

Etude et implémentation d'un classificateur basé sur des mixture models

PROJET DE SEMESTRE

Table des matières

Table des matières	2
Table des figures	3
Résumé	4
Préface	5
1 Définition du problème de classification	6
1.1 Définition générale	6
1.2 Approche déterministe	6
1.3 Approche probabiliste	7
2 Résolution du problème de classification avec l'approche déterministe	8
2.1 Perceptron multicouches	8
2.2 Algorithme backpropagation	9
3 Résolution du problème de classification avec l'approche probabiliste	11
3.1 Mixture model	11
3.2 Algorithme EM	12
4 Problème des valeurs manquantes	15
5 Utilisation de données de classe inconnue	16
5.1 Classification itérative	16
5.2 Utilisation des probabilités à priori	17
5.3 Comparaison des deux méthodes	19
6 Comparaison des performances de généralisation entre les approches déterministes et probabilistes	20
Conclusion	22
Annexe A Implémentation	23
Bibliographie	24

Table des figures

Fig. 2.1 :	Perceptron multicouches	8
Fig. 2.2 :	Algorithme backpropagation	9
Fig. 3.1 :	Classificateur	11
Fig. 3.2 :	Algorithme EM	13
Fig. 3.3 :	Vraisemblance en fonction du nombre d'itérations	13
Fig. 3.4 :	Classification de bactéries	14
Fig. 5.1 :	Classification itérative	16
Fig. 5.2 :	Algorithme EM avec probabilités à priori	17
Fig. 5.3 :	Classification avec probabilité à priori	18
Fig. 5.4 :	Comparaison des erreurs d'apprentissage	19
Fig. 5.5 :	Comparaison des erreurs des test	19
Fig. 6.1 :	Modèle probabiliste	20
Fig. 6.2 :	Modèle déterministe	21

Résumé

Le problème de la classification est un problème qui se rencontre fréquemment dans l'industrie et la recherche. Parmi les différentes solutions qui ont été développées pour de tels problèmes, il existe deux ensembles principaux : les modèles déterministes et les modèles probabilistes. Dans ce projet je me propose d'étudier et d'implémenter un classificateur basé sur le modèle probabiliste. Le classificateur est construit à partir de mixture models et utilise l'algorithme EM pour la phase d'apprentissage.

Durant ce travail j'ai également été confronté au problème des valeurs manquantes dans les données à disposition. Comme la plupart des méthodes statistiques travaillent sur des données complètes, il est nécessaire d'estimer des valeurs de substitution. L'idée consiste à estimer la distribution de probabilité de chaque caractéristique sur la base de kernels et à l'aide de l'algorithme EM. Un algorithme EM ne travaillant que sur une seule dimension est utilisé et la distribution de chacune des caractéristiques est estimée comme une somme pondérée de 2 à 4 kernels normaux univariés. Cette méthode a l'avantage d'estimer les valeurs de substitution indépendamment pour chaque caractéristique et chaque classe. La valeur de substitution est égale au centre du kernel ayant le plus grand poids.

Un autre problème abordé dans ce projet consiste à utiliser un ensemble d'exemples sans label durant la phase d'apprentissage. Deux méthodes tirant parti de ces données ont été développées :

La première méthode se base sur un processus itératif, constitué de deux phases. Durant la première phase un modèle est construit à l'aide d'un ensemble de données dont le label est connu. Puis durant la seconde phase, des données inconnues sont classifiées, et ajoutées à l'ensemble des données connues. Le processus est renouvelé jusqu'à ce que toutes les données inconnues soient classifiées. La classe attribuée correspond à celle ayant la plus forte probabilité.

La seconde méthode est proche de la classification itérative, mais évite d'attribuer définitivement une classe aux données inconnues. Elle utilise pour cela une version modifiée de l'algorithme EM qui tient compte des probabilités à priori. Cette mesure permet d'étendre le volume de données à disposition, en incluant des exemples incertains. La construction du classificateur est constitué d'un processus itératif. Un modèle initial est construit sur l'ensemble des données dont le label est connu. Les probabilités postérieures des données inconnues sont ensuite estimées à l'aide du modèle et remplacent les probabilités à priori précédentes. Le processus est itéré jusqu'à la convergence de la classification.

L'utilisation des probabilités à priori apporte une grande souplesse dans l'ajout de nouvelles données et dans la correction des probabilités estimées. Par contre, comparé à la classification itérative, cette méthode est plus sensible aux variations (migrations) et converge légèrement plus lentement.

En conclusion je peux dire que l'étude et l'implémentation d'un classificateur basé sur des mixture models m'ont permis de dégager quelques atouts qui lui donne tout son intérêt.

Premièrement, le calcul des probabilités à posteriori permettent une interprétation intuitive des résultats de classification. Cet élément est très appréciable pour la mesure de la confiance accordées aux décisions qui en découlent.

Deuxièmement, le temps calcul nécessaire à la construction du classificateur est passablement court. Cette caractéristique autorise un taux élevé d'expérimentation et se prête bien à l'ajout de données online (évolution de l'environnement, nouveaux cas, etc.).

En contrepartie, les résultats médiocres de généralisations obtenus sur les différents problèmes de test (60%, 70% de classification correcte), donnent plus de crédibilité aux méthodes déterministes.

Préface

Le problème de la classification est un problème qui se rencontre fréquemment dans l'industrie et la recherche. Il est surtout connu sous l'aspect d'un problème de reconnaissance (reconnaissance d'écriture manuscrite, reconnaissance vocale, etc.), mais se retrouve également dans beaucoup de problèmes de séparation ou de regroupement. Par exemple, la séparation de bactéries en fonction de leur espèce ou le regroupement d'assurés en classe à risque. Parmi les différentes solutions qui ont été développées pour de tels problèmes, il existe deux ensembles principaux : les modèles déterministes et les modèles probabilistes. Dans ce projet je me propose d'étudier et d'implémenter un classificateur basé sur le modèle probabiliste, et plus précisément construit à partir de mixture models.

Ce rapport expose le travail effectué et les réflexions qui en ont résulté.

- Le chapitre 1 pose le problème de la classification et introduit les approches déterministes et probabilistes.
- Le chapitre 2 expose l'approche déterministe en s'appuyant sur un perceptron multicouche et l'algorithme backpropagation.
- Le chapitre 3 expose l'approche probabiliste en utilisant une structure basée sur les mixture models et sur l'algorithme EM.
- Le chapitre 4 décrit une solution au problème des valeurs manquantes dans les systèmes basés sur l'apprentissage.
- Le chapitre 5 décrit deux approches pour l'utilisation de données sans label dans un problème d'apprentissage supervisé.
- Le chapitre 6 compare les performances de généralisation d'un perceptron multicouche et d'un classificateur à base de kernels.

Finalement le rapport se termine par une conclusion sur le travail effectué.

En annexe se trouve une description de l'implémentation en Java du classificateur.

Plusieurs raisons m'ont incité à étudier un tel modèle :

Premièrement, la réduction du nombre de paramètres libres au profit d'une plus grande utilisation de l'information contenue dans les données. Contrairement au modèle classique du perceptron multicouche qui a au moins deux paramètres libres (la taille de la couche cachée et le taux d'apprentissage), un classificateur basé sur les mixture models n'en nécessite qu'un (le nombre de kernels).

Deuxièmement, le désir de construire un classificateur intuitif. Des probabilités comme résultats de la classification, permettent une interprétation simple et intuitive.

