, dans le chapitre 5, comment nous avons réussi à résoudre ce problème grâce à un nouvel algorithme d'apprentissage permettant d'obtenir de meilleurs résultats de classification, tout en utilisant un nombre limité de patrons d'apprentissage.

#### *4.5.2.5.3 SCÉNARIOS D'APPRENTISSAGE COMMUN ET PERSONNALISÉ*

Dans la section 4.5.2.3.1, nous avons présenté les deux scénarios utilisés pour l'apprentissage et le test de nos classificateurs neuronaux. Les tableaux 4.7 et 4.9 montrent que le scénario personnalisé permet d'obtenir de meilleurs résultats pour les classificateurs avec une entrée, mais cause une dégradation des résultats obtenus avec les classificateurs ayant deux entrées. La raison de ce phonème est les nombres de patrons




(a)




(b)

Figure 4.16. Espaces de caractéristiques pour deux combinaisons des mesures inter-tracés proposées (extraites du texte #11) : (a) Dist_Paral et RLmin, (b) Dist_Paral et RLavg.

utilisés pour l'apprentissage de classificateurs. Selon le tableau 4.4, ces nombres (N = 40 et 70) sont suffisants pour l'entraînement des classificateurs avec une entrée, tandis que les autres classificateurs ont besoin d'un nombre plus élevé de patrons d'entraînement.

Nous avons limité le nombre de patrons d'apprentissage pour respecter les principes de base des systèmes interactifs ergonomiques : minimiser la tâche de l'utilisateur et maximiser la performance du système. Nous désirons ainsi utiliser le plus petit nombre de patrons pour entraîner les classificateurs, et ce, afin de permettre aux scripteurs d'accomplir cette tâche le plus rapidement possible, sans les obliger à écrire de longs textes et de les segmenter manuellement. Toutefois, comme nous en avons parlé dans la section 4.5.2.5.2, lorsque le nombre de patrons était suffisant pour l'entraînement des classificateurs, l'augmentation du nombre de caractéristiques d'un à deux nous a permis d'améliorer la performance de la classification; ce qui était le cas du scénario commun.

En somme, le compromis entre le nombre de patrons nécessaires pour l'entraînement, la complexité de la structure des classificateurs (y compris le nombre d'entrées) et le nombre de patrons disponibles, notamment dans le cas des textes cursifs et courts peut limiter l'application du scénario de l'apprentissage personnalisé. Autrement dit, si pour obtenir de meilleurs résultats de classification, l'augmentation de la complexité des classificateurs était indispensable, la nécessité de l'utilisation d'une grande base de données pour entraîner des réseaux complexes, pourrait empêcher d'utiliser ce scénario dans une application en-ligne. Dans le chapitre suivant, nous aborderons une nouvelle méthode d'entraînement des classificateurs permettant de résoudre ce problème.

## 4.6 CONCLUSION

Dans ce chapitre, nous nous sommes concentrés sur les classificateurs neuronaux de distances inter-tracés, utilisés pour la segmentation en-ligne de texte manuscrits en mots. Nous avons abordé les sujets suivants :

1- Théorie des réseaux neuronaux en général et de Perceptron en particulier;
2- Classificateurs neuronaux des distances inter-tracés utilisant les caractéristiques conventionnelles;
3- Résultats de la classification obtenus avec ces classificateurs;
4- Analyse des erreurs de segmentation en les classifiant dans trois différentes catégories;
5- Effets de la normalisation des caractéristiques sur les résultats;
6- Comparaison entre les caractéristiques conventionnelles basée sur les résultats empiriques;
7- Classificateurs neuronaux des distances inter-tracés utilisant les caractéristiques proposées;
8- Scénarios de l'apprentissage commun et personnalisé des classificateurs neuronaux;
9- Résultats de la classification obtenus avec ces classificateurs;
10- Comparaison entre les caractéristiques conventionnelles et proposées basée sur les résultats empiriques.

À partir des résultats présentés dans ce chapitre, nous pouvons tirer les conclusions suivantes :

1- Contrairement à nos attentes, la normalisation des distances inter-tracés sur les valeurs moyennes de hauteurs avgHeight et de distances entre les rectangles avgDistBB dégrade la performance de la classification.
2- Dans le cas du scénario commun, les meilleurs taux de classification appartiennent aux classificateurs neuronaux dont les entrées sont une combinaison de deux caractéristiques proposées (Dist_Paral avec RLavg ou RLmin);
3- Toujours dans le cas du scénario commun, en raison du compromis entre le nombre de patrons d'entraînement et la complexité de structure des classificateurs, la performance du classificateur utilisant toutes les

caractéristiques proposées n'est pas la meilleure parmi les classificateurs testés.

4- L'application du scénario personnalisé aux classificateurs simples (avec une seule entrée) a permis d'améliorer leurs performances par rapport à celles du scénario commun.

5- Toujours en raison du compromis entre le nombre de patrons d'entraînement et la complexité de structure des classificateurs, le scénario personnalisé ne peut pas être appliqué adéquatement sur les classificateurs ayant des structures plus complexes (avec plus d'une entrée).

Dans le chapitre suivant, nous nous pencherons davantage sur le problème du compromis entre la taille de l'ensemble d'entraînement et la complexité de la structure des classificateurs qui nous a empêché d'utiliser des classificateurs plus complexes et d'appliquer le scénario personnalisé afin d'obtenir de meilleurs résultats de segmentation de textes manuscrits en mots. En fait, nous proposerons une solution à ce problème en présentant une nouvelle technique d'apprentissage des classificateurs.

# CHAPITRE 5

# CLASSIFICATEURS BASÉS SUR L'ALGORITHME D'ADABOOST

Dans le chapitre précédent, nous avons présenté les classificateurs neuronaux de distances inter-tracés constituant la première catégorie de classificateurs implantés dans ce projet. La seconde catégorie est celle des classificateurs basés sur l'algorithme d'AdaBoost[1], choisis pour résoudre le problème des classificateurs neuronaux classiques. Comme nous l'avons expliqué dans le chapitre précédent, pour obtenir des résultats satisfaisants avec des classificateurs neuronaux, nous avons besoin d'un nombre relativement élevé de patrons d'apprentissage; notamment pour les réseaux dont la structure est complexe. Dans le cas de la segmentation en-ligne d'écriture manuscrite en mots, nous avons souvent accès à un nombre limité de patrons. De plus, puisque la production d'une grande base de données pour l'apprentissage pourrait être longue et fastidieuse pour le scripteur, on désire pouvoir entraîner adéquatement les classificateurs en utilisant le moins de patrons possibles. AdaBoost est un nouvel algorithme d'apprentissage permettant d'entraîner des classificateurs sans avoir besoin d'un nombre élevé de patrons.

Dans ce chapitre, nous présenterons d'abord la théorie de l'algorithme AdaBoost. Nous aborderons ainsi les deux versions initiale et améliorée de cet algorithme. Nous

---

[1] Adaptive Boosting

expliquerons ensuite les classificateurs AdaBoost implantés dans notre projet, tout en soulignant leurs structures, les phases d'apprentissage et de test ainsi que les résultats de classification. L'analyse de ces résultats sera présentée dans le chapitre 6. Nous terminerons le chapitre 5 avec les conclusions.

# 5.1 THÉORIE D'ADABOOST

L'algorithme d'AdaBoost fait partie d'une catégorie d'algorithmes d'apprentissage nommée *Boosting*. Les algorithmes de « Boosting » sont basés sur le principe des *comité de machines*[2] qui distribuent une tâche complexe entre plusieurs *experts*. Il s'agit donc de subdiviser une tâche complexe en plusieurs tâches simples (chacune traitée par une experte) et de combiner leur solution pour trouver la solution finale (Haykin, 1999)[3].

Dans cette section, nous présenterons les deux versions initiale et améliorée d'AdaBoost. L'algorithme d'AdaBoost est, en général, un algorithme d'apprentissage supervisé, basé sur un ensemble d'*hypothèses simples*[4]. Il suffit pour une hypothèse dite « simple », que sa précision soit plus élevée qu'une hypothèse purement aléatoire (dont le taux de succès dans le cas d'une classification binaire est 50%,). L'algorithme entraîne les hypothèses simples sur un ensemble de patrons pendant plusieurs itérations. D'une itération à l'autre, AdaBoost met plus d'emphase sur les patrons plus difficiles à apprendre. La différence principale entre ces deux versions d'AdaBoost repose sur la manière dont elles procèdent pour accroître l'importance des patrons difficiles dans la procédure d'apprentissage. L'algorithme d'AdaBoost attribue, à chaque hypothèse, un poids particulier dans le calcul de l'hypothèse finale du comité. L'hypothèse finale est ainsi une somme pondérée des hypothèses simples.

---

[2] « Committee Machines » en anglais

[3] L'idée originale de Boosting appartient à Schapire présentée dans (Schapire, 1990). Freund et Schapire ont proposé la version initiale d'AdaBoost dans (Freund, 1996a) et (Freund, 1996b) et l'ont ensuite développée dans (Freund, 1997). Schapire et Singer ont présenté récemment la version améliorée d'AdaBoost dans (Schapire, 1998).

[4] Weak hypothesis ou WeakLearn. Le terme « hypothèse » est équivalent au terme « classificateur ».

Nous proposons, pour la première fois, l'utilisation de l'algorithme d'AdaBoost pour résoudre les problèmes de classification causés par le nombre limité de patrons disponibles pour l'apprentissage. Nous allons montrer que les classificateurs basés sur l'algorithme d'AdaBoost sont de parfaits candidats pour la résolution de ce type de problèmes. Le fait de subdiviser le classificateur final en plusieurs classificateurs simples permet de les entraîner rapidement avec un petit nombre de patrons d'apprentissage. Comme l'équation (4.20) le montre, dans le cas d'un classificateur neuronal de type Perceptron multicouches, le nombre de patrons nécessaires pour l'entraînement dépend directement du nombre de paramètres libres. Par conséquent, l'utilisation d'un réseau très simple en structure, jouant le rôle d'une hypothèse simple dans l'algorithme d'AdaBoost, permet de réduire considérablement le nombre de patrons nécessaires pour l'entraîner. En plus, si chaque patron d'apprentissage est composé de plus d'une caractéristique, on peut utiliser un ou plusieurs classificateurs simples comme expert d'une caractéristique donnée. Cette capacité intéressante des classificateurs AdaBoost nous permet de présenter plusieurs caractéristiques à un classificateur sans avoir besoin d'un grand nombre de patrons d'apprentissage; ce qui est impossible avec les classificateurs neuronaux classiques (voir la section 4.5.2.3.2 et le tableau 4.6).

Nous continuons le chapitre avec la description détaillée des deux versions de l'algorithme d'AdaBoost dans les sections 5.1.1 et 5.1.2.

## 5.1.1 VERSION INITIALE D'ADABOOST

Dans cette section, nous présenterons d'abord la version initiale d'AdaBoost (appelée AdaBoost.M1 dans (Freund, 1996a)). Nous nous pencherons ensuite sur une interprétation bayésienne d'AdaBoost.M1 permettant de mieux comprendre le fonctionnement de l'algorithme.

**Algorithme AdaBoost.M1**
/* La version initiale de l'algorithme d'AdaBoost présentée dans (Freund, 1996a). */

**Entrées :** - Ensemble de $m$ patrons d'entraînement $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
avec des étiquettes $y_i \in Y = \{1, \ldots, k\}$
- Algorithme d'entraînement simple **WeakLearn**
- Entier $T$, le nombre d'itérations

**Initialiser** $D_1(i) = 1 / m$ pour tous les $i = 1, \ldots, m$ /* pour tous les patrons */
**Répéter pour** $t = 1, \ldots, T$ /* itérations */

1. Appeler **WeakLearn** en respectant la distribution $D_t$.
2. Obtenir une hypothèse $h_t : X \rightarrow Y$.
3. Calculer l'erreur de $h_t$ : $\varepsilon_t = \sum_{i:h_t(x) \neq y_i} D_t(i)$
4. **Si** $\varepsilon_t > \frac{1}{2}$, **alors** affecter $T = t - 1$ et terminer les itérations.
5. Calculer $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.

/* Mise à jour de la distribution $D_t$ */
6. **Répéter pour** $i = 1, \ldots, m$

**Si** $h_t(x_i) = y_i$, **alors** $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \beta_t$ /* $Z_t$ est une constante */
/* de normalisation utilisée */
/* pour que $D_{t+1}$ soit */
/* une distribution. */

**Sinon** $D_{t+1}(i) = \frac{D_t(i)}{Z_t}$

**Sortie :** L'hypothèse finale :

$$h_{fin}(x) = \arg \max_{y \in Y} \sum_{i:h_t(x)=y} \log \frac{1}{\beta_t}$$

Figure 5.1. Version initiale de l'algorithme AdaBoost (d'après (Freund, 1996a)).

## 5.1.1.1. DESCRIPTION DE L'ALGORITHME

Cet algorithme est montré dans la figure 5.1. L'algorithme a trois entrées : (1) un ensemble de $m$ patrons d'entraînement; (2) un algorithme d'entraînement simple WeakLearn et (3) un entier $T$ représentant le nombre maximal d'itérations.

L'ensemble de patrons d'entraînement $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ est composé des paires $(x_i, y_i)$, où $x_i \in X$ et $y_i \in Y$, représentent un vecteur de caractéristiques et la

classe associée à ce vecteur, respectivement. La seconde entrée, l'algorithme d'entraînement simple WeakLearn, sert à générer des hypothèses (classificateurs) $h_t$. La seule exigence de cet algorithme est qu'il doit classifier les patrons de façon plus précise qu'un classificateur aléatoire. L'objectif de WeakLearn consiste à minimiser l'erreur d'entraînement des hypothèses simples. La troisième entrée d'AdaBoost.M1 est le nombre maximal d'itérations $T$.

L'algorithme commence par l'initialisation de $D_t$ à une distribution uniforme. La distribution $D_t$ ne correspond pas à la probabilité de présence des patrons dans l'ensemble d'entraînement, mais plutôt à l'importance que la procédure itérative d'entraînement doit leur attribuer en fonction de leur degré de difficulté de classification. Au début, tous les patrons ont une importance identique, la $D_t$ est donc une distribution uniforme. L'algorithme se poursuit en appelant l'algorithme d'apprentissage WeakLearn auquel l'ensemble des patrons d'entraînement est présenté en respectant la distribution $D_t$. Pour la première itération, tous les patrons dans $S$ seront présentés uniformément au WeakLearn. Le résultat de l'entraînement est une hypothèse $h_t$. L'étape suivante consiste à calculer $\varepsilon_t$, l'erreur de classification de $h_t$, toujours en respectant la distribution $D_t$ :

$$\varepsilon_t = \sum_{i:h_t(x)\neq y_i} D_t(i) \tag{5.1}$$

Si cette erreur dépasse l'erreur d'un classificateur aléatoire (50%), l'hypothèse $h_t$ sera écartée et les itérations se termineront. Autrement, l'algorithme se poursuit avec le calcul du coefficient $\beta_t$ en fonction de la $\varepsilon_t$ :

$$\beta_t = \varepsilon_t / (1 - \varepsilon_t) \tag{5.2}$$

Étant donné que $0 \leq \varepsilon_t < 0.5$, nous avons $0 \leq \beta_t < 1$. Ce coefficient sert à la mise à jour de la distribution $D_{t+1}$ (qui doit être utilisée dans l'itération suivante) en fonction de la $D_t$ :

$$\text{Si } h_t(x_i) = y_i, \text{ alors } D_{t+1}(i) = \frac{D_t(i)}{Z_t}\beta_t ;$$

$$\text{Sinon } D_{t+1}(i) = 1. \tag{5.3}$$

$Z_t$ dans l'équation (5.3) est un facteur de normalisation utilisé pour que la $D_{t+1}$ devienne une distribution statistique. Il peut ainsi prendre comme valeur, la somme de $m$ éléments du vecteur $D_t$.

La génération des hypothèses $h_t$ et des coefficients $\beta_t$ continue pour $T$ itérations ou jusqu'à ce que l'erreur $\varepsilon_t$ d'une hypothèse dépasse 0.5. Le résultat de ces itérations est un ensemble de paires ($h_t$, $\beta_t$) servant à calculer l'hypothèse finale $h_{fin}$.

Selon l'algorithme de la figure 5.1, pour le calcul de $D_{t+1}$, si le $i^{ème}$ patron est classifié incorrectement, sa valeur associée dans la distribution $D_t$ est multipliée par 1, autrement par un coefficient $\beta_t$ qui est inférieur à 1 (dans tous les cas, le résultat est normalisé sur $Z_t$). Par conséquent, les patrons classifiés correctement, perdent de plus en plus de poids dans la distribution $D_{t+1}$, tandis que les patrons mal classifiés y seront plus présents. Cela permet à la procédure d'entraînement de mettre plus d'emphase, à l'itération $t+1$, sur les patrons difficiles à classifier.

L'hypothèse finale est obtenue en fonction des votes pondérés des hypothèses simples, à partir de l'équation suivante, :

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t} \tag{5.4}$$

Un patron donné $x$ est donc classifié dans une classe $y$ si la somme des poids des hypothèses qui l'ont classifié dans la classe $y$ est supérieure à celles des hypothèses qui l'ont classifié dans la(les) autre(s) classe(s). Le poids de chaque hypothèse est défini comme $\log(1/\beta_t)$; ce qui permet de donner plus de poids aux hypothèses ayant des erreurs de classification moins élevées.

Afin de respecter la distribution $D_t$ lors de la présentation des patrons à WeakLearn, on peut utiliser la technique de ré-échantillonnage qui permet de modifier la probabilité de la présence des patrons dans l'ensemble d'apprentissage en fonction de leur valeur correspondante dans la distribution $D_t$. Nous présenterons dans la section 5.2.1.3.1, l'algorithme que nous avons implanté pour le ré-échantillonnage des patrons, en respectant une distribution donnée.

Pour que l'algorithme AdaBoost.M1 fonctionne correctement, l'erreur de classification des hypothèses simples ne doit pas dépasser 50%. Dans le cas d'une classification binaire, cette condition est facile à respecter, puisqu'il suffit que les hypothèses soient légèrement plus précises qu'une simple hypothèse aléatoire. Tandis que si le nombre de classes est supérieur à 2, la condition de $\varepsilon_t < 0.5$ devient difficile à satisfaire, puisque l'erreur attendue d'une hypothèse aléatoire est égale à 1-(1/$k$), où $k$ est le nombre total des classes. (Freund, 1996a) propose un autre algorithme (AdaBoost.M2) pour le problème de classification avec $k > 2$. Étant donné que dans notre projet, nous affrontons un problème de classification binaire, nous nous contentons d'AdaBoost.M1. Toutefois, dans la section 5.2.1.3.1, nous expliquerons comment nous avons réussi à améliorer AdaBoost.M1 pour la classification des distances inter-tracés dans notre système, en utilisant un des aspects abordés dans le AdaBoost.M2.

### 5.1.1.2. INTERPRÉTATION BAYÉSIENNE D'ADABOOST.M1

On peut montrer la relation entre l'hypothèse finale d'AdaBoost.M1 (équation 5.4) et celle que l'on obtient avec une analyse bayésienne du problème. Dans cette section, à l'aide de la règle de Bayes, nous chercherons à trouver la meilleure façon de combiner les prédictions faites par un ensemble d'hypothèses binaires $h_1,...,h_T$ classifiant les patrons dans une des classes de l'ensemble {0,1}. Pour un patron donné $x$, l'hypothèse finale doit prédire 1, si l'expression suivante est vraie :

$$\Pr[y = 1 \mid h_1(x),..., h_T(x)] > \Pr[y = 0 \mid h_1(x),..., h_T(x)] \qquad (5.5)$$

Selon la règle de Bayes, l'équation (5.5) peut être écrite de la façon suivante :

$$\frac{\Pr[h_1(x),..., h_T(x) \mid y = 1] * \Pr[y = 1]}{\Pr[h_1(x),..., h_T(x)]} > \frac{\Pr[h_1(x),..., h_T(x) \mid y = 0] * \Pr[y = 0]}{\Pr[h_1(x),..., h_T(x)]} \qquad (5.6)$$

Si les prédictions des différentes hypothèses sont indépendantes, nous avons :

$$\Pr[y=1]\Pr[h_1(x)\mid y=1]\ldots\Pr[h_T(x)\mid y=1] \;>\; \Pr[y=0]\Pr[h_1(x)\mid y=0]\ldots\Pr[h_T(x)\mid y=0] \qquad (5.7)$$

Chaque terme de l'équation (5.7) peut être remplacé par une expression en fonction de $\varepsilon_t$ (équation (5.1)) qui est l'erreur de classification de l'hypothèse correspondante $h_t$. Par exemple :

$$\Pr[h_t(x)=y_t \mid y=1]=\begin{cases}\varepsilon_t, & si\ h_t(x)=0\\ 1-\varepsilon_t, & si\ h_t(x)=1\end{cases} \qquad (5.8)$$

En remplaçant (5.8) dans (5.7), on obtient la condition bayésienne suivante :

$$\Pr(y=1) * \prod_{t:h_t(x)=0}\varepsilon_t * \prod_{t:h_t(x)=1}(1-\varepsilon_t) \;>\; \Pr(y=0) * \prod_{t:h_t(x)=1}\varepsilon_t * \prod_{t:h_t(x)=0}(1-\varepsilon_t) \qquad (5.9)$$

Le terme $\Pr(y=0)$ est égal à $\varepsilon_0$, l'erreur d'une hypothèse $h_0$ prédisant toujours 1. En introduisant $\varepsilon_0$ dans l'équation (5.9), nous avons :

$$(1-\varepsilon_0) * \prod_{t:h_t(x)=0}\varepsilon_t * \prod_{t:h_t(x)=1}(1-\varepsilon_t) \;>\; \varepsilon_0 * \prod_{t:h_t(x)=1}\varepsilon_t * \prod_{t:h_t(x)=0}(1-\varepsilon_t) \qquad (5.10)$$

Dans cette équation, les termes $\varepsilon_0$ et $(1-\varepsilon_0)$ peuvent être absorbés dans les produits correspondants aux hypothèses qui prédisent 1. L'équation (5.10) sera ainsi réduite à l'équation suivante :

$$\prod_{t:h_t(x)=1}\frac{1-\varepsilon_t}{\varepsilon_t} \;>\; \prod_{t:h_t(x)=0}\frac{1-\varepsilon_t}{\varepsilon_t} \qquad (5.11)$$

En remplaçant (5.2) dans (5.10) et en prenant le logarithme des deux côtés de l'équation, nous obtiendrons :

$$\sum_{t:h_t(x)=1}\log(\frac{1}{\beta_t}) \;>\; \sum_{t:h_t(x)=0}\log(\frac{1}{\beta_t}) \qquad (5.12)$$

Si l'expression (5.12) est vraie, le patron $x$ sera classifié dans la classe 1, autrement dans la classe 0. On peut facilement constater que cette équation est équivalente à l'équation (5.4) représentant l'hypothèse finale de l'algorithme AdaBoost.M1.

## 5.1.2 ADABOOST AMÉLIORÉ

(Schapire, 1998) a apporté certaines modifications dans la version initiale d'AdaBoost. Nous aborderons dans cette section, la version améliorée d'AdaBoost, présentée dans l'article ci-dessus. Les modifications sont basées sur une version d'AdaBoost, montrée dans la figure 5.2. Cette version est appelée générale, puisqu'elle ne spécifie pas explicitement ni le critère selon lequel on entraîne les hypothèses simples ni la manière de calculer $\alpha_t$ qui est un paramètre permettant d'augmenter ou de diminuer le poids des patrons d'apprentissage, classifiés incorrectement ou correctement par une hypothèse $h_t$. Toutefois, à ce sujet, (Schapire, 1998) fait quelques propositions qui seront présentées plus loin dans cette section.

Détaillons d'abord l'algorithme de la figure 5.2. Cet algorithme a les mêmes entrées que l'algorithme AdaBoost.M1, mais les étiquettes des patrons sont binaires : il existe deux classes différentes, présentées par $-1$ et $+1$. Contrairement au AdaBoost.M1, les hypothèses peuvent produire des sorties sur l'étendue de $\Re$ au lieu de [-1, +1]. Le $\alpha_t$ dans la figure 5.2 joue un rôle semblable à celui de $\beta_t$ dans AdaBoost.M1, représentant le poids de l'hypothèse $h_t$ dans le calcul de l'hypothèse finale. Toutefois $\alpha_t$ et $\beta_t$ varient dans deux directions différentes: lorsque la précision de classification de $h_t$ augmente, $\beta_t$ diminue et $\alpha_t$ augmente.

L'erreur d'apprentissage est calculée selon l'équation suivante :

$$E_{app} = \frac{1}{m}\sum_i \| H(x_i) \neq y_i \| \qquad (5.13)$$

**Algorithme AdaBoost général**
/* La version générale de l'algorithme AdaBoost présentée dans (Schapire, 1998). */

**Entrées :** - Ensemble de $m$ patrons d'entraînement $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
avec des étiquettes $y_i \in Y = \{-1,+1\}$
- Algorithme d'entraînement simple **WeakLearn**
- Entier $T$, le nombre d'itérations

**Initialiser** $D_1(i) = 1 / m$ pour tous les $i = 1, \ldots, m$
**Répéter pour** $t = 1, \ldots, T$
1. Appeler **WeakLearn** en respectant la distribution $D_t$.
2. Obtenir une hypothèse $h_t: X \rightarrow \Re$.
3. Calculer $\alpha_t \in \Re$.

/* Mise à jour de la distribution $D_t$ */
4. **Répéter pour** $i = 1,\ldots,m$

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_t h_t(x_t))}{Z_t}$$

/* $Z_t$ est une constante */
/* de normalisation utilisée */
/* pour que $D_{t+1}$ soit */
/* une distribution. */

**Sortie :** L'hypothèse finale :

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

Figure 5.2. Version générale d'AdaBoost (d'après (Schapire, 1998)).

où $\|\pi\|$ vaut 1 si $\pi$ est vrai, autrement 0. On peut montrer que l'expression suivante donne la valeur maximale de $E_{app}$ :

$$Max(E_{app}) = \prod_{t=1}^{T} Z_t \tag{5.14}$$

Selon l'équation (5.14), pour minimiser l'erreur d'apprentissage, il faut minimiser $Z_t$ à chaque itération. On peut montrer que cette équation est déduite à partir de la règle de mise à jour de $D_t$ dans l'algorithme de la figure 5.2. Pour ce faire, nous écrivons

l'expression itérative de la dernière mise à jour de $D_t(i)$, en fonction des paramètres des itérations précédentes :

$$D_{T+1}(i) = \frac{\exp(-\sum_t \alpha_t y_i h_t(x_i))}{m\prod_t Z_t}$$

$$= \frac{\exp(-y_i f(x_i))}{m\prod_t Z_t} \tag{5.15}$$

où : $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$. Une erreur de classification se produit lorsque $H(x_i) \neq y_i$, autrement dit, si $y_i f(x_i) \leq 0$. Dans ce cas, l'expression suivante est une expression correcte:

$$\exp(-y_i f(x_i)) \geq 1 \tag{5.16}$$

Étant donné que dans le cas d'erreur, l'expression $H(x_i) \neq y_i$ est vraie, nous avons :

$$\|H(x_i) \neq y_i\| \leq \exp(-y_i f(x_i)) \tag{5.17}$$

et

$$\frac{1}{m}\sum_i \|H(x_i) \neq y_i\| \leq \frac{1}{m}\sum_i \exp(-y_i f(x_i)) \tag{5.18}$$

Et finalement, en combinant (5.15) et (5.18), l'équation suivante sera déduite :

$$\frac{1}{m}\sum_i \|H(x_i) \neq y_i\| \leq \sum_i (\prod_t Z_t)\, D_{T+1}(i) \tag{5.19}$$

Puisque $Z_t$ est indépendante de paramètre $i$ et $D_{T+1}$ est une distribution statistique, nous avons :

$$\frac{1}{m}\sum_i \|H(x_i) \neq y_i\| \leq \prod_t Z_t \tag{5.20}$$

L'équation (5.20) est équivalente à l'équation (5.14).

La minimisation de $Z_t$ devient ainsi le critère selon lequel il faut déterminer $\alpha_t$ et entraîner $h_t$. Comme nous l'avons mentionné dans l'algorithme de la figure 5.2, $Z_t$ est un

facteur de normalisation devant être choisie de manière que $D_{t+1}$ devienne une distribution statistique. On peut alors calculer $Z_t$ à partir de l'équation suivante :

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i)) \tag{5.21}$$

Pour les hypothèses simples $h_t$ dont les sorties sont entre (-1, +1), la valeur de $\alpha_t$ qui minimise $Z_t$ est calculée selon l'équation suivante (Schapire, 1998) :

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1+r_t}{1-r_t}\right) \tag{5.22}$$

où :

$$r_t = \sum_{i=1}^{m} D_t(i) y_i h_t(x_i) \tag{5.23}$$

L'augmentation de l'erreur d'apprentissage dans une itération provoque la diminution de $r_t$ et de $\alpha_t$. En fait, c'est le signe de $h_t(x_i)$ qui permet de déterminer la classe de $x_i$. Par conséquent, l'augmentation du nombre de patrons classifiés incorrectement cause une augmentation du nombre de patrons pour lesquels le produit $y_i h_t(x_i)$ est négatif; ce qui fait diminuer $r_t$ (selon l'équation (5.23)) et $\alpha_t$ (selon l'équation (5.22)) ($\alpha_t$ est un nombre positif sans borne supérieure ($\alpha_t > 0$)). Les équations (5.22) et (5.23) sont utilisables dans notre système, puisque, comme nous les présenterons dans la section 5.2.2, les hypothèses simples de notre classificateur AdaBoost sont binaires et produisent des sorties entre (-1, +1).

Dans la section 5.1.1 nous avons mentionné que l'objectif des algorithmes WeakLearn consistait à produire des hypothèses simples $h_t$ avec un nombre minimum d'erreurs de classification en respectant une distribution donnée $D_t$. Toutefois, nous avons montré que dans la version améliorée d'AdaBoost, un autre critère est proposé pour l'entraînement des hypothèses simples : minimisation de $Z_t$. L'expression de $Z_t$ est donnée dans l'équation (5.21). Il est possible de simplifier cette expression en

insérant $\alpha_t$ dans $h_t$,, puisque pour un $t$ donné, $\alpha_t$ est constante. On peut donc conclure que l'objectif des WeakLearn consiste à minimiser l'expression suivante :

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-y_i h_t(x_i)) \tag{5.24}$$

Les équations (5.24) montre que $Z_t$ , tout comme $\varepsilon_t$ (qui est calculé selon l'équation (5.1) dans l'algorithme AdaBoost.M1), est une fonction de l'erreur de classification des patrons d'apprentissage. $Z_t$ et $\varepsilon_t$ sont toutefois différents dans les deux aspects suivants : (1) $\varepsilon_t$ est calculé uniquement sur les patrons classifiés incorrectement, alors que $Z_t$ tient compte de tous les patrons d'apprentissage; (2) $Z_t$ est une somme pondérée de la distribution $D_t$, tandis que $\varepsilon_t$ est une somme non pondérée de cette distribution; (3) $Z_t$, contrairement à $\varepsilon_t$ , représente non seulement l'erreur de classification, mais aussi la «confiance» de classification effectuée par l'hypothèse $h_t$. En fait, le signe de $h_t(x_i)$, $sign(h_t(x_i))$, indique la classe du patron $x_i$ prédit par l'hypothèse $h_t$, alors que $|h_t(x_i)|$ représente la confiance de cette prédiction. Une grande amplitude pour $|h_t(x_i)|$ est interprétée comme une confiance élevée de classification de $x_i$, puisqu'elle montre que la sortie est bien loin du seuil (valeur de 0) utilisé pour discriminer les classes. Dans le calcul de $\varepsilon_t$ selon l'équation (5.1), on se sert que du signe des $h_t(x_i)$ pour les patrons classifiés incorrectement, sans tenir compte de la confiance de classification. Ces observations montrent que $Z_t$ produit une estimation plus précise de la performance de $h_t$ que $\varepsilon_t$. Nous utiliserons, dans les sections suivantes, les résultats empiriques des classifications effectuées par les deux versions d'AdaBoost (avec $Z_t$ et $\varepsilon_t$), pour comparer leur capacité de classification des distances inter-tracés.

# 5.2. CLASSIFICATEURS ADABOOST IMPLANTÉS

Après avoir expliqué les aspects théoriques de l'algorithme d'apprentissage AdaBoost, nous présenterons dans cette section, les classificateurs AdaBoost implantés en mettant l'accent sur leurs entrées, leur structure, leurs phases d'apprentissage et de test ainsi que les résultats obtenus pour la classification des distances inter-tracés. Dans les sections 5.2.1 et 5.2.2, nous expliquerons les classificateurs basés sur les versions initiale et améliorée d'AdaBoost, respectivement. Dans la section 5.2.3, nous analyserons les résultats obtenus, incluant une comparaison globale de la performance de tous les classificateurs neuronaux et AdaBoost implantés, ainsi qu'une étude sur les paramètres et les cas d'erreur du meilleur classificateur AdaBoost.

## 5.2.1 CLASSIFICATEURS BASÉS SUR LA VERSION INITIALE D'ADABOOST

Nous présenterons dans cette section, les classificateurs de distances inter-tracés implantés, basés principalement sur la version initiale d'AdaBoost. Au début de la section, nous introduirons les entrées des classificateurs qui sont les caractéristiques extraites des textes manuscrits. Nous présenterons également la structure des classificateurs implantés. Les algorithmes que nous avons utilisés pour l'apprentissage des classificateurs sont, sous quelques aspects, différents de celui de la figure 5.1. Nous avons également appliqué les scénarios commun et personnalisé[5] aux classificateurs AdaBoost. Ces scénarios seront expliqués ainsi que les résultats de classification.

