# Quatrième partie

# A) L'apprentissage mis en œuvre - Quelques applications

Cette sous-partie contient 3 applications, en imagerie, en classification de textes, enfin en reconnaissance en ligne (pour la parole ou l'écrit).

- La classification d'images: en restant bas-niveau, ce qui est permis par les Support Vector Machines qui peuvent traiter des données en très grande dimension.
- La classification de textes: les Support Vector Machines sont à mon sens peu efficaces sur les très gros benchmarks (quoique des améliorations algorithmiques peuvent sans doute y remédier).
- Méthode hybride réseaux neuronaux/modèles de Markov cachés pour la reconnaissance dans le temps. Descriptif d'une méthode classique mixte réseaux de neurones-modèles de Markov.

# Chapitre 10

# Classification d'images par méthodes à noyaux

**Résumé**

Dans cette étude, on considère de la classification d'image bas-niveau, avec différents algorithmes de machine learning adaptés à la haute dimensionnalité des problèmes. Le premier est la Support Vector Machine (SVM), le second est la Bayes Point Machine (BPM). Nous comparons ces algorithmes basés sur des résultats mathématiques forts ou de jolies considérations géométriques dans un espace de classifieurs à l'algorithme le plus simple que nous puissions imaginer qui travaille sur la même représentation en noyaux. Nous utilisons différentes données bas-niveau, expérimentant des preprocessings bas-niveau exploitant de l'information spatiale. Nos résultats suggèrent que la représentation en noyau est plus importante que l'algorithme utilisé (du moins dans ce cas). C'est un résultat positif car il existe des algorithmes beaucoup plus simples et rapides que la SVM. Notre préprocessing original n'apporte que quelques pourcents d'amélioration dans le taux de succès.

Ce chapitre est une version légèrement étendue de l'article [Teytaud et al, 2001].

# Table des matières



## 10.1 Introduction

Dans la catégorisation multiclasse d'images, deux familles d'approches coexistent; la première est basée sur l'extraction haut-niveau de caractéristiques et le second sur des algorithmes bas-niveau. Dans ce travail, nous étudions la deuxième approche, rendue pratique, comme montré par [Chapelle et al, 1999], par l'utilisation d'algorithmes de machine learning capables de gérer des données en haute-dimensionnalité: les algorithmes à noyau. [Jalam et al, 2000] ont montré qu'une approche naïve basée sur la même représentation en noyau pouvait être aussi efficace que la SVM dans le cas de catégorisation textuelle bas-niveau. Dans ce papier, nous proposons des expérimentations similaires dans le cas d'images. Notre but est de tester si des algorithmes bien connus et rapides pouvaient être aussi efficaces que d'autres algorithmes pourvu qu'ils utilisent la même représentation. Nous proposons aussi d'autres caractéristiques bas-niveau.

Dans une première partie, nous décrivons des algorithmes à noyau classiques. La section 10.3 décrit des algorithmes naïfs basés sur les noyaux. Nous présentons ensuite des résultats expérimentaux et discutons nos résultats.

## 10.2 Algorithmes à noyaux classiques

Dans la suite l'*ensemble d'apprentissage* est $D = \{(x_1,y_1),...,(x_m,y_m)\}$, avec les $x_i$ éléments de l'espace d'entrée $X$, et les $y_i$ éléments de l'espac fini de sortie $Y$. Les $x_i$ sont des images, codées par des features (voir section 10.4.1), les $y_i$ peuvent appartenir à $\{-1,1\}$ pour une discrimination bi-classe, ou à $\{1,2,\ldots,q\}$ pour de la catégorisation multiclasse. Le but est de trouver une application de $X$ à $Y$, minimisant la probabilité de mauvaise classification de nouveaux exemples. Nous notons $K : X \times X \to \mathbb{R}$ un noyau sur $X$.

### 10.2.1 Support Vector Machines et Support Vector Machines multiclasses

Les Support Vector Machines (SVM) sont définies dans [Vapnik, 1995] et dans beaucoup d'autres tutoriaux; elles sont les algorithmes les plus classiques en apprentissage à noyau. Elles sont basées sur la *maximisation de la marge*, et choisissent des séparations linéaires dans un espace de caractéristiques, utilisant le *"kernel trick"*.

Le "Kernel trick", basé sur la condition de Mercer, affirme que si $K$ vérifie certaines conditions (noyau symétrique positif défini), alors

$$K(x,y) =< \phi(x),\phi(y) >$$

avec $\phi$ une application à valeurs dans un espace de Hilbert et $< .|. >$ un produit scalaire. Si on considère des SVM à classification binaire ($y_i = \pm 1$), et le cas linéaires, on cherche des séparations de la forme

$$x \mapsto sign(< w|x > +b)$$

avec $(w,b)$ les paramètres de la séparation linéaire. Le but est de minimiser la marge:

$$(\hat{w},\hat{b}) = \arg\min_{w,b} w^2 + C\sum \xi_i$$

avec $\xi_i$ des variables de relâchement des contraintes $\xi_i = \max(0,1-y_i)$, et $C$ une constante. On peut montrer, en utilisant les conditions de Kuhn-Tucker, que le $\hat{w}$ optimal est une combinaison linéaire des $x_i$'s:

$$\hat{w} = \sum y_i\alpha_i x_i \quad \text{with} \quad \alpha_i \geq 0$$

Les $x_i$ tels que $\alpha_i > 0$ sont appelés *support vectors*. Alors, la minimisation est équivalente à la minimisation de

$$(\hat{\alpha}_i,\hat{b}) = \arg\min_{\alpha_i,b} \sum_{(i,j)\in I^2} < x_i,x_j > \times y_i \times y_j \times \alpha_i\alpha_j + C\sum \xi_i$$

Le *dual* (avec les multiplicateurs de Lagrange comme variables) est la maximisation de

$$(\hat{\alpha}_i,\hat{b}) = \arg\max_{\alpha_i} \sum_{i\in I} \alpha_i - \frac{1}{2}\sum_{(i,j)\in I^2} y_i y_j \alpha_i\alpha_j < x_i|x_j > \tag{10.1}$$

avec les contraintes $\sum y_i\alpha_i = 0$ et $0 \leq \alpha_i \leq C$. L'équation 10.1 dépend seulement de produits scalaires, et au lieu d'effectuer cette minimisation, on peut considérer une séparation linéaire dans un espace de Hilbert à noyau reproduisant, i.e.

$$x \mapsto sign\left(\sum_{i\in I} y_i\alpha_i K(x_i,x) + b\right)$$

$$\text{avec } \sum a_i - \tfrac{1}{2}\sum_{(i,j)\in I^2} y_i y_j \alpha_i\alpha_j K(x_i,x_j) \text{ maximum et } 0 \leq \alpha_i \leq C$$

pourvu que $K$ vérifie la condition de Mercer (par exemple, un noyau symétrique défini positif). Dans le cas séparable, ie le cas limite $C \to \infty$ (variables de relâchement nulles), la maximisation (10.1) se réduit à la même maximisation, mais sans la contrainte $\alpha_i \leq C$; $\alpha_i$ a seulement à être positif.