Finalement, l'envie d'avoir une expérience pratique avec le modèle probabiliste. Le cours de Monsieur W.Gerstner "Réseaux de neurones artificiels" m'a permis d'implémenter et d'expérimenter un perceptron multicouche et l'algorithme backpropagation dans le cadre d'un mini-projet, mais le modèle probabiliste est resté à l'état de théorie.

Je tiens à remercier particulièrement les personnes suivantes :

Monsieur W.Gerstner pour son cours "Réseaux de neurones artificiels" qui m'a donné goût pour le domaine de l'apprentissage (machine learning), et pour les bases théoriques qui m'ont permis de mener un projet de manière quasi autonome; Silvio Borer pour son suivi durant un semestre et pour ses conseils dans l'utilisation des données sans label; Lionel Weber pour la mise en page et l'impression du présent document.

Chapitre 1

Définition du problème de classification

1.1 Définition générale

Le problème de classification consiste à partitionner un ensemble d'objet en un ensemble de classes. Il est d'usage de considérer un ensemble d'objets physiques, par exemple une population de bactéries ou de poissons, et de vouloir les répartir en différentes classes, par exemple les espèces. Pour effectuer une telle tâche, il est nécessaire de récolter une certaine quantité d'information sur les objets physiques considérés, appelés caractéristiques ou features en anglais. Ces informations sont ensuite utilisées pour la construction d'un modèle de classification. En observant une telle démarche, deux problèmes se posent aussitôt :

Premièrement, il faut récolter certaines caractéristiques sur les objets physiques. Il faut donc décider de l'ensemble des caractéristiques utiles à la classification, et de la procédure d'acquisition (les unités utilisées, mesures brutes ou transformées, valeur absolue ou relative, etc.). En pratique, les mesures sont souvent coûteuses, susceptibles de contenir des erreurs et souvent incomplètes, réduisant ainsi le volume des données à disposition. De plus, l'ensemble des caractéristiques aptes à produire une bonne classification étant inconnu, les données choisies résulte d'un compromis entre le désir d'incorporer un maximum d'information et la faisabilité des mesures.

Deuxièmement, il faut construire un processus de classification sur les données récoltées. Il n'y a évidemment pas de modèle connu à priori ou universel, et là encore le choix dépend des données.

Deux modèles principaux existent : le modèle déterministe qui utilise une fonction de classification et le modèle probabiliste qui se base sur les distributions de probabilités des données étudiées. Dans la suite du rapport, les deux approches sont développées et comparées.

1.2 Approche déterministe

Avec l'approche déterministe, la classification consiste à trouver une fonction classify qui attribue à tout objet physique une certaine classe. Ou plutôt, qui attribue à tout vecteur de caractéristiques d'un objet physique une certaine classe. La classification est parfaite si pour tout objet physique, la classe déterminée par la fonction classify est égale à la classe de l'objet. Formellement on obtient :

Soit o un objet physique;

$c(o)$ la classe de l'objet o ;

$f(o)$ les caractéristiques ou features de o ;

Alors la classification est parfaite ssi pour tout objet o : $c(o) = \text{classify}(f(o))$ (1.1)

1.3 Approche probabiliste

Avec l'approche probabiliste, la classification consiste à déterminer la distribution des classes des objets. Il s'agit donc d'estimer pour tout objet, la probabilité d'avoir une certaine classe étant donné ses caractéristiques. Formellement on obtient :

Soient o un objet physique, c une classe;
 $f(o)$ les caractéristiques ou features de l'objet o ;
Alors on calcule pour chaque classe c et tout objet o , la probabilité $p(c | f(o))$ (1.2)

La classification est parfaite si pour tout objet physique, la probabilité calculée selon l'équation 1.2 pour sa classe, est maximale. Formellement on obtient :

Soit o un objet physique;
 $c(o)$ la classe de l'objet o ;
 $f(o)$ les caractéristiques ou features de o ;
**Alors la classification est parfaite ssi pour tout objet o ,
la probabilité $p(c(o) | f(o)) > p(c' | f(o))$ pour toute classe c' différente de $c(o)$** (1.3)

Chapitre 2

Résolution du problème de classification avec l'approche déterministe

Ce chapitre présente une résolution possible du problème de classification à l'aide de l'approche déterministe. La fonction de classification est un perceptron multicouches dont les poids sont modifiés selon une politique d'apprentissage supervisé avec l'algorithme backpropagation.

2.1 Perceptron multicouches

Le Perceptron multicouches est un réseau de neurones constitués de plusieurs couches de neurones formels (type proposé par McCulloch Pitts, 1943). En général, il est constitué de trois couches : une couche d'entrée, une couche intermédiaire appelée couche cachée et une couche de sortie. Dans un problème de classification la taille de la couche d'entrée est égale à la dimension des données étudiées, la taille de la couche de sorties est égale au nombre de classes (classification 1 parmi n) et finalement la couche cachée est de taille variable. Cette couche intermédiaire permet au réseau de construire une représentation interne des données étudiées. Le nombre de neurones nécessaires croît avec la complexité du problème. Pour terminer on peut ajouter que ce réseau ne comporte aucun cycle (type feedforward), est totalement connecté (fully connected) et que chaque neurone possède une fonction d'activation du type $\arctan(x)$.

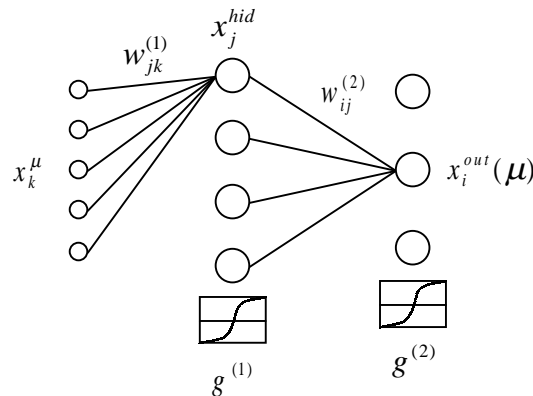


Fig. 2.1 : Perceptron multicouches

La sortie est calculée selon la formule :

$$x_i^{out}(\mu) = g^{(2)} \left[\sum_j w_{ij}^{(2)} g^{(1)} \left(\sum_k w_{jk}^{(1)} x_k^\mu \right) \right] \quad (2.1)$$

La phase d'apprentissage modifie les poids w à l'aide de l'algorithme backpropagation.

2.2 Algorithme backpropagation

L'algorithme backpropagation est basé sur une descente de gradient. Il essaie de minimiser l'erreur quadratique totale :

$$\Delta w_{ij}^{(k)}(\boldsymbol{\mu}) = -\eta \frac{dE}{dw_{ij}^{(k)}} \text{ où } \eta \text{ est le taux d'apprentissage} \quad (2.2)$$

L'erreur est minimisée sur l'ensemble des données, on parle dans ce cas d'une version batch. Il existe aussi une version online qui minimise à chaque itération l'erreur de l'exemple courant :

$$E(w^{(1)}, w^{(2)}, \boldsymbol{\mu}) = \frac{1}{2} \sum_i [t_i^\mu - x_i^{out}(\boldsymbol{\mu})]^2 \quad (2.3)$$

La version online correspond à une descente du gradient stochastique et est préférée en pratique pour des raisons de performances. La mise à jour des poids w se fait donc selon la formule :

$$E(w^{(1)}, w^{(2)}) = \frac{1}{2} \sum_{\mu} \sum_i [t_i^\mu - x_i^{out}(\boldsymbol{\mu})]^2 \quad (2.4)$$

Après quelques calculs on obtient finalement l'algorithme suivant :