### 5.2.1.1 ENTRÉES DES CLASSIFICATEURS

Les résultats obtenus avec les classificateurs neuronaux de distances inter-tracés, présentés dans le chapitre 4, montrent que, parmi les sept caractéristiques extraites de

[5] Voir la section 4.4.2.3.1 pour la définition de ces termes.

textes manuscrits, les trois mesures de distances produisent les meilleurs résultats de classification. Il s'agit de Dist_Paral, RLmin et RLavg. Compte tenu du nombre très élevé de combinaisons possibles des sept caractéristiques et à la lumière de ces résultats, nous n'avons utilisé, dans un premier temps, que ces trois mesures de distance comme les entrées de classificateurs AdaBoost. Nous devons ajouter que ces mesures sont les caractéristiques proposées dans le chapitre 3 : elles sont extraites après la phase de la reconnaissance des Accent_Point_TBar et la distance entre les parallélogrammes des tracés (Dist_Paral) remplace la distance entre les rectangles (Dist_BB).

En plus des combinaisons des trois mesures de distances inter-tracés, nous avons implanté des classificateurs qui reçoivent à l'entrée les sept caractéristiques et cela pour observer l'effet des quatre autres caractéristiques extraites, sur les résultats de classification. Nous rappelons que ces caractéristiques sont les suivantes : prHghtExtrema, curHghtExtrema, prNbExtrema et curNbExtrema. prHghtExtrema et curHghtExtrema sont les distances moyennes entre les extrema consécutifs pour le mot précédent et le tracé actuel, respectivement. prNbExtrema est le nombre d'extrema du mot précédent et curNbExtrema est le nombre d'extrema du tracé actuel (voir la figure 4.10).

Dans la section 5.2.1.4, nous présenterons différentes combinaisons des caractéristiques utilisées comme entrées de la première catégorie de nos classificateurs AdaBoost.

### 5.2.1.2 STRUCTURE DES CLASSIFICATEURS

Comme l'algorithme de la figure 5.1 le montre, l'implantation d'un classificateur AdaBoost requiert des hypothèses ou des classificateurs simples ainsi qu'un algorithme d'apprentissage WeakLearn. Pour la tâche de classification des distances inter-tracés, nous avions utilisé, des classificateurs neuronaux de type Perceptron à une couche ou multicouche entraînés par l'algorithme d'apprentissage de rétro-propagation d'erreur. Pour implanter nos classificateurs AdaBoost, nous avons choisi la structure la plus

simple des Perceptrons, soit un seul neurone avec une entrée et une sortie. Chaque neurone joue donc le rôle d'une hypothèse simple. La rétro-propagation d'erreur est utilisée comme l'algorithme d'apprentissage WeakLearn.

La figure 5.3 montre la structure générique de nos classificateurs inspirés de la version initiale d'AdaBoost. Dans cette figure, $x^1,...,\ x^N$ représentent les $N$ caractéristiques d'un patron $x$. Chaque caractéristique est présentée à une banque d'hypothèses simples qui agissent comme expertes de cette caractéristique. Ces hypothèses ainsi que leurs $\beta$ associés sont les résultats de l'algorithme AdaBoost. Chaque hypothèse génère en fonction de la valeur de l'entrée, une sortie dans l'intervalle (-1,+1). La classe du patron à l'entrée du classificateur est déterminée à partir de la valeur de sortie. Les sorties supérieures et inférieures à 0 représentent respectivement les distances inter-mots et intra-mot. La sortie de chaque classificateur est l'entrée de deux blocs, produisant des sorties binaires {0, 1} en fonction de la classe prédite. La sortie du module *«intra»* sera 1, si la classe prédite est intra-mot; la sortie du module *«inter»* sera alors 0. Par contre, une classe prédite inter-mot permet de produire 1 à la sortie «inter» et 0 à la sortie «intra». Ces valeurs sont ensuite multipliées par $log(1/\beta)$ de l'hypothèse correspondante qui représente son poids dans l'ensemble des hypothèses. Étant donnée qu'une des sorties des blocs « intra » et « inter » vaut 0 et l'autre vaut 1, le résultat d'une des pondérations est 0 et celui de l'autre est $log(1/\beta)$. Les votes intra-mot pondérés sont ensuite additionnés. La somme des votes inter-mots pondérés est également calculée. Parmi les votes intra et inter-mots pondérés, celui qui est le plus grand détermine le résultat final de classification.

Comme nous l'avons mentionné précédemment, la figure 5.3 représente le schéma générique des classificateurs AdaBoost. Dans nos classificateurs spécifiques, les caractéristiques utilisées à l'entrée varient d'un classificateur à l'autre. Le nombre d'hypothèses générées pour chaque caractéristique, est également déterminé pendant la phase d'apprentissage. Cette phase sera expliquée dans la section suivante et les classificateurs AdaBoost spécifiques seront présentés dans la section 5.2.1.4, lors de la présentation des résultats de classification.

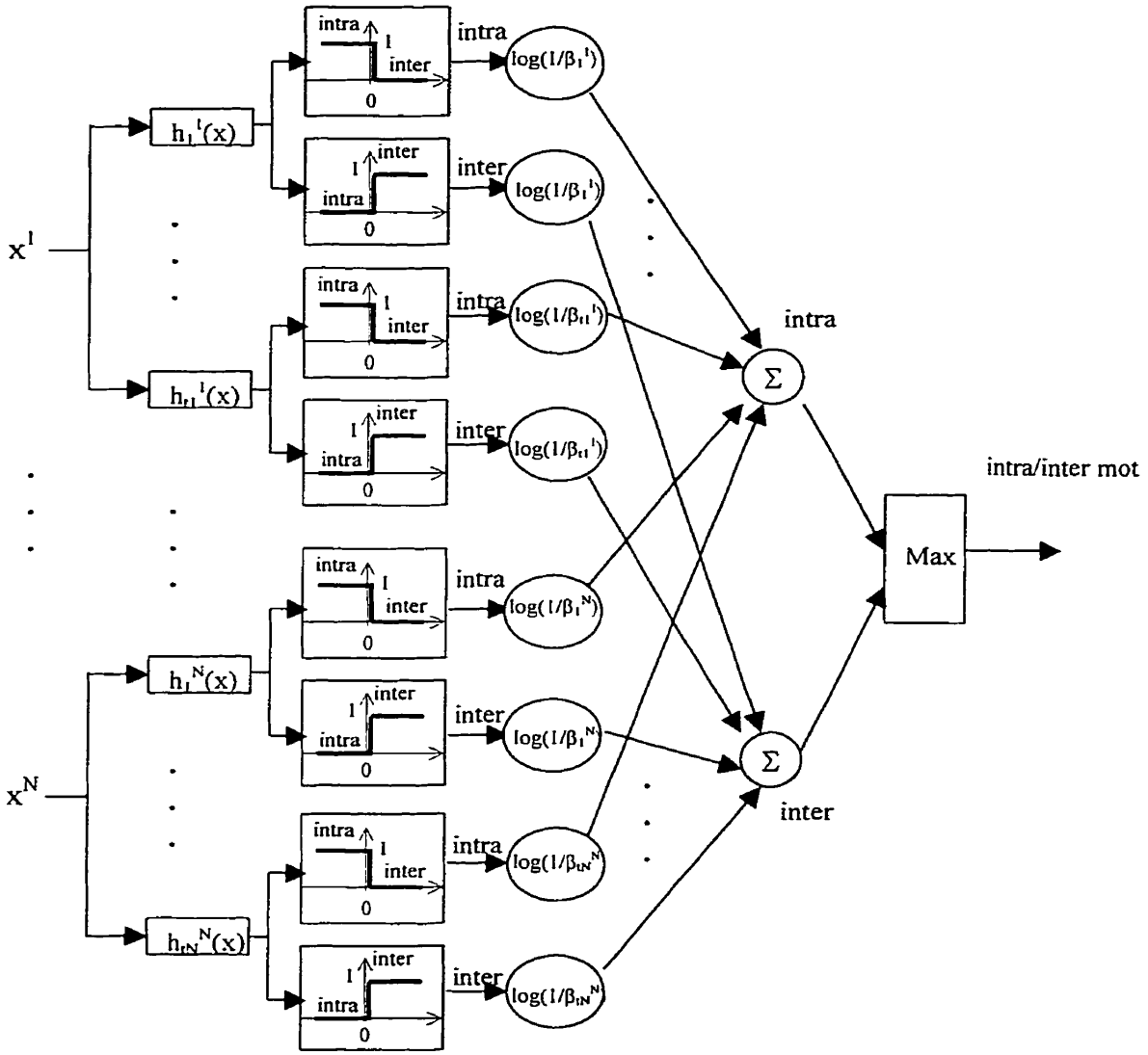



Figure 5.3. Structure générique des classificateurs basés sur la version initiale d'AdaBoost.

### 5.2.1.3 APPRENTISSAGE ET TEST DES CLASSIFICATEURS

La phase d'apprentissage de nos classificateurs AdaBoost est basée sur l'algorithme de la figure 5.1. Notre algorithme est toutefois différent, sous certains

aspects qui seront abordés dans cette section, de l'algorithme générique. Nous présenterons également les deux scénarios commun et personnalisé d'apprentissage et de validation des classificateurs.

#### *5.2.1.3.1 ALGORITHME D'APPRENTISSAGE IMPLANTÉ*

La figure 5.4 montre l'algorithme que nous avons implanté pour l'apprentissage de nos classificateurs AdaBoost.M1.

À l'entrée, l'algorithme dispose d'un ensemble de $m$ patrons $S^0$, de **BackProp** comme l'algorithme d'entraînement simple (WeakLearn) dans AdaBoost.M1, de l'algorithme de ré-échantillonnage **ReSampel** et de l'entier $T$ servant à déterminer le nombre maximal d'itérations.

Au début, la distribution $D$, qui représente l'importance de chacun des patrons dans l'algorithme d'apprentissage, est initialisée à une distribution uniforme. La partie principale de l'algorithme est formée de deux boucles imbriquées permettant de générer une ou plusieurs hypothèses (expertes) pour chacune de $N$ caractéristiques des patrons d'entraînement. Comme on le verra plus loin, il est toutefois possible qu'aucune hypothèse ne soit obtenue pour une caractéristique donnée. Le nombre d'experts pour chaque caractéristique, peut donc varier entre 0 et $T$. Autrement dit, pour $N$ caractéristiques et $T$ itérations, le nombre totale d'hypothèses générées est entre 0 et $N*T$. Pour la $n^{ième}$ caractéristique, l'algorithme calcule les numéros de la première et de la dernière itération ($T_{min}^n$, $T_{max}^n$) en fonction du numéro de la dernière itération effectuée pour la dernière caractéristique ($T_{max}^n$). On lui prévoit ainsi $T$ itérations au maximum ($T_{max}^n - T_{min}^n = T$).

$T_{min}^n$ et $T_{max}^n$ servent à construire la boucle de calcul des hypothèses et de leur $\beta$. La première étape dans cette boucle consiste à ré-échantillonner l'ensemble des patrons pour que l'ensemble résultant ait la distribution $D^t$. Pour ce faire, on utilise l'algorithme ReSampel dont les entrées sont l'ensemble des patrons de l'itération précédente $S^{t-1}$ ainsi que la distribution $D^t$. La sortie de l'algorithme est l'ensemble $S^t$ (Cet algorithme

**Algorithme Entraîne_AdaBoost.M1**

/* Entraînement d'un classificateur AdaBoost.M1 pour classifier les distances inter-tracés */

**Entrées :** - Ensemble de $m$ patrons d'entraînement $S^0 = \{(x_1^0, y_1^0), \ldots, (x_m^0, y_m^0)\}$

où : $x_i^0 = \{x_{i1}^0, x_{i2}^0, \ldots, x_{iN}^0\}$ est le vecteur de $N$ caractéristiques
et $y_i^0 \in Y = \{-1,+1\}$ est la classe du patron (-1 : intra-mot, 1 : inter-mots)

- Algorithme d'entraînement **BackProp**
- Algorithme de ré-échantillonnage **ReSampel**
- Entier $T$, le nombre maximal d'itérations

**Initialiser** $D^1(i) = 1 / m$ pour tous les $i = 1, \ldots, m$ /* Distribution uniforme */
**Initialiser** $T_{max}^0 = 0$.
**Répéter pour** n = 1 : $N$ /* Pour toutes les caractéristiques */
**Initialiser** $T_{min}^n = T_{max}^{n-1} + 1$ et $T_{max}^n = T_{max}^{n-1} + T$
**Répéter pour** $t = T_{min}^n, \ldots, T_{max}^n$ /* $T$ itérations au maximum */

1. Ré-échantillonner les patrons d'entraînement en respectant la distribution $D^t$ :
   $S^t = \textbf{ReSampel}(S^{t-1}, D^t)$;
2. Extraire les vecteur de $n^{ième}$ caractéristique $X_n^t$ et des étiquettes $Y^t$ à partir de $S^t$:
   $X_n^t = \{x_{1n}^t, \ldots, x_{mn}^t\}$ et $Y^t = \{y_1^t, \ldots, y_m^t\}$
3. Construire le sous-ensemble d'entraînement $S_n^t$ à partir de $X_n^t$ et $Y^t$ :
   $S_n^t = \{(x_{1n}^t, y_1^t), \ldots, (x_{mn}^t, y_m^t)\}$
4. Entraîner le réseau neuronal avec l'algorithme **BackProp** appliqué sur $S_n^t$.
5. Obtenir une hypothèse $h_n^t : X_n^t \rightarrow (-1, 1)$.
6. Calculer l'erreur de $h_n^t$ : $\varepsilon_n^t = 0.5 * \sum_{j=1}^{m} D^t(j) \left| h_n^t(x_{jn}^t) - y_j^t \right|$
7. **Si** $\varepsilon_n^t > \frac{1}{2}$, **alors** affecter $T_{max}^n = t - 1$ /* Erreur supérieure à celle */
   /* d'une hypothèse aléatoire */
   terminer les itérations.
8. Calculer $\beta_n^t = \varepsilon_n^t / (1 - \varepsilon_n^t)$.
9. **Si** $\varepsilon_n^t = 0$, **alors** affecter $T_{max}^n = t$ /* Classification parfaite */
   terminer les itérations.
10. **Répéter pour** $j = 1,\ldots,m$ /* Mise à jour de la distribution $D^t$ */
    **Si** $h_n^t(x_{jn}^t) = y_j^t$, **alors** $D^{t+1}(j) = D^t(j)\beta_n^t$
    **Sinon** $D^{t+1}(j) = 1$

/* Normalisation de $D^{t+1}$ pour qu'elle devienne une distribution. */

11. **Répéter pour** i = 1,...,m

$$D^{t+1}(i) = \frac{D^{t+1}(i)}{\sum_{j=1}^{m} D^{t+1}(j)}$$

Figure 5.4. Algorithme d'entraînement des classificateurs AdaBoost.M1 implantés pour la classification des distances inter-tracés.

sera détaillé plus loin dans cette section). Après la construction de $S^t$, il faut en extraire le sous-ensemble d'entraînement $S_n^t$ qui est formé des vecteurs $X_n^t$ et $Y^t$. $X_n^t$ contient la $n^{ième}$ caractéristique des patrons dans $S^t$ et $Y^t$ est le vecteur des étiquettes (des classes) des éléments de $X_n^t$.

L'étape suivante consiste à entraîner le classificateur neuronal formé d'un neurone avec une entrée et une sortie. Pour ce faire, nous utiliserons l'algorithme de rétro-propagation d'erreur (BackProp) appliqué sur le sous-ensemble d'entraînement $S_n^t$. Les paramètres utilisés dans l'algorithme BackProp sont ceux que nous avons employés pour entraîner nos classificateurs neuronaux, mentionnés dans le chapitre 4 (section 4.5.2.3.2). Le résultat de l'entraînement est une hypothèse produisant une sortie continue dans l'intervalle (-1,1) en fonction de la valeur de caractéristique à l'entrée. À partir de la valeur de sortie, on peut déterminer la classe du patron d'entrée, en utilisant un seuil fixe de 0.

On calcule ensuite l'erreur pondérée de l'apprentissage $\varepsilon_n^t$. Contrairement à la version initiale de AdaBoost.M1, montrée dans la figure 5.1, pour obtenir $\varepsilon_n^t$ sur le sous-ensemble d'entraînement $S_n^t$, l'erreur est calculée pour tous les patrons de $S_n^t$ et non pas uniquement pour les patrons classifiés incorrectement. En plus, vu que la sortie de $h_n^t$ est continue plutôt que binaire {0,1}, on peut utiliser la confiance de classification $\left|h_n^t(.) - y^t\right|$ au lieu de l'erreur binaire, permettant ainsi d'obtenir des résultats plus précis en ce qui concerne l'erreur de classification. L'hypothèse obtenue doit être meilleure qu'une simple hypothèse aléatoire. Les itérations vont donc s'arrêter pour la caractéristique actuelle, si $\varepsilon_n^t$ dépasse 0.5. Cela impose également le rejet de la dernière hypothèse. Autrement, on calcule $\beta_n^t$ en fonction de $\varepsilon_n^t$ et les itérations vont se poursuivre.

En plus de l'introduction de la confiance de classification dans le calcul de l'erreur d'apprentissage, nous avons ajouter une étape à notre algorithme par rapport à l'algorithme original. Il s'agit d'arrêter les itérations, si la classification parfaite est atteinte. Cela se produit lorsque l'erreur de l'apprentissage $\varepsilon_n^t$ devient 0. Autrement, l'algorithme se poursuit avec la mise à jour de la distribution $D_t$, suivi de sa

normalisation afin d'obtenir une distribution $D^{t+1}$ qui sera utilisée dans l'itération suivante.

Selon l'algorithme de la figure 5.4, la distribution des patrons d'entraînement est modifiée pour une caractéristique pendant les différentes itérations. En plus, lorsque l'on change de caractéristique, cette distribution n'est pas initialisée à la distribution uniforme, mais plutôt à celle que l'on a obtenue lors de la dernière itération de la caractéristique précédente. Par conséquent, les patrons utilisés afin d'obtenir les hypothèses pour une caractéristique donnée, dépendent des résultats des itérations précédentes. Nous appelons cette façon d'entraînement « stratégie dépendante » dont l'alternative est « stratégie indépendante » où la distribution de l'ensemble des patrons est initialisée à la distribution uniforme avant la première itération pour chaque caractéristique. Nous parlerons davantage de ces stratégies d'apprentissage dans la section suivante, lors de la présentation des résultats de classification.

Les résultats de cet algorithme sont un ensemble d'hypothèses $h_n^t$ et leurs $\beta_n^t$ associés qui seront utilisés pour construire le classificateur final montré dans la figure 5.3.

Nous avons présenté l'algorithme d'apprentissage des classificateurs. Avant de terminer cette section et d'expliquer les deux versions d'apprentissage commun et personnalisé, nous détaillerons l'étape de ré-échantillonnage de l'algorithme de la figure 5.4.

**Algorithme de ré-échantillonnage des patrons d'apprentissage**

La figure 5.5 montre l'algorithme « ReSampel » qui sert à ré-échantillonner les patrons d'entraînement. Cet algorithme est basé sur une stratégie de rejet. À l'entrée, il existe un ensemble de patrons et une distribution désirée pour l'ensemble résultant. La sortie de l'algorithme est l'ensemble de patrons respectant la distribution à l'entrée. Le principe des opérations est simple. Pour obtenir un ensemble de $m$ patrons, l'algorithme utilise une boucle dont le nombre de répétitions est plus grand ou égal à $m$. À chaque

tour de boucle, deux nombres aléatoires avec la distribution uniforme sont générés et une comparaison est effectuée pour accepter ou rejeter les nombres générés.

On peut montrer que l'ensemble de patrons à la sortie $S_{out}$ aurait la distribution désirée $D$. Les étapes 1 et 2 de l'algorithme génèrent un point aléatoire dans un rectangle de largeur $m$ et de hauteur $D\_Max$. Considérons maintenant l'expression suivante :

$$P[\text{un point est généré dans une région}] = \text{la surface de la région} / (m * D\_Max) \qquad (5.25)$$

où $(m * D\_Max)$ représente la surface totale du rectangle. En plus, la probabilité de l'acceptation de $X$ est égale à la probabilité que $Y \leq D(X)$ ou que $X$ se trouve dans la région sous la courbe $D$. Selon (5.25), puisque $D$ est une distribution, cette probabilité est égale à $1/(m * D\_Max)$. Nous avons donc les expressions suivantes :

$$P[x< X \leq x+dx \mid X \text{ est accepté}] = \frac{P[\{x < X \leq x+dx\} \cap \{X \text{ est accepté}\}]}{P[X \text{ est accepté}]}$$

$$= \frac{D(x)dx / (m * D\_Max)}{1/(m * D\_Max)}$$

$$= D(x)dx \qquad (5.26)$$

**Algorithme ReSampel**
/* Ré-échantillonnage d'un ensemble de patrons afin de respecter une distribution donnée. */

**Entrées :** - $S_{in}$ : Ensemble de $m$ patrons
- $D$ : Distribution désirée pour l'ensemble de patrons obtenu

**Sortie :** - $S_{out}$ : Ensemble de $m$ patrons avec la distribution $D$

Chercher $D\_Max$, la valeur maximale dans la distribution $D$
**Initialiser** $i = 1$
**Répéter tant que** $i \leq m$ /* Pour tous les patrons */
1. Générer un nombre entier aléatoire (distribution uniforme) entre [1,$m$] : $X$.
2. Générer un nombre entier aléatoire (distribution uniforme) entre [0, $D\_Max$] : $Y$.
3. **Si** $Y \leq D(X)$, **alors** $S_{out}(i) = S_{in}(X)$ et incrémenter $i$ : $i = i + 1$
   **Sinon** rejeter $X$.

Figure 5.5. Algorithme de ré-échantillonnage d'un ensemble de patrons pour respecter une distribution donnée.

L'équation (5.26) montre que si $X$ est accepté, sa distribution est égale à $D$. Par conséquent, $S_{out}$ a également la distribution désirée.

#### *5.2.1.3.2 SCÉNARIOS COMMUN ET PERSONNALISÉ*

Les deux scénarios d'apprentissage et de validation commun et personnalisé des classificateurs neuronaux sont expliqués à la section 4.5.2.3.1. Nous avons appliqué ces scénarios sur nos classificateurs AdaBoost.M1. Pour les deux scénarios, l'algorithme de la phase d'apprentissage est celui de la figure 5.4 et le résultat de cette phase est un ensemble de ($h$, $\beta$) utilisé dans les classificateurs dont la structure générique est celle de la figure 5.3. En plus, les caractéristiques utilisées sont choisies parmi nos caractéristiques proposées, présentées dans la section 5.2.1.1. Ce qui différencie les deux scénarios est donc le choix des patrons d'entraînement et de validation parmi les patrons disponibles.

En ce qui concerne le premier scénario, les patrons extraits d'un texte ont été utilisés pour l'entraînement et ceux des autres textes pour la validation. Comme tous les classificateurs neuronaux entraîné et testé avec ce scénario, le texte employé pour l'apprentissage est le texte no. 7 (figure A.7 de l'annexe A). Les classificateurs implantés, avec différentes combinaisons de caractéristiques à l'entrée, seront présentés dans la section suivante.

Dans le cas du scénario personnalisé, un sous-ensemble de patrons de chaque texte a été utilisé pour l'entraînement des classificateurs AdaBoost.M1 et le reste des patrons du même texte pour la validation. Pour implanter nos classificateurs, nous avons utilisé différentes combinaisons et ordres de présentation des caractéristiques, nombres de patrons d'entraînement et nombres maximaux d'itérations pour chacune des caractéristiques ($T$). Nous donnerons les détails de ces classificateurs dans la section suivante, lors de la présentation des résultats de classification.

## 5.2.1.4 RÉSULTATS DE CLASSIFICATION

Dans cette section, nous présentons les résultats de classification obtenus avec l'application des deux scénarios d'apprentissage et de validation commun et personnalisé sur les classificateurs AdaBoost.M1. Ces résultats comprennent les taux de classification correcte des distances inter-tracés.

### *5.2.1.4.1 RÉSULTATS DU SCÉNARIO COMMUN*

Nous avons appliqué ce scénario à plusieurs classificateurs avec différentes entrées, différents ordres de présentation des entrées et nombres maximaux des itérations[6]. Ces informations sont présentées dans le tableau 5.1.

Tableau 5.1. Spécifications des classificateurs AdaBoost.M1 implantés (scénario commun).

| Numéro du classificateur | Entrées | *T* | Stratégie de l'apprentissage |
|---|---|---|---|
| 1 | Dist_Paral | 5 | dépendante |
| 2 | Dist_Paral | 10 | dépendante |
| 3 | Dist_Paral et RLmin | 5 | dépendante |
| 4 | Dist_Paral et RLmin | 10 | dépendante |
| 5 | Dist_Paral et Rlavg | 5 | dépendante |
| 6 | Dist_Paral et Rlavg | 10 | dépendante |
| 7 | RLmin et Dist_Paral | 5 | dépendante |
| 8 | RLmin et Dist_Paral | 10 | dépendante |
| 9 | RLavg et Dist_Paral | 5 | dépendante |
| 10 | RLavg et Dist_Paral | 10 | dépendante |
| 11 | 7 entrées* | 5 | indépendante |
| 12 | 7 entrées* | 10 | indépendante |
| 13 | 7 entrées* | 5 | dépendante |
| 14 | 7 entrées* | 10 | dépendante |

* 7 entrées : Dist_Paral, RLmin, RLavg, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema.

[6] Toutefois, tous ces classificateurs ont la même structure générique, présentée dans la figure 5.3.

La première colonne du tableau 5.1 contient le numéro des classificateurs. Ce numéro sera utilisé dans le tableau des résultats de classification. La deuxième colonne montre la (les) entrée(s) et leur ordre de présentation séquentielle aux classificateurs. Par exemple, les classificateurs 3 et 7 ont les mêmes entrées mais différents ordres de présentation. Dans la troisième colonne, se trouvent les nombres d'itérations $T$ utilisés pour entraîner les classificateurs. Et, finalement, la quatrième colonne contient la stratégie d'entraînement des classificateurs «dépendante » ou «indépendante» (voir la section 5.2.1.3.1 pour la définition de ces termes).

Le tableau 5.2 montre les résultats obtenus avec l'application du scénario commun aux classificateurs présentés dans le tableau 5.1. Nous rappelons que, dans ce scénario, les patrons du septième texte de notre base de données ont été utilisés pour l'entraînement des classificateurs. La dernière colonne du tableau 5.2 contient les valeurs moyennes des taux de classification correcte pour les différents classificateurs.

Tableau 5.2. Pourcentages de classification correcte des distances inter-tracés pour les classificateurs AdaBoost.M1 (scénario commun).

| Numéro de classificateur \ Numéro de Texte | *1* | *2* | *3* | *4* | *5* | *6* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *6* | 96.2 | 98.7 | 99.8 | 100 | 99.4 | 99.6 | 99.1 | 99 | 98.4 | 96.7 | 99 | 97.8 | **98.64** |
| *5* | 95.1 | 98.7 | 99.8 | 100 | 99.4 | 99.6 | 98.6 | 99.5 | 98.9 | 97.1 | 99 | 97.8 | **98.63** |
| *8* | 96.2 | 98.7 | 99.8 | 100 | 99.4 | 99 | 99.1 | 98.6 | 97.5 | 96.7 | 98.6 | 98.4 | **98.50** |
| *10* | 94.5 | 98.7 | 99.8 | 100 | 99.4 | 99.6 | 99.1 | 99 | 98.6 | 96.2 | 98.6 | 98.4 | **98.49** |
| *9* | 96.7 | 98.7 | 99.8 | 100 | 99.4 | 99 | 99.1 | 98.6 | 97.5 | 96.2 | 98.6 | 97.8 | **98.45** |
| *3* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 96.2 | 99.3 | 98.4 | **98.28** |
| *7* | 96.7 | 98.7 | 99.6 | 100 | 99.4 | 98.3 | 99.1 | 98.6 | 96.4 | 95.8 | 98.6 | 97.8 | **98.25** |
| *1* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *2* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *12* | 88.5 | 99.4 | 99.8 | 100 | 99.4 | 99.8 | 97.6 | 99 | 99.2 | 95.4 | 99.3 | 98.4 | **97.98** |
| *11* | 86.8 | 99.4 | 99.8 | 100 | 99.4 | 99.8 | 97.2 | 98.6 | 98.6 | 95.4 | 99.3 | 98.4 | **97.73** |
| *4* | 86.8 | 99.4 | 99.8 | 100 | 98.8 | 99.8 | 96.7 | 98.1 | 98.1 | 96.7 | 99.3 | 97.8 | **97.61** |
| *13* | 76.9 | 100 | 99.8 | 99.8 | 98.2 | 99.4 | 90 | 90 | 98.9 | 94.1 | 99 | 98.4 | **95.38** |
| *14* | 94.5 | 98.7 | 91.8 | 89.8 | 98.8 | 82.9 | 88.6 | 91.4 | 83.6 | 80.8 | 83.7 | 96.2 | **90.07** |

Tableau 5.3. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 5.2.

| Numéro de classificateur | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *6* | 1.19 | 98.64 | (98.02, 99.26) |
| *5* | 1.39 | 98.63 | (97.90, 99.35) |
| *8* | 1.17 | 98.50 | (97.90, 99.10) |
| *10* | 1.59 | 98.49 | (97.67, 99.32) |
| *9* | 1.18 | 98.45 | (97.84, 99.06) |
| *3* | 2.46 | 98.28 | (97.00, 99.55) |
| *7* | 1.33 | 98.25 | (97.56, 98.94) |
| *1* | 2.62 | 98.14 | (96.78, 99.50) |
| *2* | 2.62 | 98.14 | (96.78, 99.50) |
| *12* | 3.25 | 97.98 | (96.30, 99.67) |
| *11* | 3.68 | 97.73 | (95.82, 99.63) |
| *4* | 3.59 | 97.61 | (95.75, 99.47) |
| *13* | 6.88 | 95.38 | (91.81, 98.94) |
| *14* | 6.29 | 90.07 | (86.81, 93.33) |

Le tableau 5.3 montre le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 5.2, incluant les estimés pour l'écart type et la moyenne et l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

La figure 5.6 est une représentation graphique des valeurs moyennes et de leurs intervalles de confiance montrés dans le tableau 5.3 (à l'exception du dernier résultat qui a été exclu de la figure, en raison de son grand écart par rapport aux autres résultats).

Nous analyserons les résultats présentés dans cette section, à la section 5.3.

### *5.2.1.4.2 RÉSULTATS DU SCÉNARIO PERSONNALISÉ*

Le scénario d'apprentissage et de validation personnalisé des classificateurs a également été appliqué sur tous les textes de notre base de données. L'objectif de la phase d'apprentissage consiste toujours à déterminer les paires $(h, \beta)$ dans les différentes itérations. Ces paires seront ensuite utilisées pour construire le

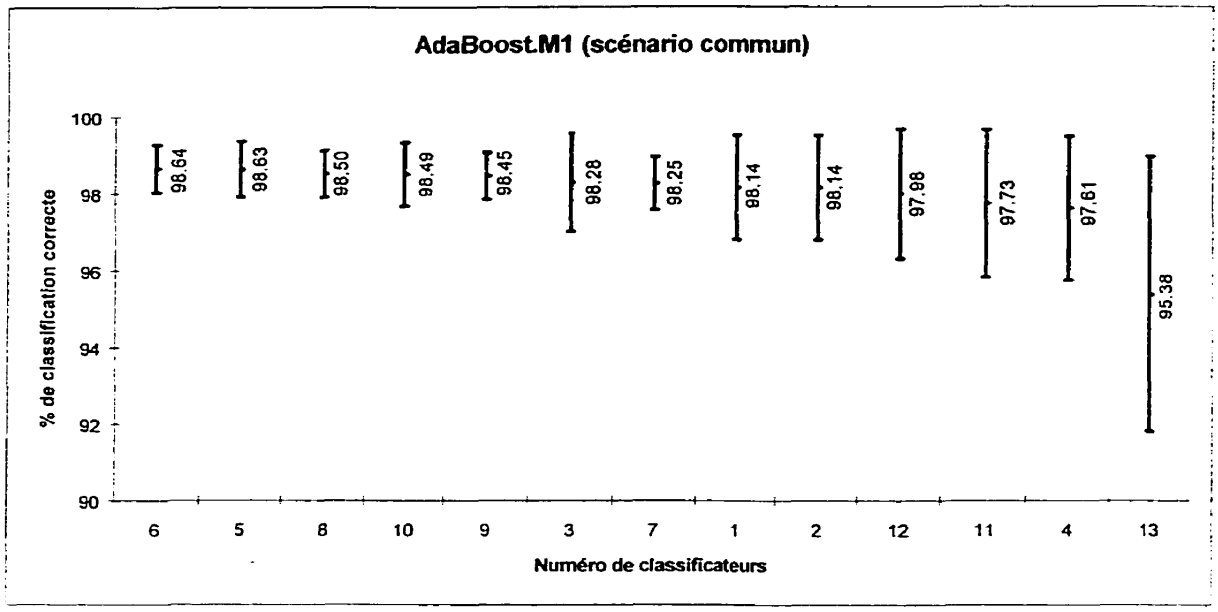



Figure 5.6. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs AdaBoost.M1 (scénario commun).

classificateur AdaBoost.M1 complet dont la structure est montrées dans la figure 5.3.