L'extension de la SVM au cas multiclasse était faite par une approche un-contre-tous dans le livre [Vapnik, 1995]; donc $x \mapsto f_k(x) = sign(\sum y_i\alpha_i K(x_i,x) + b)$ était calculé pour chaque classe $k$ contre toutes les autres, et $x$ était classé dans la classe $k$ telle que $f_k(x)$ soit maximum. [Guermeur et al, 2000] ont proposé une autre version de SVM multiclasse, mathématiquement justifiée mais plus coûteuse en termes de temps de calcul, parfois meilleure en termes de performances.

### 10.2.2 Bayes Point Machines

[Herbrich et al, 1999] considère que la SVM fonctionne principalement en tant qu'approximation d'un algorithme Bayésien défini par [Rujan, 1997]. Leurs expérimentations suggèrent que, dans le cas séparable, l'algorithme appelé *Bayes Point Machine*, est plus efficace que la SVM. Le principe de l'algorithme est le suivant:

1. Soit $K$ un noyau. $x \mapsto K(x,x)$ est supposé constant égal à 1 (vrai par exemple pour le noyau gaussien $K(x,y) = \exp(-\frac{\|x-y\|^2}{\sigma^2})$.
2. Considérons $H$ l'ensemble des classifieurs $H = \{x \mapsto \sum_{i=1}^{m} \alpha_i K(x_i,x) \ / \alpha_i \in \mathbb{R}\}$.
3. L'espace des versions $V$ est défini comme étant l'espace des hypothèses consistantes de norme 1: $V = \{w \in H \ / \ \| w \| = 1 \text{ et } \forall i \ y_i w(x_i) \geq 0\}$.
4. Le but est de sélectionner comme classifieur le centre de gravité de $V$, pour la métrique dans $H$: $\| w \| = \sum \alpha_i\alpha_j K(x_i,x_j) y_i y_j$. Ceci est fait en *jouant au billard* dans l'espace des versions.

Le billard se joue comme suit:

Choisir pour centre de masse le point moyen de ce billard (en prenant en compte la métrique: voir [Rujan, 1997] pour des détails). Comme le billard n'est pas nécessairement ergodique, un pas supplémentaire

1: Trouver un point initial $x_0 \in V$.
2: Choisir au hasard une direction $d_0$ dans $H$. Soit $i$ égal à 0.
3: Trouver $x_{i+1}$, l'intersection entre $\{x_i + \lambda d_i / \lambda \in [0,\infty[\}$ et la frontière de $V$.
4: Calculer $d_{i+1}$, nouvelle direction après rebond sur la frontière de $V$.
5: Remplacer $i$ par $i+1$ et retourner en 3).

est ajouté dans les réflections afin que l'espace des versions soit uniformément couvert (cette ergodicité "artificielle" n'a pas été prouvée).

En supposant que tous les vecteur ont norme 1, on peut voir que les SVM dans le cas séparable (ie minimisation de $w^2$ sous contrainte $\xi_i = 0$) choisissent le $w$ dans l'espace des versions qui est au centre de la sphère inscrite maximale. [Herbrich et al, 1999] expliquent qu'un tel point est une approximation de la moyenne de l'espace des versions, et que quand l'espace des versions a une forme irrégulière l'approximation n'est pas efficace. Dans ces cas, leur algorithme donne de meilleurs résultats que les SVM. L'extension au cas multiclasse peut être faite dans une approche un-contre-tous.

Nous proposons ici une Bayes Point Machine modifiée, dans laquelle nous ne moyennons pas $wx$ mais $sign(wx)$. Une telle technique a l'inconvénient de ne pas choisir un classifieur parmi une famille de classifieurs linéaires, mais est théoriquement plus proche de l'inférence Bayésienne et était expérimentalement meilleure sur ce benchmark particulier.

## 10.3 Algorithmes à noyaux naïfs

### 10.3.1 Description

L'algorithme à noyau naïf (NKBA pour Naive Kernel-Based Algorithm) utilisé dans nos expérimentations est résumé ci-dessous:

• Soit $(x_i)_{i \in I}$ la famille des exemples classifiés (utilisée pour l'apprentissage). Soit $(x'_i)$ la famille des points à classifier.

• Soit $O$ matrice telle que $O_{i,j} = 1$ si $x_i$ appartient à la classe $j$, $-1$ sinon.

• Soit $K_{i,j} = K(x_i,x_j)$ et soit $K'_{i,j} = K(x'_i,x_j)$

• Soit $W$ tel que $K1 \times W = O$. $K1$ est la matrice $K$, plus une colonne remplie de 1 ($K'1$ ci-dessous définie de manière similaire). $W$ est choisi par Décomposition en valeurs singulières avec norme quadratique minimale [1].

• Soit $O' = K'1 \times W$. On classe $x'_i$ dans la classe $\hat{k} = \arg\max_k \ O'(i,k)$.

Avec $K$ une fonction à base radiale, ceci est l'algorithme RBF classique. Ici nous décidons d'utiliser d'autres fonctions de même. Notez que les résultats de [Scholkopf, 1997], soulignant la supériorité des SVM sur les approches RBF classiques, ne s'appliquent pas ici car nous n'utilisons la rétropropagation pour aucune couche de poids.