0. Initialization of weights

1. Choose pattern x^μ , apply at input :

$$x_k^{(0)} = x_k^\mu$$

2. Forward propagation of signals :

$$x_j^{(n)} = g^{(n)}(h_j^{(n)}) = g^{(n)}\left(\sum_k w_{jk}^{(n)} x_k^{(n-1)}\right)$$

$$x_i^{out}(\boldsymbol{\mu}) = x_i^{(n_{\max})}$$

3. Computations of error in output :

$$\delta_i^{out} = g'(h_i^{out}) [t_i^\mu - x_i^{out}(\boldsymbol{\mu})]$$

4. Backward propagations of errors :

$$\delta_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij} \delta_i^{(n)}$$

5. Update weights :

$$\Delta w_{ij}^{(n)} = \eta \delta_i^{(n)} x_j^{(n-1)}$$

6. Return to step 1

Fig. 2.2 : Algorithme backpropagation

Quelques observations importantes découlent d'un tel modèle :

Premièrement, il y a au minimum deux paramètres libres à fixer, à savoir la taille de la couche cachée et le taux d'apprentissage. Il peut y avoir des paramètres supplémentaires si le processus est affiné pour prendre en considération des méthodes de régulations, comme par exemple la gestion d'un momentum ou la mise en place du weight decay.

Deuxièmement, la fonction d'erreur converge souvent lentement et la complexité de calcul est de l'ordre du nombre de poids $O(w)$. Ces deux facteurs font que cette méthode d'apprentissage est très gourmande en temps de calcul et peut devenir critique pour des problèmes réels.

Troisièmement, si la durée d'apprentissage est trop longue, le réseau se spécialise trop sur les données d'apprentissage et diminue son degré de généralisation. On parle dans ce cas d'overfitting. Il est donc nécessaire de mettre en place des méthodes de contrôle pour prévenir l'overfitting comme par exemple l'early-stopping, le weight decay ou l'élagage du réseau.

Finalement, il est difficile pour un humain d'interpréter le modèle de classification que le réseau a construit et il manque une mesure de la confiance avec laquelle une donnée est classifiée.

Malgré ces quelques points à observer, ce modèle donne souvent de bons résultats de généralisation et possède l'avantage d'avoir une méthode d'apprentissage online.

Chapitre 3

Résolution du problème de classification avec l'approche probabiliste

Ce chapitre présente une résolution possible du problème de classification à l'aide de l'approche probabiliste. Le classificateur est basé sur des kernels dont les paramètres sont estimés avec l'algorithme Expectation-Maximisation (EM).

3.1 Mixture model

n classificateur construit sous la forme d'un mixture model utilise trois couches, comme le perceptron présenté au chapitre 2 :

La première répartit des kernels dans l'espace des données étudiées. Les kernels sont basés sur une distribution normale multivariée de la forme :

$$N(\vec{\mu}_j, \sigma_j, \vec{x}) = \frac{1}{(2\pi)^{d/2} \sigma^d} \exp\left(\frac{-1}{2\sigma^2} \|\vec{x} - \vec{\mu}_j\|^2\right) \text{ où } d \text{ est la dimension de } x \quad (3.1)$$

Le nombre de kernels utilisés est un paramètre libre qui dépend de la complexité du problème.

La seconde couche estime pour chaque classe la distribution des données étudiées. Cette distribution est égale à une somme pondérée des kernels de la première couche.

Finalement, la troisième couche calcul pour chaque classe la probabilité qu'un vecteur de l'espace étudié appartienne à cette dernière.

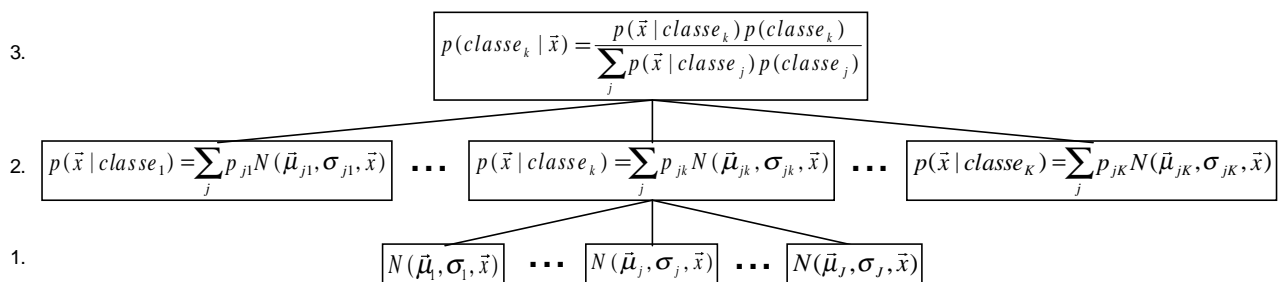


Fig. 3.1 : Classificateur

Pour être complète, la description du modèle nécessite l'ajout de quelques remarques importantes :

Premièrement, les kernels ne sont pas partagés entre les classes. Chaque classe possède un nombre identique de kernels uniques.

Deuxièmement, l'estimation des distributions de probabilités dans la seconde couche se fait indépendamment pour chaque classe. A ce niveau là, le processus construit autant de modèles qu'il existe de classes.

Troisièmement, la dernière couche combine les résultats de la couche précédente pour classifier un objet étudié, mais ne participe pas à l'apprentissage.

Ces trois remarques soulèvent des différences fondamentales avec le perceptron multicouches présentés au chapitre 2. Dans le perceptron, les deux dernières couches participent à l'apprentissage, le modèle de classification est global et le réseau est totalement connecté.

Pour classifier un vecteur x , le système calcule pour chaque classe c la probabilité $p(c | x)$. Le résultat est la classe dont la probabilité est maximale. Les sorties sous la forme de probabilités représentent un avantage fondamental par rapport au perceptron, car elles permettent une interprétation intuitive de la marge de confiance. Si le système ne parvient pas à classifier l'objet étudié, les probabilités seront égales à $1/\text{nombre de classes}$.

3.2 Algorithme EM

Dans un tel modèle, la phase d'apprentissage se concentre sur l'estimation des distributions de probabilités de la seconde couche :

$$p(\vec{x} | \text{classe}_k) = \sum_j p_j N(\vec{\mu}_j, \sigma_j, \vec{x}) \quad (3.2)$$

La méthode détermine les paramètres qui maximisent la vraisemblance (likelihood L) des données à disposition

$$\begin{aligned} L(p, \mu, \sigma) &= \sum_n \ln p(\vec{x}^n | \text{classe}_k) \\ &= \sum_n \ln \left[\sum_j p_j N(\vec{\mu}_j, \sigma_j, \vec{x}^n) \right] \end{aligned} \quad (3.3)$$

à l'aide d'un algorithme itératif basé sur une montée du gradient (hill climbing) :

0. Initialization of μ , σ and p for all kernels :

μ_i = a vector in learning examples $\mu_i \neq \mu_j \forall j$

$\sigma_i = \min \|\mu_i - \mu_j\| \forall j$

$p_i = 1/\text{number of kernels}$

$\Rightarrow \theta_i^{\text{old}} = (\mu_i, \sigma_i)$

1. Repeat until likelihood convergence :

Mstep : maximisation

$$\begin{aligned} \vec{\mu}_j &= \frac{\sum_n p(\theta_j | \vec{x}^n) \vec{x}^n}{\sum_n p(\theta_j | \vec{x}^n)} \\ \sigma_j^2 &= \frac{1}{d} \frac{\sum_n p(\theta_j | \vec{x}^n) \|\vec{x}^n - \mu_j\|^2}{\sum_n p(\theta_j | \vec{x}^n)} \\ p_j &= \frac{1}{N} \sum_n p(\theta_j | \vec{x}^n) \end{aligned}$$

where :

d is the dimension of x ;