Le tableau 5.4 montre les spécifications des classificateurs AdaBoost.M1 entraînés avec le scénario personnalisé, incluant les caractéristiques à l'entrée et les nombres de patrons d'apprentissage ($N$) et d'itérations ($T$). Les hypothèse simples sont des neurones ayant une entrée et une sortie. Pour un tel classificateur, selon le tableau 4.6, pour une erreur permise de généralisation de 0.1, le nombre minimal de patrons d'apprentissage est égal à 20. Dans un premier temps, nous avons entraîné nos classificateurs avec ce nombre de patrons d'apprentissage. Afin d'évaluer l'effet de l'augmentation du nombre de patrons d'apprentissage sur les résultats, nous avons également effectué l'entraînement des classificateurs avec un nombre plus élevé de patrons soit 70. Deux valeurs différentes ont été utilisées comme nombre maximal d'itérations $T$ : 5 et 10.

Tableau 5.4. Spécifications des classificateurs AdaBoost.M1 implantés (scénario personnalisé).

| Numéro de classificateur | Entrées | $T$ | $N$ |
|---|---|---|---|
| 1 | Dist_Paral | 5 | 20 |
| 2 | Dist_Paral | 10 | 20 |
| 3 | Dist_Paral | 5 | 70 |
| 4 | Dist_Paral | 10 | 70 |
| 5 | Rlavg | 5 | 20 |
| 6 | Rlavg | 10 | 20 |
| 7 | Rlavg | 5 | 70 |
| 8 | Rlavg | 10 | 70 |
| 9 | RLmin | 5 | 20 |
| 10 | Rlmin | 10 | 20 |
| 11 | Rlmin | 5 | 70 |
| 12 | Rlmin | 10 | 70 |
| 13 | Dist_Paral et Rlavg | 5 | 20 |
| 14 | Dist_Paral et Rlavg | 10 | 20 |
| 15 | Dist_Paral et Rlavg | 5 | 70 |
| 16 | Dist_Paral et Rlavg | 10 | 70 |
| 17 | RLavg_et Dist_Paral | 5 | 20 |
| 18 | RLavg_et Dist_Paral | 10 | 20 |
| 19 | RLavg_et Dist_Paral | 5 | 70 |
| 20 | RLavg_et Dist_Paral | 10 | 70 |
| 21 | Dist_Paral et RLmin | 5 | 20 |
| 22 | Dist_Paral et RLmin | 10 | 20 |
| 23 | Dist_Paral et RLmin | 5 | 70 |
| 24 | Dist_Paral et RLmin | 10 | 70 |
| 25 | RLmin_et Dist_Paral | 5 | 20 |
| 26 | RLmin_et Dist_Paral | 10 | 20 |
| 27 | RLmin_et Dist_Paral | 5 | 70 |
| 28 | RLmin_et Dist_Paral | 10 | 70 |
| 29 | 7 entrées* | 5 | 20 |
| 30 | 7 entrées* | 10 | 20 |
| 31 | 7 entrées* | 5 | 70 |
| 32 | 7 entrées* | 10 | 70 |

* 7 entrées : Dist_Paral, RLmin, RLavg, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema.

Les classificateurs ayant les mêmes caractéristiques à l'entrée sont entraînés de façon indépendante sur différents textes de la base de données. En conséquence, ils peuvent avoir différentes valeurs de poids, de biais, de $\beta$ des hypothèses simples et surtout différents nombres d'hypothèses générées. Par exemple, si le nombre maximal

d'itérations $T$ est 10 et l'entrée des classificateur est RLmin, le nombre d'hypothèses générées avec l'entraînement du premier et du quatrième texte pourrait être égal à 10 et 5, respectivement. Cela montre que l'entraînement du quatrième texte est arrêté avant que le nombre maximal d'itérations ne soit atteint (voir l'algorithme de la figure 5.4).

Quant à l'initialisation de la distribution des patrons d'apprentissage lors du passage d'une caractéristique à l'autre, tous les classificateurs du tableau 5.4 sont entraînés avec la stratégie «dépendante» qui est la stratégie utilisée dans l'algorithme de la figure 5.4.

Le tableau 5.5 montre les résultats de classification des distances inter-tracés pour les classificateurs AdaBoost.M1 entraînés avec le scénario personnalisé. La première colonne de ce tableau contient le numéro de classificateurs dont les spécifications sont données dans le tableau 5.4. Les autres colonnes (sauf la dernière) montrent, pour les textes de la base données (le numéro de chaque texte se trouve dans la première ligne), les pourcentages de classification correcte obtenus avec différents classificateurs. Les moyennes de ces valeurs sont présentées dans la dernière colonne du tableau 5.5.

Dans le tableau 5.6, nous présenterons le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 5.5, incluant les estimés pour l'écart type et la moyenne et l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

La figure 5.7 est une représentation graphique des valeurs moyennes et de leurs intervalles de confiance montrés dans le tableau 5.6 (à l'exception du dernier résultat qui a été exclu de la figure, en raison de son grand écart par rapport aux autres résultats).

Les résultats présentés dans cette section seront analysés à la section 5.3. Le reste de cette section est consacré aux classificateurs implantés avec l'algorithme amélioré d'AdaBoost.

Tableau 5.5. Pourcentages de classification correcte des distances inter-tracés pour les classificateurs AdaBoost.M1 (scénario personnalisé).

| Numéro de classificateur | Numéro de Texte | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
| *4* | 98.9 | 98.7 | 98.9 | 99.1 | 99.4 | 99.8 | 98.3 | 99.1 | 99.5 | 99.2 | 95 | 99.3 | 98.4 | **98.74** |
| *1* | 98.4 | 98.1 | 99.6 | 100 | 99.4 | 99.4 | 94.4 | 97.2 | 99 | 99.2 | 95.8 | 98.6 | 97.8 | **98.22** |
| *2* | 92.9 | 98.7 | 99.6 | 100 | 99.4 | 99.8 | 97.2 | 98.6 | 96.2 | 97.5 | 96.2 | 98.6 | 98.4 | **97.93** |
| *10* | 97.8 | 98.1 | 97.3 | 96.2 | 98.8 | 99.8 | 98.7 | 97.6 | 95.2 | 96.4 | 97.1 | 94.2 | 97.8 | **97.31** |
| *24* | 80.8 | 100 | 84.8 | 100 | 99.4 | 99.8 | 85.1 | 98.6 | 98.5 | 98.9 | 96.4 | 99.3 | 97.1 | **95.28** |
| *31* | 97.8 | 97.5 | 99.1 | 100 | 96.9 | 99.2 | 94.7 | 98.1 | 66.8 | 93.8 | 97.2 | 97.9 | 97.1 | **95.08** |
| *23* | 91.8 | 98.8 | 99.6 | 100 | 99.4 | 99.8 | 98.7 | 70.5 | 97.1 | 82.1 | 97.2 | 99.3 | 95.4 | **94.59** |
| *27* | 98.4 | 100 | 84.8 | 100 | 99.4 | 99.2 | 85.1 | 67.6 | 98 | 98.6 | 96.4 | 98.6 | 98.3 | **94.18** |
| *28* | 99.5 | 91.9 | 99.8 | 84.8 | 97.5 | 85.1 | 98.7 | 98.1 | 98.5 | 82.1 | 96.4 | 98.6 | 87.9 | **93.76** |
| *14* | 90.1 | 94.4 | 84.8 | 96 | 96.3 | 99.8 | 97.1 | 99 | 99 | 82.1 | 95.2 | 77.4 | 96 | **92.86** |
| *19* | 94 | 100 | 99.8 | 100 | 99.4 | 85.1 | 96.7 | 67.1 | 95.1 | 99.2 | 73.4 | 94.1 | 97.7 | **92.43** |
| *13* | 99.5 | 98.8 | 84.3 | 100 | 96.3 | 99.8 | 97.1 | 70.5 | 99 | 90.8 | 96.4 | 77.4 | 87.9 | **92.14** |
| *26* | 97.3 | 98.1 | 84.8 | 100 | 95.6 | 99.8 | 85.1 | 99 | 71.7 | 73.9 | 96.8 | 98.6 | 96 | **92.05** |
| *18* | 94.5 | 91.9 | 84.3 | 99.1 | 95 | 99.4 | 98.5 | 99 | 99 | 82.1 | 60.5 | 95.8 | 97.1 | **92.02** |
| *29* | 91.8 | 97.5 | 99.8 | 100 | 95 | 30.9 | 98.7 | 98.1 | 97.1 | 98.4 | 96.4 | 94.1 | 95.4 | **91.78** |
| *20* | 94.5 | 91.9 | 99.8 | 100 | 95 | 85.1 | 98.5 | 71 | 99 | 82.1 | 96.4 | 77 | 93.7 | **91.08** |
| *3* | 97.3 | 100 | 99.8 | 100 | 99.4 | 99.8 | 97.2 | 97.2 | 72.4 | 81.9 | 72.4 | 78 | 88.5 | **91.07** |
| *25* | 80.8 | 95 | 99.8 | 84.8 | 95.6 | 85.1 | 85.1 | 98.6 | 99 | 82.1 | 96.8 | 76.3 | 87.3 | **89.72** |
| *22* | 99.5 | 98.8 | 84.8 | 84.8 | 96.3 | 99.8 | 85.1 | 70.5 | 84.8 | 82.1 | 97.2 | 77.4 | 97.1 | **89.09** |
| *15* | 99.5 | 91.9 | 99.8 | 100 | 95 | 99.4 | 97.1 | 71 | 71.7 | 98.9 | 58.5 | 77.4 | 96 | **88.94** |
| *21* | 97.8 | 91.9 | 84.8 | 100 | 95.6 | 99.8 | 85.1 | 70.5 | 66.8 | 82.1 | 96.8 | 77.4 | 87.9 | **87.42** |
| *17* | 95.1 | 99.4 | 99.8 | 84.8 | 95 | 99.4 | 84.2 | 71 | 71.7 | 80.4 | 58.5 | 77 | 97.1 | **85.65** |
| *16* | 98.4 | 91.9 | 84.8 | 84.8 | 95 | 85.1 | 85.1 | 70.5 | 66.8 | 82.1 | 96 | 77.4 | 87.9 | **85.06** |
| *32* | 81.9 | 93.1 | 99.8 | 99.3 | 98.1 | 99.8 | 85.1 | 70.5 | 66.8 | 39.4 | 73.4 | 98.3 | 87.9 | **84.11** |
| *12* | 97.3 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 67.6 | 81.9 | 72.4 | 78 | 88.5 | **83.45** |
| *5* | 80.8 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 67.6 | 81.9 | 72.4 | 78 | 88.5 | **82.18** |
| *9* | 80.8 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 72.4 | 81.9 | 56.9 | 78 | 88.5 | **81.36** |
| *7* | 80.8 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 67.6 | 81.9 | 56.9 | 78 | 88.5 | **80.99** |
| *6* | 80.8 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 67.6 | 81.9 | 56.9 | 78 | 88.5 | **80.99** |
| *8* | 80.8 | 91.8 | 84.9 | 84.9 | 95.3 | 85.8 | 85.4 | 71.1 | 67.6 | 81.9 | 56.9 | 78 | 88.5 | **80.99** |
| *30* | 94 | 91.9 | 33 | 97.3 | 94.4 | 85.1 | 32.2 | 91.3 | 93.2 | 93.2 | 58.5 | 96.5 | 87.9 | **80.65** |
| *11* | 97.8 | 94.9 | 33.4 | 33.6 | 95.9 | 34.2 | 33.7 | 76.3 | 81 | 38.9 | 67.4 | 51.5 | 92.9 | **63.96** |

Tableau 5.6. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 5.5.

| Numéro de classificateur | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *4* | 1.20 | 98.74 | (98.15, 99.33) |
| *1* | 1.61 | 98.22 | (97.42, 99.02) |
| *2* | 1.97 | 97.93 | (96.96, 98.90) |
| *10* | 1.52 | 97.31 | (96.56, 98.06) |
| *24* | 6.83 | 95.28 | (91.91, 98.66) |
| *31* | 8.67 | 95.08 | (90.80, 99.37) |
| *23* | 8.75 | 94.59 | (90.27, 98.92) |
| *27* | 9.53 | 94.18 | (89.47, 98.90) |
| *28* | 6.51 | 93.76 | (90.54, 96.98) |
| *14* | 7.12 | 92.86 | (89.34, 96.38) |
| *19* | 10.72 | 92.43 | (87.13, 97.73) |
| *13* | 9.54 | 92.14 | (87.42, 96.86) |
| *26* | 9.89 | 92.05 | (87.16, 96.94) |
| *18* | 10.99 | 92.02 | (86.58, 97.45) |
| *29* | 18.44 | 91.78 | (82.67, 100) |
| *20* | 9.37 | 91.08 | (86.44, 95.71) |
| *3* | 11.01 | 91.07 | (85.63, 96.51) |
| *25* | 8.01 | 89.72 | (85.76, 93.67) |
| *22* | 9.58 | 89.09 | (84.36, 93.83) |
| *15* | 14.15 | 88.94 | (81.95, 95.93) |
| *21* | 10.98 | 87.42 | (82.00, 92.85) |
| *17* | 13.35 | 85.65 | (79.05, 92.24) |
| *16* | 9.42 | 85.06 | (80.41, 89.72) |
| *32* | 17.84 | 84.11 | (75.29, 92.93) |
| *12* | 9.12 | 83.45 | (78.94, 87.96) |
| *5* | 8.13 | 82.18 | (78.17, 86.20) |
| *9* | 10.05 | 81.36 | (76.40, 86.33) |
| *7* | 10.48 | 80.99 | (75.81, 86.17) |
| *6* | 10.48 | 80.99 | (75.81, 86.17) |
| *8* | 10.48 | 80.99 | (75.81, 86.17) |
| *30* | 23.51 | 80.65 | (69.03, 92.27) |
| *11* | 27.16 | 63.96 | (50.53, 77.39) |

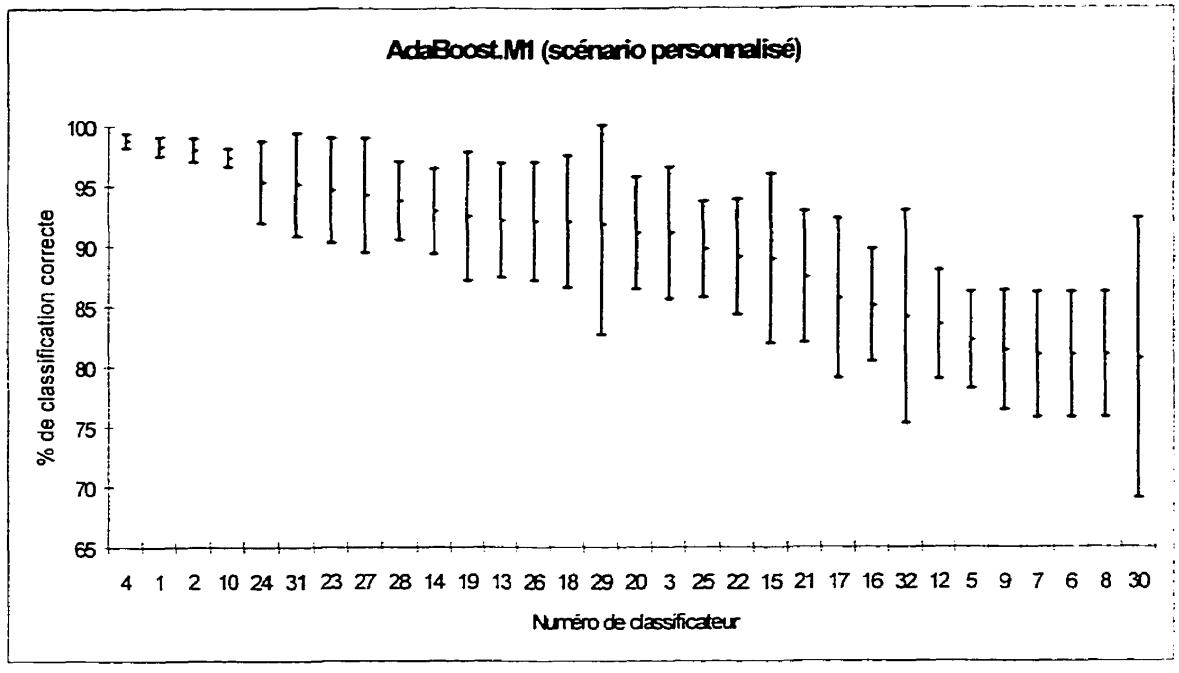



Figure 5.7. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs AdaBoost.M1 (scénario personnalisé) (voir le tableau 5.6).

## 5.2.2 CLASSIFICATEURS BASÉS SUR LA VERSION AMÉLIORÉE D'ADABOOST

Dans cette section, nous présenterons les classificateurs que nous avons implantés en nous inspirant de la version améliorée d'AdaBoost (figure 5.2). Puisque notre algorithme est relativement différent de cette version d'algorithme, nous l'appellerons dorénavant «AdaBoost.Z»[7].

Les entrées des classificateurs AdaBoost.Z ont été choisies parmi les mêmes caractéristiques utilisées pour les classificateurs AdaBoost.M1 (voir la section 5.2.1.1).

[7] Le poste fixe 'Z' montre que l'algorithme tente de minimiser le facteur Z comme critère d'entraînement.

### 5.2.2.1 STRUCTURE DES CLASSIFICATEURS

La figure 5.8 montre la structure générique de nos classificateurs AdaBoost.Z. Dans cette figure, $x^1, ..., x^N$ représentent les $N$ caractéristiques du patron $x$. Chaque caractéristique est présentée à une banque d'hypothèses simples $h$ qui sont expertes de cette caractéristique. Ces hypothèses (qui sont des classificateurs neuronaux) ainsi que leur $\alpha$ sont les résultats de l'algorithme AdaBoost.Z (voir la figure 5.9). Chaque hypothèse produit une sortie entre (-1, +1) en fonction de la valeur de l'entrée. Ces valeurs sont ensuite multipliées par $\alpha$ (> 0) de l'hypothèse correspondante qui représente son poids dans l'ensemble des hypothèses. Les votes pondérés de toutes les hypothèses sont ensuite additionnés. Le signe du résultat de cette sommation détermine la classe du patron d'entrée : les sorties négative et positive correspondent à des classes intra et inter-mots, respectivement.

Comme nous l'avons mentionné précédemment, la figure 5.8 présente le schéma générique des classificateurs AdaBoost.Z. Dans nos classificateurs spécifiques, les caractéristiques utilisées à l'entrée varient d'un classificateur à l'autre. Le nombre d'hypothèses générées pour chaque caractéristique est déterminé pendant la phase d'apprentissage. Cette phase sera expliquée dans la section suivante et les classificateurs AdaBoost.Z spécifiques seront présentés dans la section 5.2.2.3, lors de la présentation des résultats de classification.

La complexité de la structure des classificateurs dépend du nombre d'hypothèses générées dans la phase d'apprentissage. Si $T$ est le nombre maximal d'hypothèses générées pour chaque patron et $N$ est le nombre de caractéristiques, le nombre total d'hypothèses dans le classificateur pourra varier entre [1, $N*T$]. En général, les hypothèses sont des classificateurs très simples, comme par exemple un seul neurone. Dans le cas de ce dernier, selon la figure 5.8, pour classifier un patron, le nombre maximal d'opérations peut être calculé à partir de l'expression suivante :

$$Max\_N_{op} = (2*N*T)\ Multiplication + (N*T+1)\ Addition \qquad (5.27)$$

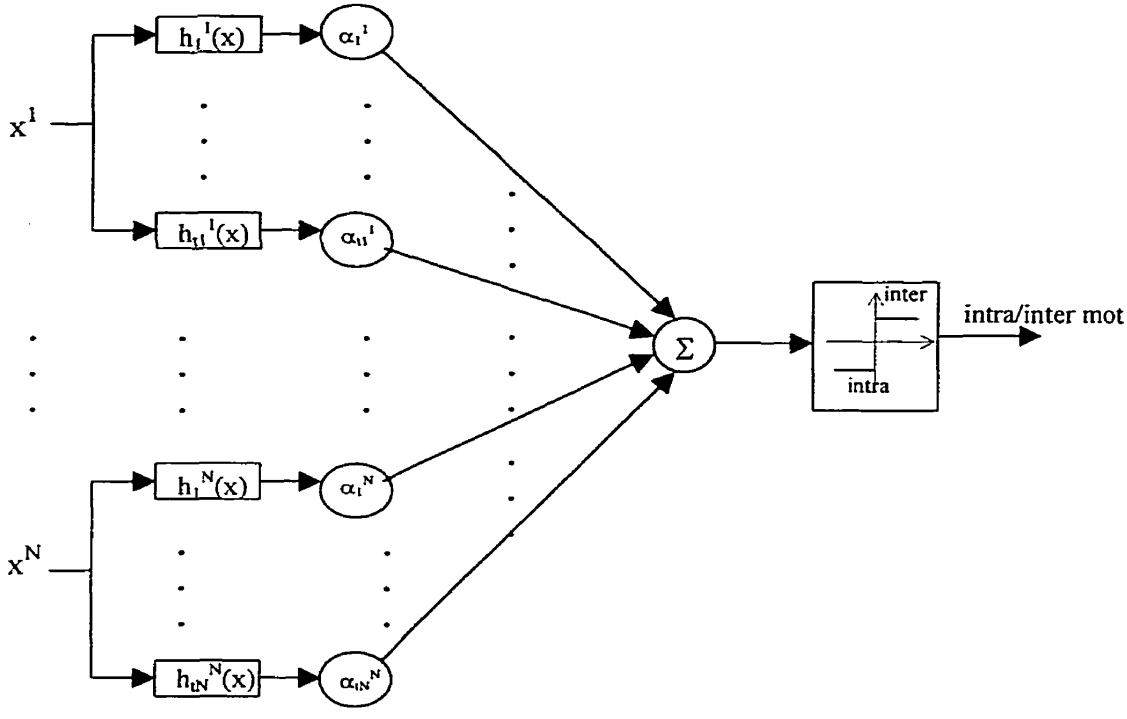



Figure 5.8. Structure générique des classificateurs AdaBoost.Z implantés.

## 5.2.2.2 ALGORITHME D'APPRENTISSAGE IMPLANTÉ

La figure 5.9 montre l'algorithme que nous avons implanté pour l'apprentissage des classificateurs AdaBoost.Z.

Les entrées de l'algorithme sont les suivantes : un ensemble de $m$ patrons d'entraînement, un algorithme d'entraînement **Minim_Z** et un entier $T$ représentant le nombre maximal d'itérations.

L'objectif de l'algorithme consiste à entraîner des experts pour chacune des $N$ caractéristiques des patrons d'apprentissage. $N$ peut être inférieur ou égal au nombre total de caractéristiques disponibles. Tout comme l'algorithme d'AdaBoost.M1 (figure 5.4), le sous-ensemble d'entraînement $S_n$ est construit pour chaque caractéristique.

**Algorithme AdaBoost.Z**
/* L'algorithme AdaBoost.Z pour l'entraînement des classificateurs de distance inter-tracés */

**Entrées :** - Ensemble de $m$ patrons d'entraînement $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
avec des étiquettes $y_i \in Y = \{-1,+1\}$
- Algorithme d'entraînement **Minim_Z**
- Entier $T$, le nombre d'itérations

**Initialiser** $D_1(i) = 1 / m$ pour tous les $i = 1, \ldots, m$
**Initialiser** $T_{max}^{0} = 0$.
**Répéter pour** n = 1 : $N$ /* Pour toutes les caractéristiques */

1. Extraire le vecteur de $n^{ième}$ caractéristique $X_n$ à partir de l'ensemble d'entrée:
$X_n = \{x_{1n}, \ldots, x_{mn}\}$
2. Construire le sous-ensemble d'entraînement $S_n$ :
$S_n = \{(x_{1n}, y_1), \ldots, (x_{mn}, y_m)\}$
3. **Initialiser** $T_{min}^{n} = T_{max}^{n-1} + 1$ et $T_{max}^{n} = T_{max}^{n-1} + T$
4. **Répéter pour** $t = T_{min}^{n}, \ldots, T_{max}^{n}$ /* $T$ itérations au maximum */
5. Appeler l'algorithme **Minim_Z** pour entraîner le réseau neuronal:
Minim_Z( $D_t$, $S_n$)
6. Obtenir une hypothèse $h_t : X_n \rightarrow (-1, +1)$.
7. Calculer $r_t$:

$$r_t = \sum_{i=1}^{m} D_t(i) y_i h_t(x_{in})$$

8. **Si** $r_t < 0$, **alors :** affecter $T_{max}^{n} = t - 1$ /* Performance pire que celle */
/* d'une hypothèse aléatoire */
terminer les itérations.
9. Calculer $\alpha_t$ :

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 + r_t}{1 - r_t}\right)$$

/* Mise à jour de la distribution $D_t$ */
10. **Répéter pour** $i = 1,\ldots,m$

$$D_{t+1}(i) = D_t(i) \exp(-\alpha_t y_i h_t(x_{in}))$$

/* Normalisation de $D^{t+1}$ pour qu'elle devienne une distribution. */
11. **Répéter pour** i = 1,...,m

$$D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_{j=1}^{m} D_{t+1}(j)}$$

Figure 5.9. Algorithme d'entraînement des classificateurs AdaBoost.Z implantés pour la classification des distances inter-tracés.

Toutefois, contrairement à AdaBoost.M1, le même sous-ensemble est utilisé pour entraîner les hypothèses simples (expertes), sans être ré-échantillonné pour respecter une

distribution donnée. Entre *0* et *T* experts sont générés pour chaque caractéristique. Les experts sont des réseaux neuronaux ayant une structure très simple. En fait, chaque expert n'est qu'un neurone avec une entrée et une sortie. La fonction d'activation du neurone est une *tansig* (équation (4.5)) pour qu'il puisse produire des sorties situées entre (-1, +1). Comme nous l'avons mentionné dans la section 5.1.2, chaque expert $h_t$ est entraîné dans le but de minimiser le facteur $Z_t$ (équation (5.9)), plutôt que l'erreur quadratique moyenne. L'algorithme utilisé pour ce faire, s'appelle Minim_Z et sera détaillé plus loin dans cette section. Après avoir entraîné le réseau neuronal, on calcule le paramètre $r_t$ représentant la confiance de classification.

Nous avons ajouté une condition d'arrêt à l'algorithme qui consiste à comparer la performance de l'expert obtenu avec celle d'un classificateur purement aléatoire. Pour un tel classificateur, la valeur de $r_t$ est égale à 0. Par conséquent, si l'expert n'arrive pas à classifier mieux qu'un classificateur aléatoire, les itérations seront arrêtées et l'algorithme passera au traitement de la caractéristique suivante. Autrement, l'algorithme se poursuit avec le calcul de $\alpha_t$, la mise à jour et la normalisation de la distribution $D_t$. La paire ($h_t$, $\alpha_t$) obtenue sera utilisée dans la structure complète du classificateur AdaBoost.Z (figure 5.8) et la distribution $D_t$ servira à spécifier l'importance de chacun des patrons d'entraînement dans la prochaine itération de l'algorithme.

#### *5.2.2.2.1 ALGORITHME D'ENTRAÎNEMENT « MINIM_Z »*

La figure 5.10 montre l'algorithme Minim_Z implanté pour entraîner les classificateurs neuronaux dans l'algorithme AdaBoost.Z (figure 5.9). On utilise la technique de descente de gradient pour minimiser le facteur *Z* du classificateur neuronal.

L'objectif de l'algorithme consiste à modifier les paramètres libres du réseau afin de minimiser *Z* dans un processus itératif. Dans chaque époque (itération), tous les patrons d'entraînement sont présentés au classificateur et le vecteur *Z* est calculé pour ces patrons. On calcule ensuite *Z* total de l'époque, Z(*p*), où *p* est le numéro de l'époque. Z(*p*) est égal à la somme des éléments du vecteur *Z*. Si cette valeur est inférieure à un

**Algorithme Minim_Z**
/* L'algorithme d'entraînement d'un classificateur neuronal composé d'un neurone avec le critère de minimisation de *Z* */

**Entrées :** - Ensemble de *m* patrons d'entraînement $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ avec des étiquettes $y_i \in Y = \{-1,+1\}$
- Distribution *D* pour les *m* patrons d'entraînement
- Paramètres d'entraînement du réseau neuronal :
*Max_epochs* : nombre maximal d'époques;
$\eta$ : Taux d'apprentissage
*Z_goal* : *Z* désirée de l'apprentissage
*mnt* : moment (momentum)
*Z_ratio* : ratio de Z d'une époque sur celui de l'époque précédente

**Initialiser** le poids *W* et le biais *B* du neurone.
**Initialiser** le changement du poids *dW* et le changement du biais *dB*
$dW = 0$ et $dB = 0$.
**Initialiser** l'ancien *Z* (*Z_Préc*) à une très grande valeur.
**Répéter pour** $p = 1 : Max\_epochs$ /* itérations pour *Max_epochs* époques */
**Répéter pour** $i = 1 : m$ /* pour tous les patrons d'entraînement */
1. Calculer la sortie du neurone:
$v_i = W*x_i+B;$
$h_i = \varphi(v_i)$ /* $\varphi$ est la fonction *tansig* */
2. Calculer *Z* pour le neurone actuel :
$Z_i(p) = D(i)*exp((-1)*h_i*y_i)$
/* fin de Répéter pour $i = 1 : m$ */

3. Calculer *Z* total de l'époque :

$$Z(p) = \sum_{i=1}^{m} Z_i(p)$$

4. **Si** $Z(p) < Z\_goal$, **alors** terminer les itérations.

/* ajustement du poids et du biais du neurone */
**Si** ( $Z(p) / Z\_Préc$ ) $< Z\_ratio$, **alors** /* Bonne direction de gradient (descente) */
**Répéter pour** $i = 1 : m$ /* pour tous les patrons d'entraînement */
5. $dW = \eta*y_i*Z_i(p)*\varphi'(v_i)*x_i$
6. $dB = \eta*y_i*Z_i(p)*\varphi'(v_i)$
7. $W = W + dW$
8. $B = B + dB$
9. $Z\_Préc = Z(p)$
/* fin de Répéter pour $i = 1 : m$ */
**Sinon** /* Mauvaise direction de gradient */
9. $W = W - dW$
10. $B = B - dB$
/* Fin de Répéter pour $p = 1 : Max\_epochs$ */

Figure 5.10. Algorithme d'entraînement d'un réseau neuronal avec le critère de minimisation de facteur Z.

seuil (*Z_goal*), que l'utilisateur doit spécifier au début de l'algorithme, l'objectif est atteint et les itérations vont s'arrêter. Autrement, on procède à l'ajustement du poids et du biais du neurone. Pour ce faire, on calcule d'abord le ratio de *Z(p)/Z_Préc*, où *Z_Préc* représente le *Z* de l'époque précédente. Si ce ratio est inférieur à un seuil prédéfini (*Z_ratio*), l'algorithme est en train de diminuer la valeur de *Z*. Il faut donc continuer le changement de poids et de biais dans la même direction. Les nouveaux pas de changement *dW* et *dB* seront ainsi calculés suivi de la mise à jour du poids et du biais du neurone avec ces valeurs (dans les paragraphes suivants, nous présenterons les étapes de calcul de *dW* et *dB*). Si le ratio *Z(p)/Z_Préc* dépasse un seuil prédéfini, *Z(p)* sera rejeté et les anciens poids et biais seront restaurés sans calculer les nouveaux *dW* et *dB*.

Nous présentons maintenant une justification mathématique pour l'expression que nous avons proposée dans l'algorithme de la figure 5.10 pour l'ajustement du poids et du biais du neurone. L'objectif de l'algorithme consiste à modifier les paramètres libres du classificateur pour minimiser le paramètre *Z(p)* de l'époque *p* :

$$Z(p) = \sum_{i=1}^{m} Z_i(p) = \sum_{i=1}^{m} D(i) \exp(-y_i h_i) \tag{5.28}$$

Selon «la règle de delta»[8], pour minimiser *Z(p)*, la correction nécessaire dans le poids du neurone est calculée à partir de l'équation suivante :

$$\Delta W = -\eta \frac{\partial Z(p)}{\partial W} = -\eta \sum_{i=1}^{m} \frac{\partial Z_i(p)}{\partial W} \tag{5.29}$$

L'expression $\frac{\partial Z_i(p)}{\partial W}$ peut être écrite sous forme d'une chaîne de dérivées partielles :

$$\frac{\partial Z_i(p)}{\partial W} = \frac{\partial Z_i(p)}{\partial h_i} \frac{\partial h_i}{\partial v_i} \frac{\partial v_i}{\partial W} \tag{5.30}$$

où : $v_i$ est la sortie du neurone avant la fonction d'activation $\varphi$ et $h_i$ est la sortie finale après cette fonction.

---

[8] « Delta rule » en anglais

Les trois termes de l'équation (5.30) sont calculées selon les expressions suivantes :

$$\frac{\partial Z_t(p)}{\partial h_t} = -y_t D(i) \exp(-y_t h_t) = -y_t Z_t(p) \tag{5.31}$$

$$\frac{\partial y_t}{\partial v_t} = \varphi'(v_t) \tag{5.32}$$

$$\frac{\partial v_t}{\partial W} = x_t \tag{5.33}$$

En combinant les équations (5.30) à (5.33), l'équation suivante sera déduite :

$$\frac{\partial Z_t(p)}{\partial W} = -y_t Z_t(p) \varphi'(v_t) x_t \tag{5.34}$$

Et si l'on remplace l'équation (5.34) dans l'équation (5.29), on aura :

$$\Delta W = \sum_{t=1}^{m} \eta y_t Z_t(p) \varphi'(v_t) x_t \tag{5.35}$$

À l'aide d'une procédure mathématique semblable à celle que nous avons montrée ci-dessus, l'équation suivante peut être déduite pour le calcul de l'ajustement de biais du neurone:

$$\Delta B = \sum_{t=1}^{m} \eta y_t Z_t(p) \varphi'(v_t) \tag{5.36}$$

Les équations (5.35) et (5.36) sont les expressions que nous avons utilisées dans l'algorithme Minim_Z (figure 5.10) pour ajuster les paramètres libres du classificateur neuronal.