On peut discuter des justifications mathématiques de notre algorithme, car nous ne pouvons le justifier par la notion de marge géométrique $w^2$ (comme terme de régularisation) et les bornes de VC-dim basées dessus. Toutefois, [Bartlett, 1998] rappelle que l'erreur $\gamma$-empirique (nombre de points dans $D$ pour lesquels $y_f(x_i) \leq \gamma$) en classification est bornée par l'erreur quadratique divisée par $(1-\gamma)^2$. Pour un $\gamma$ fixé, l'erreur $\gamma$-empirique est bornée linéairement par l'erreur quadratique moyenne. Une borne est aisément dérivée, dépendant de la taille des poids, en combinant les résultats de [Bartlett, 1998]: l'ensemble des fonctions $H = \{x \mapsto \exp(-\frac{d^2(x,y)}{\sigma^2})\}$ pour tout $d$ considéré ici sont $L$-lipschitziennes; ainsi leur $\gamma$-fat-shattering dimension est

1. L'élimination gaussienne est la méthode la plus rapide, mais la décomposition en valeurs singulières et l'algorithme de House Holder sont meilleurs en raison de leurs stabilités numériques (voir [Golub, 1996], cité dans *HDI Functional Analysis, Optimizations and Implementation Trade-offs,* `http://www.atl.external.lmco.com/projects/rassp/RASSP_legacy/casestudies/SAIP/hdi1/`).

bornée par le $\frac{\gamma}{2L}$-nombre de couverture de $X$, et ainsi est $O(\frac{1}{\gamma^{dim}})$ avec $dim$ la dimension de $X$. Si la somme des poids est bornée par $A$, alors avec $d$ la $\frac{\gamma}{32A}$-fat-shattering dimension de $H$, quand la $\gamma$-fat-shattering dimension de $F = \left\{ \sum_{i=1}^{M} w_i f_i / M \in q\mathbb{N}, w_i \in \mathbb{R}, f_i \in H, \sum_i |w_i| \leq A \right\}$ est bornée par $\frac{cA^2 d}{\gamma^2} \log^2(Ad/\gamma)$, avec $c$ une constante universelle. Ainsi un compromis entre l'eeeur empirique et la norme des poids est justifié. On peut noter que dans nos expérimentations (partie 10.4), permettant de hautes valeurs pour $M$ (nombre de fonctions) accroit le taux de succès empirique, mais ne conduit jamais à l'over-fitting. Cela suggère que la fat-shattering dimension est plus adaptée pour construire des bornes pour cet algorithme que des arguments basés sur le nombre de poids. Ceci était aussi une conclusion de [Bartlett, 1998] dans un autre cadre, avec des arguments mathématiques rigoureux.

### 10.3.2 Améliorations

Une amélioration classique possible (à la fois en temps de calcul et en espace mémoire) consiste à définir $K_{i,j} = K(x_i,x_j)$ et $K'_{i,j} = K(x'_i,x_j)$ seulement pour un sous-ensemble $S \subset I$ de valeurs possibles de $j$. Ce sous-ensemble peut être choisi par $K$-means (voir [Bishop, 1995] par exemple) ou au hasard. Nous avons opté pour un choix aléatoire pour préserver la vitesse et la simplicité de notre approche. La taille du sous-ensemble est choisi minimal parmi les tailles permettant un taux de succès empirique optimal.

Une autre amélioration inspirée de [Rujan, 1997] et [Herbrich et al, 1999] a été testée: moyenner $O'$ sur divers essais (en pondérant par un a priori, par exemple une norme du classifieur). Ceci est possible en introduisant un bruit aléatoire sur $O$, préservant le signe de ses éléments (les multiplier par une variable aléatoire iid comprise entre 0 et 1). On note que cette technique combine deux paradigmes d'apprentissage: l'inférence Bayésienne (moyennage parmi différents modèles possibles) et addition d'exemples bruités à l'ensemble d'apprentissage (solution à l'overfitting et résistance au bruit). On peut aussi moyenner sur une autre étape: le choix du sous-ensemble $S$.

L'inférence Bayésienne peut être rapidement calculée en évaluant $K1^{-1}$. Alors calculer un nouveau $O'$ est fait simplement en:

• Calculant $K1^{-1}$ (pseudo-inverse de $K1$, coût en $O(m^3)$ avec $m$ le nombre d'exemples, par Décomposition en valeurs singulières).

• Générer $O$ (dépendant du générateur aléatoire, peu coûteux en général).

• Calculer $W$ par un produit matriciel ($O(mpQ)$ avec $m$ le nombre d'exemples $p$ le nombre d'exemples sélectionnés, les éléments $x_j$ pour lesquels $K(x_i,x_j)$ est calculé, et $Q$ le nombre de classes).

• Vérifier $K1 \times W = O$ (à cause de l'heuristique consistant à choisir le plus petit cardinal de $S$ qui assure une erreur empirique nulle).

• Calculer $O'$ par un produit matriciel : $O(tpQ)$ avec $t$ le nombre d'exemples à tester.

Avec cette modification, notre inférence Bayésienne s'avère aussi rapide que la Bayes Point Machines.

## 10.4 Expérimentations: résultats en classification d'image

### 10.4.1 Matériel et méthode

On utilise le dataset appelé Corel14, décrit dans [Chapelle et al, 1999, Carson et al, 1998], composé de 1400 images classifiées en 14 classes de 100 images (des exemples de classes sont: "polar bears", "air shows"...). Une image est notée $I : \prod_{i=1}^{2}[1..n_i] \mapsto \mathcal{D}$, avec $n_i$ le nombre d'exemples le long de l'axe $i$, $\mathcal{D}$ le domaine de valeur des pixels. Ici les images sont des images couleurs en 24 bits et $\mathcal{D}$ est composé de trois canaux 8 bits R, G et B (Rouge, Vert, Bleu).