N is the number of learning examples

Estep : expectation

$$\begin{aligned}
 p(\theta_j | \bar{x}^n) &= \frac{p(\bar{x}^n | \theta_j^{old}) p(\theta_j^{old})}{\sum_i p(\bar{x}^n | \theta_i^{old}) p(\theta_i^{old})} \\
 &= \frac{N(\bar{\mu}_j^{old}, \sigma_j^{old}, \bar{x}^n) p_j^{old}}{\sum_i N(\bar{\mu}_i^{old}, \sigma_i^{old}, \bar{x}^n) p_i^{old}}
 \end{aligned}$$

where :

$N(\mu, \sigma, x)$ is a normal multivariate distribution

Fig. 3.2 : *Algorithme EM*

La mise à jour des paramètres (phase M) nécessite l'ensemble des exemples d'apprentissage, empêchant ainsi la création d'une version online. Mais cet inconvénient par rapport à l'algorithme backpropagation, n'empêche nullement la vraisemblance de converger extrêmement rapidement. La figure 3.3 montre qu'il suffit souvent d'une vingtaine d'itérations pour atteindre un maximum, qui est souvent global !

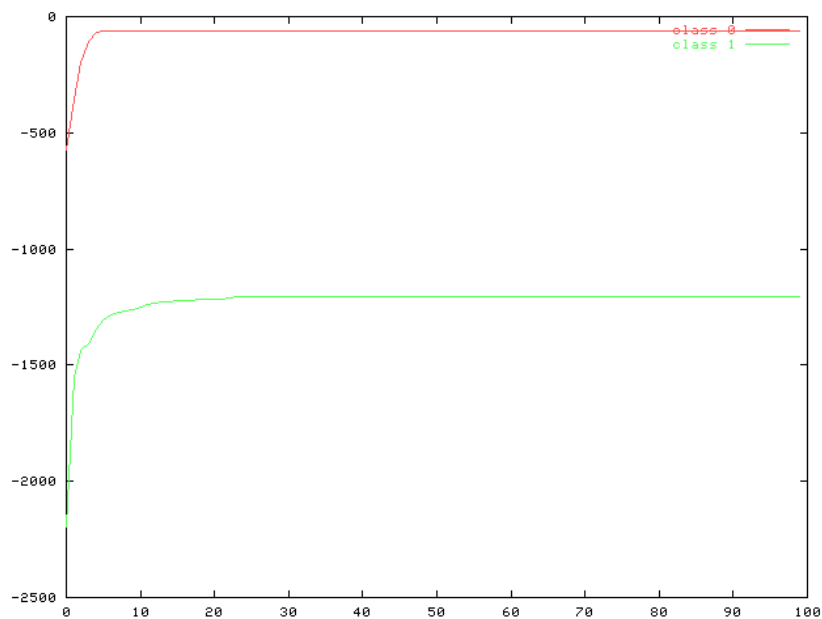


Fig. 3.3 : *Vraisemblance en fonction du nombre d'itérations*

Pour illustrer les propos ci-dessus, considérons un exemple consistant à classifier des bactéries en deux espèces à partir de 13 mesures effectuées sur chacune d'elle.

Le classificateur a 4 kernels par classe à disposition et 36 exemples de classe connue pour l'apprentissage. Après environ 30 itérations de l'algorithme EM, le classificateur a réparti ses kernels dans l'espace des données comme illustré par la figure 3.4.

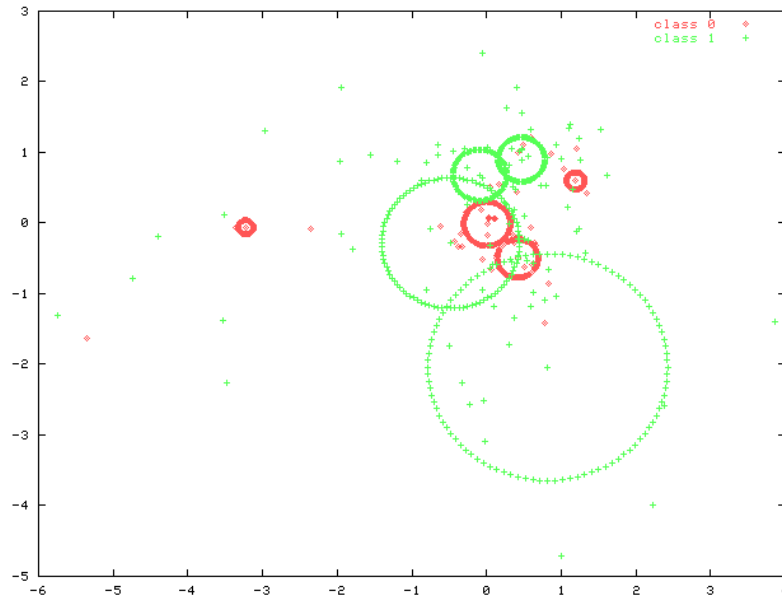


Fig. 3.4 : *Classification de bactéries*

Cette figure est une projection selon les deux premières composantes principales (PCA) de la répartition des kernels de chacune des classes dans l'espace à 13 dimensions. Chaque kernel est représenté par un cercle de rayon égal à sigma.

Chapitre 4

Problème des valeurs manquantes

En réalité, les données disponibles sont souvent incomplètes en raison d'un manque ou d'une perte d'information. Comme la plupart des méthodes statistiques travaillent sur des données complètes, il est nécessaire d'estimer les valeurs manquantes. La méthode la plus simple consiste à substituer les valeurs manquantes par la moyenne des valeurs connues. Cette méthode a le désavantage d'introduire un biais dans les données. Si l'on considère l'exemple suivant : des données sont séparées en deux clusters C1 et C2 avec $p(C1) > p(C2)$. La moyenne se situera donc en $p(C1)*\mu_1 + p(C2)*\mu_2$ où μ_1 et μ_2 sont les centres de C1 et C2.

Remplacer des valeurs manquantes par cette moyenne aura tendance à fusionner les deux clusters en un dont la variance est plus élevée. Partant de ce constat, l'idée consiste à estimer la distribution de probabilité de chaque caractéristique sur la base de kernels et à l'aide de l'algorithme EM.

La distribution de probabilité de la caractéristique i pour la classe k s'exprime par une somme pondérée de distributions normales univariées :

$$p(x_i | classe_k) = \sum_j p_j N(\mu_j, \sigma_j, x_i) \quad (4.1)$$

Un algorithme EM semblable à celui présenté dans le chapitre 3 (figure 3.2), mais ne travaillant que sur une seule dimension, est utilisé pour estimer les paramètres libres de l'équation 4.1. En pratique, il suffit de prendre 2 à 4 kernels, car les données d'une caractéristique sont souvent groupées autour de quelques valeurs. Cette méthode a l'avantage d'estimer les valeurs de substitution indépendamment pour chaque caractéristique et chaque classe. Elle peut donc utiliser dans l'apprentissage un vecteur contenant des valeurs manquantes pour autant qu'elles ne concernent pas la caractéristique estimée. Cela réduit la quantité de données nécessaire à l'apprentissage, mais néglige le potentiel d'information contenu dans les dépendances entre les caractéristiques.