Nous avons expliqué dans cette section, la phase de l'entraînement des classificateurs AdaBoost.Z. Dans la section suivante, nous présenterons les résultats de classification de distances inter-tracés obtenus avec ce type de classificateurs.

## 5.2.2.3 RÉSULTATS DE CLASSIFICATION

Nous avons appliqué les deux scénarios commun et personnalisé sur les classificateurs AdaBoost.Z implantés. Les caractéristiques utilisées sont obtenues après la phase de reconnaissance des Accent_Point_Tbars et correspondent à celles que nous avons proposées dans le chapitre 3. Nous présenterons d'abord les résultats du scénario commun et ensuite ceux du scénario personnalisé.

### *5.2.2.3.1 RÉSULTATS DU SCÉNARIO COMMUN*

Différentes combinaisons de caractéristiques et du nombre d'itérations ont été utilisées pour tester les classificateurs AdaBoost.Z entraînés avec le scénario commun. Ces informations, pour chacun des classificateurs, sont montrées dans le tableau 5.7.

La première colonne du tableau 5.7 contient les numéro des classificateurs. Ces numéros seront utilisés dans le tableau des résultats de classification. La deuxième colonne montre les entrées et leur ordre de présentation aux différents classificateurs et la troisième colonne contient le nombre maximal d'itérations de l'algorithme.

Tableau 5.7. Spécifications des classificateurs AdaBoost.Z implantés (scénario commun).

| Numéro de classificateur | Entrées | *T* |
|---|---|---|
| 1 | Dist_Paral | 5 |
| 2 | Dist_Paral | 10 |
| 3 | RLmin | 5 |
| 4 | RLmin | 10 |
| 5 | Dist_Paral et RLavg | 5 |
| 6 | Dist_Paral et RLavg | 10 |
| 7 | Dist_Paral et RLmin | 5 |
| 8 | Dist_Paral et RLmin | 10 |
| 9 | Dist_Paral, RLmin, RLavg, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema | 5 |
| 10 | Rlavg, Rlmin, Dist_Paral, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema | 5 |
| 11 | Rlmin, Rlavg, Dist_Paral, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema | 5 |
| 12 | Dist_Paral, Rlavg, RLmin, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema | 5 |

Pour tous les classificateurs, la stratégie de l'apprentissage «dépendante» a été utilisée: la distribution $D_t$ produite par des itérations effectuées pour une caractéristique a été présentée aux itérations de la caractéristique suivante, sans être réinitialisée à une distribution uniforme.

**Stratégie adoptée pour choisir les caractéristiques**

Les caractéristiques utilisées et leurs combinaisons dans le tableau 5.7 ont été choisies selon les discussions et les résultats des expérimentations précédentes, notamment ceux du tableau 5.6.

Pour les classificateurs à une entrée, les deux mesures de distances inter-tracés Dist_Paral et RLmin ont été sélectionnées, en raison de leurs performances toujours bien meilleures que celle de la RLavg.

Quant aux classificateurs ayant deux entrées, nous avons choisi « Dist_Paral et RLmin » et « Dist_Paral et RLavg », puisque ces deux combinaisons ont donné des résultats meilleurs dans le tableau 5.6 que ceux des combinaisons de mêmes caractéristiques, mais avec l'ordre inversé[9].

Le classificateur 9 dans le tableau 5.7 utilise les mêmes entrées que les classificateurs à sept entrées dans le tableau 5.4. Le résultat d'un de ces classificateurs (numéro 31) est parmi les meilleurs dans le tableau 5.6. Pour cette raison et pour l'importance d'utilisation de toutes les caractéristiques proposées dans la procédure de segmentation, cette combinaison a été ajoutée au tableau 5.7. Et en plus, nous avons testé trois autres combinaisons des sept caractéristiques (classificateurs 10 à 12), afin d'évaluer l'importance de l'ordre de présentation séquentielle des trois mesures principales de distances inter-tracés pendant la phase d'apprentissage, dans les résultats de segmentation. Nous rappelons que, dans la phase d'apprentissage, selon l'algorithme d'Adaboost, les caractéristiques sont présentées séquentiellement aux classificateurs simples correspondants. En conséquence, il est important de présenter en premier les

[9] En d'autres termes, «Dist_Paral et RLmin » par rapport à «RImin et Dist_Paral» et «Dist_Paral et RLavg» par rapport à «RLavg et Dist_Paral».

caractéristiques principales afin d'exploiter au maximum leur capacité de discrimination et de minimiser les erreurs de classification. C'est, selon ce principe, que nous avons choisi les combinaisons des caractéristiques dans les classificateurs 9 à 12, sans tester aucune autre combinaison.

**Résultats numériques de classification**

Le tableau 5.8 montre les résultats de classification des distances inter-tracés obtenus avec les classificateurs AdaBoost.Z entraînés avec le scénario commun. Ce tableau contient les taux de classification correcte des distances inter-tracés pour chacun des textes de notre base de données (voir l'annexe A), à l'exception du texte no. 7 qui a été utilisé pour l'apprentissage des classificateurs. La dernière colonne du tableau montre les valeurs moyennes des taux de classification.

Dans le tableau 5.9, nous présenterons le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 5.8, incluant les estimés pour l'écart type et la moyenne et l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

Tableau 5.8. Pourcentages de classification correcte des distances inter-tracés pour les classificateurs AdaBoost.Z (scénario commun).

| Numéro de classificateur \ Numéro de Texte | *1* | *2* | *3* | *4* | *5* | *6* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *9* | 98.4 | 98.7 | 99.6 | 100 | 99.4 | 98.3 | 98.1 | 98.6 | 96.7 | 96.2 | 98.6 | 97.8 | **98.37** |
| *12* | 98.4 | 98.7 | 99.6 | 100 | 99.4 | 98.3 | 98.1 | 98.6 | 96.4 | 95.8 | 98.6 | 97.8 | **98.31** |
| *10* | 90.7 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 98.6 | 99.2 | 97.5 | 99.3 | 98.4 | **98.29** |
| *11* | 90.1 | 98.7 | 99.8 | 100 | 99.4 | 100 | 98.1 | 98.6 | 99.5 | 97.5 | 99.3 | 98.4 | **98.28** |
| *1* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *2* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *5* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *6* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *7* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *8* | 91.2 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.1 | 99.5 | 98.9 | 94.6 | 99.3 | 98.4 | **98.14** |
| *3* | 86.8 | 99.4 | 99.8 | 100 | 98.8 | 99.8 | 96.7 | 98.1 | 98.1 | 96.7 | 98.3 | 97.3 | **97.48** |
| *4* | 86.8 | 99.4 | 99.8 | 100 | 98.8 | 99.8 | 96.7 | 98.1 | 98.1 | 96.7 | 98.3 | 97.3 | **97.48** |

Tableau 5.9. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 5.8.

| Numéro de classificateur | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *9* | 1.10 | 98.37 | (97.80, 98.94) |
| *12* | 1.22 | 98.31 | (97.68, 98.94) |
| *10* | 2.51 | 98.29 | (96.99, 99.59) |
| *11* | 2.69 | 98.28 | (96.89, 99.68) |
| *1* | 2.62 | 98.14 | (96.78, 99.50) |
| *2* | 2.62 | 98.14 | (96.78, 99.50) |
| *5* | 2.62 | 98.14 | (96.78, 99.50) |
| *6* | 2.62 | 98.14 | (96.78, 99.50) |
| *7* | 2.62 | 98.14 | (96.78, 99.50) |
| *8* | 2.62 | 98.14 | (96.78, 99.50) |
| *3* | 3.56 | 97.48 | (95.64, 99.33) |
| *4* | 3.56 | 97.48 | (95.64, 99.33) |

La figure 5.11 est une représentation graphique des valeurs moyennes et de leurs intervalles de confiance montrés dans le tableau 5.9.

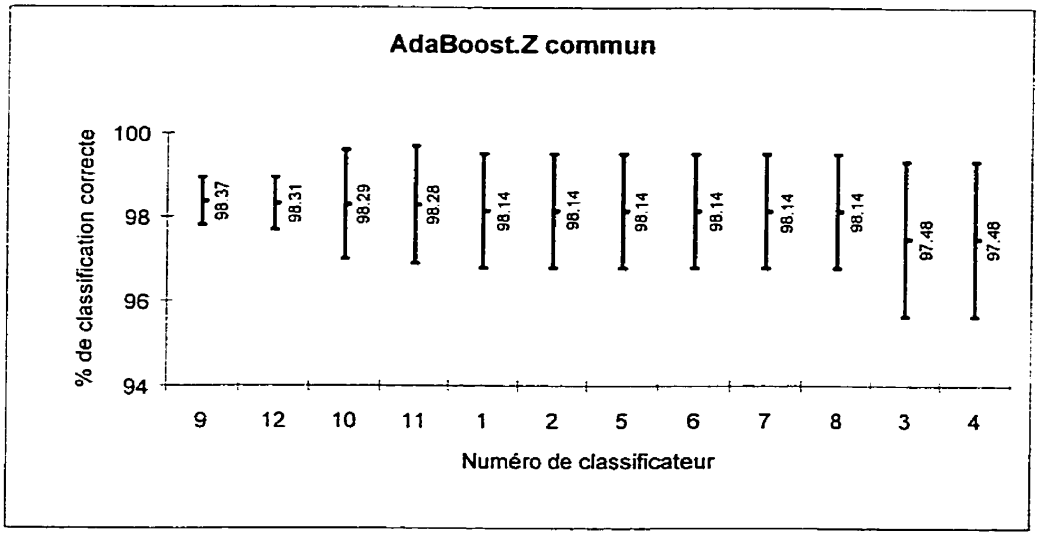



Figure 5.11. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs AdaBoost.Z (scénario commun).

### *5.2.2.3.2 RÉSULTATS DU SCÉNARIO PERSONNALISÉ*

Le tableau 5.10 montre les spécifications des classificateurs AdaBoost.Z implantés avec le scénario d'apprentissage personnalisé, incluant le numéros, la(les) caractéristique(s) à l'entrée et leur ordre de présentation séquentielle, le nombre maximal d'hypothèses générées $T$ pour chaque entrée et le nombre de patrons $N$ utilisés pour les entraîner. La stratégie de l'apprentissage «dépendante» est utilisée pour les classificateurs de plus d'une entrée, comme ce fut le cas pour le scénario commun.

**Stratégie adoptée pour choisir les caractéristiques**

Tout comme le scénario commun, nous nous sommes servis des résultats précédents pour choisir les caractéristiques et leurs ordres de présentation séquentielle aux hypothèse simples. Pour ce faire, les pourcentages de classification montrés dans le tableau 5.9 ont été utilisés.

Quant aux classificateurs avec une entrée, les résultats du tableau 5.9 montrent que, du point de vue de la valeur moyenne et de sa confiance, Dist_Paral est plus efficace que la RLmin. En conséquence, pour implanter les classificateurs à une entrée, nous avons choisi Dist_Paral.

Tableau 5.10. Spécifications des classificateurs AdaBoost.Z implantés (scénario personnalisé).

| Numéro de classificateur | Entrées | $T$ | $N$ |
|---|---|---|---|
| 1 | Dist_Paral | 10 | 20 |
| 2 | Dist_Paral | 20 | 20 |
| 3 | Dist_Paral | 10 | 70 |
| 4 | Dist_Paral et RLavg | 10 | 20 |
| 5 | Dist_Paral et RLavg | 10 | 40 |
| 6 | Dist_Paral et RLmin | 10 | 20 |
| 7 | Dist_Paral et RLmin | 10 | 40 |
| 8 | 7 entrées* | 5 | 40 |
| 9 | 7 entrées* | 5 | 70 |

* Dist_Paral, RLmin, RLavg, prNbExtrema, prHghtExtrema, curNbExtrema, curHghtExtrema

En ce qui concerne les combinaisons de deux caractéristiques, les deux combinaisons utilisées dans le tableau 5.9, « Dist_Paral et Rlavg » et « Dist_Paral et RLmin » donnent des résultats identiques. Elles ont donc été choisies pour implanter les classificateurs AdaBoost.Z ayant deux entrées.

Et, finalement, pour ce qui est des combinaisons à sept entrées, les entrées du classificateur 9 du tableau 5.9 ont été utilisées. Les classificateurs 9 à 12, dans ce tableau, ont tous les mêmes caractéristiques, mais différents ordres de présentation séquentielle aux hypothèses simples dans la phase d'apprentissage. Parmi ces classificateurs, le meilleur résultat appartient au classificateur 9, qui produit, en plus, le meilleur résultat parmi tous les classificateurs du tableau 5.9.

**Résultats numériques de classification**

Le tableau 5.11 montre les résultats de classification des distances inter-tracés obtenus avec nos classificateurs AdaBoost.Z pour le scénario personnalisé. Ce tableau contient les taux de classification correcte des distances inter-tracés pour chacun des textes de notre base de données (voir l'annexe A). La dernière colonne du tableau présente les valeurs moyennes des taux de classification.

Tableau 5.11. Pourcentages de classification correcte des distances inter-tracés pour les classificateurs AdaBoost.Z (scénario personnalisé).

**Numéro de Texte**

| Numéro de classificateur | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *Moy.* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *9* | 98.4 | 99.4 | 99.8 | 100 | 99.4 | 99.8 | 98.9 | 98.6 | 99 | 99.7 | 97.1 | 99.3 | 98.4 | **99.06** |
| *3* | 97.8 | 100 | 99.8 | 100 | 99.4 | 99.8 | 98.3 | 99.1 | 98.6 | 99.2 | 96.2 | 99.3 | 97.8 | **98.87** |
| *8* | 97.3 | 98.7 | 99.8 | 100 | 99.4 | 99.6 | 98.7 | 99.1 | 99 | 98.6 | 97.5 | 98.3 | 98.4 | **98.80** |
| *6* | 99.5 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.9 | 99.1 | 98.6 | 97.5 | 96.7 | 98 | 96.2 | **98.63** |
| *7* | 97.3 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.3 | 99.1 | 98.6 | 97.5 | 97.1 | 98.6 | 97.3 | **98.58** |
| *5* | 97.8 | 98.7 | 99.8 | 100 | 99.4 | 99.8 | 98.5 | 99.1 | 98.6 | 97.5 | 96.2 | 98.6 | 97.3 | **98.56** |
| *2* | 99.5 | 98.7 | 99.8 | 100 | 99.4 | 99.4 | 97.2 | 99.1 | 98.6 | 97.5 | 96.2 | 98 | 96.2 | **98.43** |
| *4* | 99.5 | 98.7 | 99.8 | 100 | 99.4 | 99.4 | 97.2 | 99.1 | 98.6 | 97.5 | 96.2 | 98 | 96.2 | **98.43** |
| *1* | 80.8 | 98.7 | 84.9 | 100 | 99.4 | 99.8 | 85.4 | 99.1 | 98.6 | 97.5 | 96.2 | 98 | 96.2 | **94.96** |

Tableau 5.12. Mesures statistiques sur les intervalles de confiance des résultats moyens dans le tableau 5.11.

| Numéro de classificateur | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *9* | 0.80 | 99.06 | (98.67, 99.46) |
| *3* | 1.11 | 98.87 | (98.32, 99.42) |
| *8* | 0.82 | 98.80 | (98.40, 99.20) |
| *6* | 1.21 | 98.63 | (98.03, 99.23) |
| *7* | 1.03 | 98.58 | (98.07, 99.08) |
| *5* | 1.12 | 98.56 | (98.01, 99.11) |
| *2* | 1.30 | 98.43 | (97.79, 99.08) |
| *4* | 1.30 | 98.43 | (97.79, 99.08) |
| *1* | 6.61 | 94.97 | (91.70, 98.24) |

Dans le tableau 5.12, nous présenterons le résultat d'analyse statistique (voir la section 4.5.1.5.1) des taux de classification du tableau 5.11, incluant les estimés pour l'écart type et la moyenne et l'intervalle de confiance des moyennes des taux de classification correcte pour les textes de la base de test.

La figure 5.12 est une représentation graphique des valeurs moyennes et de leurs intervalles de confiance montrés dans le tableau 5.12.

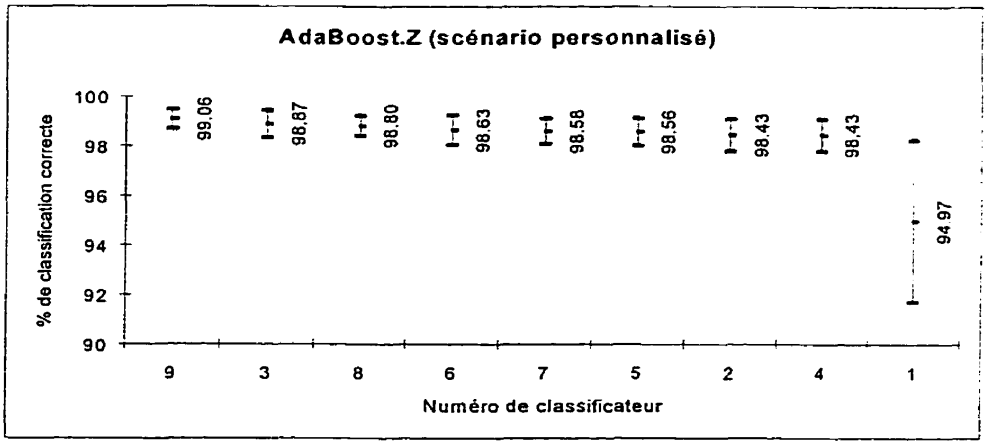



Figure 5.12. Les pourcentages moyens de classification correcte des distances inter-tracés et leurs intervalles de confiance pour les classificateurs AdaBoost.Z (scénario personnalisé).

## 5.3 CONCLUSION

Dans ce chapitre, nous avons présenté les classificateurs inspirés de l'algorithme AdaBoost pour la segmentation de textes manuscrits en mots. Nous avons opté pour ce type de classificateurs pour résoudre le problème des classificateurs neuronaux classiques concernant le compromis entre le nombre de patrons d'entraînement et la complexité de la structure des classificateurs. Dans une application en-ligne telle que notre système de segmentation, on souhaite avoir un entraînement rapide des classificateurs basé sur un petit nombre d'exemples et, en même temps, de très bons résultats de généralisation afin de minimiser l'effort des scripteurs pour la correction d'erreurs de segmentation. Nous avons ainsi exploité, pour la première fois, la capacité de l'algorithme d'AdaBoost pour la résolution de ce type de problème de classification.

Les sujets suivants ont été abordés dans ce chapitre:

1- Théorie de l'algorithme AdaBoost, incluant ses deux versions initiale (AdaBoost.M1) et améliorée (AdaBoost.Z);
2- Un interprétation bayésienne d'AdaBoost;
3- Classificateurs AdaBoost.M1 et AdaBoost.Z implantés pour la classification de distances inter-tracés, incluant nos propositions pour les améliorer par rapport à leurs versions théoriques;
4- Résultats de classification obtenus avec l'application des scénarios commun et personnalisé aux classificateurs AdaBoost.M1 et AdaBoost.Z implantés.

Nous présenterons dans le chapitre 6, une analyse des résultats obtenus dans ce chapitre et comparerons également ces derniers avec ceux des classificateurs neuronaux obtenus dans le chapitre 4.

# CHAPITRE 6

# ANALYSE ET DISCUSSION

Dans le chapitre 5, nous avons présenté les résultats de classification des distances inter-tracés obtenus avec les classificateurs AdaBoost.M1 et AdaBoost.Z. Nous avions déjà mentionné, qu'en proposant ces classificateurs, nous cherchions à résoudre le problème des classificateurs neuronaux simples qui nous empêchait d'utiliser toutes les caractéristiques proposées dans un scénario personnalisé afin d'obtenir une meilleure performance de segmentation.

Dans ce chapitre, à la lumière des résultats présentés dans la section 5.2 et ceux du chapitre 4, nous allons d'abord évaluer la performance des classificateurs basés sur l'algorithme d'AdaBoost et les comparer avec celle des classificateurs neuronaux. Après avoir déterminé le meilleur classificateur implanté, nous ferons une analyse plus profonde des paramètres et des cas d'erreurs de ce classificateur en particulier.

## 6.1. COMPARAISON GLOBALE DES RÉSULTATS DE CLASSIFICATION

Dans les chapitres 4 et 5, nous avons testé 102 classificateurs de distances inter-tracés dans sept groupes différents. Pour chaque groupe de classificateur, nous avons présenté les tableaux comparatifs des résultats de classification. Nous avons également comparé, à plusieurs endroits, la performance globale d'un groupe donné de classificateur avec celle d'un ou de plusieurs autres groupes.

Dans cette section, pour déterminer le(les) meilleur(s) classificateur(s) et combinaison(s) de caractéristiques, nous comparerons la performance de tous les

Tableau 6.1. Tableau comparatif des meilleurs résultats de classification.

| # | Type de classificateur | Scénario d'apprentissage | Entrées | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|---|---|---|
| 1 | AdaBoost.Z | Personnalisé | 7 entrées* | 0.80 | 99.06 | (98.67, 99.46) |
| 2 | AdaBoost.Z | Personnalisé | D_Prl | 1.11 | 98.87 | (98.32, 99.42) |
| 3 | AdaBoost.M1 | Personnalisé | D_Prl | 1.20 | 98.74 | (98.15, 99.33) |
| 4 | Neuronal | Personnalisé | RLmin | 1.58 | 98.66 | (97.88, 99.44) |
| 5 | AdaBoost.M1 | Commun | D_Prl , RLavg | 1.19 | 98.64 | (98.02, 99.26) |
| 6 | AdaBoost.M1 | Commun | D_Prl , RLavg | 1.39 | 98.63 | (97.90, 99.35) |
| 7 | Neuronal | Commun | D_Prl , RLavg | 0.98 | 98.63 | (98.12, 99.13) |
| 8 | Neuronal | Commun | D_Prl , RLmin | 1.73 | 98.40 | (97.50, 99.30) |
| 9 | AdaBoost.Z | Commun | 7 entrées* | 1.10 | 98.37 | (97.80, 98.94) |
| 10 | Neuronal | Personnalisé | D_Prl | 1.47 | 98.35 | (97.62, 99.07) |
| 11 | AdaBoost.Z | Commun | 7 entrées** | 1.22 | 98.31 | (97.68, 98.94) |
| 12 | AdaBoost.M1 | Personnalisé | D_Prl | 1.61 | 98.22 | (97.42, 99.02) |
| 13 | Neuronal | Commun | Dist_BB | 2.00 | 97.68 | (96.65, 98.72) |

7entrées*: 1- Dist_Paral, 2- RLmin, 3- RLavg, 4- prevNbExtrema, 5- pHeightExtrema, 6- curNbExtrema, 7- curHgtExtrema.

7entrées**: 1- Dist_Paral, 2- RLavg, 3- RLmin, 4- prevNbExtrema, 5- pHeightExtrema, 6- curNbExtrema, 7- curHgtExtrema.

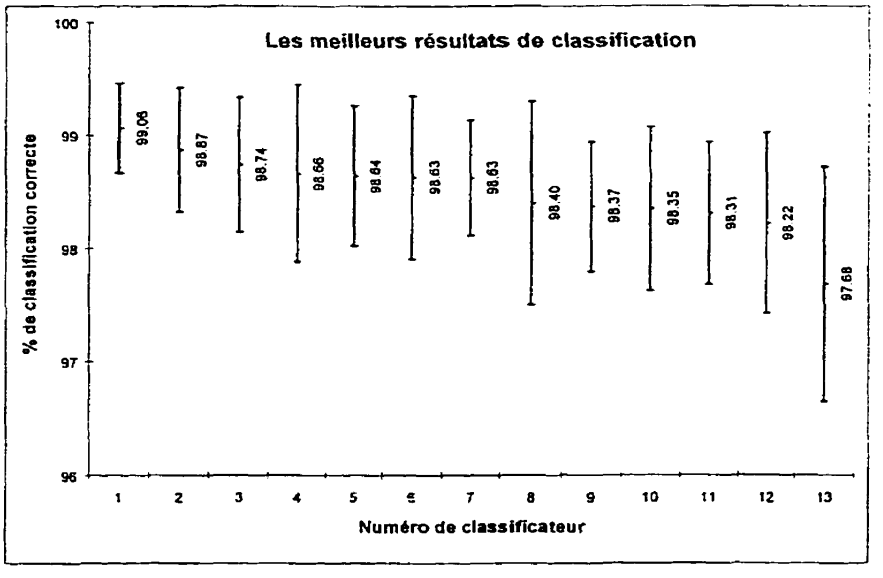



Figure 6.1. Pourcentages moyens de classification correcte et leurs intervalles de confiance pour les meilleurs classificateurs testés (voir le tableau 6.1).

classificateurs implantés. Pour ce faire, nous nous servirons du tableau 6.1 et de la figure 6.1. Le premier est un tableau comparatif des résultats de classification, contenant les meilleurs résultats de chaque groupe de classificateurs. La figure 6.1 est une représentation graphique des deux dernières colonnes du tableau 6.1.

Dans le tableau 6.1, la première colonne représente le classement des classificateurs et les colonnes de $\bar{y}_{med}$ et $(\bar{y}_{min}, \bar{y}_{max})$ contiennent les taux moyens de classification correcte et leurs intervalles de confiance, respectivement. $s$ représente l'écart type des taux moyens.

Dans un premier temps, nous présenterons le meilleur classificateur implanté et ensuite, nous comparerons les classificateurs AdaBoost et neuronaux pour les scénarios d'apprentissage commun et personnalisé.

## 6.1.1. MEILLEUR CLASSIFICATEUR IMPLANTÉ

La conclusion la plus importante et également la plus intéressante que l'on peut tirer des résultats du tableau 6.1, est la suivante : la meilleure performance de segmentation appartient à un classificateur AdaBoost.Z qui utilise les sept caractéristiques et qui est entraîné avec le scénario personnalisé. La supériorité de ce classificateur réside dans plusieurs aspects :

(1) Le meilleur taux moyen de classification correcte;
(2) La meilleur stabilité de résultats;
(3) L'utilisation de toutes les caractéristiques extraites;
(4) Le nombre d'hypothèses très bas;
(5) Le petit nombre de patrons utilisé dans la phase d'apprentissage.

Le classificateur mentionné nous a permis d'obtenir un taux moyen de classification $\bar{y}_{med}$ très élevé (environ 99.1%). En plus, il est le classificateur le plus robuste parmi tous les classificateurs testés : avec le plus petit écart type de valeurs

moyennes ($s$ = 0.8) et donc le plus petit intervalle de confiance de taux moyen ($\bar{y}_{min}$ , $\bar{y}_{max}$), il a produit le résultat le plus stable.

Un autre point fort de ce classificateur est qu'il utilise toutes les caractéristiques extraites : cet aspect, combiné avec l'application du scénario personnalisé, permet d'avoir la meilleure adaptabilité à l'écriture des scripteurs. La méthode que nous avons utilisée pour combiner les hypothèses simples, consiste à attribuer une caractéristique à chaque hypothèse et à consacrer plusieurs hypothèses consécutives à une caractéristique donnée. En appliquant le scénario personnalisé pour entraîner ce classificateur, nous permettons aux paramètres et surtout au poids de chaque hypothèse simple, dans l'ensemble des hypothèses, de s'adapter aux styles d'écriture utilisés dans les différents textes. Et comme on peut le voir dans la section 6.2, en présentant toutes les caractéristiques au classificateur, nous lui offrons les avantages d'une combinaison adaptative de caractéristiques : il peut choisir pour chaque texte, la (les) caractéristique(s) qui permettent de mieux extraire ses particularités.

Ce classificateur est composé d'un petit nombre d'hypothèses : seulement cinq hypothèses/caractéristique au maximum. Ceci constitue un grand avantage, puisqu'il permet d'avoir un apprentissage rapide et une classification encore plus rapide. D'après l'équation (5.27), le nombre maximal d'opérations pour classifier un patron est égal à :

$$Max\_N_{op} = 70\ Multiplication + 36\ Addition \tag{6.1}$$

puisque $N$ = 7 et $T$ = 5. En comparant ce nombre avec celui d'un classificateur K plus proches voisins, qui utilise seulement 100 patrons de référence[1], on constate que le nombre d'opérations de notre meilleur classificateur implanté est plus de dix fois plus petit que celui du classificateur K plus proches voisins (calculé selon l'équation (D.1)).

Et, finalement, seulement 70 patrons sont utilisés pour entraîner ce classificateur; ce qui permet d'appliquer le scénario personnalisé en utilisant un court texte. Il s'agit

[1] Ce nombre de patrons de référence est, en principe, insuffisant pour un tel classificateur, voir l'annexe D.

d'un point fort très important qui permet de bien alléger la part des scripteurs pour entraîner le classificateur.

### 6.1.2. CLASSIFICATEURS ADABOOST VERSUS NEURONAUX : SCÉNARIOS COMMUN ET PERSONNALISÉ

Nous commencerons par comparer le scénario commun des classificateurs neuronaux et AdaBoost. Comme on pouvait le prévoir, les résultats présentés dans le tableau 6.1, ainsi que ceux des tableaux 4.8, 5.3 et 5.9 montrent que dans le cas des classificateurs utilisant une ou deux caractéristiques, les classificateurs AdaBoost ne sont pas plus performants que les classificateurs neuronaux. En d'autres termes, l'utilisation de plusieurs experts pour la classification des patrons pouvant être classifiés (avec une erreur acceptable) à l'aide d'un seul expert, n'apporte pas d'amélioration dans la performance de classification.

Toutefois, les résultats des tableaux 6.1, 4.10, 5.6 et 5.12 montrent l'avantage des classificateurs AdaBoost sur les classificateurs neuronaux dans le cas du scénario personnalisé. D'après ces résultats, les taux de classification correcte des classificateurs AdaBoost.Z avec une ou deux entrées sont plus élevés que ceux de leurs classificateurs neuronaux correspondants. En plus, grâce à l'algorithme d'AdaBoost, nous avons réussi à appliquer le scénario personnalisé à un classificateur avec toutes les caractéristiques et avec un petit nombre de patrons d'apprentissage, permettant d'obtenir des résultats impressionnants. Nous avions déjà souligné l'utilité de l'algorithme AdaBoost lorsque les patrons d'entraînement sont peu nombreux. C'est alors le cas du scénario personnalisé et les résultats obtenus prouvent l'exactitude de cet énoncé.

## 6.2. ANALYSE DÉTAILLÉE DU MEILLEUR CLASSIFICATEUR IMPLANTÉ

Dans la section précédente, nous avons identifié le meilleur classificateur implanté et avons également présenté ses points forts. Dans cette section, nous allons l'étudier plus profondément en mettant l'accent sur ses paramètres et sa performance de segmentation.

### 6.2.1. PARAMÈTRES DU CLASSIFICATEUR

Puisque les valeurs des paramètres du meilleur classificateur implanté (le classificateur AdaBoost.Z avec sept entrées) sont obtenues dans un scénario d'apprentissage personnalisé, elles peuvent varier d'un texte à l'autre (pour mieux s'adapter aux caractéristiques des textes). En fait, le classificateur détermine les paramètres attribués à chaque caractéristique en fonction de l'importance et de la difficulté de classification des patrons. Le tableau 6.2 montre, à titre d'exemple, les paramètres de deux classificateurs entraînés et testés l'un sur le texte no. 1 (figure A.1) et l'autre sur le texte no. 11 (figure A.11). Ce tableau contient pour toutes les caractéristiques, le poids ($\omega$) et le biais ($B$) de chacune des hypothèses simples (dont le numéro est montré par $t$) ainsi que le paramètre $\alpha$ servant à pondérer la décision de chaque hypothèse pour obtenir la décision finale.