Ces images sont des images réelles. Il y a de nombreuses caractéristiques qui peuvent être extraites pour décrire des images. Nous avons décider de nous limiter à des caractéristiques bas-niveau, en haute-dimension, invariantes par translation et rotation: typiquement des caractéristiques basés sur des histogrammes. Les caractéristiques que nous avons sélectionnées sont présentées en table 10.1 (avec $\boldsymbol{x} = (x,y)$ les coordonnées d'un pixel). En table 10.1, $\#E$ est la taille de $E$. Les deux premiers histogrammes $H_{RGB}$ et $H_{HSV}$ sont de simples histogrammes de couleurs basés sur différents espaces de couleurs. $HSV$ (Hue Saturation Value) est considéré comme *orienté-utilisateur* (by contrast with RGB considered as *orienté hardware*), basé sur

les notions intuitives de teinte, ombre et ton. Un tel espace de couleur est généralement considéré comme une meilleure caractéristique de l'image pour le machine learning que $RGB$ (voir [Foley et al, 1990] pour la conversion de $RGB$ à $HSV$). $\nabla_x I = \frac{\delta I}{\delta x}$ et $\nabla_y I = \frac{\delta I}{\delta y}$ sont les deux gradients orthogonaux aux points $I(\boldsymbol{x})$, $\Delta(\boldsymbol{x}) = \frac{\delta^2 I}{\delta^2 x} + \frac{\delta^2 I}{\delta^2 y}$ le Laplacien. L'histogramme des horientations de graident $H_\nabla$ est forcé à être invariant par translation et rotation en ordonnant les angles $\theta$ selon $\theta_{max} = \arg\max_\theta H(\theta)$ comme origine. $H_\Delta$ est l'histogramme du laplacien. $H_{\bigcirc,d}(i,j)$ est le nombre d'occurences des couleurs $i$ et $j$ pour des points plus proches qu'une distance $d$. A part les deux premiers, chauqe histogramme est calculé après une régularisation par un filtre gaussien (de taille $55 \times 55$ et $\sigma = 9$) de manière à diminuer l'effet des pixels bruités. Aucun autre préprocessing n'a été appliqué. Nous pensons que $H_{\bigcirc,d}$ amène des caractéristiques intéressantes dans l'algorithme de machine learning, parce qu'il amène une notion de proximité. Toutefois, la valeur de $d$ est difficile à choisir *a priori* (nous avons utilisé $d = 25$). En outre, dans cette première étude, nous n'avons pas considéré des histogrammes en multi-résolution. Notez que, comme les images sont en RGB, il y a différentes manières de considérer les valeurs des pixels: par exemple, il serait possible de combiner les trois canaux. Toutefois, nous avons décidé de prendre en compte les trois canaux afin de bénéficier des capacités de haute-dimension de l'apprentissage à noyaux. Donc, le gradient et le Laplacien sont calculés indépendamment sur chaque canal et chaque histogramme prend sa valeur dans $\mathcal{D}$ qui a 3 dimensions. Dans un future travail, nous pourrions prendre en compte plusieurs valeurs de $d$ et des caractéristiques en multirésolution. Les images étaient décrites par l'ensemble de caractéristiques présentées et les trois méthodes présentes (SVM, BPM, NKBA) ont été testées.

L'ensemble d'apprentissage est constitué pour les deux tiers d'images et l'ensemble de validation est composé du reste. Les résultats sont moyennés sur un grand nombre de tests. Les noyaux testés sont le linéaire ($K(x,y) =< x|y >$) et le noyau de Jambu (une version symétrique de la dissimilarité du $\chi^2$) $K(x,y) = \exp\left(-\frac{\sum \frac{(x_i - y_i)^2}{x_i + y_i}}{\sigma^2}\right)$ avec $\sigma$ choisi heuristiquement (de manière à ce que les valeurs de $K(x_i,x_j)$ pour $(i,j) \in I^2$ sont à peu près homogènes).

TAB. 10.1 – *Ensemble de caractéristiques bas-niveau considérées.*

| | | |
|---|---|---|
| $H_{RGB}$ | $\mathcal{D} \mapsto \mathbb{R}$ | $H(i) = \#\{\boldsymbol{x} \ / I(\boldsymbol{x}) = i\}$ |
| $H_{HSV}$ | $\mathcal{D} \mapsto \mathbb{R}$ | $H(i) = \#\{\boldsymbol{x} \ / I(\boldsymbol{x}) = i\}$ |
| $H_\nabla$ | $[0 : 2\pi] \mapsto \mathbb{R}$ | $H(\theta) = \sum \parallel (\nabla_x I, \nabla_y I) \parallel / angle\,(\nabla_x I, \nabla_y I) = \theta$ |
| $H_\Delta$ | $\mathcal{D} \mapsto \mathbb{R}$ | $H(i) = \#\{\boldsymbol{x} \ / \Delta(\boldsymbol{x}) = i\}$ |
| $H_{\bigcirc,d}$ | $\mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ | $H(i,j) = \#\{(\boldsymbol{x},\boldsymbol{x'}) \ / I(\boldsymbol{x}) = i\,, I(\boldsymbol{x'}) = i$ et $dist(\boldsymbol{x},\boldsymbol{x'}) \leq d\}$ |

TAB. 10.2 – *Différents taux de succès selon les différentes méthodes et les différentes caractéristiques utilisées. Les deux premières lignes sont basées sur le noyau linéaire, les autres sur le noyau de Jambu.*

| Comparaison SVM/NKBA | | |
|---|---|---|
| Données | SVM | NKBA |
| RGB (linéaire) | 57.6 % | 60.0 % |
| HSV (linéaire) | 63.4 % | 63.4 % |
| RGB (Jambu) | 81.4 % | 83.6 % |
| HSV (Jambu) | 81.4 % | 83.5 % |

| Expérimentations supplémentaires avec NKBA | |
|---|---|
| Gradient (Jambu) | 58.5 % |
| Laplacian (Jambu) | 60.0 % |
| $H_\bigcirc$ (Jambu) | 73.4 % |
| Combinée | 85.9 % |