La valeur de substitution est égale au centre du kernel ayant le plus grand poids. Avec un seul kernel, la substitution est égale à la moyenne.

Cette méthode est valide sous l'hypothèse d'une distribution des valeurs manquantes indépendante des clusters de données. On parle de données MCAR - missing completely at random (Little and Rubin, 1987, 1990). Dans le cas contraire, il faudrait essayer d'estimer la distribution des valeurs manquantes et choisir le centre du kernel couvrant la plus grande proportion.

Chapitre 5

Utilisation de données de classe inconnue

Il est souvent impossible de disposer d'un nombre important de données dont le label est connu en raison des coûts de mesures prohibitifs. Par contre, il est assez facile de rassembler un volume important d'exemples dont le label est inconnu. Le problème consiste à tirer le maximum d'information de ces données pour la construction du classificateur.

Dans la suite du chapitre deux méthodes tirant parti de ces données sont présentées. La première est basée sur une classification itérative et la seconde utilise des probabilités à priori.

5.1 Classification itérative

Cette méthode se base sur un processus itératif, constitué de deux phases. Durant la première phase un modèle est construit à l'aide d'un ensemble de données dont le label est connu. Puis durant la seconde phase, des données inconnues sont classifiées, et ajoutées à l'ensemble des données connues. Le processus est renouvelé jusqu'à ce que toutes les données inconnues soient classifiées. La classe attribuée correspond à celle ayant la plus forte probabilité. Cette méthode peut être affinée en choisissant la classe dont la probabilité est la plus forte parmi toutes les classes de probabilité supérieure à une certaine marge de confiance. Si aucune classe n'est trouvée, la donnée demeure inconnue. Cette méthode permet de classifier à chaque itération uniquement les données les plus sûres. Si le modèle initial n'est pas biaisé, il se renforcera jusqu'à la convergence.

En appliquant cette méthode au problème de classification des bactéries présenté au chapitre 3 (section 3.2), dont l'ensemble des données d'apprentissage est augmenté d'une centaine d'exemples de classe inconnue, la convergence est nettement visible.

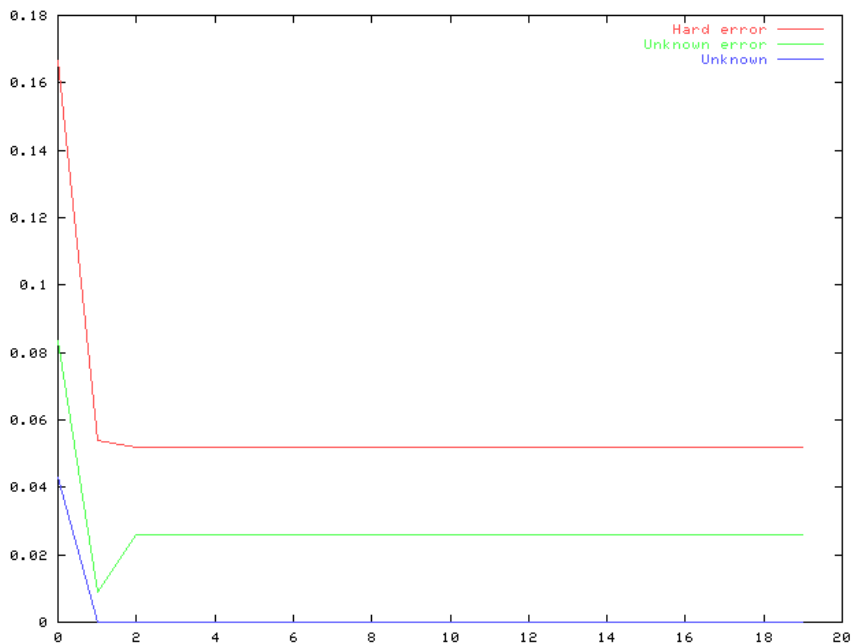


Fig. 5.1 : Classification itérative

La figure 5.1 montre qu'après 2 à 3 itérations, le taux d'erreur de classification des données connues (hard error), le taux d'exemples connus impossibles à classifier (unknown error) et le taux d'exemples inconnus (unknown) convergent vers des valeurs constantes, respectivement 5%, 3% et 0%. La chute très rapide du taux d'exemples inconnus, montre que le classificateur est capable d'intégrer rapidement de nouvelles données.

Même si cette méthode n'apporte pas forcément de grandes améliorations au niveau des performances de généralisations, elle permet d'augmenter progressivement le nombre de données à disposition pour l'apprentissage. Si le volume initial est relativement restreint mais bien distribué, cette méthode peut aider dans la construction d'un classificateur performant.

5.2 Utilisation des probabilités à priori

Cette méthode est proche de la classification itérative, mais évite d'attribuer définitivement une classe aux données inconnues. Elle utilise pour cela une version modifiée de l'algorithme EM qui tient compte des probabilités à priori. L'idée est d'ajouter à chaque exemple d'apprentissage x , sa probabilité à priori $q(C | x)$ d'appartenir à une classe C . Cette mesure permet d'étendre le volume de données à disposition, en incluant des exemples incertains.

On désire par exemple utiliser un vecteur x dans l'apprentissage d'un modèle à deux classes C_0 et C_1 . Si la classe de x est connue, par exemple C_0 , alors $q(C_0 | x) = 1$ et $q(C_1 | x) = 0$. Si par contre la classe de x est inconnue, mais x est supposé appartenir à la classe C_0 avec 80% de chance, alors $q(C_0 | x) = 0.8$ et $q(C_1 | x) = 0.2$.

En insérant la probabilité à priori $q(x)$ dans l'équation de la vraisemblance (cf. équation 3.3), on obtient :

$$\begin{aligned}
 L(p, \mu, \sigma) &= \sum_n q(x^n) \ln p(\vec{x}^n | \text{classe}_k) \\
 &= \sum_n q(x^n) \ln \left[\sum_j p_j N(\vec{\mu}_j, \sigma_j, \vec{x}^n) \right] \tag{5.1}
 \end{aligned}$$

Et finalement l'algorithme EM (cf. fig 3.2) devient :

0. Initialization of μ , σ and p for all kernels :

μ_i = a vector in learning examples $\mu_i \neq \mu_j \forall j$
 $\sigma_i = \min \|\mu_i - \mu_j\| \forall j$
 $p_i = 1/\text{number of kernels}$
 $\Rightarrow \theta_i^{\text{old}} = (\mu_i, \sigma_i)$

1. Repeat until likelihood convergence :

Mstep : maximisation

$$\begin{aligned}
 \vec{\mu}_j &= \frac{\sum_n q(x^n) p(\theta_j | \vec{x}^n) x^n}{\sum_n q(x^n) p(\theta_j | \vec{x}^n)} \\
 \sigma_j^2 &= \frac{1}{d} \frac{\sum_n q(x^n) p(\theta_j | \vec{x}^n) \|\vec{x}^n - \vec{\mu}_j\|^2}{\sum_n q(x^n) p(\theta_j | \vec{x}^n)} \\
 p_j &= \frac{1}{N} \sum_n q(x^n) p(\theta_j | \vec{x}^n)
 \end{aligned}$$

where :

d is the dimension of x ;

N is the number of learning examples;

$q(x^n)$ is the prior probability of example x

Estep : expectation

$$\begin{aligned}
 p(\theta_j | \vec{x}^n) &= \frac{p(\vec{x}^n | \theta_j^{\text{old}}) p(\theta_j^{\text{old}})}{\sum_i p(\vec{x}^n | \theta_i^{\text{old}}) p(\theta_i^{\text{old}})} \\
 &= \frac{N(\vec{\mu}_j^{\text{old}}, \sigma_j^{\text{old}}, \vec{x}^n) p_j^{\text{old}}}{\sum_i N(\vec{\mu}_i^{\text{old}}, \sigma_i^{\text{old}}, \vec{x}^n) p_i^{\text{old}}}
 \end{aligned}$$

where :

$N(\mu, \sigma, x)$ is a normal multivariate distribution

Fig. 5.2 : Algorithme EM avec probabilités à priori

La construction du classificateur est constitué d'un processus itératif. Un modèle initial est construit sur l'ensemble des données dont le label est connu, en utilisant la structure présenté dans la section 3.1 (mixture model) et l'algorithme EM modifié (figure 5.2). Les probabilités postérieures $p(c | x)$ des données inconnues sont ensuite estimées à l'aide du modèle et remplacent les probabilités à priori $q(c | x)$ précédentes. Le processus est itéré jusqu'à la convergence de la classification.