Le tableau 6.2 prouve que, non seulement la valeur des paramètres reliés à une caractéristique donnée, mais aussi les caractéristiques impliquées dans le calcul du classificateur final, peuvent varier d'un texte à l'autre. Par exemple, pour le texte no. 1, les caractéristiques *RLavg, prevNbExtrema* et *curNbExtrema* et pour le texte no. 11, *RLmin* et *prevHeightExtrema* sont exclues du classificateur final. Ceci est le résultat de l'algorithme que nous avons conçu et implanté pour entraîner les classificateurs AdaBoost.Z (voir la figure 5.9). Dans l'étape 7 de cet algorithme, si la performance de classification d'une hypothèse simple, qui utilise seulement une caractéristique, est pire que celle d'une hypothèse aléatoire, la génération des hypothèses pour la caractéristique courante sera arrêtée et l'algorithme continuera avec l'entraînement d'autres hypothèses

Tableau 6.2. Paramètres des classificateurs AdaBoost.Z (no. 9 du tableau 5.12) pour le premier et le onzième textes de notre base de données (figures A.1 et A.11 de l'annexe A).

| Paramètres / Caractéristique | Texte no. 1 | | | | Texte no. 11 | | | |
|---|---|---|---|---|---|---|---|---|
| | $t$ | $\omega$ | $B$ | $\alpha$ | $t$ | $\omega$ | $B$ | $\alpha$ |
| *Dist_Paral* | 1 | 0.344 | -0.757 | 1.032 | 1 | 0.211 | -1.311 | 0.876 |
| | 2 | 0.344 | -1.606 | 0.985 | 2 | 0.212 | -1.767 | 0.701 |
| | 3 | 0.327 | -1.687 | 0.604 | 3 | 0.169 | -1.458 | 0.361 |
| | 4 | 0.247 | -1.115 | 0.259 | 4 | 0.092 | -0.725 | 0.138 |
| | 5 | 0.163 | -0.665 | 0.151 | 5 | 0.057 | -0.389 | 0.074 |
| *RLmin* | 1 | 0.055 | -0.305 | 0.049 | *NA* | *NA* | *NA* | *NA* |
| | 2 | 0.048 | -0.256 | 0.042 | *NA* | *NA* | *NA* | *NA* |
| | 3 | 0.045 | -0.238 | 0.038 | *NA* | *NA* | *NA* | *NA* |
| | 4 | 0.043 | -0.226 | 0.035 | *NA* | *NA* | *NA* | *NA* |
| | 5 | 0.041 | -0.217 | 0.033 | *NA* | *NA* | *NA* | *NA* |
| *RLavg* | *NA* | *NA* | *NA* | *NA* | 1 | -0.725 | -0.223 | 0.006 |
| | *NA* | *NA* | *NA* | *NA* | 2 | -0.744 | -0.319 | 0.002 |
| | *NA* | *NA* | *NA* | *NA* | 3 | -0.760 | -0.406 | 0.002 |
| | *NA* | *NA* | *NA* | *NA* | 4 | -0.774 | -0.485 | 0.002 |
| | *NA* | *NA* | *NA* | *NA* | 5 | -0.785 | -0.558 | 0.001 |
| *prevNbExtrema* | *NA* | *NA* | *NA* | *NA* | 1 | -0.028 | 0.274 | 0.044 |
| | *NA* | *NA* | *NA* | *NA* | 2 | -0.037 | 0.418 | 0.056 |
| | *NA* | *NA* | *NA* | *NA* | 3 | -0.040 | 0.461 | 0.056 |
| | *NA* | *NA* | *NA* | *NA* | 4 | -0.039 | 0.460 | 0.053 |
| | *NA* | *NA* | *NA* | *NA* | 5 | -0.037 | 0.444 | 0.048 |
| *prevHeightExtrema* | 1 | -0.032 | 0.274 | 0.057 | *NA* | *NA* | *NA* | *NA* |
| | 2 | -0.069 | 0.773 | 0.122 | *NA* | *NA* | *NA* | *NA* |
| | 3 | -0.095 | 1.098 | 0.151 | *NA* | *NA* | *NA* | *NA* |
| | 4 | -0.111 | 1.300 | 0.156 | *NA* | *NA* | *NA* | *NA* |
| | 5 | -0.120 | 1.411 | 0.147 | *NA* | *NA* | *NA* | *NA* |
| *curNbExtrema* | *NA* | *NA* | *NA* | *NA* | 1 | 0.112 | -0.658 | 0.090 |
| | *NA* | *NA* | *NA* | *NA* | 2 | 0.099 | -0.581 | 0.073 |
| | *NA* | *NA* | *NA* | *NA* | 3 | 0.088 | -0.519 | 0.061 |
| | *NA* | *NA* | *NA* | *NA* | 4 | 0.080 | -0.471 | 0.053 |
| | *NA* | *NA* | *NA* | *NA* | 5 | 0.074 | -0.435 | 0.046 |
| *curHeightExtrema* | 1 | -0.039 | 0.379 | 0.008 | 1 | -0.018 | 0.149 | 0.036 |
| | 2 | -0.031 | 0.307 | 0.007 | 2 | -0.038 | 0.470 | 0.069 |
| | 3 | -0.027 | 0.262 | 0.006 | 3 | -0.047 | 0.604 | 0.077 |
| | 4 | -0.023 | 0.235 | 0.005 | 4 | -0.049 | 0.646 | 0.075 |
| | 5 | -0.021 | 0.218 | 0.005 | 5 | -0.049 | 0.643 | 0.069 |

à l'aide de la caractéristique suivante. Or, si lors du passage d'une caractéristique à l'autre, la caractéristique courante ne permet pas de générer une hypothèse meilleure qu'une hypothèse aléatoire, cette caractéristique sera totalement exclue de la structure finale du classificateur. Ceci est un point fort de notre algorithme qui permet de construire des classificateurs adaptatifs en fonction du degré de représentativité de chacune des caractéristiques dans la procédure de segmentation.

## 6.2.2. PERFORMANCE DU CLASSIFICATEUR

Dans cette section, nous analyserons la performance du meilleur classificateur implanté dans les cas difficiles de segmentation. Nous présenterons le pourcentage des cas corrigés sur toutes les erreurs du meilleur classificateur conventionnel, suivi des améliorations pour chaque catégorie d'erreur de segmentation et de la performance du classificateur dans le cas de textes difficiles, hors de notre base de données.

### 6.2.2.1. POURCENTAGE TOTAL DE CORRECTION D'ERREURS

Les résultats de classification présentés dans les tableaux des chapitres 4 et 5 reflètent la performance globale des classificateurs sur l'ensembles des patrons de la base de test. Une mesure plus spécifique d'évaluation de la performance d'un classificateur est sa capacité de correction des erreurs d'un autre classificateur. Comme il a été mentionné dans la section 4.5.1.6, nous avions identifié les cas d'erreurs de segmentation de la meilleure méthode de segmentation conventionnelle. Pour la meilleure méthode implantée, nous avons calculé le taux de correction totale des erreurs de segmentation : il s'agit d'une amélioration de près de 79% sur les cas problématiques de la meilleure méthode conventionnelle; ce qui montre que, grâce à nos propositions, une portion très importante des cas problématiques de segmentation a été éliminée.

Tableau 6.3. Pourcentages de correction de trois types d'erreurs obtenus avec le meilleur classificateur implanté.

| *Type d'erreur* | *Pourcentage de correction* |
|---|---|
| **Signes diacritiques** | **87.8** |
| **Inclinaison** | **100** |
| **Composition-Classification** | **72.8** |

## 6.2.2.2. CORRECTION DES TROIS TYPES D'ERREURS

Dans la section 4.5.1.6, nous avons identifié les trois catégories d'erreurs de segmentation pour la meilleure méthode conventionnelle : il s'agit des erreurs causées par les signes diacritiques, l'inclinaison d'écriture et l'imprécision de la composition ou de classification. Nous avons également calculé les pourcentages de ces erreurs, présentés dans le tableau 4.5. Dans cette section, nous présenterons les pourcentages et les exemples concernant la correction de ces erreurs grâce au meilleur classificateur implanté.

Le tableau 6.3 montre les pourcentage de correction de ces catégories d'erreur dans l'ensemble des patrons classifiés par notre meilleur classificateur. Selon ces résultats, une portion très élevée des erreurs de segmentation du meilleur classificateur conventionnel est corrigée.

### *6.2.2.2.1. ERREURS CAUSÉES PAR LES SIGNES DIACRITIQUES*

Quant à cette catégorie, l'implantation d'un module performant de reconnaissance de ces entités, nous a permis de corriger près de 88% des erreurs. La figure 6.2 montre, dans cette catégorie, des exemples des erreurs corrigées et non corrigées.

Dans la figure 6.2(a), la barre de « t » cause un chevauchement horizontal des tracés et la meilleure méthode conventionnelle regroupe les mots « faut » et « nous » ensemble dans un mot. Toutefois, l'ajout du module de reconnaissance des signes diacritiques permet d'éliminer le chevauchement et de mesurer l'espacement entre la partie principale de ces mots. Il faut quand même ajouter que, dans ce cas particulier, en

(a) (b) (c)

Figure 6.2. Exemples des erreurs corrigées et non corrigées de segmentation après la reconnaissance des signes diacritiques : (a) et (b) erreurs corrigées et (c) erreur non corrigée.

raison de l'inclinaison d'écriture, le calcul de la distance entre les parallélogrammes, plutôt que de la distance entre les rectangles, contribue également à corriger l'erreur de segmentation.

Dans le cas de la figure 6.2(b), la reconnaissance de l'accent de « à » a permis de mesurer la distance entre les tracés « mis » et « a », qui est ensuite reconnue correctement comme une distance inter-mots. Ce même accent était la source d'erreur de la meilleure méthode conventionnelle (voir la figure 4.6(a)).

Un léger pourcentage (12%) des erreurs de ce type existe encore dans les résultats du meilleur classificateur. La figure 6.2(c) montre un exemple de ces erreurs non corrigées où « , » n'a pas été reconnu. La raison de cette erreur de reconnaissance est la suivante : la hauteur du tracé « , » est comparable à celle des extrema du tracé « images ». Rappelons que, dans la procédure de reconnaissance des signes diacritiques (figure 3.20), pour un tracé simple comme « , » avec deux extrema, si sa hauteur n'est pas suffisamment petite par rapport à la hauteur moyenne des extrema du mot précédent et que le tracé n'est pas situé en dessous de ce mot, il ne sera pas considéré comme un Accent_Point_TBar. En fait, cette combinaison de test a été ajoutée à l'algorithme pour empêcher que des lettres comme « i » qui sont simples en structure et souvent petites ne soient reconnues comme un signe diacritique. Les résultats de reconnaissance montrent que ces vérifications permettent de résoudre le problème de ces lettres. Toutefois, le prix à payer est d'avoir des situations comme celles de la figure 6.2(c). Cependant, ces dernières se produisent rarement.

Figure 6.3. Exemples de correction d'erreurs de segmentation causées par l'inclinaison d'écriture.

#### *6.2.2.2.2. ERREURS CAUSÉES PAR L'INCLINAISON DES TRACÉS*

Le tableau 6.3 montre que, dans les résultats obtenus avec le meilleur classificateur implanté, toutes les erreurs de cette catégorie sont corrigées; ce qui démontre la capacité de notre module d'estimation de l'inclinaison.

La figure 6.3 montre des exemples d'erreurs corrigées dans cette catégorie. Le calcul précis des inclinaisons des tracés et le remplacement des rectangles englobant par des parallélogrammes de cette inclinaison, ont permis de segmenter les tracés dans trois mots différents, alors que dans les résultats du meilleur classificateur conventionnel, ils se trouvaient tous dans un mot (voir la figure 4.6(b)).

#### *6.2.2.2.3. ERREURS CAUSÉES PAR L'IMPRÉCISION DE LA COMPOSITION OU DE LA CLASSIFICATION*

Comme le tableau 6.3 le démontre, près de 73% des erreurs de cette catégorie ont été corrigées. Cette amélioration est obtenue grâce à la capacité de classification de l'algorithme d'AdaBoost.Z implanté qui permet de trouver une frontière de décision plus précise en mettant l'accent sur les patrons les plus difficiles à apprendre, positionnés près de la frontière de décision (dans la section 6.1.1, nous avons énuméré les points forts du classificateur ayant permis d'obtenir le meilleur résultat de segmentation de texte en mots).

Bien évidemment, le travail fait pour améliorer la précision de classification permet de corriger uniquement les erreurs causées par la procédure de classification, plutôt que celles causées par l'imprécision de la composition du texte par le scripteur.

(a) (b) (c)

Figure 6.4. Exemples d'erreurs corrigées et non corrigée de segmentation pour la catégorie des erreurs causées par l'imprécision de classification: (a) et (b) erreurs corrigées et (c) erreur non corrigée.

Cette partie d'erreurs, produites par les irrégularités des espacements (comme un espacement inter-mots trop petit par rapport aux autres espacements inter-mots dans le texte ou un espacement intra-mot trop grand par rapport aux autres espacements) sont considérées comme des erreurs inévitables produites par les patrons situés dans la région de chevauchement des distributions des caractéristiques. Environ 50% des erreurs non corrigées de la troisième catégorie d'erreurs sont de cette nature.

La figure 6.4 montre des exemples d'erreurs corrigées et non corrigée dans cette catégorie. Dans le cas de la figure 6.4(a), le meilleur classificateur conventionnel reconnaît l'espacement entre les tracés « n » et « ces » comme une distance inter-mots et divise le mot « sciences » en deux mots (voir la figure 4.6(c)), alors que cette erreur est corrigée par notre meilleur classificateur. Dans la figure 6.4(b) également, la distance entre les deux lettres « I » et « f » est considérée correctement comme une distance intra-mot par le meilleur classificateur implanté, alors que le meilleur classificateur conventionnel place ces lettres dans deux mots distincts. La figure 6.4(c) montre un exemple des erreurs de composition non corrigées par le meilleur classificateur implanté. En fait, dans cet exemple, l'espacement entre les deux mots « toutes » et « nos » est trop petit (relativement aux autres espacements intra-mot dans le même texte) pour être reconnu comme une distance inter-mots.

### 6.2.2.3. PERFORMANCE DU CLASSIFICATEUR SUR D'AUTRES TEXTES DIFFICILES

Après avoir entraîné et testé notre meilleur classificateur implanté sur notre base de données, nous avons évalué sa performance sur d'autres textes. Les textes de notre base de données sont des textes écrits librement par différents sujets et représentent leur style d'écriture naturel. Toutefois, les trois textes présentés dans les figures 6.5, 6.6 et 6.7 sont des textes intentionnellement composés de cas difficiles de segmentation. Ces textes, qui ne font pas partie de notre base de données de prototypage (voir l'annexe A), ont été utilisés exclusivement pour tester le meilleur classificateur implanté et comparer sa performance avec celle du meilleur classificateur conventionnelle. Bien qu'ils n'aient pas l'apparence de vrais textes manuscrits, ces textes sont utiles puisqu'ils permettent de mieux contraster la différence entre les performances des meilleurs classificateurs conventionnel et proposé.

Le texte «ext1» montré dans la figure 6.5 a les particularités suivantes : (1) l'écriture est inclinée, (2) les mots sont très proches les uns des autres et (3) les signes diacritiques, comme les barres de « t », qui couvrent les espacements inter-mots, sont nombreux.

Le texte «ext2» montré dans la figure 6.6 a les particularités suivantes : (1) la grandeur d'écriture est variable, (2) les espacements inter-tracés varient avec la grandeur de texte et (3) les signes diacritiques, comme les barres de « t », qui couvrent les espacements inter-mots sont nombreux.

Le texte «ext3» montré dans la figure 6.7 a les particularités suivantes : (1) Il n'est pas intelligible, (2) la grandeur d'écriture est variable et peut même changer dans une ligne, (3) l'écriture est inclinée, son inclinaison varie et peut même changer de signe (par rapport à la verticale) dans une ligne, (4) les espacements inter-tracés sont variables, même dans une ligne, et (5) l'écriture dans une ligne peut changer de niveau (position verticale), (6) les signes diacritiques, comme les barres de « t », qui couvrent les espacements inter-mots sont nombreux.

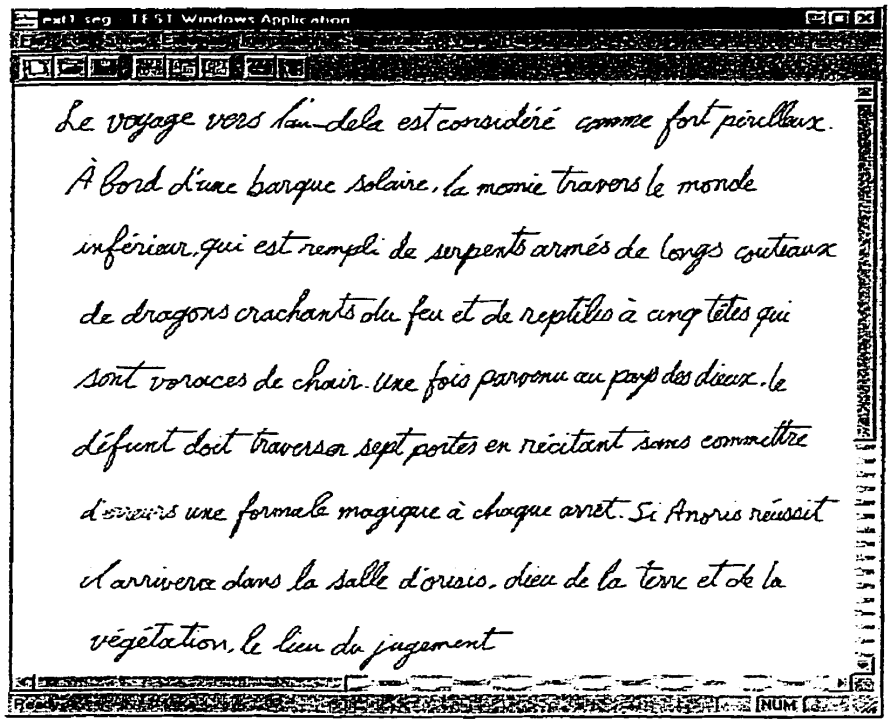



Figure 6.5. Texte difficile «ext1».

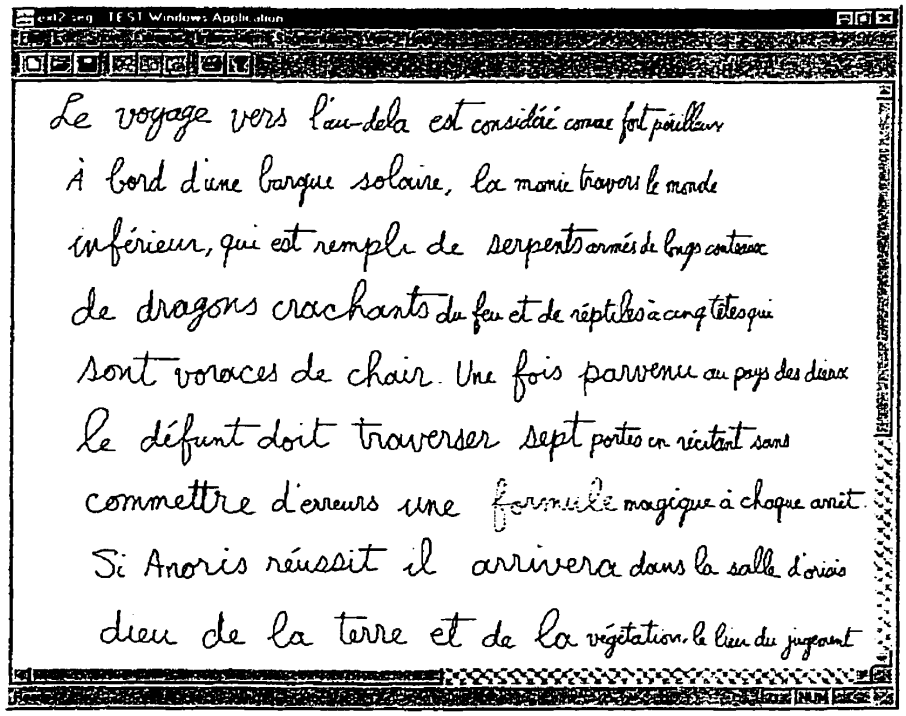



Figure 6.6. Texte difficile «ext2».

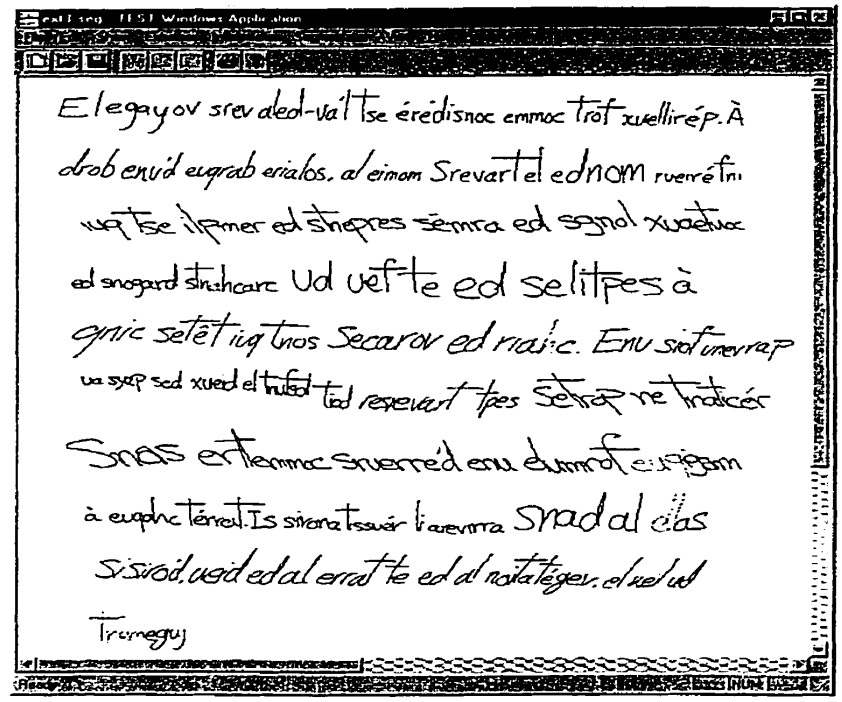
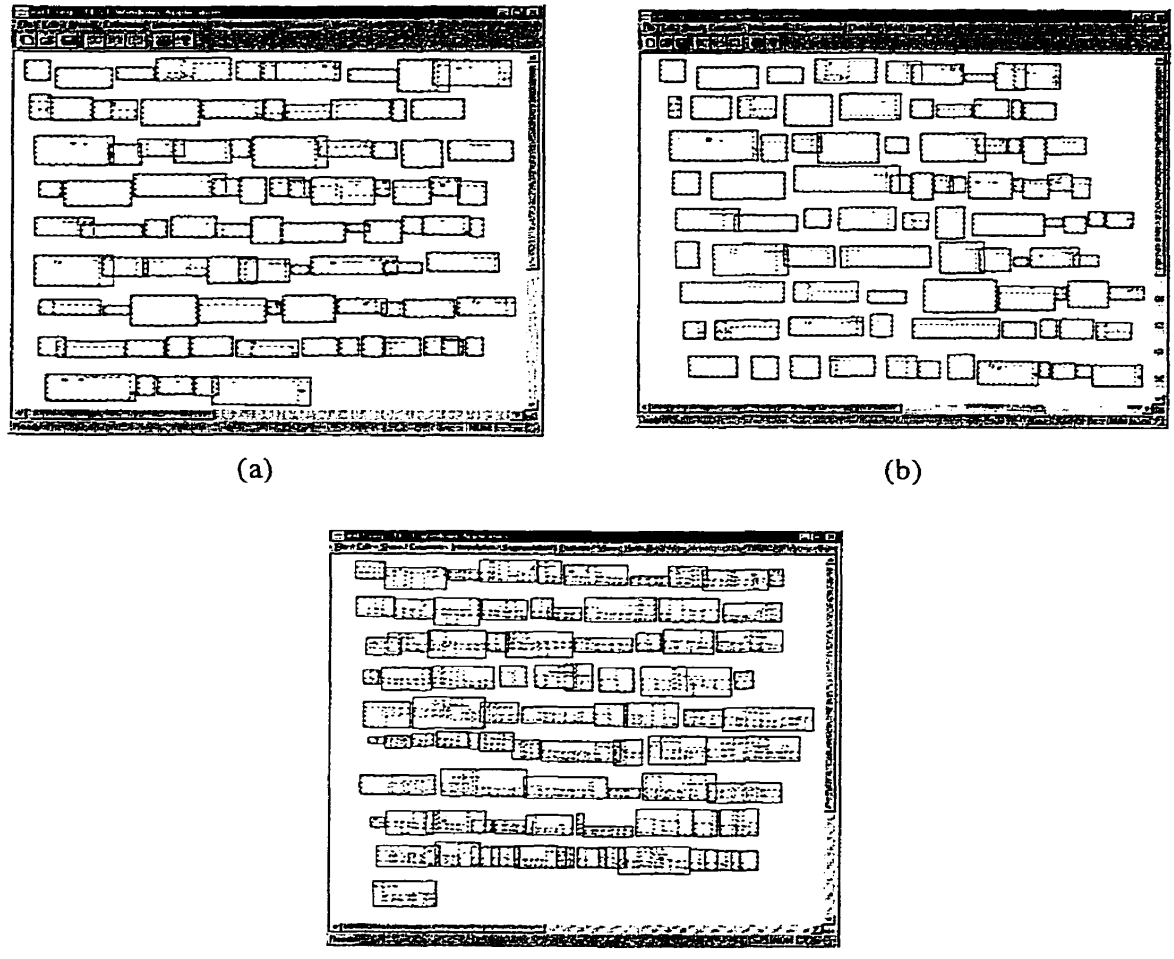


Figure 6.7. Texte difficile «ext3».

En plus, les textes utilisés ont différents styles d'écriture : les deux premiers sont d'écriture cursive et le troisième est d'écriture isolée.

Les figures 6.8(a) à (c) sont présentées comme indicateurs visuels de difficulté de segmentation de textes des figures 6.5 à 6.7 avec une méthode conventionnelle. Elles montrent les rectangles englobant des tracés et des mots se trouvant dans les textes. Comme nous l'avons déjà mentionné, les textes des figures 6.5 à 6.7 sont utilisés pour comparer la performance du meilleur classificateur implanté avec celle du meilleur classificateur conventionnel sur des exemples autres que ceux de notre base de données. Les pourcentages de classification correcte des distances inter-tracés, pour ces classificateurs, sont présentés dans le tableau 6.4. Ces résultats montrent encore une fois, la supériorité de notre approche sur les méthodes conventionnelles de segmentation. Comme on peut le voir dans le tableau 6.4, notre meilleur classificateur produit des taux de classification correcte bien élevés pour ces textes très difficiles à segmenter. Les



(a) (b)

(c)

Figure 6.8. Les rectangles englobant des tracés (pointillés) et des mots : (a) pour le texte de la figure 6.5, (b) pour le texte de la figure 6.6 et (c) pour le texte de la figure 6.7.

pourcentages de réussite du meilleur classificateur conventionnel sont bas; ce qui est la meilleure preuve de la difficulté de segmentation de ces textes. En fait, la conclusion la plus importante que l'on peut tirer du tableau 6.4 est le grand écart entre les résultats obtenus avec les deux méthodes conventionnelle et proposée, lequel varie entre 8% et 34.9%. Cela montre que notre meilleur classificateur proposé produit systématiquement

Tableau 6.4. Taux de classification correcte pour les meilleurs classificateurs conventionnels et proposés pour les textes des figures 6.5 à 6.7.

| Spécifications de classificateurs | | Textes et résultats de segmentation | | |
|---|---|---|---|---|
| Type | Entrée(s) | Ext1 | Ext2 | Ext3 |
| neuronaux | Dist_BB | 63.5% | 78.9% | 89.4% |
| AdaBoost | 7 caractéristiques proposées | 98.4% | 97.4% | 97.4% |

des résultats meilleurs que ceux du meilleur classificateur conventionnel et que sa performance est bien plus stable.

# 6.3 CONCLUSION

Dans ce chapitre, nous avons analysé la performance des classificateurs basés sur l'algorithme d'AdaBoost et nous les avons comparé avec celle des classificateurs neuronaux.

Les sujets suivants ont été abordés dans ce chapitre:

1- Comparaison globale des résultats obtenus avec les classificateurs neuronaux et AdaBoost implanté, incluant un tableau comparatif des meilleurs résultats de classification;

2- Comparaison entre la performance des classificateurs neuronaux, AdaBoost.M1 et AdaBoost.Z, à la lumière des résultats empiriques présentés dans les chapitres 4 et 5;

3- Analyse détaillée du meilleur classificateur implanté, incluant ses paramètres internes et sa performance par rapport au meilleur classificateur conventionnel dans les cas difficiles de segmentation et pour les textes difficiles autres que ceux de notre base de données. Dans cette analyse de performance, nous nous sommes basés sur les résultats de plusieurs études qualificatives et quantitatives sur l'ensemble des erreurs, de façon générale, et les différents types d'erreurs, de façon spécifique.

À partir de cette analyse, nous pouvons tirer les conclusions suivantes :

1- Les meilleurs résultats de classification, parmi tous les classificateurs implantés, sont obtenus avec l'application du scénario personnalisé à un classificateur AdaBoost.Z, en utilisant l'ensemble des caractéristiques proposées.
2- Le meilleur classificateur implanté possède, à la fois le taux moyen de classification le plus élevé (presque 99.1%), et l'intervalle de confiance de taux moyen le plus petit (un écart type de seulement 0.8%) de tous les classificateurs.
3- En général, près de 80% des cas d'erreurs du meilleur classificateur conventionnel ont été corrigés.
4- Dans un contexte plus spécifique, nous avons réussi à corriger près de 88% des erreurs causées par les signes diacritiques, 100% des erreurs causées par l'inclinaison d'écriture et près de 73% des erreurs causées par l'imprécision dans la composition du texte ou la classification des distances.
5- En plus des tests effectués sur les textes de notre base de données, les résultats obtenus avec d'autres textes difficiles à segmenter, ont montré la supériorité de notre meilleur classificateur proposé sur le meilleur classificateur conventionnel et ce, en terme de taux de classification correcte et de stabilité des résultats de segmentation.
6- Ces résultats sont obtenus avec seulement 70 patrons d'apprentissage par texte et cinq hypothèses générées (au maximum) par caractéristique; ce qui permet d'avoir des phases d'apprentissage et de test assez rapides : une situation idéale pour un système en-ligne.
7- Un point fort du meilleur classificateur est l'utilisation de l'ensemble des caractéristiques extraites, qui permet d'exploiter toutes les informations spatiales et sémantiques disponibles pour augmenter la précision de la segmentation. Ces informations sont reliées aux mesures de distances

extraites, à la grandeur et à la longueur des mots et de tracés impliqués dans la procédure de segmentation.

8- L'application du scénario personnalisé d'apprentissage sur le meilleur classificateur grâce à l'algorithme AdaBoost.Z, conçu et réalisé dans ce travail, permet d'obtenir la meilleure flexibilité et adaptabilité d'un système de segmentation en-ligne de textes manuscrits en mots, dans un contexte multi-scripteur.

# CHAPITRE 7
# CONCLUSION GÉNÉRALE

Cette thèse présente les travaux de recherche effectués pour la conception et la réalisation d'un système original et efficace de segmentation en-ligne de textes manuscrits en mots. Ce travail s'inscrit dans le cadre d'un système de traitement en-ligne de structures de documents manuscrits. À ce sujet, nous avons proposé le schéma du système complet, incluant les modules de reconnaissance et d'édition de structures physique et logique. Quant à la représentation interne de la structure logique, nous avons présenté la forme des structures logiques générique (DTD) et spécifique d'un document manuscrit, en conformité avec la norme SGML.

L'emphase a été mise sur les modules de reconnaissance et d'édition physique, qui jouent un rôle primordial dans le système complet. Nous avons ainsi présenté les spécifications des modules conçus et réalisés, y compris les stratégies adoptées pour la reconnaissance des lignes et des blocs (des objets physiques) et les commandes d'édition de la structure physique. Dans la phase de conception, notamment celle du système de reconnaissance, nous avons tenté d'offrir aux scripteurs, une liberté temporelle et séquentielle maximale pour la composition de texte, tout en respectant les contraintes des systèmes en-ligne. En fait, la difficulté principale constituait le compromis entre la liberté des scripteurs exigeant le minimum de contraintes, et la forme structurelle ne pouvait être atteinte qu'en imposant certaines contraintes aux scripteurs. Nous devions donc trouver la(les) contrainte(s) à la fois tolérable(s) et fonctionnelle(s). Nous avons ainsi décidé de permettre aux scripteurs de choisir librement le temps et la séquence de

composition de textes. La seule contrainte imposée constitue la composition du texte entre les lignes horizontales pré-dessinées. Il s'agit d'une contrainte tolérable et minimale, puisqu'elle donne aux scripteurs la sensation d'écrire sur un papier ligné auquel ils sont habitués. De plus, le scripteur peut choisir la distance entre les lignes, peut même écrire incliné et changer la grandeur de l'écriture dans une ligne de texte. Ces lignes nous permettent de reconnaître deux entités principales de la structure physique, c'est-à-dire les lignes et les blocs du texte.

L'accent a donc été mis sur l'autre entité de la structure physique : les mots. Cette thèse se penche, principalement, sur le problème de la segmentation en-ligne de textes manuscrits en mots. Il s'agit de la phase à la fois la plus importante et la plus compliquée de la procédure de reconnaissance de la structure physique. Le module de segmentation doit, entre chaque levée et pose de crayon, prendre une décision sur le type de transition (espacement inter-tracés).

Dans un style d'écriture libre, la difficulté principale de la segmentation réside en l'interprétation des espacements inter-tracés, constituant une traduction fiable du geste du scripteur à une(des) mesure(s) d'espacement. En somme, on doit choisir, à partir de cette(ces) mesure(s) et éventuellement d'autres propriétés de l'écriture, la(les) caractéristique(s) qui permet(tent) d'effectuer une classification précise des espacements inter-tracés. Nous devons rappeler que dans le choix des mesures et des caractéristiques, il faut respecter les contraintes des systèmes en-ligne, notamment les calculs uniquement basés sur les entités précédentes (dans le temps) et la rapidité dans l'extraction des caractéristiques. Une autre difficulté consiste à concevoir une méthode de classification qui soit à la fois efficace et rapide, nécessitant, de plus, peu d'exemples pour l'apprentissage des particularités des espacements inter-tracés. La méthode choisie doit également être, autant que possible, adaptative, pour tenir compte des divers styles d'écriture. Elle doit offrir aux scripteurs la liberté de composition de textes et doit minimiser leurs tâches pour l'entraînement du système, en minimisant la taille de base d'apprentissage nécessaire et, pour la correction des erreurs, en minimisant le nombre d'erreurs de segmentation.

Comme le premier pas vers la conception d'un système de segmentation, nous avons effectué une étude bibliographique sur les méthodes de segmentation existantes, notamment sur les caractéristiques utilisées dans ces méthodes. Nous avons présenté une taxinomie originale des méthodes de segmentation et nous en avons souligné les points forts et faibles. Dans une étude comparative, nous avons choisi les meilleures caractéristiques conventionnelles pour une application en-ligne. Il s'agit de la distance entre les rectangles englobant Dist_BB et des distances horizontales minimale RLmin et moyenne RLavg entre les tracés et les mots. Toutefois, Nous avons démontré que Dist_BB souffre du problème de la sous-estimation des distances inter-tracés, causée principalement par les tracés inclinés et les barres de « t ». Les deux autres mesures de distances, RLmin et RLavg, sont sensibles au chevauchement vertical des tracés. En plus, les accents et points situés au-dessus ou en dessous de la zone médiane des tracés sont souvent les sources de la surestimation ou l'inexistence des distances horizontales.