### 10.4.2 Résultats

[Chapelle et al, 1999] suggèrent des préprocessings éventuels (tels qu'élever les fréquences à la puissance $\frac{1}{8}$). Il est probable que des améliorations usuelles en catégorisation de texte pourraient être efficaces aussi (voir par exemple [Sahami, 1999], remplaçant les fréquences $\alpha$ par $\frac{1}{1+\alpha}$ entre autres). Nous n'étudions pas

de tels préprocessings ici. Sur la même base de donnée, avec différentes approches (basées sur l'extraction de caractéristiques et des arbres de décision), [Carson et al, 1998] trouvent environ 50 % de succès (cité dans [Carson et al, 1998]). Les $k$ plus proches voisins ne font pas mieux, du moins avec les similarités utilisées dans notre algorithme à noyau. Les tables 10.2 présentent les taux de succès selon différentes méthodes et caractéristiques. En utilisant les SVM et $H_{RGB}$, [Chapelle et al, 1999] présentent des résultats un peu meilleurs sur le même benchmark avec le même algorithme. Nous supposons qu'une petite différence de préprocessing peut expliquer cette différence. Différentes approches (RGB, Gradient, Laplacien, $H_{\bigcirc}$, toute avec le noyau de Jambu) sont combinées linéairement (en utilisant un algorithme de décomposition en valeur singulière pour choisir les poids de la combinaison); ceci amène aux résultats de la ligne "combinée". Notre inférence Bayésienne et notre BPM modifiée on atteint respectivement 82.0 % et 82.8 % dans le cas du noyau de Jambu sur RGB. Les algorithmes linéaires n'ont pas atteint d'aussi bons résultats que le noyau de Jambu: leur force réside plutôt dans la vitesse de reconnaissance.

## 10.5 Conclusion

Nous n'avons pas réussi à améliorer significativement les résultats en incorporant une information sur la proximité des couleurs, ou les gradients, le laplacien. Il est un peu décevant de ne pas atteindre sur un tel problème des taux comme 95 %. Un travail futur pourrait inclure une évaluation de la direction spatiale, des Dépendence Spatiales en Niveaux de Gris, des filtres de Gabor. La différence avec les résultats données dans [Chapelle et al, 1999] pourrait être attribuée à la différence de préprocessing ou pourrait aussi être due à une faiblesse de notre implémentation de SVM. SVM sont toujours un algorithme d'apprentissage récent, et différentes implémentations donnent des résultats différents, parce que des algorithmes de programmation quadratique différents ont des comportements différents (principalement, à cause de la fonction objectif quasi-horizontale). Comme déjà observé en catégorisation de texte ([Jalam et al, 2000]), on peut remplacer des algorithmes complexes comme la Bayes Point Machine ou la Support Vector Machine par des algorithmes vieux, facilement et gratuitement accessibles, et bien connus. La condition de Mercer ne semble pas indispensable au bon fonctionnement d'une SVM. A la fois d'un point de vue théorique (grâce aux nombres de couverture fournis par la fat-shattering dimension) et d'un point de vue pratique, des algorithmes simples peuvent être utilisés pour cette tâche. En particulier, le temps de calcul peut en être grandement réduit, même par rapport à une SVM basée sur SMO (Voir [Smola, 1998] par exemple). Un point intéressant pour un travail à suivre pourrait être l'utilisation de Lp-machines, un algorithme à noyau basé sur la programmation linéaire au lieu de quadratique, et qui apparait théoriquement (à cause des bornes basées sur les poids au lieu des marges, voir section 10.3.1) et pratiquement (à cause de son comportment numérique) intéressant. En catégorisation de texte, nous n'avons pas obtenu d'améliorations en utilisant ECOC (voir [Dietterich, 1995]), un paradigme classique pour étendre la classification bi-classe à la classification multi-classe. En outre, la version multiclasse de SVM proposé dans [Guermeur et al, 2000] a fourni de meilleurs résultats que les SVM un-contre-tous, alors que [Weston et al, 1998] ne mentionne aucune supériorité d'une SVM multiclasse sur l'autre. Nous supposons que le grand nombre de classes et la grande dimensionnalité des problèmes expliquent l'écart ici observé. Dans un travail à suivre, nous pourrions tester si ces résultats peuvent être vérifiés aussi dans le cas de la catégorisation d'images.

# Bibliographie

[Bartlett, 1998] Bartlett P.L. 1998, *The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, IEEE transactions on Information Theory, 44:525-536*

[Bishop, 1995] Bishop C.M. 1995, *Neural Networks for Pattern Recognition, Oxford*

[Carson et al, 1998] Carson C., S. Belongie, H. Greenspan, and J. Malik 1998, *Color and texture-based images segmentation using EM and its application to image querying and classification, submitted to Pattern Anal. Machine Intell.*

[Chapelle et al, 1999] Chapelle O., P. Haffner, and V.-N. Vapnik 1999, *Support Vector Machines for Histogram Based Image Classification, IEEE transactions on Neural Networks, Vol 10*

[Golub, 1996] Golub G., and C. Van Loan 1996, *Matrix computations, third edition, The Johns Hopkins University Press Ltd., London*

[Herbrich et al, 1999] Herbrich R., T. Graepel, and C. Campbell 1999, *Bayes Point Machines: Estimating the Bayes Point in Kernel Space, in Proceedings of IJCAI Workshop Support Vector Machines, pages 23-27, 1999*

[Dietterich, 1995] Dietterich T.-G., and G. Bakiri 1995, *Solving Multiclass Learning Problems via Error-Correcting Output Codes. Journal of Artificial Intelligence Research 2: 263-286,1995*

[Foley et al, 1990] Foley J., A. van Dam, S.Feiner and J. Hughes 1990, *Computer graphics – Principles and Practice–* Addison-Wesley, 2nd edition

[Guermeur et al, 2000] Guermeur Y., A. Elisseeff, and H. Paugam-Moisy 2000, *A new multi-class SVM based on a uniform convergence result. Proceedings of IJCNN'00, IV-183*

[Jalam et al, 2000] Jalam R., and O. Teytaud 2000, *Text Categorization based on N-grams, research report, Laboratoire ERIC*

[Rujan, 1997] Ruján P. 1997, *Playing Billiard in Version Space. Neural Computation 9, 99-122*

[Sahami, 1999] Sahami M. 1999, *Using Machine Learning to Improve Information Access, Ph.D. in Computer Science, Stanford University*

[Scholkopf, 1997] Schölkopf B., K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V.N. Vapnik 1997, *Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. IEEE Transactions on Signal Processing, 45(11):2758-2765*

[Smola, 1998] Smola, A.J. 1998, *Learning with kernels, Ph.D. in Computer Science, Technische Universität Berlin*

[Teytaud et al, 2001] O. Teytaud, D. Sarrut, *Kernel-based image classification, Proceedings of Icann 2001, 2002.*

[Vapnik, 1995] Vapnik V.N. 1995, *The Nature of Statistical Learning, Springer*

[Weston et al, 1998] Weston J., and C. Watkins 1998, *Multiclass Support Vector Machines, Univ. London, U.-K.,m Tech. Rep. CSD-TR-98-04*

# Chapitre 11

# Classification de textes par méthodes à noyaux

**Résumé**

Ce travail est une version étendue de l'article "Identification de la langue et catégorisation de textes", R. Jalam, O. Teytaud, proceedings of EGC 2001.