En comparaison avec la classification itérative (section 5.1), la convergence pour le problème des bactéries est légèrement plus lente.

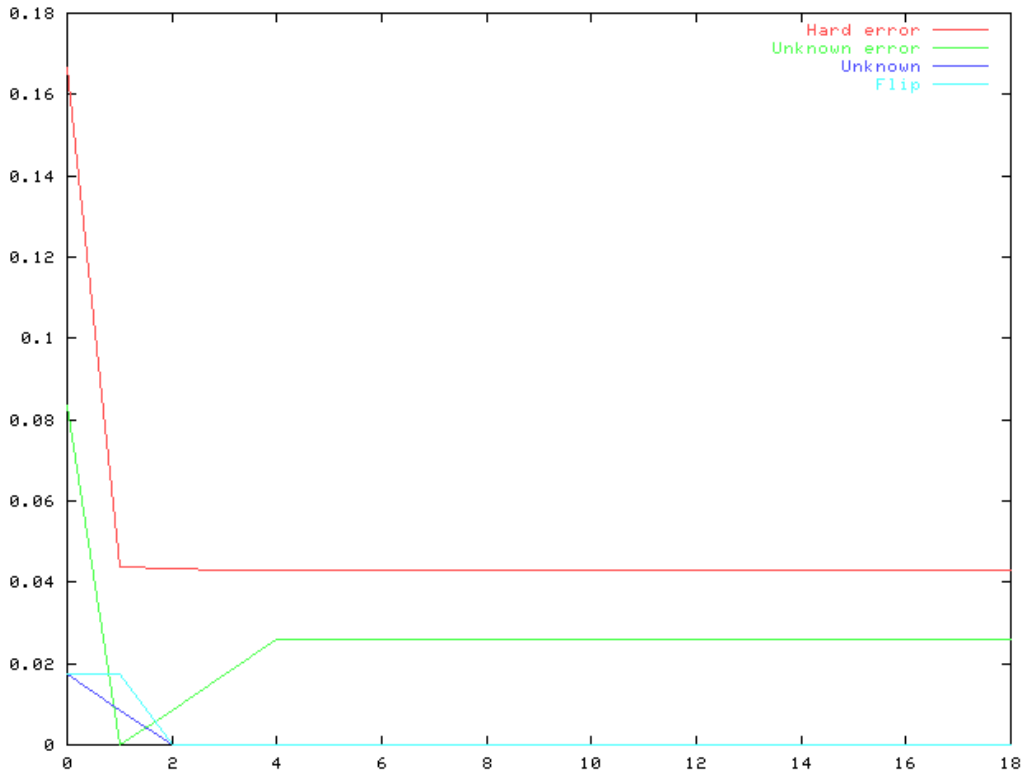


Fig. 5.3 : Classification avec probabilité à priori

Quatre itérations sont en effet nécessaires pour atteindre des taux d'erreurs constants. Le taux d'erreur de classification des données connues (hard error) est légèrement inférieur (4%) à la méthode itérative (5%). Comme les données ne sont pas attachées définitivement à une classe, elles peuvent changer d'identité au cours de l'apprentissage. Le nombre de migrations à chaque itération (courbe Flip sur le graphique) donnent des résultats intéressants. Durant les 3 premières itérations environ 2% des données changent de classe. Les données concernées sont essentiellement situées sur les zones frontières des classes. Puis à partir de la quatrième itération, les migrations cessent, toutes les données sont adoptées par une classe ou sont situées exactement sur la frontière empêchant toute décision. Cette dernière possibilité expliquerait la croissance du taux d'exemples connus impossible à classifier (unknown error) de 0 à 3%.

Dans ce projet, j'ai choisi d'itérer le processus jusqu'à la convergence, mais on aurait pu imaginer exécuter la boucle seulement deux fois : une première fois pour construire un modèle sur les données dont le label est connu, et une seconde fois pour intégrer les données dont le label est inconnu. Dans ce cas le modèle final risque d'être légèrement moins biaisé, mais il n'est pas évident que la généralisation donne de meilleurs résultats.

5.3 Comparaison des deux méthodes

Cette section compare l'erreur d'apprentissage et l'erreur de généralisation de la méthode itérative avec celle utilisant les probabilités à priori pour le problème des bactéries.

La figure 5.4 montre l'erreur d'apprentissage en fonction du nombre de kernels par classe :

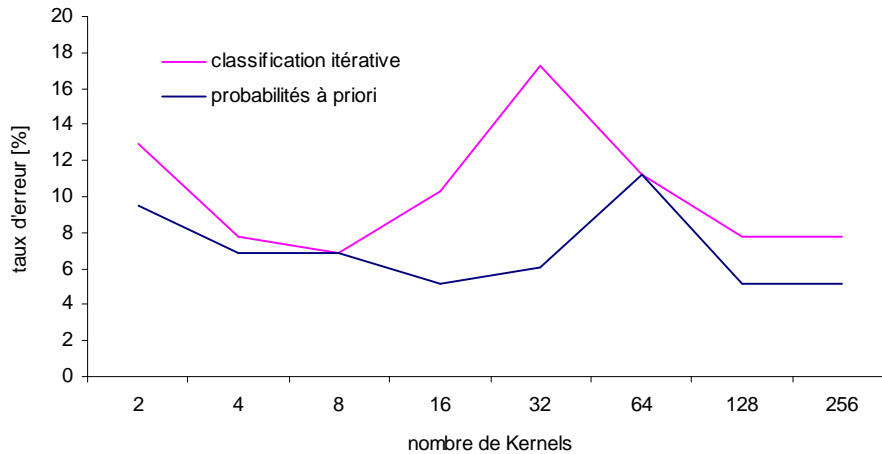


Fig. 5.4 : Comparaison des erreurs d'apprentissage

L'utilisation des probabilités à priori donne des résultats d'apprentissage légèrement supérieurs à la méthode itérative. Les probabilités à priori apportent en effet une souplesse supplémentaire au modèle qui évite de classifier trop impérativement certains exemples difficiles.

Le graphique montre également une certaine précarité de la configuration. Avec 8 kernels le taux d'erreur d'apprentissage est minimal et égal pour les deux méthodes, mais il suffit de prendre 32 kernels pour que la méthode itérative donne un taux d'erreur avoisinant les 18%, alors que l'autre méthode plafonne à 6%.