Nous avons ensuite présenté nos propositions pour résoudre les problèmes des mesures conventionnelles des distances inter-tracés. En principe, il s'agit d'effectuer, premièrement, une phase préalable de reconnaissance des signes diacritiques pour les enlever du texte et, deuxièmement, de remplacer Dist_BB par la distance entre les parallélogrammes englobant Dist_Paral. Les difficultés reliées à ces traitements ont été soulignées, notamment la nécessité des calculs rapides, la quantité limitée des informations disponibles par rapport à un système hors-ligne et les variations intrinsèques dans l'écriture.

Pour tester les deux catégories de mesures, conventionnelles et proposées, sur les textes manuscrits en-ligne, et les comparer ensemble, nous avons implanté une interface d'acquisition de textes manuscrits. Les aspects matériel et logiciel de ces interfaces ont été présentés, et leurs points forts ont été soulignés, notamment la liberté offerte aux scripteurs pour composer leur texte sans aucune contrainte temporelle ni séquentielle. Nous avons également détaillé les étapes d'extraction des caractéristiques, constituant dans le cas de caractéristiques conventionnelles, Dist_BB, RLmin et RLavg et deux autres mesures représentant la hauteur moyenne des tracés et la distance moyenne entre

les rectangles. Ces deux dernières, ainsi que les distances horizontales sont utilisées pour la première fois dans un système en-ligne de segmentation. Les distances horizontales sont parmi les meilleures caractéristiques conventionnelles, mais n'étaient utilisées que dans les systèmes hors-ligne. La raison principale d'absence des distances horizontales dans les systèmes en-ligne est la difficulté du calcul de ces mesures par rapport aux systèmes hors-ligne. Cette difficulté est, en fait, causée par l'aspect dynamique et les différentes séquences possibles des tracés lors de la composition du texte. Toutefois, les stratégies que nous avons proposées pour l'acquisition des tracés et d'extraction de caractéristiques, permettent de résoudre ces problèmes et d'utiliser les distances horizontales dans les systèmes en-ligne.

Afin d'extraire les caractéristiques proposées, une phase de reconnaissance des signes diacritiques est nécessaire. Notre algorithme original de reconnaissance est basé sur les extrema des tracés; lesquels permettent d'estimer la complexité de structure, la grandeur et la position relative des tracés. Cet algorithme, les étapes de calcul des extrema ainsi que les résultats numériques de reconnaissance, ont été présentés. Ces derniers affichent un taux bien élevé de reconnaissance des signes diacritiques (93%) dans notre base de données.

Pour un calcul précis, rapide et adaptatif des distances entre les parallélogrammes, nous avons conçu et implanté un algorithme d'estimation d'inclinaison d'un tracé ainsi qu'une méthode originale de calcul adaptatif de pentes moyennes des tracés.

Outre la distance entre les parallélogrammes, Dist_Paral et les distances horizontales minimale, RLmin, et moyenne, RLavg, quatre autres caractéristiques ont été utilisées comme des caractéristiques proposées. Il s'agit du nombre d'extrema du mot précédent preNbExtrema et du tracé actuel curNbExtrema ainsi que des hauteurs moyennes d'extrema du mot précédant preHghtExtrema et du tracé actuel curHghtExtrema. Ces caractéristiques, qui sont toutes calculées à partir des extrema des entités de l'écriture, permettent d'introduire l'effet de grandeur et de longueur des tracés dans la procédure de segmentation

Afin d'observer l'effet de nos propositions sur les résultats de segmentation de texte en mots, trois différents types de classificateurs ont été implantés : K plus proches voisins, neuronaux et AdaBoost.

Nous avons choisi les classificateurs K plus proches voisins pour évaluer la performance d'une méthode classique de classification dans notre système. Les deux scénarios commun et personnalisé ont été appliqués à ces classificateurs et les taux de classification correcte ont été calculé en utilisant les différentes valeurs de K de 1 à 19. Le meilleur taux obtenu est de 98.5% et il appartient à un classificateur entraîné avec le scénario commun en utilisant les caractéristiques proposées. Malgré la facilité d'implantation des classificateurs K plus proches voisins, leurs taux de classification n'ont pu atteindre nos attentes et étaient inférieurs à ceux des classificateurs neuronaux et AdaBoost. En plus, en raison de la nécessité d'un grand nombre de patrons de référence, du temps élevé de classification et de la performance inférieure dans le cas du scénario personnalisé par rapport au scénario commun, les classificateurs K plus proches voisins ne peuvent être considérés comme de bons classificateurs dans un système de segmentation en-ligne de textes manuscrits en mots.

Nous avons également conçu des classificateurs neuronaux de distances inter-tracés. Il s'agit des Perceptron à une couche et multicouches. Les caractéristiques conventionnelles et proposées ont été présentées à ces classificateurs. Les entrées, la structure, les phases d'apprentissage et de test, ainsi que les scénarios commun et personnalisé de l'apprentissage et de test des classificateurs, ont été expliqués.

Nous avons présenté les résultats de classification des distances inter-tracés, en utilisant les caractéristiques conventionnelles et proposées et en appliquant les deux scénarios commun et personnalisé aux classificateurs neuronaux. Nous avons également fait une analyse des erreurs de segmentation, en les classifiant dans trois différentes catégories : les erreurs causées par (1) les signes diacritiques, (2) l'inclinaison de l'écriture et (3) l'imprécision, soit dans la composition du texte, soit dans la classification des espacements par le classificateur.

Les résultats obtenus avec les caractéristiques conventionnelles démontrent que: (1) Dist_BB sous-estime les distances inter-mot, à cause de l'inclinaison d'écriture et des signes diacritiques chevauchant le mot suivant ou précédent; (2) Dans le cas des signes diacritiques, RLmin et RLavg surestiment les espacements inter-tracés et, en plus, RLavg surestime les distances intra-mot et (3) la normalisation des distances inter-tracés sur avgHeight et avgDistBB n'améliore pas la performance de classification.

À partir des résultats obtenus avec les caractéristiques proposées, on peut tirer les conclusions suivantes : (1) Grâce aux caractéristiques proposées, la performance des classificateurs neuronaux entraînés avec le scénario commun est améliorée; (2) Toujours dans le cas du scénario commun, la combinaison des caractéristiques proposées (Dist_Paral avec RLavg ou RLmin) a un effet positif sur les résultats obtenus et (3) L'application du scénario personnalisé aux classificateurs simples (avec une seule entrée) permet d'obtenir de meilleurs taux de classification par rapport au scénario commun. En fait, l'utilisation des caractéristiques proposées et l'application du scénario personnalisé ont permis d'améliorer le taux de segmentation de tracés en mots, par rapport au meilleur classificateur conventionnel (de 97.7% à près de 98.7%). Toutefois, en raison du compromis entre le nombre de patrons d'entraînement et la complexité des structures des classificateurs, le scénario personnalisé ne peut être appliqué adéquatement aux classificateurs ayant un nombre plus élevé de caractéristiques (et donc une structure plus complexe). Nous rappelons que notre objectif consiste à effectuer un entraînement rapide et efficace avec le moins de patrons nécessaires. Le défi à relever est donc de pouvoir utiliser un grande nombre de caractéristiques extraites dans la phase de classification, afin d'offrir aux classificateurs toutes les informations disponibles sans pourtant avoir besoin d'un grand nombre de patrons d'entraînement.

Dans une approche originale, nous avons proposé d'utiliser l'algorithme AdaBoost pour résoudre les problèmes de classification causés par le compromis entre le nombre de patrons d'entraînement et la complexité de structure des classificateurs. AdaBoost est une « machine à comité » qui permet de regrouper plusieurs classificateurs et de combiner leurs votes pondérés dans une procédure de classification. Dans la phase

d'apprentissage, l'emphase est mise sur les patrons difficiles à apprendre. Nous avons présenté la théorie d'AdaBoost, incluant ses deux versions, AdaBoost.M1 et AdaBoost.Z, une interprétation bayésienne de cet algorithme ainsi que les classificateurs implantés, leurs phases d'apprentissage et de test et nos propositions pour les améliorer. Dans le cas de nos classificateurs AdaBoost, les experts sont des classificateurs neuronaux très simples, nécessitant un nombre très petit de patrons d'apprentissage. En plus, chacun est un expert d'une caractéristique; ce qui permet d'obtenir plusieurs experts pour chaque caractéristique. Le nombre d'experts par caractéristique est déterminé dans la phase d'apprentissage.

En appliquant les deux scénarios commun et personnalisé aux classificateurs AdaBoost.M1 et AdaBoost.Z, nous avons obtenus des résultats de classification des distances inter-tracés. Nous avons aussi fait une comparaison de la performance des classificateurs neuronaux et AdaBoost, en regroupant, dans un tableau comparatif, ceux qui ont produit les meilleurs taux moyen de classification. Ce tableau contient également les intervalles de confiance des taux moyens. En plus d'une analyse globale des résultats, nous avons présenté une analyse détaillée du meilleur classificateur implanté, incluant ses paramètres et sa performance par rapport au meilleur classificateur conventionnel dans les cas difficiles de segmentation. Nous avons testé notre meilleur classificateur implanté sur des textes manuscrits intentionnellement composés de cas difficiles de segmentation, tels que les variations dans les espacements inter-tracés, l'inclinaison, la grandeur et la position verticale de l'écriture et ce, même dans une ligne de texte.

Nous avons réussi à améliorer considérablement les résultats de segmentation en-ligne de textes manuscrits en mots. En appliquant l'algorithme AdaBoost, pour la première fois, dans un système de segmentation en-ligne, nous avons atteint un taux de classification considérablement élevé. Ce résultat intéressant appartient à un classificateur AdaBoost.Z qui est entraîné avec le scénario personnalisé et qui utilise toutes les caractéristiques proposées pour effectuer la classification des distances inter-tracés. Les points forts de ce classificateur sont les suivants :

1- Le taux moyen de classification bien élevé (environ 99.1%);
2- La meilleure stabilité de résultats parmi tous les classificateurs implantés : il a produit le plus petit intervalle de confiance de taux moyen (écart type de 0.8% seulement);
3- L'adaptabilité au style d'écriture des scripteurs. En fait, l'utilisation de toutes les caractéristiques extraites permet de présenter toutes les informations disponibles au classificateur. C'est ce dernier qui détermine le poids de chaque caractéristique dans la décision prise. Grâce au scénario personnalisé, le classificateur s'adapte avec le style d'écriture des différents scripteurs, en déterminant, dans chaque cas, la(les) caractéristique(s) la(les) plus pertinente(s) parmi les caractéristiques disponibles et la façon de les combiner, si plus d'une caractéristique est choisie.
4- L'utilisation d'un petit nombre de patrons d'apprentissage (seulement 70). Par conséquent, le classificateur est idéal pour l'application du scénario personnalisé et ce, même aux courts textes. En plus, cela permet d'alléger la tâche des scripteurs pour préparer les exemples d'apprentissage et d'abaisser le temps nécessaire pour l'entraînement.
5- Le petit nombre d'hypothèses générées; ce qui permet de réduire la complexité de structure du classificateur et le temps de segmentation.

L'analyse des cas difficiles de segmentation montre que l'utilisation de ce classificateur permet de corriger près de 80% des erreurs du meilleur classificateur conventionnel. En outre, dans un contexte plus spécifique, près de 88% des erreurs causées par les signes diacritiques, 100% des erreurs causées par l'inclinaison d'écriture et près de 73% des erreurs causées par l'imprécision dans la composition du texte ou dans la classification des distances ont été corrigées.

En plus des tests effectués sur les textes de notre base de prototypage, les résultats obtenus avec d'autres textes difficiles à segmenter ont montré la supériorité systématique de notre meilleur classificateur proposé sur le meilleur classificateur

conventionnel et ce, en terme de taux de classification correcte et de stabilité des résultats de segmentation. Dans le cas de ces textes, l'écart entre les taux de classification des deux classificateurs varie entre 8% et 35%; ce qui montre, encore une fois, la performance de notre meilleur classificateur dans les cas difficiles de segmentation.

En somme, la contribution majeure de cette thèse repose sur le développement d'un système original et performant de segmentation en-ligne de textes manuscrits en mots. Les points originaux dans ce travail sont les suivants :

1- Sujet de la thèse où, pour la première fois, nous avons présenté l'idée de traitement des structures (en particulier la structure logique) des documents manuscrits.

2- Propositions originales pour la représentation interne de la structure logique d'un document manuscrit, en conformité avec SGML qui est une norme internationale de représentation de structures de documents typographiques. À cet égard, les deux points originaux sont les suivants :
   - DTD d'un document manuscrit;
   - Instance d'un document manuscrit structuré.

3- Conception et réalisation d'un nouveau système de segmentation en-ligne de textes manuscrits en mots. Les aspects originaux de ce système sont les suivants :
   - Revue de la littérature critique et complète des méthodes de segmentation de l'écriture manuscrite en mots, dans le contexte d'une application en-ligne. Voici les points importants de cette étude :
     - Présenter une nouvelle taxinomie des méthodes;
     - Analyser les méthodes et préciser leurs points forts et faibles dans un système en-ligne;
     - Choisir les meilleures caractéristiques parmi les caractéristiques conventionnelles.

- Les propositions originales pour résoudre les lacunes des meilleures caractéristiques conventionnelles et pour classifier adéquatement les espacements inter-tracés; toujours dans un contexte en-ligne. Les aspects originaux de ces propositions sont les suivants :
    - Utilisation des informations extraites d'extrema des tracés pour l'estimation adaptative de la grandeur de l'écriture.
    - Algorithme rapide et efficace de la reconnaissance des signes diacritiques et des ponctuations à l'aide des informations extraites d'extrema des tracés.
    - Ensemble des caractéristiques proposées pour la segmentation en-ligne d'un texte manuscrit en mots. Il faut ajouter que : (1) à l'exception de la distance entre les rectangles englobant, ces caractéristiques n'avaient jamais été utilisées dans un système de segmentation en-ligne de textes manuscrits en mots et (2) les méthodes existantes de segmentation en-ligne n'utilisent qu'une caractéristique pour la classification, alors que nous avons utilisé une combinaison de plusieurs caractéristiques.
    - Utilisation d'un nouvel algorithme d'apprentissage, AdaBoost, pour résoudre les problèmes de classification causés par le nombre limité de patrons disponibles pour l'apprentissage.
    - Application de l'algorithme d'AdaBoost dans un système en-ligne de traitement de l'écriture manuscrite et la stratégie adoptée pour la présentation des caractéristiques aux classificateurs AdaBoost afin de construire le classificateur final.
    - La stratégie adoptée pour présenter les caractéristiques extraites au classificateur AdaBoost, afin de mieux exploiter dans la phase de classification, les informations discriminantes de ces caractéristiques.

4- Conception et réalisation d'un module de reconnaissance et d'édition en-ligne de la structure physique de documents manuscrits. Voici les points originaux de cet aspect du travail:
   - La méthode d'acquisition des tracés qui offre aux scripteurs la liberté temporelle et séquentielle pour la composition de textes;
   - La reconnaissance instantanée des entités physiques;
   - Les commandes d'édition de la structure physique.

Le développement de notre système de segmentation de textes manuscrits en mots permet de compléter la procédure de reconnaissance de la structure physique de documents manuscrits. En plus, grâce à un éditeur spécial que nous avons conçu et implanté, l'utilisateur est capable de modifier la structure physique de son texte. Dans un système complet de traitement de documents manuscrits structurés, proposé dans (Moin, 1997), le balisage automatique de la structure physique est l'étape qui suit l'édition de cette structure; ce qui permet de construire la structure logique spécifique de documents. Nous avons proposé dans (Moin, 1997), les idées initiales pour une reconnaissance de structure logique basée sur la norme internationale de présentation de documents structurés SGML. Nous y avons également présenté un modèle de document pour un texte manuscrit simple, incluant les caractéristiques potentielles pour la reconnaissance de la structure logique.

Nous croyons que la suite de ce travail se situe principalement dans l'implantation des blocs de reconnaissance et l'édition de la structure logique de documents manuscrits. Dans ce but, les travaux suivants peuvent être envisagés :

1- Effectuer une étude profonde sur les caractéristiques adéquates pour la reconnaissance de la structure logique;
2- Écrire des modèles de documents pour différents types de documents manuscrits tels que des listes, des mémos, etc.

3- Concevoir et implanter une stratégie de balisage automatique, en utilisant les caractéristiques extraites de la structure physique de document et les informations disponibles dans le modèle de document.
4- Concevoir et réaliser un scénario de balisage manuel comme une alternative au balisage automatique ou comme un outil d'édition de documents balisés automatiquement;
5- Concevoir et réaliser un éditeur de la structure logique de documents manuscrits, particulièrement avec les commandes gestuelles d'édition.

Ce travail a permis de franchir les premiers pas vers la conception d'un système complet de traitement de documents manuscrits. Il s'agit de la reconnaissance et de l'édition de la structure physique de documents. La réalisation du système au complet permettra à l'utilisateur de convertir facilement une séquence de tracés manuscrits à une base de données de haut niveau (la structure logique de son document). Il aura également la possibilité d'éditer la structure logique. En plus, les résultats de la reconnaissance de la structure logique peuvent être utilisés pour améliorer la performance des traitements tels que la transmission et le stockage de documents ainsi que la reconnaissance de caractères et de mots manuscrits, si nécessaire.

# BIBLIOGRAPHIE

**Amari, S., Murata, N., Müller, K.-R., Finke, M. et Yang, H., 1996,** « Statistical theory of overtraining - Is cros-validation asymptotically effective? », *Advances in Neural Information Processing Systems*, Vol. 8, p.176-182.

**Blayo, Fraçois et Verleysen, Michel, 1996**, « Les réseaux de neurones artificiels», Presses universitaires de France, 126 p.

**Buxton, W., Hill, R. et Rowley, P., 1985**, « Issues and techniques in touch-sensitive tablet input », *Computer Graphics, Proceedings of SIGGRAPH'85,* Vol. 19, No. 3, p. 69-85.

**Chen, C., 1973**, *Statistical Pattern Recognition*, Hayden Book Co., 236 p.

**Chen, C., 1999**, *Handbook of pattern recognition & computer vision*, World Scientific, 1019 p.

**Cormen, T., Leiserson, C. et Rivest, R., 1990**, *Introduction to Algorithms,* MIT Press, 1028 p.

**Dimitriadis, ,Y. et Coronado, L., 1995**, « Towards an Art based mathematical editor, that uses on-line handwritten symbol recognition », *Pattern Recognition*, Vol. 28, No. 6, p. 807-822.

**Fox, A.S. et Tappert, C.C., 1987,** « On-line external word segmentation for handwriting recognition », *Proceedings of the third Symposium on Handwriting and Computer Applications*, p. 53-55.

**Fox, A.S., Kim, J. et Tappert, C.C., 1984,** « Segmenter for known number of characters », *IBM Technical Disclosure Bulletin*, Vol. 27, No. 7A, p. 3691-3693, décembre 1984.

**Freund, Y. et Schapire, Robert E., 1996a,** « Experiments with a New Boosting Algorithm », *Machine Learning : Proceedings of the Thirteenth International Conference*, p. 148-156.

**Freund, Y. et Schapire, Robert E., 1996b,** « Game Theory, On-line Prediction and Boosting », *Proceedings of the Ninth Annual Conference on Computational Learning Theory* , p. 325-332.

**Freund, Y. et Schapire, Robert E., 1997,** « A decision-theoretic generalization of on-line learning and an application to boosting », *Journal of Computer and System Sciences*, Vol. 55, No.1, p. 119-139.

**Fukunaga, K., 1972,** *Introduction to Statistical Pattern Recognition*, Academic Press, 369 p.

**Goodwin, N.C., 1975**, « Cursor positioning on an electronic display using lightpen, lightgun, or keyboard for three basic tasks », *Human Factors*, Vol. 17, No. 3, p. 289-295.

**Guerfali, W., 1990,** W. Guerfali, « Éditeur de texte manuscrit », mémoire de Maîtrise, École Polytechnique de Montréal.

**Haller, R., Mutschler, H. et Voss, M., 1984**, « Comparison of input devices for correction of typing errors in office systems », in B. Shackel (Ed.), *Human-Computer*

*Interaction - INTERACT '84,* Elsevier Science Publishers B.V. (North-Holland), p. 177-182.

**Hanaki, S., Temma T. et Yoshida, H., 1976** , « An on-line character recognition aimed at a substitution for a billing machine keyboard », *Pattern Recognition*, Vol. 8, p. 63-71.

**Hatamian, M. et Brown, E.F., 1985**, « A New Light Pen With Subpixel Accuracy », *AT&T Technical Journal*, Vol. 64, No 5, p. 1065- 1075.

**Hatamian, M., Budrikis, Z.L., Kubik, P.S. et Netravali, N., 1987**, « Accurate Light Pen », *Computer Vision, Graphics and Image Processing,* Vol. 39, p. 246-257.

**Haykin, Simon, 1999,** « Neural Networks, A Comprehensive Foundation », Printice-Hall, 842 p.

**Kimura, F., Miyake, Y. et Shridhar, M., 1995,** « Handwritten ZIP code recognition using lexicon free word recognition algorithm », *Proc. ICDAR'95*, p. 906-910.

**Kutzberg, J. M. et Tappert, C.C., 1982,** « Segmentation procedure for handwritten symbols and words », *IBM Technical Disclosure Bulletin*, Vol. 25, No. 7B, p. 3848-3852.

**Leroy, A., 1994**, « Segmentation de texte dans un éditeur cursif », *3ème Colloque National sur l'Écrit et le Document*, p. 407-417.

**Loy, W.W., et Landau, I.D., 1982**, « An on-line procedure for recognition of handprinted alphanumeric characters », *IEEE Trans. on PAMI*, Vol. PAMI-4, p. 422-427.

**Maarse, F. et Thomassen, A.J.W.M., 1983,** « Produced and Perceived Writing Slant: Difference between Up and Down Strokes », *Acta Psychologica*, Vol. 54, p. 131-147.

**Mahadevan, U. et Nagabushnam, R.C, 1995,** « Gap metrics for word separation in handwritten lines », *Proc. ICDAR'95*, p. 124-127.

**McCulloch, W. et Pitts, W., 1943,** « A Logical Calculus of the Ideas Immanent in Nervous Activity », *Bulletin of Mathematical Biophysiology*, Vol. 3, p. 115-133.

**Moin, S. et Brault, J.J., 1997,** « Structure Recognition of On-line Handwritten Documents Based on SGML », *Fourth International Conference on Document Analysis and Recognition ICDAR'97*, Ulm, p. 649-652.

**Nouboud F. et Plamondon, R., 1992,** « Bloc-notes électronique description et manuel d'utilisation », Rapport Technique, École Polytechnique de Montréal.

**Oulhadj, H., Bennacer, L., Lemoine, J., Wehbi, H. et Petit, E., 1994,** « Système de reconnaissance hierarchique d'écriture à architecture modulaire », *RFIA, 9ème congrès de Reconnaissance de Formes et Intelligence Artificiel*, volume I, p. 20.

**Plamondon, R., 1991**, « Step toward the production of an electronic pen-pad », *Proc. ICDAR'91*, p. 361-371.

**Plamondon, R., Nouboud, F., Parizeau M., et Guerfali, W., 1990,** « Conception d'un bloc-notes électronique », *Colloque International sur les Industries de la Langue*, p. 929- 951.

**Rosenblatt, F., 1958,** « The Perceptron : A probabilistic Model for Information Storage and Organization in the Brain », *Pscychological Review*, Vol. 65, p. 386-408.

**Schapire, Robert E. et Singer, Y., 1998,** « Improved Boosting Algorithms Using Confidence-rated Predictions », *Proceeding of the Eleventh Annual Conference on Computational Learning Theory.*

**Schapire, Robert E., 1990,** « The Strength of Weak Learnability », *Machine Learning*, Vol. 5, p. 197-227.

**Schomaker, L., 1998,** « From Handwriting Analysis to Pen-Computer Applications », *Electronics & Communication Engineering Journal*, June 1998, p. 93-102.

**Seni, G. et Cohen, E., 1994,** « External word segmentation of off-line handwritten text lines », *Pattern Recognition*, Vol. 27, No. 1, p. 41-52.

**SGML, 1986,** « ISO 8879 : 1986 Information Processing – Text and Office System – Standard Generalized Markup Language (SGML) », Geneva, 15 October 1986.

**Struckman-Johnson, D. L., Swierenga, S.J. et Shieh, K., 1984,** « Alternative cursor control devices: an empirical comparison using a text editing task », *Final*

*Report: Task II.2.* Vermillion, SD: University of South Dakota, Human Factors Laboratory.

**Suen, C.Y., 1979,** « A study of man-machine interaction problems in character recognition », *IEEE Trans. System, Man, and Cybernetics*, vol. 9, p. 732-738.

**Tappert, C.C., 1982,** « Cursive script recognition by elastic matching », *IBM J. Research and Development* , 26, p. 765-771.

**Tappert, C.C., 1984,** « Adaptive on-line handwriting recognition », *Proc. of Seventh Int. Conf. On Pattern Recognition*, p. 1004-1007.

**Tappert, C.C., 1990,** « Rationale for adaptive online handwriting recognition », *Frontiers in Handwriting Recognition*, p. 13-22.

**Tappert, C.C., Fox, A.S., Kim, J., Levy, S.E. et Zimmerman, L.L., 1986,** « Handwriting recognition on transparent tablet over flat display », *1986 SID Int. Symposium Digst of Technical Papers*, p. 308-312.

**Vapnik, V.N.,** « Estimation of Dependences Based on Empirical Data », Springer-Verlag, Berlin, 1982.

**Welbourn, L. et Withrow, R., 1987,** « A pen driven editor for editing text », Technical Repport, Trent Polytechnique, UK.

**Welbourn, L. et Withrow, R., 1990,** « A gesture based text and diagram editor », *Computer Processing of Handwriting*, p. 221-235.

# ANNEXE A

# TEXTES MANUSCRITS UTILISÉS POUR L'ENTRAÎNEMENT ET LE TEST DES CLASSIFICATEURS DE DISTANCES INTER-TRACÉS

Dans cette annexe, nous présenterons les treize textes manuscrits sur lesquels nous avons appliqué les différentes méthodes de segmentation de textes en mots (figures A.1 à A.13). Ces textes, dix en français et trois en anglais, ont été transcrits par huit sujets. Le tableau A.1 montre le numéro de textes écrits par chaque sujet. Nous avons demandé aux sujets d'écrire les textes entre les lignes horizontales. Toutefois, ils avaient la liberté de choisir la distance entre les lignes. Aucune contrainte spatiale ni temporelle n'a été imposée aux scripteurs. Comme on peut constater, les textes sont de différents types d'écriture, isolée, cursive pure et cursive mixte. Les méthodes de segmentation avaient donc à tenir compte de tous les aspects temporel, séquentiel et spatial de différents types d'écriture manuscrit.

Nous présenterons d'abord les images des textes, suivies d'un tableau contenant le nombre de distances inter-tracés pour chacun des textes. Ce nombre correspond au nombre de patrons utilisables pour l'entraînement et le test des classificateurs inter-tracés, à l'exception des distances horizontales conventionnelles pour lesquelles le nombre de patrons valides peut être inférieur au nombre des distances inter-tracés (voir la section 4.5.1.5).

Les textes présentés dans cet annexe forment notre base de données du prototypage. Afin de comparer la performance de notre meilleur classificateur proposé avec celle du meilleur classificateur conventionnel, en plus des textes de la base de données du prototypage, nous avons utilisé trois autres textes contenant uniquement des cas difficiles de segmentation. Ces textes sont présenté dans les figures 6.5 à 6.7.

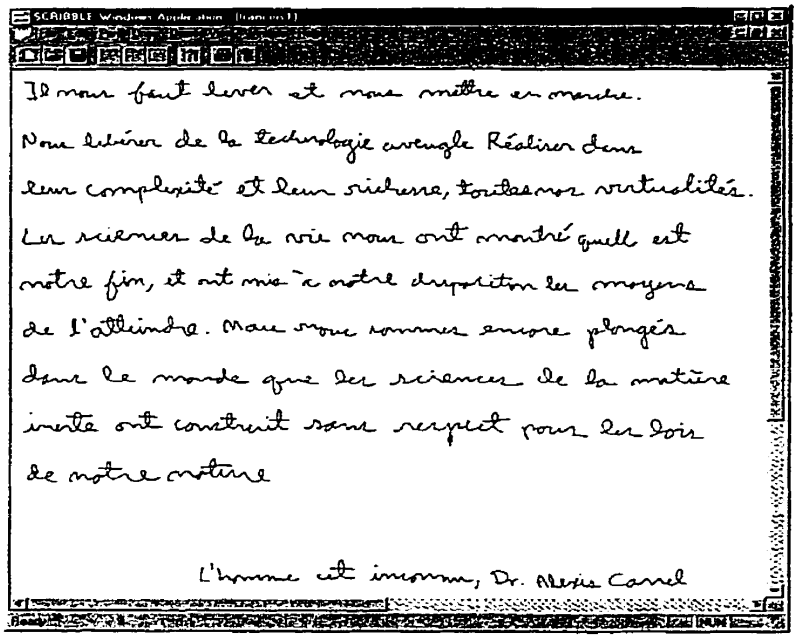



Figure A.1. Texte no. 1 écrit par le premier sujet.

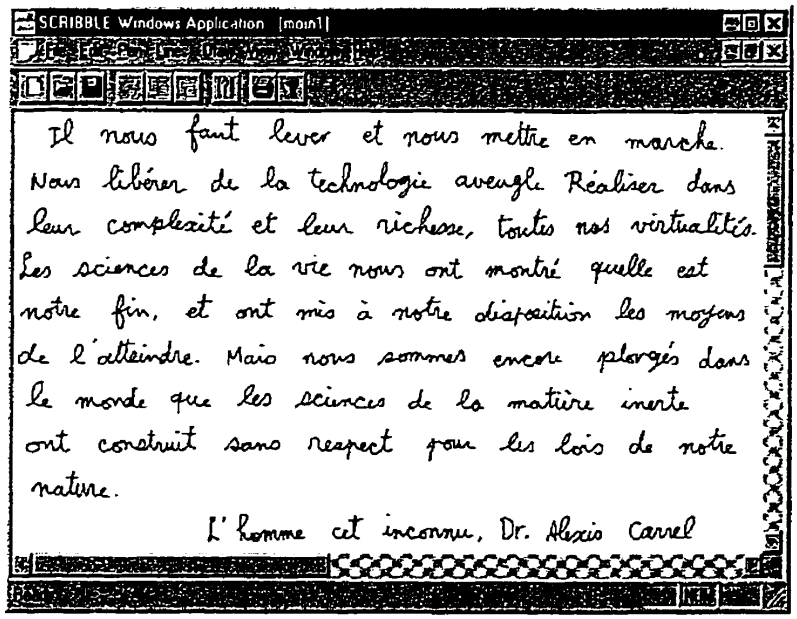



Figure A.2. Texte no. 2 écrit par le deuxième sujet.

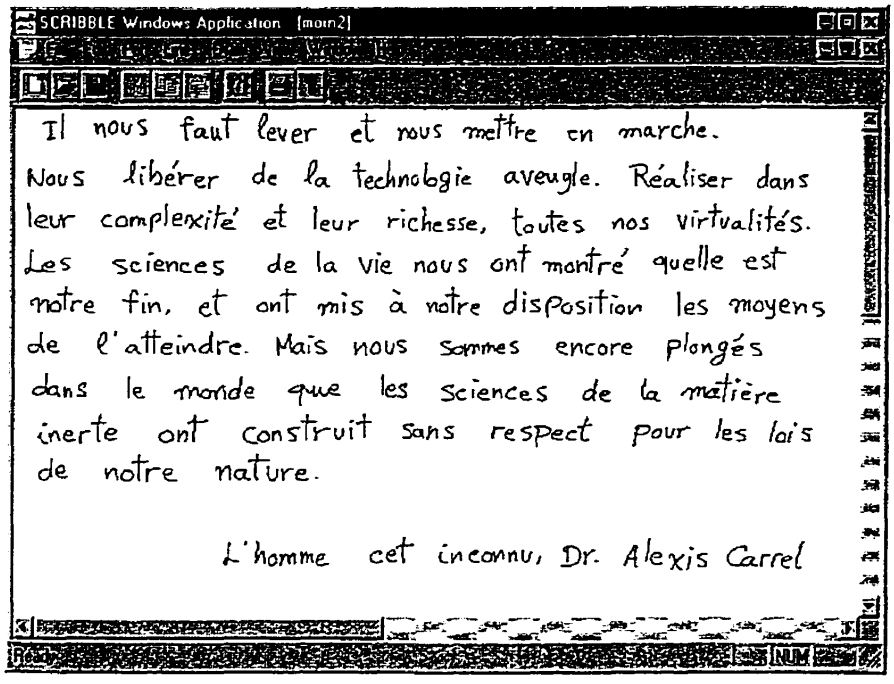



Figure A.3. Texte no. 3 écrit par le deuxième sujet.