Nous rappelons tout d'abord diverses techniques utilisées en catégorisation de textes. Nous nous penchons ensuite sur une approche prometteuse: les $N$-grammes. Généralisable à d'autres types de données (ADN en particulier), cette méthode s'avère efficace. Nous développons un nouveau noyau pour le text mining, basé sur la dissimilarité du $\chi^2$, et l'appliquons aux Support Vector Machines, et à un algorithme simple de type Fonctions à Bases Radiales. Le cas de la discrimination de la langue est plus détaillé en raison des spécificités des approches spécialisées sur ces problèmes, mais l'approche est illustrée sur un problème de catégorisation plus difficile et est généralisable à d'autres problèmes, notamment tous les problèmes pouvant se ramener à la classification de distributions (par exemple la classification d'images, travail en cours).

**Mots clé:** Catégorisation de textes, Identification de langue, Distribution, $N$-grammes, $\chi^2$, Support Vector Machines, Fonctions à bases radiales, Text Mining

# Table des matières



## 11.1 Introduction & état de l'art

Plusieurs éléments représentatifs d'une langue peuvent être considérés pour de la catégorisation de langue: la présence de certains caractères [Mustonen, 1965] [Souter et al, 1994]; la présence de certains mots [Souter et al, 1994] [Giguet, 1998]; la fréquence de $n$-grammes [Beesley, 1988] [Cavnar et al, 1994] [Dunning, 1994] [Grefenstette, 1995]. Nous pouvons distinguer trois familles d'approches : les approches **linguistiques**, les approches **statistiques** et **probabilistes**, et les approches basées sur des méthodes générales de **classification** (réseaux de neurones, SVM...). Après avoir détaillé ces approches, nous proposerons un nouvel algorithme et verrons quels algorithmes s'avèrent efficaces dans différents cas.

### 11.1.1 Les approches basées sur des connaissances linguistiques

Ces approches d'identification supposent la construction des ressources linguistiques et nécessitent une connaissance préalable. Elles sont peu généralisables de la classification de langues à la catégorisation de textes.

Une approche consiste à trouver, empiriquement ou en utilisant des compétences linguistiques, des **chaînes de caractères uniques** pour chaque langue. Mais ces chaînes ne sont pas un indicateur fiable [Souter et al, 1994]; comme l'explique [Dunning, 1994], il est difficile d'accepter d'attribuer la langue italienne à tout texte qui parle de Zucchini ou Pinocchio ou d'associer la langue française à un texte anglais qui cite le mot français «milieux». De plus, ces chaînes de caractères uniques sont relativement rares et par conséquent il est difficile de les avoir dans des passages de textes courts.

Si on définit la langue d'un énoncé par la langue des mots qui le composent, le simple recours à des **lexiques de mots** de chaque langue est suffisant pour identifier la langue : il suffit de reconnaître la langue du segment comme étant celle pour laquelle le lexique contient tous les mots. L'incomplétude des lexiques, la technicité des termes, le risque de faute (de frappes ou de reconnaissance automatique), perturbent ces méthodes [Giguet, 1998].

Une autre approche, plus intuitive, consiste à choisir les **mots grammaticaux** comme discriminants pour l'identification de la langue [Grefenstette, 1995]. L'utilisation de ces mot grammaticaux (**prépositions**, **conjonctions**, **déterminants**, **pronoms**, **adverbes**, **auxiliaires**) se justifie par le fait qu'ils permettent un diagnostic sûr; ils représentent en moyenne 50% des mots d'une phrase dans la plupart des langues et leur présence est indispensable [Giguet, 1998]. Les mots grammaticaux ont la propriété d'être courts et en nombre limité ce qui permet d'envisager la construction de listes exhaustives. Cette approche, bien que très rapide et relativement efficace (plus que la méthode précédente), est faible sur des séquences dépourvues de tels mots, comme souvent les titres de sections [Grefenstette, 1995]. En outre, étant peu informateurs, ces mots sont souvent extraits par des prétraitements lors d'indexation pour recherches à grande échelle («stop words», [Sahami, 1999]). La méthode souffre aussi d'une difficile adaptabilité au traitement de séquences d'autres types (biologie), et de langues difficiles à segmenter en mots. On peut combiner ces techniques et leur adjoindre la vérification de l'**alphabet** et des **affixes** (**préfixes** et **suffixes**).

### 11.1.2 Les approches statistiques et probabilistes

Ces approches utilisent des connaissances construites automatiquement à partir d'un corpus textuel représentatif de la langue. L'objectif est de capturer au moyen de modèles statistiques ou probabilistes

certaines régularités des langues et de leur associer une fréquence ou probabilité d'apparition. Elles se généralisent de la reconnaissance de langue à la classification de textes et à part dans le cas de mots, à la catégorisation de séquences sur un alphabet discret.

ts: la nécéssité de segmentation en mots, le manque de robustesse devant des mots particuliers.

Une solution pour transformer des textes en vecteurs de fréquences, plus faciles à manier pour les algorithmes d'apprentissage usuels, consiste à découper les textes en mots, quitte à ne garder que les mots les plus fréquents ou les plus discriminants. La plupart des méthodes évoquées ci-dessous pourraient se faire sur ce principe, mais nous choisissons d'utiliser une autre façon de segmenter les textes: les $n$-grammes, parfaitement généralisables à toutes formes de séquences sur un alphabet discret; les mots étant, eux, difficiles à définir pour le Chinois ou pour des séquences ADN.

Un ***n*-gramme** est une séquence de $n$ caractères. Bien que, dans la littérature, ce terme désigne parfois des séquences qui ne sont pas ordonnées, ici un $n$-gramme désignera une chaine de $n$ caractères consécutifs [Cavnar et al, 1994].