Au niveau des performances de généralisation, les deux méthodes sont équivalentes. La figure 5.5 montre l'erreur de généralisation en fonction du nombre de kernels par classe :

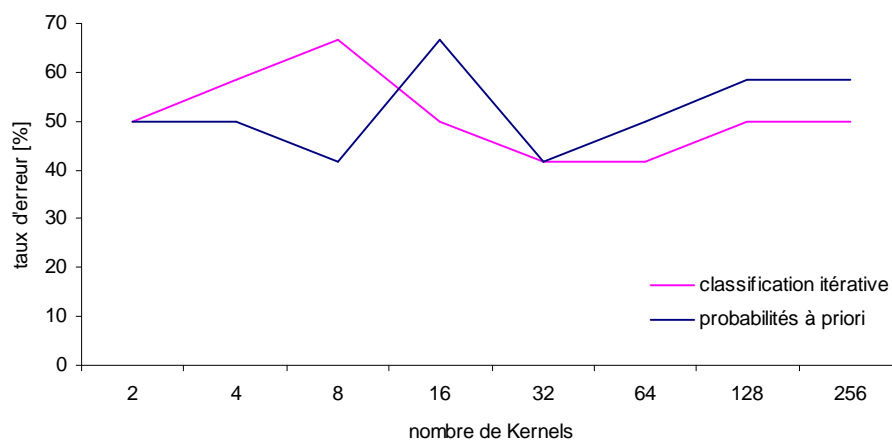


Fig. 5.5 : Comparaison des erreurs des test

En conclusion, on peut dire que l'utilisation des probabilités à priori apporte une grande souplesse dans l'ajout de nouvelles données et dans la correction des probabilités estimées. Il devient possible d'ajouter des données au fur et à mesure de leur disponibilité, ou de modifier les distributions des données si l'environnement extérieur venait à changer. Par contre, comparé à la classification itérative, cette méthode est plus sensible aux variations (migrations) et converge légèrement plus lentement.

Mais dans tous les cas, le choix de la méthode dépend des besoins et les résultats finaux restent fortement influencé par la configuration.

Chapitre 6

Comparaison des performances de généralisation entre les approches déterministes et probabilistes

Un classificateur basé sur des kernels a une vision très locale. Un kernel par sa construction ne s'occupe que d'une petite partie de l'espace. Et la somme de toutes ces contributions locales permet de construire un modèle global. Il faut souvent beaucoup de données pour construire un modèle performant sur une large étendue. Par contre, le modèle s'adapte très bien à des distributions de probabilités complexes, constitués par exemple de petits îlots émergents dans un vaste espace. On pourrait donc parler d'une généralisation prudente qui construit un modèle sur les données réellement disponibles.

Un classificateur basé sur un perceptron multicouches construit une surface de séparation entre une classe et le reste. La surface par construction coupe tout l'espace; ensuite suivant la flexibilité du réseau (le nombre de neurones dans la couche cachée), la surface peut prendre une forme plus ou moins complexe. Une différence fondamentale avec le modèle probabiliste provient de la portée globale de la classification. En effet, à partir de quelques données, une surface de séparation qui coupe tout l'espace est induite. On pourrait donc parler d'une généralisation risquée, dans le sens où le modèle prend une décision même pour les régions qu'il n'a jamais eu connaissance.

Afin de comparer la puissance de généralisation du modèle probabiliste par rapport au modèle déterministe, j'ai utilisé les données du problème de classification étudié dans le cadre du mini-projet du cours "Réseaux de neurones artificiels" de Monsieur W.Gerstner. Le problème consiste à séparer des séquences d'ADN de 60 bases en trois classes IE, EI, N selon qu'elles contiennent respectivement une transition Intron-Exon, une transition Exon-Intron ou aucune (None). La base d'apprentissage contient 3175 séquences de classe connue. Le modèle déterministe est basé sur un perceptron multicouches et l'algorithme backpropagation. Pour des raisons de temps de calcul, il utilise 600 exemples d'apprentissages et 300 exemples de test. Le modèle probabiliste utilise le classificateur présenté dans ce rapport, et travaille avec 1000 exemples d'apprentissage et 100 exemples de test.

La figure 6.1 montre l'erreur de classification du modèle probabiliste en fonction du nombre de kernels.

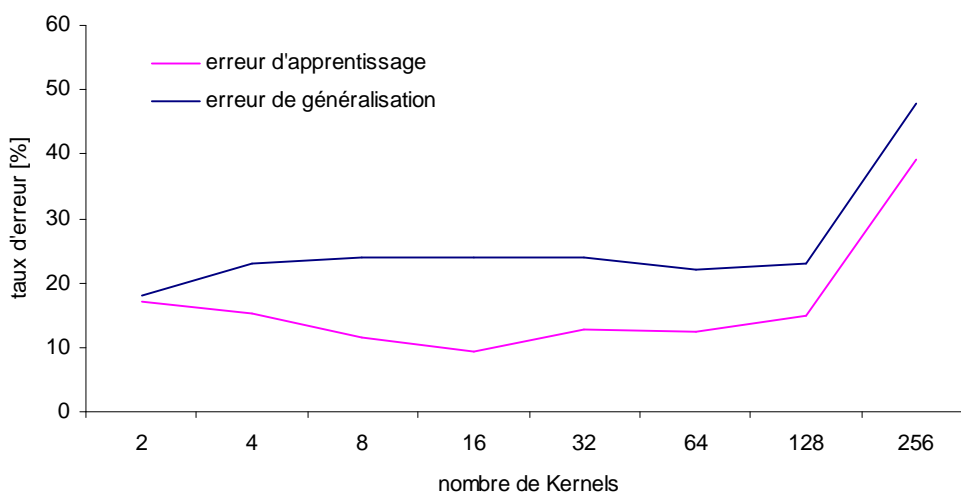


Fig. 6.1 : *Modèle probabiliste*

La figure 6.2 montre l'erreur de classification du modèle déterministe en fonction du nombre de neurones dans la couche cachée pour trois méthodes d'apprentissage : un apprentissage simple à l'aide de l'algorithme backpropagation, un apprentissage régularisé avec la méthode early stopping et enfin un apprentissage régularisé avec la méthode du weight decay.

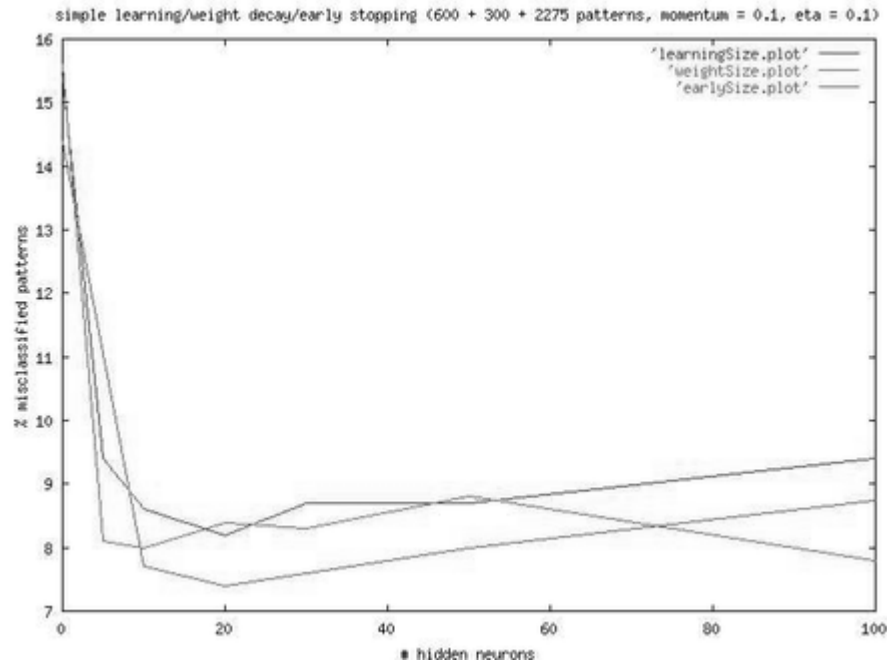


Fig. 6.2 : *Modèle déterministe*

En observant ces deux figures, on remarque que l'erreur moyenne de classification pour le modèle probabiliste est de 20% tandis que pour le modèle déterministe elle est de 8.5%. Les performances de généralisation du modèle probabiliste sont nettement inférieures et posent donc la question de la validité d'un tel modèle dans des espaces à très grande dimension.