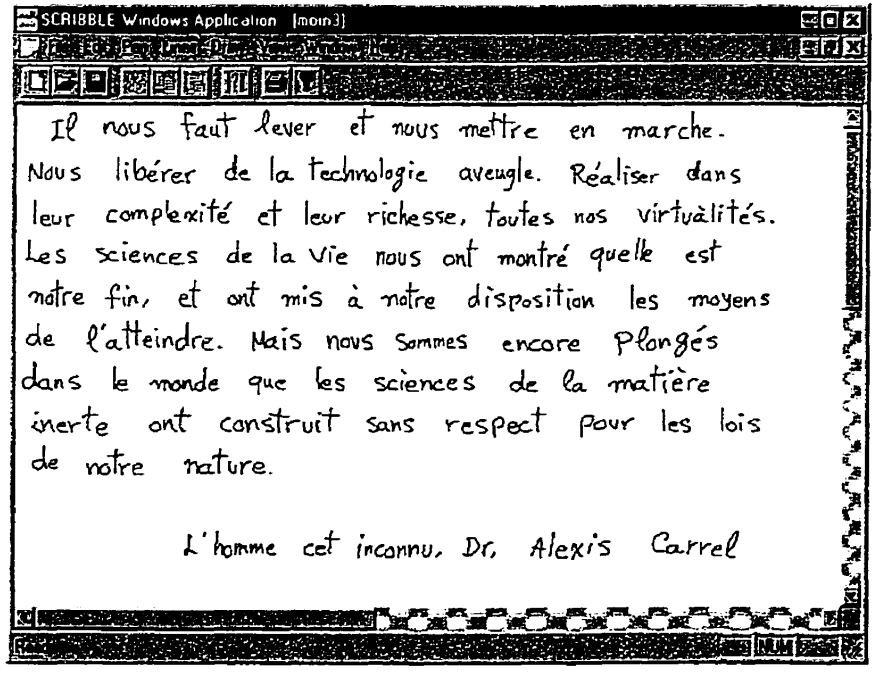
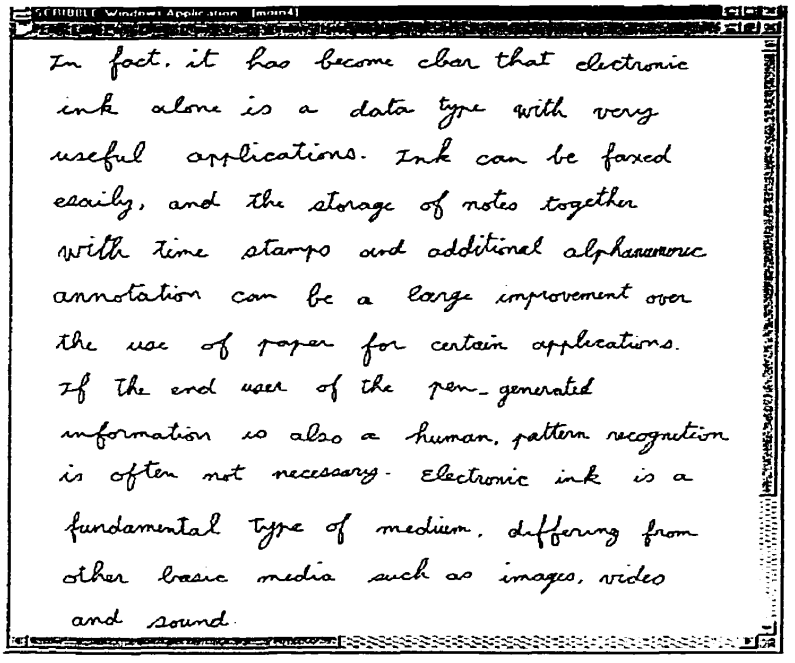
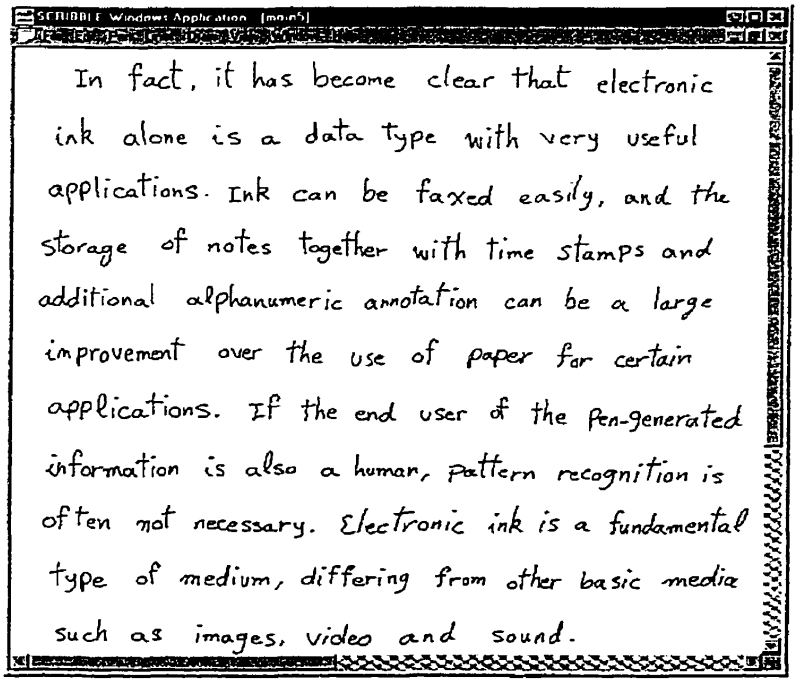



Figure A.4. Texte no. 4 écrit par le deuxième sujet.



In fact, it has become clear that electronic ink alone is a data type with very useful applications. Ink can be faxed easily, and the storage of notes together with time stamps and additional alphanumeric annotation can be a large improvement over the use of paper for certain applications. If the end user of the pen-generated information is also a human, pattern recognition is often not necessary. Electronic ink is a fundamental type of medium, differing from other basic media such as images, video and sound.

Figure A.5. Texte no. 5 écrit par le deuxième sujet.



In fact, it has become clear that electronic ink alone is a data type with very useful applications. Ink can be faxed easily, and the storage of notes together with time stamps and additional alphanumeric annotation can be a large improvement over the use of paper for certain applications. If the end user of the pen-generated information is also a human, pattern recognition is often not necessary. Electronic ink is a fundamental type of medium, differing from other basic media such as images, video and sound.

Figure A.6. Texte no. 6 écrit par le deuxième sujet.

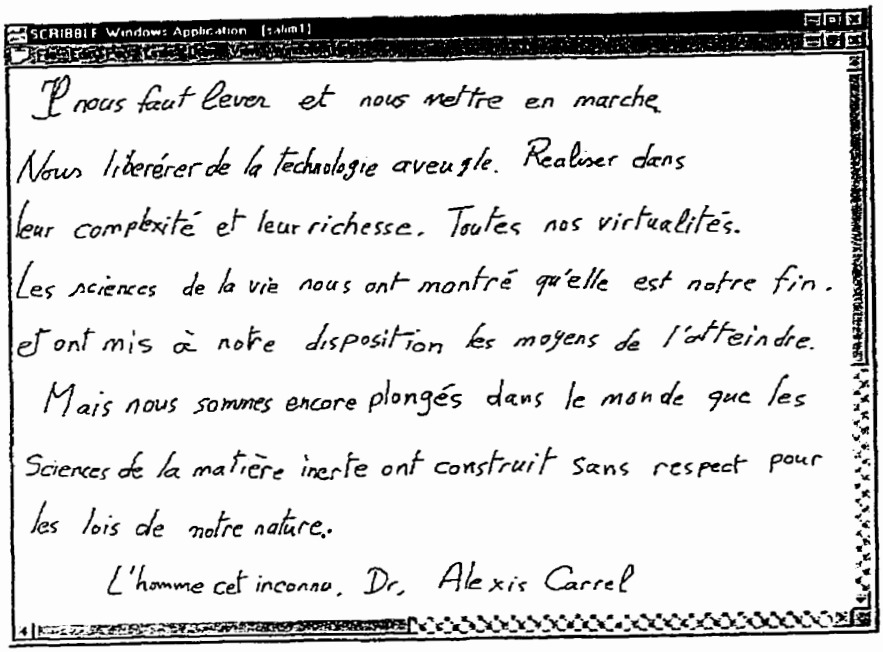



Figure A.7. Texte no. 7 écrit par le troisième sujet.

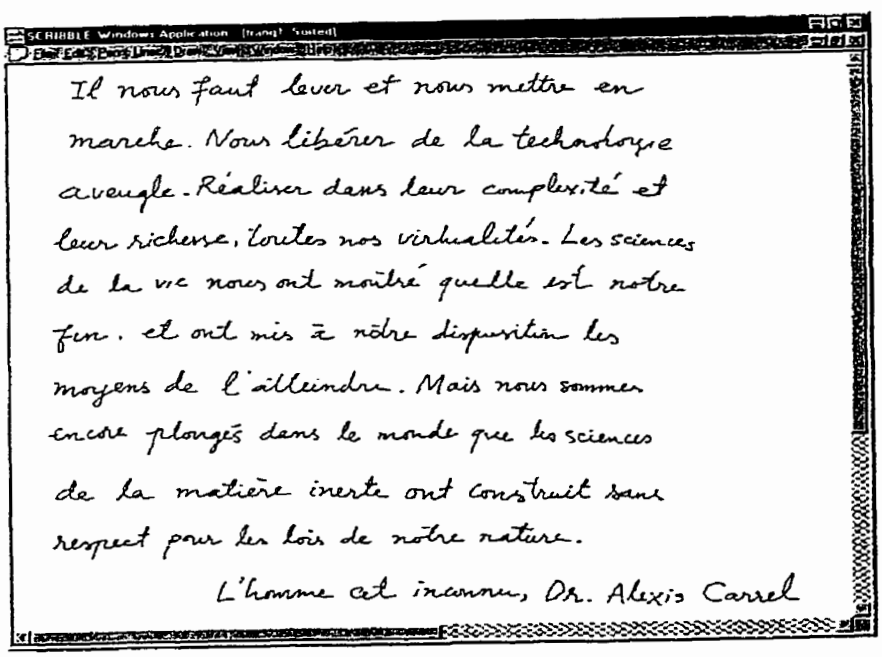



Figure A.8. Texte no. 8 écrit par le quatrième sujet.

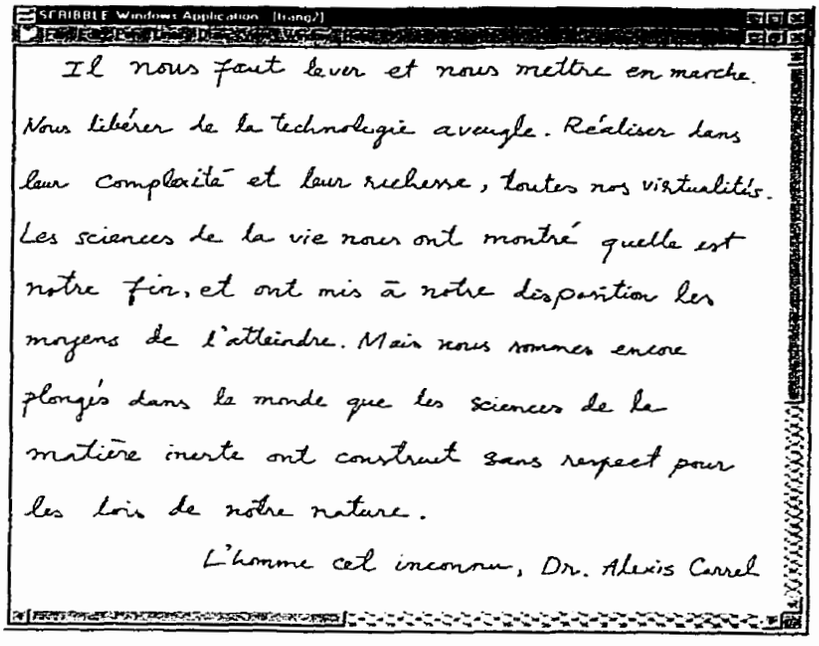



Figure A.9. Texte no. 9 écrit par le quatrième sujet.

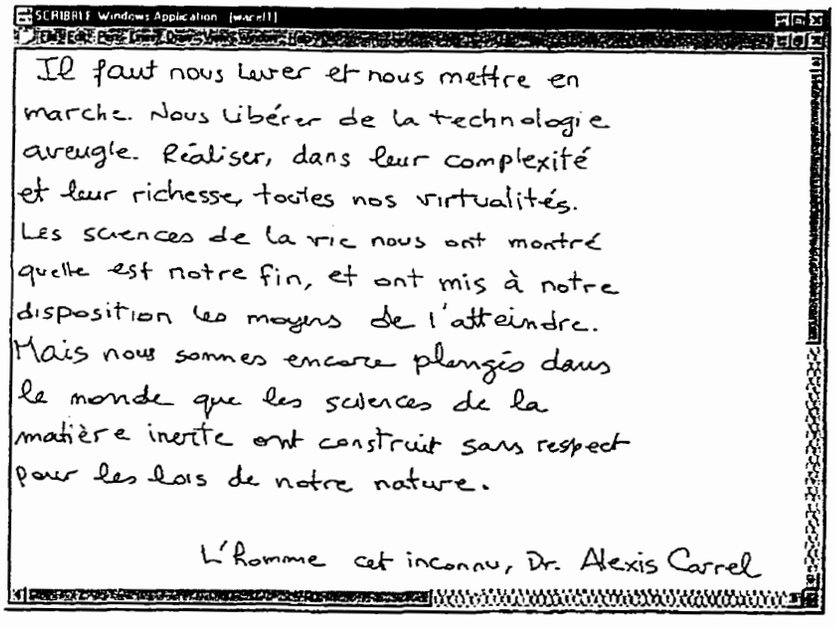
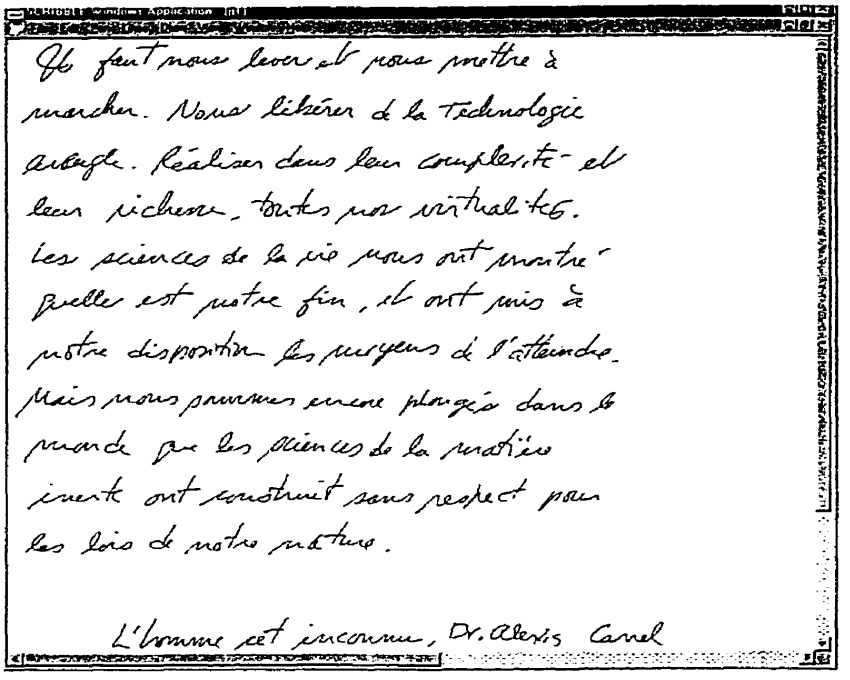


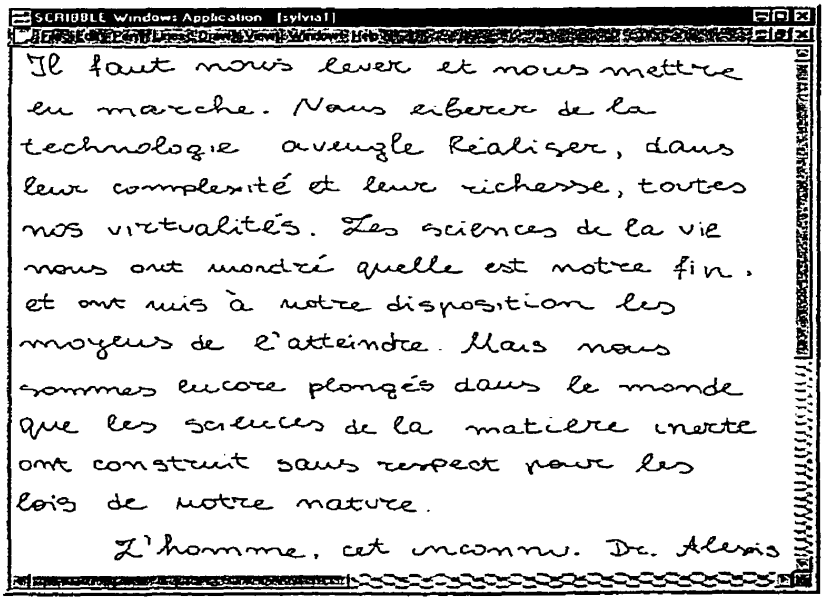

Figure A.10. Texte no. 10 écrit par le cinquième sujet.



Il faut nous lever et nous mettre à marcher. Nous libérer de la technologie aveugle. Réaliser dans leur complexité et leur richesse, toutes nos virtualités. Les sciences de la vie nous ont montré quelle est notre fin, et ont mis à notre disposition les moyens de l'atteindre. Mais nous sommes encore plongés dans le monde que les sciences de la matière inerte ont construit sans respect pour les lois de notre nature.

L'homme, cet inconnu, Dr. Alexis Carel

Figure A.11. Texte no. 11 écrit par le sixième sujet.




Il faut nous lever et nous mettre en marche. Nous libérer de la technologie aveugle Réaliser, dans leur complexité et leur richesse, toutes nos virtualités. Les sciences de la vie nous ont montré quelle est notre fin, et ont mis à notre disposition les moyens de l'atteindre. Mais nous sommes encore plongés dans le monde que les sciences de la matière inerte ont construit sans respect pour les lois de notre nature.

L'homme, cet inconnu. Dr. Alexis

Figure A.12. Texte no. 12 écrit par le septième sujet.

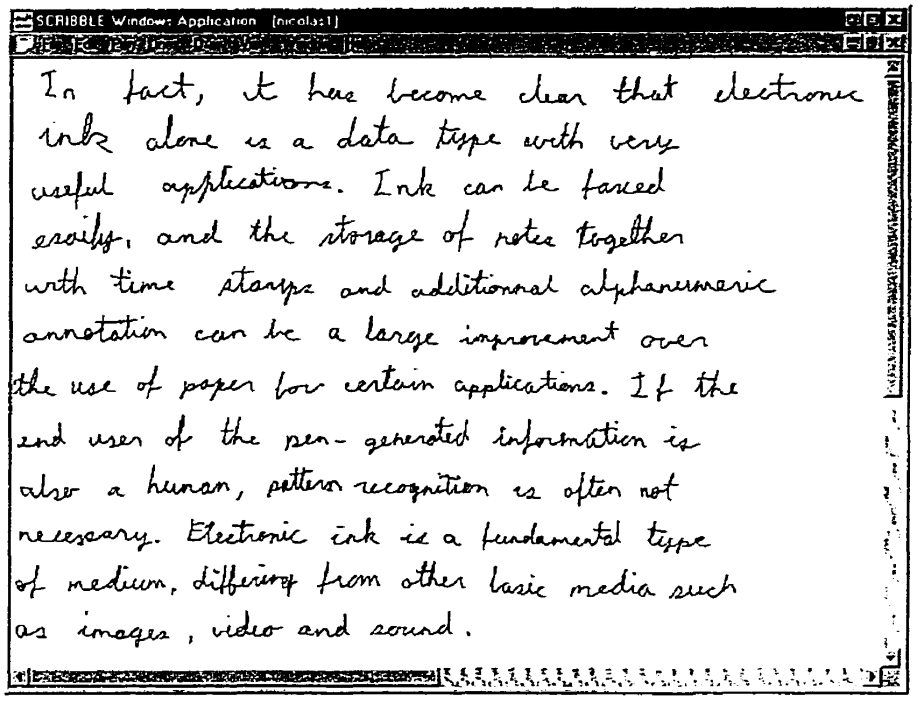



Figure A.13. Texte no. 13 écrit par le huitième sujet.

Tableau A.1. Numéro(s) de texte(s) composé(s) par les différents sujets.

| Sujet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Texte | 1 | 2-6 | 7 | 8-9 | 10 | 11 | 12 | 13 |

Tableau A.2. Le nombre de distances inter-tracés dans les textes manuscrits de la base de données.

| Numéro de Texte | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nombre de distances inter-tracés | 169 | 149 | 438 | 440 | 158 | 509 | 452 | 198 | 194 | 350 | 221 | 282 | 166 | 3726 |

# ANNEXE B

# DÉFINITION DES CARACTÉRISTIQUES ET DES MESURES UTILISÉES

Dans cette annexe, nous avons regroupé la définition des différentes caractéristiques et mesures utilisées pour la segmentation de textes manuscrits en mots. Nous avons essayé de donner, autant que possible, une représentation graphique et/ou un équation mathématique pour chacune de ces mesures et caractéristiques. La liste est en ordre alphabétique.

**avgDistBB :** Distance moyenne entre les rectangles des tracés précédents pour un tracé donné.

$$avgDistBB_i = \frac{\sum_{j=0}^{N-1} Dist\_BB_{i-j}}{N} \qquad \text{pour } i = 1 : NbTracés$$

où : $N$ est la largeur de fenêtre de calcul (en nombre des tracés);
$avgDistBB_i$ est la valeur moyenne de $N$ derniers $Dist\_BB$ pour le *i-ème* tracé;
$Dist\_BB_{i-j}$ est la distance entre les rectangles pour le *j-ième* tracé avant le tracé $i$.
*NbTracés* est le nombre de tracés du texte.

**avgHeight :** Hauteur moyenne des tracés précédents pour un tracé donné.

$$avgHeight_i = \frac{\sum_{j=0}^{N-1} h_{i-j}}{N} \qquad \text{pour } i = 1 : NbTracés$$

où : $N$ est la largeur de fenêtre de calcul (en nombre de tracés);
$avgHeight_i$ est la hauteur moyenne de $N$ derniers tracés pour le i-ème tracé;
$h_{i-j}$ est la hauteur de *j-ième* tracé avant le tracé $i$.
*NbTracés* est le nombre de tracés du texte.

**curHghtExtrema :** Distance verticale moyenne entre extrema consécutifs de tracé actuel (voir l'exemple de la figure B.1).

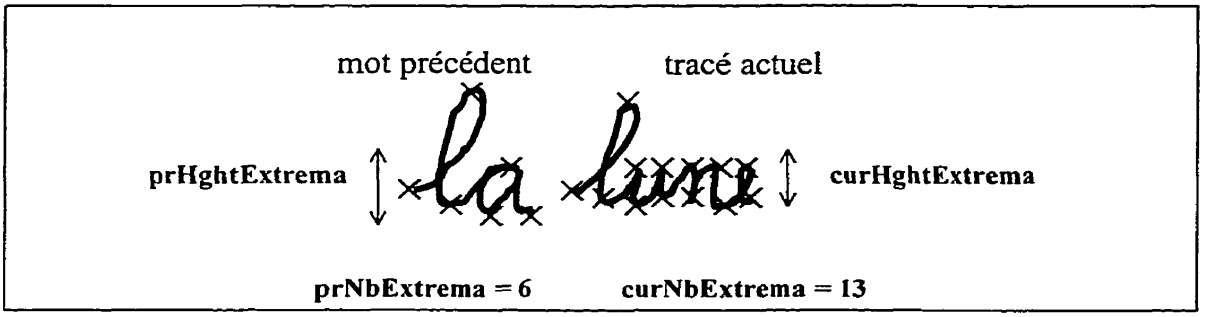



Figure B.1. Extrema (les « × ») d'un tracé et de son mot précédent et une représentation graphique de quatre caractéristiques extraites de ces extrema.

**curNbExtrema :** Nombre d'extrema de tracé actuel (voir l'exemple de la figure B.1).

**Dist_BB :** Distance horizontale entre les rectangles de deux tracés ou d'un tracé et d'un mot (voir l'exemple de la figure B.2).

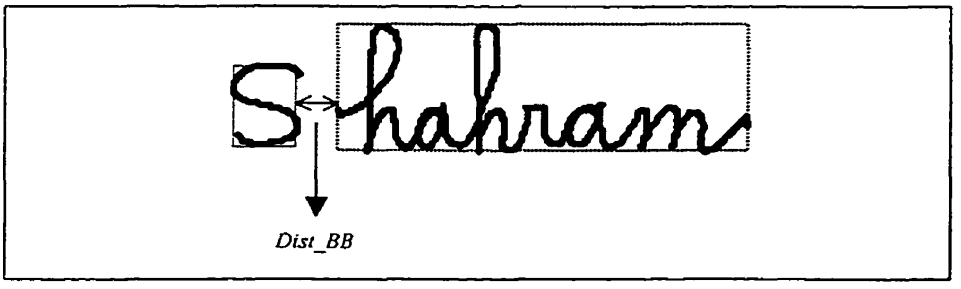



Figure B.2. Distance horizontale entre les rectangles de deux tracés.

**Dist_Paral (ou D_Prl) :** Distance horizontale entre les parallélogrammes de deux tracés ou d'un tracé et d'un mot (voir l'exemple de la figure B.3).

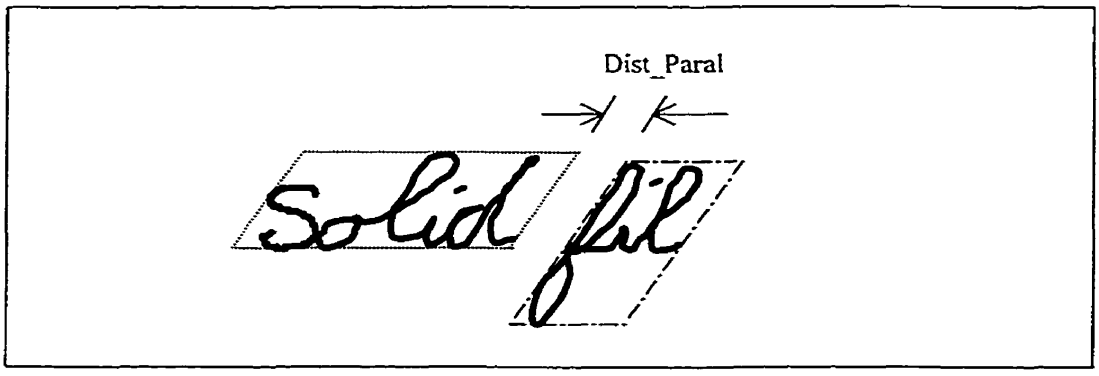



Figure B.3. Distance horizontale entre les parallélogrammes de deux mots.

**Haut_Moy_Extrema :** Distance moyenne verticale entre les paires de minima et maxima consécutives d'un tracé (voir l'exemple de la figure B.4).

$$Haut\_Moy\_extrema = \frac{\sum_{i=1}^{N_{min\_max}} h\_ext_i}{N_{min\_max}}$$

où : $h\_ext_i$ est la distance verticale entre la i-ème paire de minima et maxima;
$N_{min\_max}$ est le nombre de paires de minima et maxima consécutives.

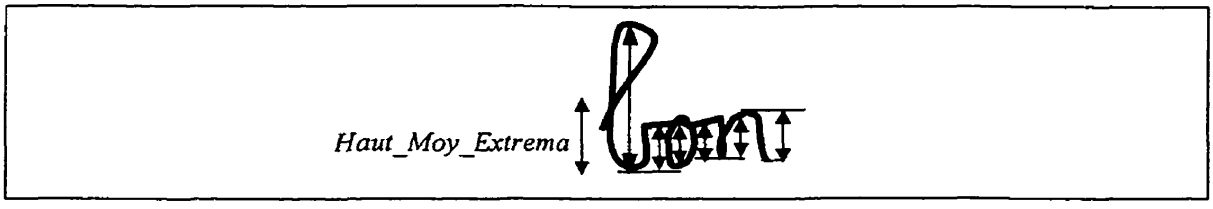



Figure B.4. Hauteur moyenne des minima et maxima consécutifs. La *Haut_Moy_Extrema* est la valeur moyenne des distances verticales montrées dans cette figure.

**prHghtExtrema (ou pHghtExt) :** Distance verticale moyenne entre les extrema consécutifs pour le mot précédent (voir l'exemple de la figure B.1).

**prNbExtrema (ou pNbExt) :** Nombre d'extrema de mot précédent (voir l'exemple de la figure B.1).

**RLavg :** Distance horizontale moyenne entre deux tracés ou entre un tracé et un mot (voir l'exemple de la figure B.5).

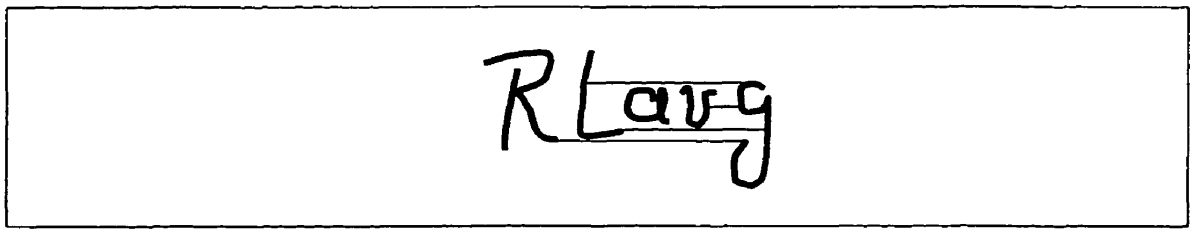



Figure B.5. Distances horizontales entre le tracé « g » et le mot « RLav ». RLavg et RLmin sont les valeurs moyenne et minimale de toutes les distances horizontales, respectivement.

**RLmin :** Minimum de distances horizontales entre deux tracés ou entre un tracé et un mot (voir l'exemple de la figure B.5).

$y_{max\_moy}$ : Moyenne des coordonnées $y$ de maxima d'un tracé ou d'un mot (voir la figure B.6). $y_{max\_moy}$ représente la borne supérieure de la zone médiane.

$y_{med\_moy}$ : Moyenne des coordonnées $y$ de tous les minima et maxima d'un tracé ou d'un mot (voir la figure B.6). $y_{med\_moy}$ représente la coordonnée $y$ de ligne médiane de la zone médiane.

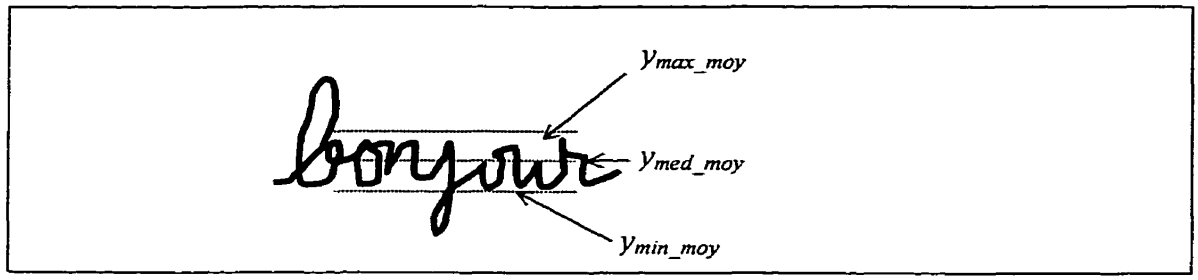



Figure B.6 Coordonnées $y$ des bornes supérieure et inférieure ($y_{max\_moy}$ et $y_{min\_moy}$) et la ligne médiane $y_{med\_moy}$, de la zone médiane d'un tracé, extraites de coordonnées $y$ de ses minima et maxima.

$y_{min\_moy}$ : Moyenne des coordonnées $y$ de minima d'un tracé ou d'un mot (voir la figure B.6). $y_{min\_moy}$ représente la borne inférieure de la zone médiane.

# ANNEXE C

# ÉDITEUR DE LA STRUCTURE PHYSIQUE DES DOCUMENTS MANUSCRITS

Dans cette annexe, nous présenterons le fonctionnement d'un logiciel que nous avons conçu et implanté pour éditer la structure physique d'un document manuscrit. Il s'agit d'une application orientée objet pour Windows, développée dans l'environnement Visual C++. La figure C.1 montre l'éditeur de documents manuscrits structurés.

Notre éditeur de documents manuscrits structurés met à la disposition du scripteur les opérations et les commandes d'édition suivantes :

1- Initialiser le nombre de lignes horizontales (la distance entre les lignes);
2- Afficher et effacer les lignes horizontales;
3- Afficher et effacer les rectangles englobant des mots, des lignes et des blocs;
4- Effacer un ou plusieurs mots, une ligne ou un bloc;
5- Déplacer un ou plusieurs mots ou un bloc;
6- Copier un ou plusieurs mots ou un bloc;
7- Insérer un saut de ligne;
8- Insérer un mot dans une ligne;
9- Ajouter un tracé à un mot sans affecter d'autres mots;
10- Uniformiser les distances inter-mots dans une ligne ou dans un bloc;
11- Sauvegarder le texte;
12- Imprimer le texte.