Pour un document quelconque, l'ensemble des $n$-grammes qu'on peut générer est la famille de photos qu'on obtient en déplaçant une fenêtre de $n$ cases sur le corps de texte. Ce déplacement se fait par étapes, une étape (qui correspond à un caractère) pour chaque déplacement. À chaque étape une prise de photo se fait, l'ensemble des « photos » qu'on obtient constitue l'ensemble de tous les $n$-grammes qu'on puisse générer. Par exemple, les 5-grammes de la phrase «Je suis un génie», sont: je␣su, e␣sui, ␣suis, suis␣, uis␣u, ..., etc [Miller et al, 1999]. Un **profil *n*-grammes** d'un document consiste en la liste des contiguïtés de $n$ caractères les plus fréquentes, par ordre décroissant de leur fréquence d'apparition dans le document et munis de leurs fréquences. Les profils s'obtiennent en temps quasi-linéaire grâce à des tables de hachage (sous l'hypothèse que les $N$-grams sont convenablement répartis par la table de hachage).

La notion de $n$-grammes est introduite par Shannon [Shannon, 1948] où il s'intéressait à la prédiction d'apparition de certains caractères en fonction des autres caractères. Depuis, les $n$-grammes sont utilisés dans plusieurs domaines comme l'identification de la parole, la recherche documentaire, etc.

Cette technique est tolérante aux déformations causées lors de l'utilisation des OCR et tolérante aux fautes d'orthographes. [Miller et al, 1999] montre qu'il y a des systèmes de recherches documentaires basés sur les $n$-grammes qui ont gardé leurs performances avec des taux de déformations de 30%, un taux avec lequel aucun système basé sur les mots ne peut fonctionner correctement.

nts, stemming: suppression des affixes non discréminants). A titre d'exemple, lorsqu'un mot se rencontre plusieurs fois, sa fréquence sera augmentée ainsi que les fréquence des $n$-grammes correspondants.

Les systèmes d'identification de langues basés sur les $n$-grammes suivent en général le même schéma. Dans une première phase, la phase d'acquisition automatique des connaissances linguistiques, on choisit un corpus représentatif pour chaque langue puis on génère un profil caractéristique qui fera office de référence: calculer les nombres d'occurrences des différents $n$-grammes (souvent $n$ est compris entre 1 et 5) et les transformer en fréquences. Dans la deuxième phase, nommée la phase de diagnostic, pour chaque texte à identifier, on construit son profil $n$-grammes et on détermine sa classe.

### Les méthodes de plus proche(s) voisin(s)

Plusieurs algorithmes, utilisés pour la catégorisation de texte, sont basés sur la distance ou plus généralement sur la similarité ou dissimilarité. L'idée générale est de chercher le texte, parmi l'ensemble d'apprentissage, qui soit le plus proche du texte $T$ à classer pour ensuite lui attribuer la classe de ce texte. La généralisation à $k$ plus proches voisins est immédiate; il suffit de choisir la classe la plus représentée parmi les classes de $k$ textes les plus proches. La difficulté pour ces approches est de définir une distance. En pratique, on utilise des pseudo-distances, ne vérifiant pas tous les axiomes des distances au sens mathématique.

• **La distance de Beesley** La méthode de [Beesley, 1988] est basée sur des modèles mathématiques linguistiques qui étaient proposées essentiellement pour décoder les textes cryptés. L'identification comporte deux phases: dans la phase d'apprentissage, établir, pour chaque langue, un profil bi-grammes (sans pourtant utiliser une lettre plus d'une fois - il ne s'agit pas du profil au sens usuel: "ordinateur" donne "or", "di", "na", etc, mais pas "rd","in") qui fera office de référence et évaluer les probabilités d'apparitions de chaque bi-gramme. Dans la phase de diagnostic: rechercher, pour le texte $T$ à identifier, le profil de référence le plus proche en utilisant pour mesure de similarité avec la langue $L$ le produit des fréquences des bi-grammes de $T$ dans la langue $L$ (méthode Bayésienne Naïve).

Cette méthode suppose la segmentation du texte en mots ce qui empêche son utilisation pour des langues dont les frontières entre les mots ne sont pas claires. Elle se limite également à prendre en

compte uniquement les bi-grammes alors qu'il est important de travailler aussi avec des tri-grammes ou quadri-grammes pour mieux concerver la spécificité de chaque langue: par exemple, la séquence «tion» est typique du français et de l'anglais, si on la découpe en «ti» et «on» alors le système aura du mal à pénaliser les autres langues comme l'espagnol et le portugais qui utilisent «ti» et «on» mais pas «tion». Cela explique que la notion de profil gardée par la suite soit remaniée.

• **La distance de Cavnar et Trenkle (CT)** Cette distance date de [Cavnar et al, 1994]. L'identification comporte deux phases: pendant la phase d'acquisition, il faut établir pour chaque langue un profil tri-grammes caractéristique acquis automatiquement qui fera office de référence. Ensuite il faudra rechercher, pour chaque énoncé particulier, le profil de référence le plus proche en utilisant une mesure de similarité. Cette mesure entre deux profils $P_1$ et $P_2$ se calcule comme suit:

$$CT(P_1,P_2) = \sum_{w \in P_1,\, R_{P_1}(w) < NMAX} min(|R_{P_2}(w) - R_{p_1}(w)|, DMAX)$$

avec $|x|$ la valeur absolue de $x$ et $R_P(w)$, avec $w$ un $n$-gramme et $P$ un $N$-profil, désigne le rang de $w$ dans le profil $P$, si $w$ fait parti de $P$, et $DMAX$ dans le cas contraire (e.g., $NMAX = 500$ et $DMAX = 1000$). On conçoit facilement que cette notion un peu tranchée est moins sensible aux nuances que certaine des distances qui suivent.