Il est en tout cas certain que le choix de la méthode de classification dépend du problème à résoudre et de contraintes extérieures comme le nombre de données disponibles ou le temps de calcul nécessaire. Pour des problèmes facilement séparables, le perceptron donnera sûrement de meilleurs résultats. Pour des problèmes avec un volume important de données disponibles ou nécessitant une interprétation intuitive des résultats, un classificateur basé sur les mixture models est préférable.

Conclusion

L'étude et l'implémentation d'un classificateur basé sur des mixture models m'ont permis de dégager quelques atouts qui lui donne tout son intérêt.

Premièrement, le calcul des probabilités à posteriori permettent une interprétation intuitive des résultats de classification. Cet élément est très appréciable pour la mesure de la confiance accordées aux décisions qui en découlent.

Deuxièmement, le temps calcul nécessaire à la construction du classificateur est passablement court. Cette caractéristique autorise un taux élevé d'expérimentation et se prête bien à l'ajout de données online (évolution de l'environnement, nouveaux cas, etc.).

En contrepartie, les résultats médiocres de généralisations obtenus sur les différents problèmes de test (60%, 70% de classification correcte), donnent plus de crédibilité aux méthodes déterministes. Un travail futur possible consisterait à reprendre comme base l'implémentation des classes de gestion des entrées-sorties et des distributions de probabilités, et d'étendre la structure du classificateur pour lui inclure une vision plus globale de l'espace étudié...

Mais quelle que soit la méthode retenue, l'apprentissage supervisé requiert une quantité importante de travail au niveau du choix des caractéristiques utilisées pour la classification. La succession des étages de transformation ne faisant que réduire la quantité d'information véhiculée par les données, implique qu'il est crucial d'avoir une quantité suffisante d'information au départ.

Annexe A Implémentation

Le projet a été entièrement développé sous Java 1.2. Il est composé de 4 paquetages : epfl.io, epfl.pdf, epfl.classifier et epfl.experiment.

package epfl.io

Ce paquetage contient toutes les classes nécessaires à la gestion des entrées-sorties. Il existe deux ensembles de classes principaux :

1) la lecture ou l'écriture de fichiers tagués

Les classes TagReader et TagWriter permettent de lire ou d'écrire des tags dans un flot de caractères. Elles permettent de lire ou d'écrire des tags de début <tagName>, de fin </tagName> ou contenant un type primitif <tagName> int </tagName>, <tagName> double </tagName>, etc.

2) la gestion de flots de vecteurs de nombres réels

Les réseaux de neurones et les méthodes statistiques nécessitent une gestion simple et efficace d'un ensemble important de vecteurs de nombres réels. La solution retenue consiste à étendre le concept de flot de caractères en flot de vecteurs. Deux interfaces VectorReader et VectorWriter définissent les flots de vecteurs d'entrée et de sortie. L'idée d'emboîtement successif de flots, qui existe dans les flots de caractères de Java, a été reprise. On peut ainsi appliquer des transformations successives à un ensemble de vecteurs. Par exemple : ensemble initial → normalisation → PCA → ensemble final

Différents utilitaires de gestion de flots ont été écrits, notamment des flots périodiques (VectorSimplePeriodReader, VectorManualPeriodReader) qui sont très utiles dans les processus itératifs, ou encore un démultiplexeur (VectorDemultiplexer) qui redirige le flot suivant une fonction de classification.

package epfl.pdf

Ce paquetage contient un ensemble de classes relatives aux distributions de probabilités. On trouve par exemple une implémentation de la distribution normale multivariée, d'une distribution basée sur des kernels normaux ou encore un utilitaire de PCA. Mais les classes principales sont surtout constituées de trois implémentations de l'algorithme EM :

1) EM sur une distribution basée sur des kernels normaux univariés;

2) EM sur une distribution basée sur des kernels normaux multivariés;

3) EM sur une distribution basée sur des kernels normaux multivariés avec probabilités à priori.

Pour utiliser l'algorithme, il suffit de créer une distribution basée sur des kernels, d'avoir un ensemble de données sous la forme d'un flot de vecteurs périodique et d'appeler la méthode maximizeLikelihood().

package epfl.classifier

Ce paquetage contient l'implémentation du classificateur basé sur le modèle probabiliste présenté dans ce projet, ainsi que deux méthodes d'apprentissage basés sur l'algorithme EM. La première méthode (MultMixtureModelLearning) utilise l'algorithme EM standard pour les kernels multivariés et la seconde (PMultMixtureModelLearning) utilise les probabilités à priori.

package epfl.experiment, epfl.experiment.bin

Ce paquetage contient les classes spécifiques aux données étudiées et les programmes principaux. Dans le paquetage epfl.experiment.bin se trouvent deux programmes principaux P1Mult et P1PMult qui classifient les données des bactéries présentés en exemple dans le rapport. P1Mult implémente la méthode itérative pour l'utilisation des données sans label (cf. chapitre 5), tandis que P1PMult implémente la méthode basée sur les probabilités à priori. On trouve également le programme P6Mult qui classifient les séquences d'ADN présentées au chapitre 6, et un utilitaire ShuffleVectors qui mélange aléatoirement un ensemble de vecteurs.

Bibliographie

- AFFIFI A.A. and CLARK V. (1996), *Computer-Aided Multivariate Analysis*, Third edition, Chapman & Hall, Texts in Statistical Science Series, London
- GEOFFREY, J.Mc. (1992), *Discriminant Analysis and Statistical Pattern Recognition*, Wiley Series in Probability and Mathematical Statistics, Wiley-Interscience, New-York
- GEOFFREY, J.Mc. and THRIYAMBAKAM, K. (1997), *The EM Algorithm and Extensions*, Wiley Series in Probability and Mathematical Statistics, Wiley-Interscience, New-York
- GERSTNER, W. (2000), *Réseaux de Neurones Artificiels : Cours pour les Informaticiens*, Polycopié EPFL, Ecole Polytechnique Fédérale de Lausanne, Lausanne
- HAND, D.J. (1997), *Construction and Assessment of Classification Rules*, Wiley Series in Probability and Mathematical Statistics, Wiley-Interscience, Chichester
- LITTLE, R.J.A. and RUBIN D.B. (1987), *Statistical Analysis with Missing Data*, Wiley, New-York
- SCOTT, D.W. (1992), *Multivariate Density Estimation : Theory, Practice and Visualization*, Wiley Series in Probability and Mathematical Statistics, Wiley-Interscience, New-York
- THIRIA, S., LECHEVALLIER, Y. et al. (1997), *Statistique et méthodes neuronales, 2e cycle : Ecole d'Ingénieurs*, Dunod, Paris
- WAND, M.P. and JONES M.C. (1995), *Kernel Smoothing*, Chapman & Hall, Monographs on Statistics and Applied Probability, London