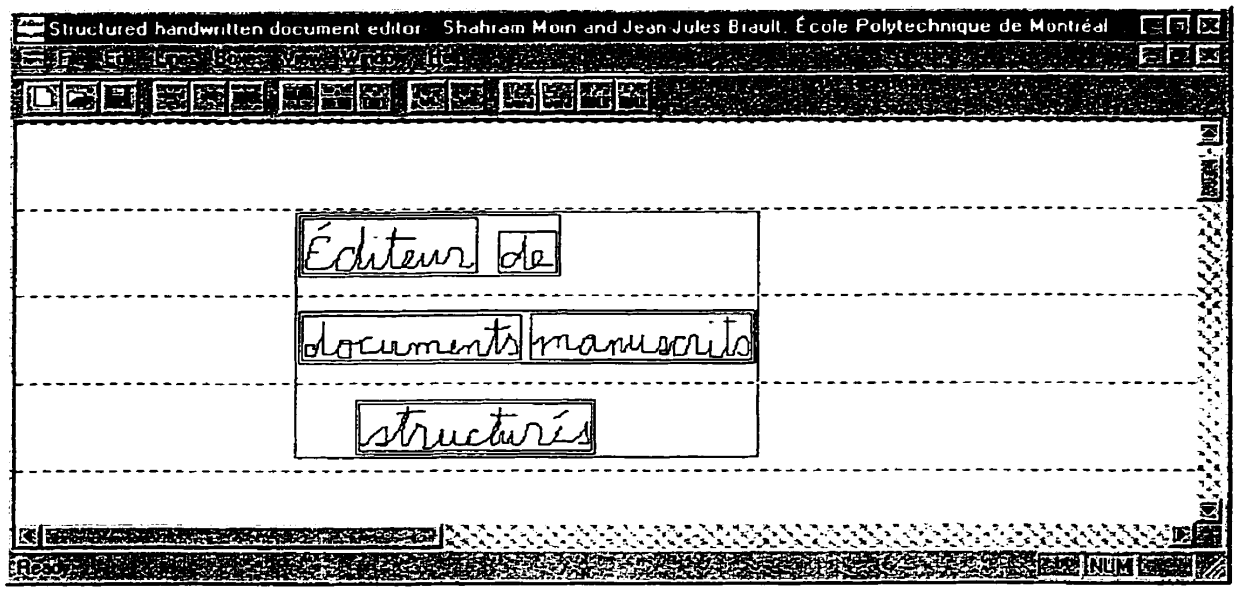



Figure C.1. Exemple d'un texte composé dans l'éditeur de documents manuscrits structurés.

Les trois objets principaux de cet éditeur sont les objets « Mot », « Ligne » et « Bloc », sur lesquels les commandes d'édition peuvent être effectuées. L'éditeur permet à l'utilisateur d'entrer un nouveaux texte via un crayon ou d'ouvrir un texte sauvegardé pour le modifier. Pour entrer un texte manuscrit, le scripteur détermine d'abord le nombre de lignes horizontales qui divise la fenêtre d'édition. Ces lignes, qui sont ensuite affichées automatiquement, peuvent être effacées et réaffichées, en tout temps, via une option de menu. Le scripteur doit écrire entre ces lignes pour créer un document structuré. Aucune autre contrainte temporelle, ni séquentielle ne lui est imposée. À chaque nouveau tracé, l'éditeur réorganise la structure physique du document pour mettre à jour les listes des ses blocs, les lignes de chaque bloc et les mots de chaque ligne, si cela est nécessaire. La figure C.1 montre un bloc de texte, ses lignes et les mots de ses lignes. Les rectangles englobant les mots, les lignes et le bloc sont affichés dans cette figure. Ces rectangles, qui peuvent être effacés et réaffichés par l'utilisateur, servent à montrer les résultats de segmentation d'un document en objets physiques. Nous devons ajouter que l'on peut afficher ou effacer les rectangles des objets de même

type, indépendamment de ceux des objets d'autres types. Par exemple, il est possible d'afficher les rectangles des lignes sans afficher les rectangles des mots et des blocs.

Les commandes d'édition implantées sont les suivantes : effacer, déplacer, copier, insérer, ajouter et uniformiser. Toutes ces commandes, qui seront détaillées dans les paragraphes suivants, sont disponibles sous forme de boutons affichés en dessous du menu principal (boutons 4 à 15 dans la figure C.1). L'objet sur lequel une commande doit être appliquée, est sélectionné après le choix de la commande. Par exemple, pour effacer un mot, il faut d'abord appuyer, avec le crayon ou la souris, sur le bouton correspondant et ensuite désigner le mot à effacer. On peut sélectionner un mot, une ligne ou un bloc du texte en dessinant un tracé, en appuyant le crayon ou en cliquant sur le bouton gauche de la souris sur une partie de l'objet. Pour désigner plusieurs mots, il faut dessiner un tracé qui croise tous les mots.

La plupart des commandes peuvent être appliquées sur plusieurs objets l'un après l'autre, tant qu'elles sont actives. Pour désactiver une commande, on peut appuyer de nouveau sur son bouton ou sélectionner une autre commande.

La commande « effacer » est applicable sur un ou plusieurs mots, une ligne ou un bloc de texte (les boutons 4, 5 et 6). En effaçant un objet, tous les objets qui le suivent dans le texte seront déplacés automatiquement pour remplir le vide produit. Ainsi, la suppression d'une ligne d'un bloc fait remonter toutes les lignes suivantes du bloc et tous les blocs situés en dessous (voir la figure C.2). En plus, si une ligne n'a qu'un seul mot, en effaçant le mot la ligne sera effacée, et s'il s'agit de la seule ligne du bloc, le bloc sera supprimé au complet.

La deuxième commande est « déplacer » qui est applicable sur un ou plusieurs mots ou un bloc (boutons 12 et 14). L'exemple de la figure C.3 montre le résultat du déplacement du deuxième mot de la première ligne au début de la deuxième ligne de la figure C.2. Pour déplacer une ligne au complet, on peut choisir la commande de déplacement de mot(s) et sélectionner tous les mots de la ligne. Après avoir sélectionné l'objet ou les objets à déplacer, ils seront effacés de l'écran et les objets suivants

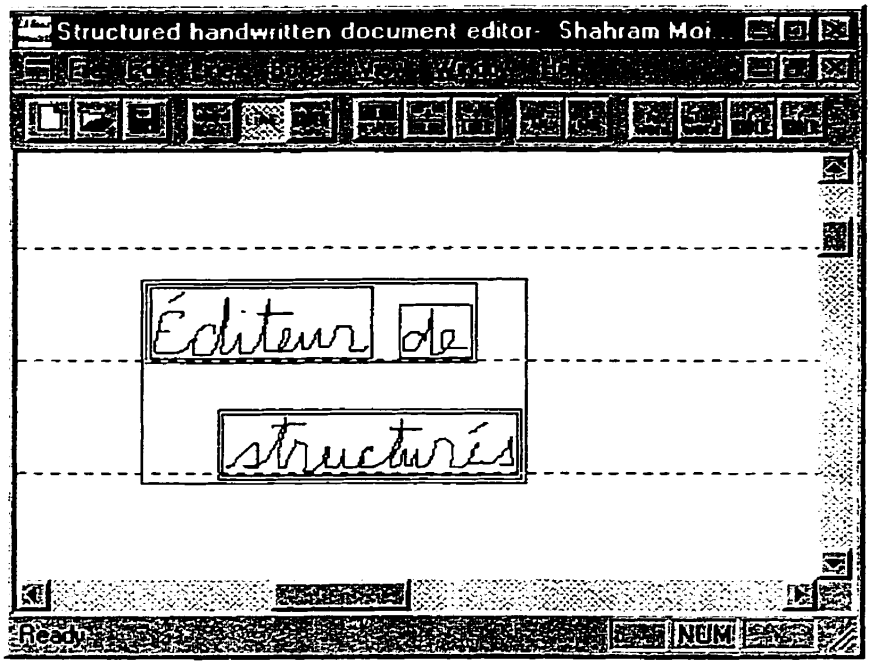



Figure C.2. Résultat de la suppression d'une ligne de texte.

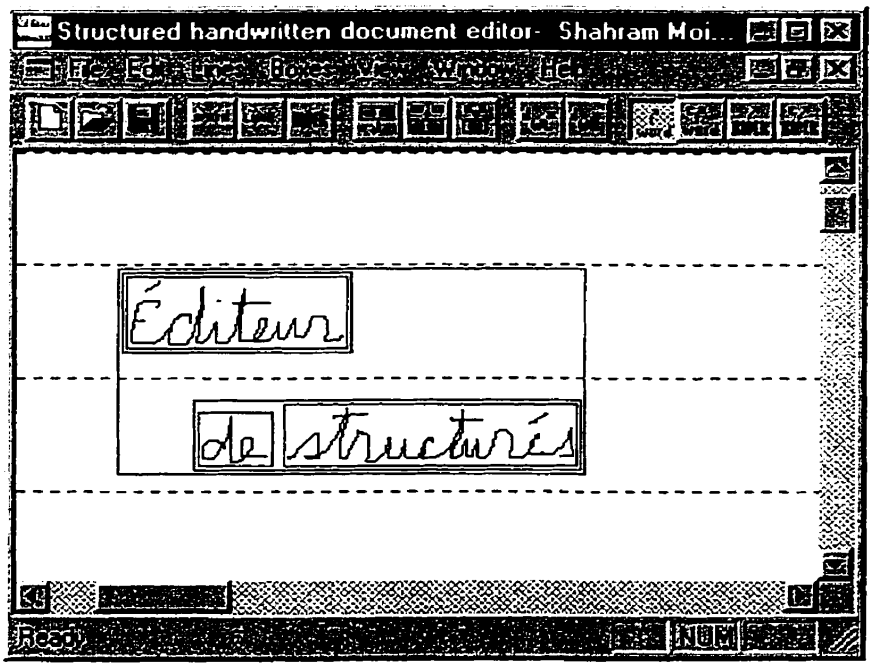



Figure C.3. Résultat de l'application de la commande de «déplacer un mot».

seront déplacés pour remplir le vide produit. Ensuite, l'utilisateur doit spécifier l'endroit du déplacement en appuyant sur le bouton gauche de la souris ou en touchant l'écran avec le crayon. Le résultat est l'apparition des objets sélectionnés à l'endroit désiré. Cette opération peut être utilisée pour fusionner deux blocs, à condition que l'un soit déplacé en dessous ou au-dessus de l'autre, sans une ou plusieurs lignes vides séparatrices.

La commande « copier » permet de copier un ou plusieurs mots ou un bloc de texte (boutons 13 et 15). Les mêmes étapes expliquées ci-dessus pour la commande « déplacer » doivent également être suivies pour copier un ou plusieurs objets. La figure C.4 montre le résultat de copier le bloc de la figure C.3.

La commande « insérer » peut être utilisée pour ajouter un saut de ligne (bouton 7) ou un ou plusieurs mots (bouton 8 de gauche) dans une ligne de texte. Pour insérer un saut de ligne, il faut appuyer sur le bouton correspondant et pointer avec le crayon ou la souris sur le point d'insertion. Le résultat est le déplacement à la ligne suivante de tous

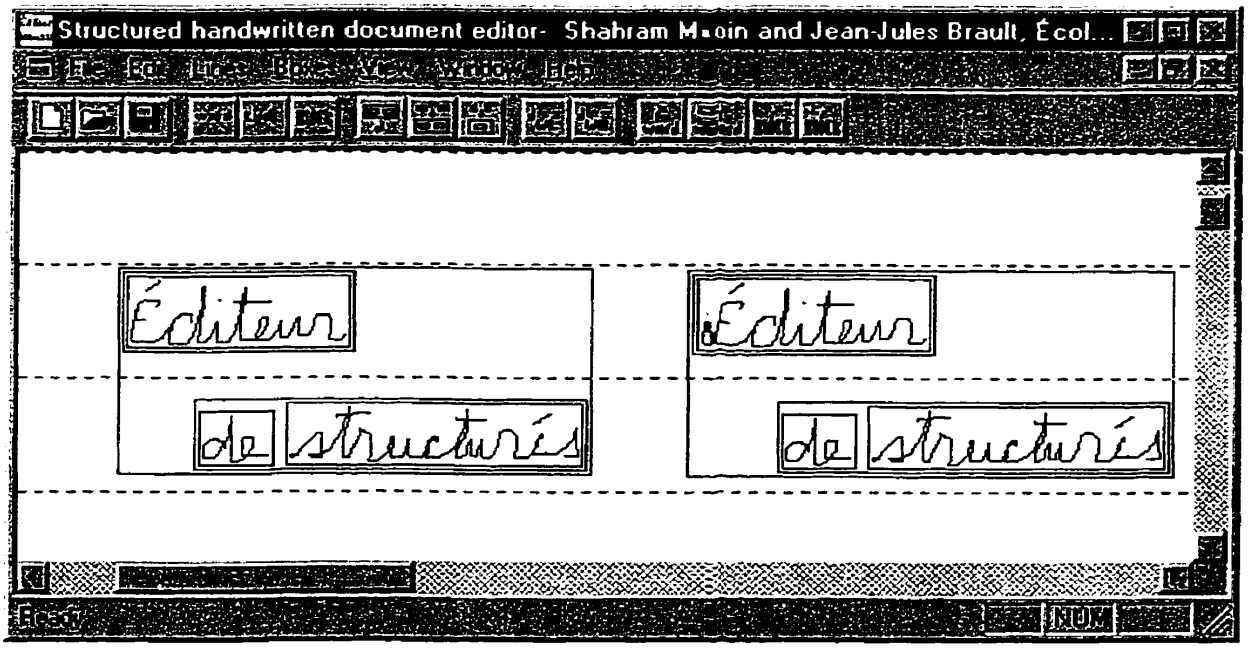



Figure C.4. Résultat de l'application de la commande de «copier un bloc».

les mots situés après le point d'insertion et la descente de toutes les lignes et les blocs se trouvant en dessous de cette ligne. La figure C.5 montre le résultat de l'application de cette commande sur le texte de la figure C.3. La commande d'insertion de saut de ligne peut être utilisée pour diviser un bloc en deux.

Pour insérer un ou plusieurs mots dans une ligne de texte, on appuie d'abord sur le bouton correspondant et on pointe ensuite l'endroit désiré de l'insertion. Ce point doit être situé avant le premier mot, après le dernier mot ou entre deux mots d'une ligne de texte. Une petite flèche montre le point d'insertion et l'utilisateur doit écrire le(s) mot(s) à ajouter, n'importe où dans la fenêtre d'édition. Une fois que les mots sont entrés, on doit appuyer de nouveau sur le bouton 8 pour signaler la fin de la composition. Les mots composés seront automatiquement insérés dans la ligne au point désigné, en déplaçant les mots suivants dans la ligne vers la droite. La figure C.6 montre le résultat d'insertion de deux mots « documents manuscrits » dans la deuxième ligne du texte de la figure C.3.

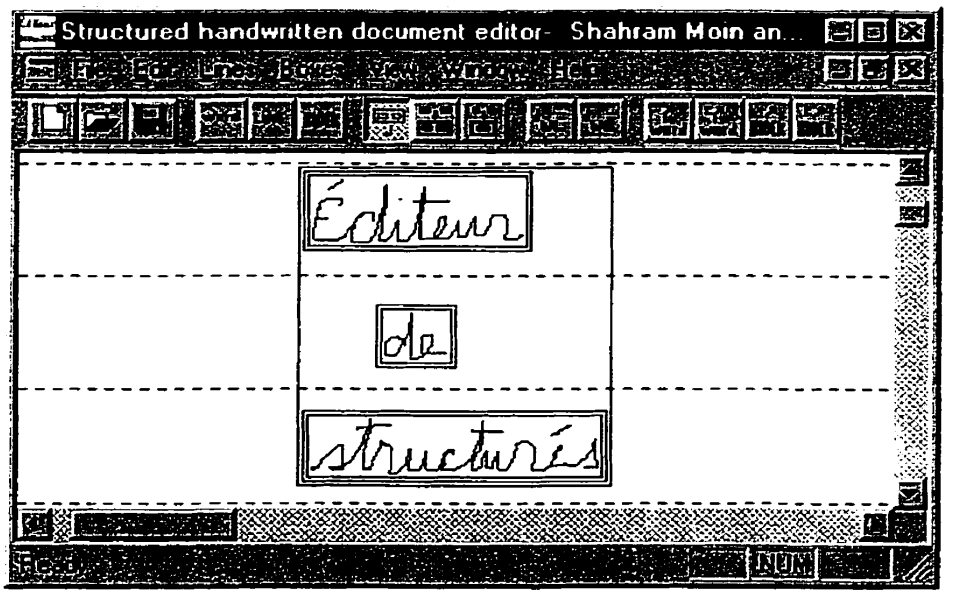



Figure C.5. Résultat de l'application de la commande d' «insérer un saut de ligne».

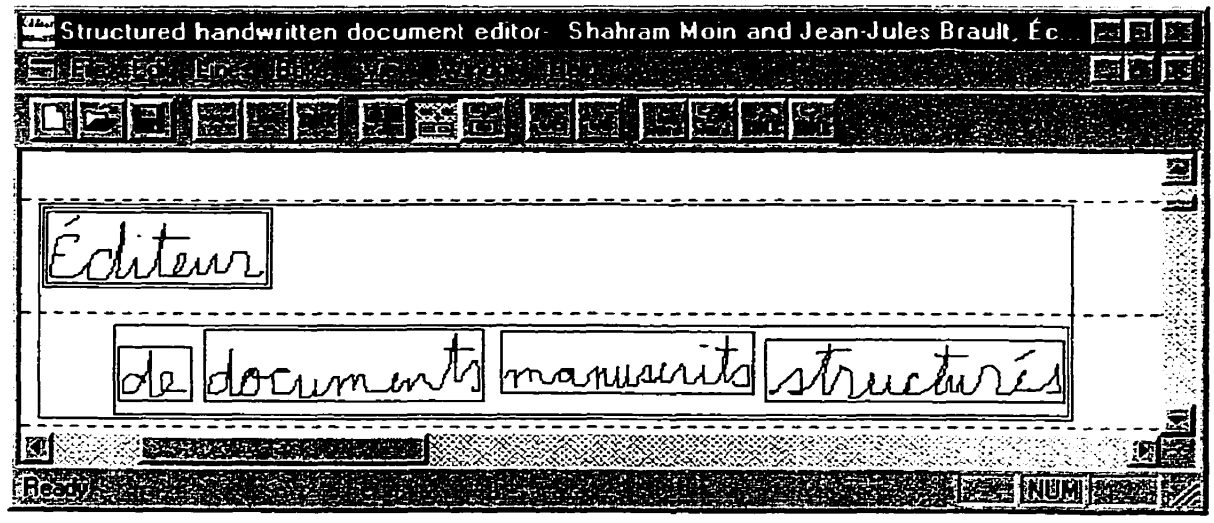



Figure C.6. Résultat de l'application de la commande d'«insérer des mots dans une ligne».

Une autre commande d'édition est la commande « ajouter » (bouton 9). Elle permet d'ajouter un tracé à un mot, sans affecter d'autres mots de la ligne. En fait, comme nous l'avons déjà mentionné, le scripteur a la liberté de composer un texte à son choix. Il peut ainsi composer deux lignes au complet et ensuite ajouter un tracé à un mot qui existe déjà. Toutefois, l'ajout d'un tracé peut causer la fusion de deux mots si le tracé a des chevauchements horizontaux avec les deux ou s'il est assez proche des deux. Autrement dit, cette opération peut affecter non seulement le mot auquel on veut ajouter un tracé, mais aussi les mots en voisinage. Par conséquent, si l'on ne veut pas que l'ajout d'un tracé dans un mot affecte les autres mots, on doit utiliser la commande « ajouter ». Pour ce faire, il faut appuyer sur le bouton correspondant et cliquer avec le crayon ou la souris sur le mot pour le sélectionner. Il est ensuite possible d'entrer le tracé directement à l'endroit désiré, même s'il touche les mots suivants ou précédents lors de la composition. À la levée du crayon, le tracé sera ajouté dans le mot et les mots suivants seront déplacés, si cela est nécessaire. La figure C.7 montre le résultat d'ajouter le tracé

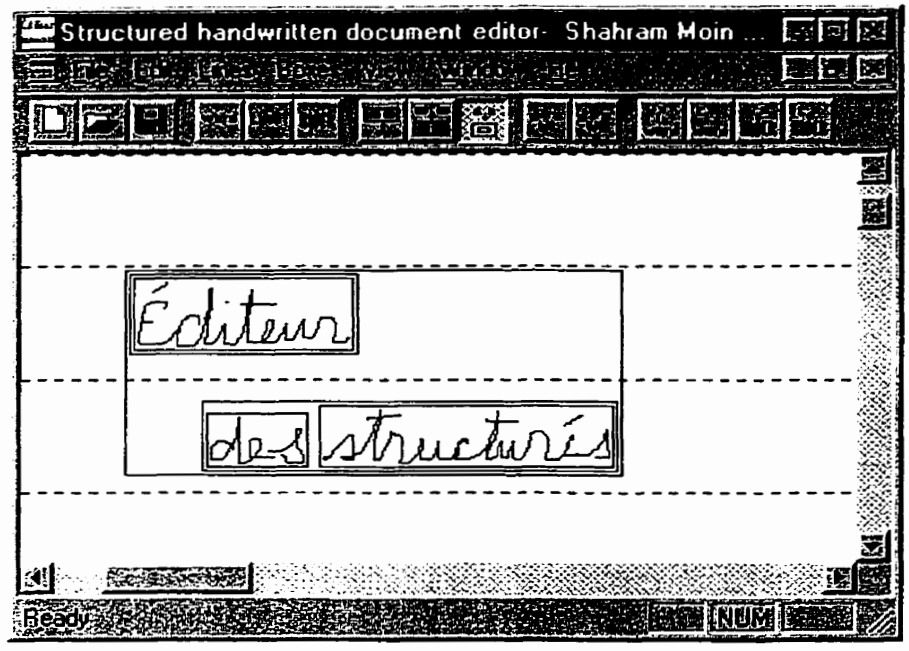



Figure C.7. Résultat de l'exécution de la commande d'«ajouter».

« s » à la fin du mot « de » dans la figure C.3 et ce, sans faire fusionner ce mot avec le mot « structurés ».

La dernière commande d'édition de notre système est « uniformiser ». Cette commande, qui est applicable sur une ligne ou toutes les lignes d'un bloc, permet d'uniformiser les distances inter-mots de l'objet sélectionné. La figure C.8 montre le résultat de l'exécution de la commande « uniformiser » sur les distances inter-mots d'un bloc de texte manuscrit. Pour obtenir cette figure, nous avons d'abord composer le bloc à gauche. Nous en avons ensuite fait une copie pour créer le bloc à droite et avons appliqué la commande « uniformiser » sur ce bloc.

En plus des commandes d'édition ci-dessus, l'éditeur permet à l'utilisateur d'effectuer d'autres opérations sur son texte comme sauvegarder et imprimer.

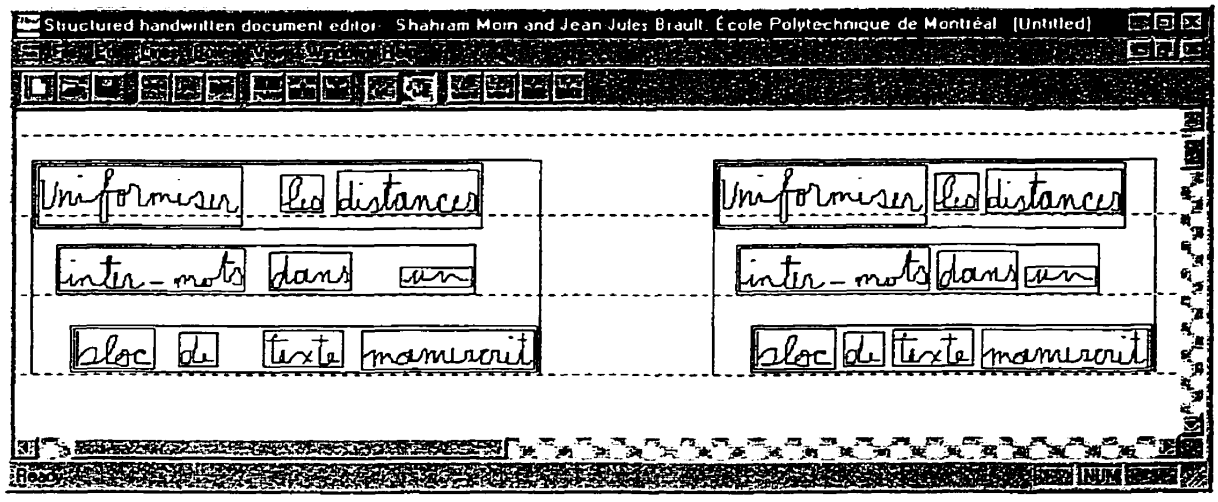



Figure C.8. Résultat de l'exécution de la commande d'édition d'«uniformiser».

# ANNEXE D

# RÉSULTATS DE CLASSIFICATION OBTENUS AVEC LES CLASSIFICATEURS K PLUS PROCHES VOISINS

Afin de mesurer la performance des classificateurs classiques pour la segmentation de textes manuscrits en mots, nous avons implanté des classificateurs K plus proches voisins (voir la section 4.1). Les deux scénarios commun et personnalisé[1] ont été appliqués à ces classificateurs. Dans le cas du scénario commun, les mêmes exemples d'apprentissage des classificateurs neuronaux et AdaBoost ont été utilisés comme des patrons de référence. Quant au scénario personnalisé, une partie des patrons (70 patrons) dans chaque texte servaient de références pour la classification des autres patrons du même texte. Pour les deux scénarios, nous avons utilisé toutes les 7 caractéristiques proposées et avons calculé les taux de classification correcte obtenus avec différentes valeurs impaires de K de 1 à 19. Les résultats obtenus sont présentés dans le tableau D.1 et la figure D.1 pour le scénario commun et le tableau D.2 et la figure D.2 pour le scénario personnalisé. Dans les tableaux D.1 et D.2, $K$, $s$, $\bar{y}_{med}$ et ($\bar{y}_{min}$, $\bar{y}_{max}$) représentent respectivement le nombre de voisins utilisés, l'écart type et la moyennes des taux de classification et l'intervalle de confiance des valeurs de moyennes.

---

[1] voir la section 4.5.2.3.1

Tableau D.1. Taux de classification correcte des classificateurs KNN (scénario commun).

| $K$ | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *1* | 1.76 | 97.91 | (97.00, 98.82) |
| *3* | 1.46 | 98.05 | (97.29, 98.81) |
| *5* | 1.46 | 98.28 | (97.52, 99.03) |
| *7* | 1.29 | 98.25 | (97.58, 98.92) |
| *9* | 1.40 | 98.52 | (97.79, 99.24) |
| *11* | 1.59 | 98.32 | (97.49, 99.14) |
| *13* | 1.61 | 98.30 | (97.47, 99.13) |
| *15* | 1.78 | 98.21 | (97.29, 99.13) |
| *17* | 2.11 | 98.03 | (96.94, 99.13) |
| *19* | 2.13 | 97.97 | (96.86, 99.07) |

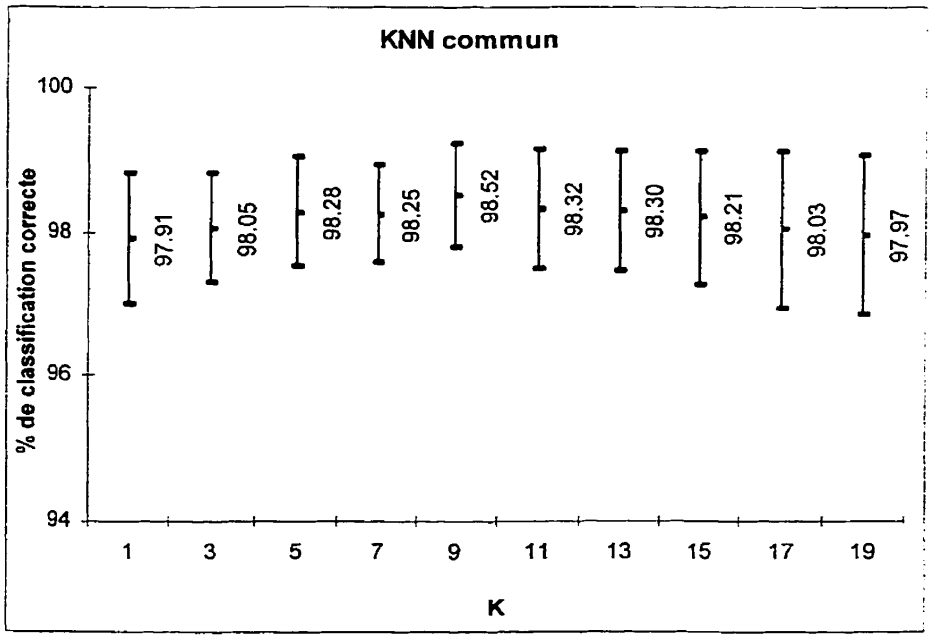



Figure D.1. Taux de classification correcte des classificateurs KNN (scénario commun).

Tableau D.2. Taux de classification correcte des classificateurs KNN (scénario personnalisé).

| $K$ | $s$ | $\bar{y}_{med}$ | $(\bar{y}_{min}, \bar{y}_{max})$ |
|---|---|---|---|
| *1* | **1.63** | **98.38** | **(97.57, 99.18)** |
| *3* | **1.65** | **98.21** | **(97.39, 99.02)** |
| *5* | **2.62** | **97.78** | **(96.48, 99.07)** |
| *7* | **3.80** | **97.15** | **(95.27, 99.03)** |
| *9* | **4.87** | **96.02** | **(93.62, 98.43)** |
| *11* | **7.33** | **95.11** | **(91.48, 98.73)** |
| *13* | **7.17** | **94.25** | **(90.70, 97.79)** |
| *15* | **6.79** | **92.70** | **(89.34, 96.06)** |
| *17* | **7.11** | **92.01** | **(88.49, 95.52)** |
| *19* | **6.91** | **90.65** | **(87.23, 94.06)** |

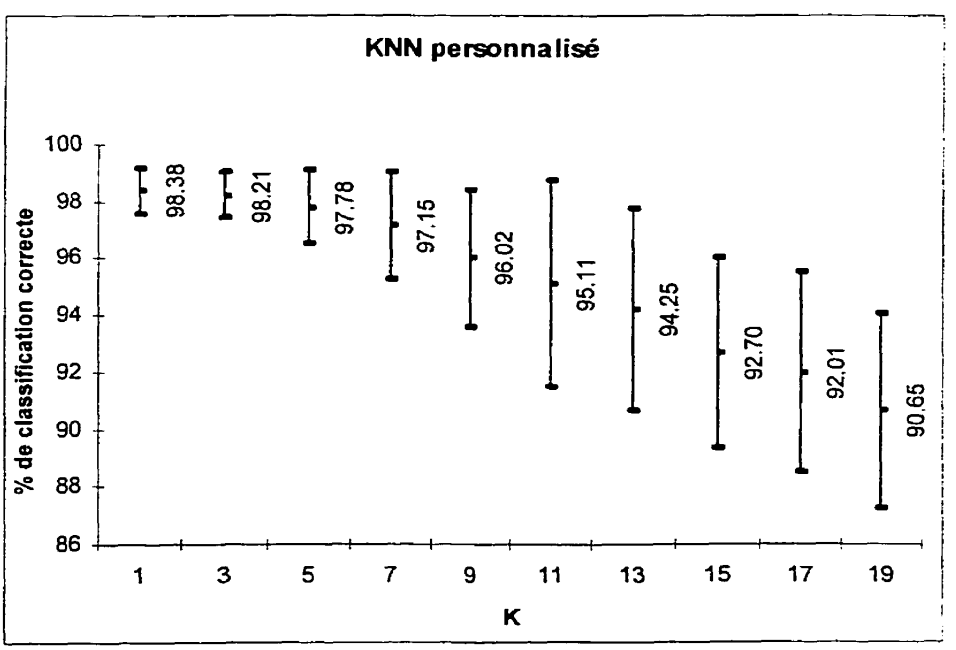



Figure D.2. Taux de classification correcte des classificateurs KNN (scénario personnalisé).

À partir des tableaux D.1 et D.2, on peut tirer les conclusions suivantes. Premièrement, le meilleur taux obtenu est de 98.52%, lequel appartient à un classificateur KNN avec K = 9, dans le cas du scénario commun. Deuxièmement, ce taux est inférieur à ceux de nos meilleurs classificateurs neuronaux (98.66%) et AdaBoost (99.06%). Troisièmement, les résultats du scénario personnalisé sont pires que ceux du scénario commun et cela à cause de l'importance de l'utilisation d'un grand nombre de patrons de référence dans l'algorithme de K plus proches voisins.

En plus des résultats numériques de classification, les classificateurs KNN souffrent des deux problèmes suivants : la quantité élevée de mémoire nécessaire pour garder tous les exemples en même temps et le temps élevé d'exécution de l'algorithme. Si $N_{ref}$ est le nombre de patrons de référence, le nombre d'opérations à effectuer pour classifier un patron donné sera égal à :

$$N_{op} = N_{ref}\ Racine + (6*N_{ref})\ Multiplication + (5*N_{ref})\ Addition \qquad (D.1)$$

Comme nous l'avons mentionné ci-dessus, il est important d'utiliser un nombre élevé de patrons de référence $N_{ref}$ dans l'algorithme de K plus proches voisins. Le prix à payer, selon l'équation (D.1) sera l'augmentation considérable du nombre de calculs et de temps de classification.

En somme, le taux de classification inférieur, les résultats moins stables (l'intervalle de confiance plus grand), le temps d'exécution et la quantité de mémoire supérieure par rapport aux classificateurs neuronaux et AdaBoost ainsi que la nécessité d'utiliser un grand nombre d'exemples de référence pour avoir des résultats satisfaisants (ce qui est la cause de leur sous-performance dans le cas du scénario personnalisé) montrent l'infériorité des classificateurs KNN par rapport aux classificateurs neuronaux et AdaBoost pour la segmentation en-ligne de textes manuscrits, et cela malgré leur simplicité de structure et facilité relative d'implantation.