• **La distance de Kullbach-Leibler(KL)** [Sibun et al, 1996] proposent une méthode qui utilise l'entropie relative de Kullbach et Leibler comme mesure de distance. Une entropie relative entre deux distributions de probabilité reflète la quantité d'informations supplémentaires nécessaire pour coder la deuxième distribution en utilisant un code optimal généré pour la première. Formellement,

$$KL(T_1,T_2) = \sum_{N_g} f_2(N_g).\log(\frac{f_2(N_g)}{f_1(N_g)})$$

ici, la somme est prise sur tous les $n$-grammes présents dans $T_2$, avec $T_1$ et $T_2$ des textes, $f_i(N_g)$ est la fréquence du $n$-gramme $N_g$ dans le texte $T_i$. Si le $n$-gramme $N_g$ est absent dans $T_i$, une demi-fréquence est alors ajoutée pour éviter que le score tombe vers moins l'infini.

• **La distance du $\chi^2$** Cette distance peut être présentée comme suit :

$$\chi^2(T_1,T_2) = \sum_{N_g} \frac{(f_1(N_g) - f_2(N_g))^2}{f_2(N_g)}$$

Dans nos expérimentations, on symétrise cette distance en utilisant $\chi^2(T_1,T_2) = 2\frac{(f_1(N_g)-f_2(N_g))^2}{f_1(N_g)+f_2(N_g)}$. Si $f_1(N_g) + f_2(N_g)$ est égal à 0, alors on remplace $\frac{(f_1(N_g)-f_2(N_g))^2}{f_1(N_g)+f_2(N_g)}$ par 0 (son prolongement par continuité).

• **La méthode de [Cowie et al, 1998]** Cette méthode est intéressante pour sa capacité à mêler différents profils $N$-grammes (pour différentes valeurs de $N$) d'un même texte.

**La phase d'apprentissage:** [Cowie et al, 1998] remplit un ensemble, nommé CNP, avec des $n$-grammes communs ($n$ = 1,2,...,$k$,...,$N$) choisis. Plus précisément, pour choisir la liste des 1-grammes qui feront partie du CNP, un poids $W(a_1)$, pour chaque 1-gramme $a_1$ présent dans l'ensemble d'apprentissage, est calculé de la manière suivante:$W(a_1) = -p(a_1).\log p(a_1)$. Ensuite, les 1-grammes seront classés par ordre décroissant de leurs poids. Seulement les premiers $m_1$ seront classés dans le CNP.

[Cowie et al, 1998] refait la même procédure pour les 2-grammes, $k$-grammes, ..., $n$-grammes. Afin de calculer le poids de chaque $k$-gramme, ils distinguent deux cas:

1. Si le $[k-1]$-gramme $a_2a_3...a_k$ n'est pas présent dans le CNP alors

$$W(a_1a_2a_3...a_k) = -p(a_1a_2a_3...a_k).\log p(a_k|a_1a_2a_3...a_{k-1})$$

2. Sinon $W(a_1a_2a_3...a_k)$
$= -p(a_1a_2a_3...a_k).(\log p(a_k|a_1a_2a_3...a_{k-1}) - \log p(a_k|a_2a_3...a_{k-1}))$

puis, ils classent les $k$-grammes par ordre décroissant de poids et placent les premiers $m_k$ dans le CNP. Une fois l'ensemble CNP rempli, [Cowie et al, 1998] construit un vecteur primaire de poids $PRW$ pour chaque $k$-gramme, le $i$-ième composant $PRW_i$ est le poids de ce $k$-gramme dans la langue $i$. Ce poids $PRW_i$ , pour un 1-gramme $a_1$ et la langue $i$, est calculé de la manière suivante: $PRW_i = -\log p_i(a_1)$. Pour les autres $k$-grammes $a_1a_2...a_k$, ce poids est: $PRW_i = -\log p_i(a_k|a_1a_2...a_{k-1})$. Si un $k$-gramme est absent dans une langue, alors [Cowie et al, 1998] définit une valeur maximale $MAX$, et on pose $PRW_i = MAX$. A la fin du processus, ils calculent un vecteur de poids de reconnaissance $RW_n$ pour chaque $n$-gramme comme suit:

$$RW(a_1a_2a_3...a_n) = \begin{cases} PRW(a_1a_2...a_n) & a_1a_2...a_n \in CNP \\ PRW(a_2a_3...a_n) & a_2a_3...a_n \in CNP \\ PRW(a_3a_4...a_n) & a_3a_4...a_n \in CNP \\ PRW(a_n) & a_n \in CNP \\ vecteur\ de\ MAX & autrement \end{cases}$$

Egalement, pour chaque langue $i$, [Cowie et al, 1998] calcule le poids moyen et sa variance. Ils segmentent l'ensemble d'apprentissage de chaque langue $i$ en $K$ segments $x_1x_2...x_{500}$ de taille 500 octets, puis calculent la moyenne $WA_i$ et la variance $D_i$:

$$\text{pour } k \in [1,K] d_{i,k} = \sum_{s=N}^{500} \frac{RW_i(x_{s-N+1}x_{s-N+2}...x_s)}{500}$$

$$WA_i = \frac{1}{K}\sum_{k=1}^{K} d_{i,k} \qquad D_i = \frac{1}{K}\sum_{k=1}^{K}(d_{i,k} - WA_i)^2$$

**La phase de diagnostic:** Cette phase comporte deux étapes: la classification et la validation. Pour chaque segment de texte $X = x_1x_2x_3...x_s$ dont la classe est inconnue, on calcule son vecteur de poids $RRW(X)$:

$$RRW_i(X) = \frac{1}{S}\sum_{s=N}^{S} RW(x_{s-N+1}x_{s-N+2}...x_s)$$

On détermine alors $i^* = \arg\min RRW_i(X)$. Le texte $X$ est de langue $i^*$ si on a $RRW_{i^*} - WA_{i^*} \leq 1.5D_{i^*}$ sinon la langue du texte sera considérée comme inconnue. La méthode se généralise bien sûr à la catégorisation de séquences.

### Modèles de Markov

Un modèle de Markov est principalement un ensemble fini d'états entre lesquels sont définies des probabilités de transition. Une distribution de probabilité sur les états est donnée aussi, qui détermine l'état initial. Formellement, soit $E = [1,n]$, et soit $M$ une matrice $n \times n$ de nombres réels positifs tels que la somme sur chaque ligne soit égale à 1. Soit $p$ un vecteur de $n$ réels positifs et de somme 1. Soit $X_0$ une variable aléatoire égale à $i$ avec prob bilité $p_i$. Définissons $X_{m+1}$ comme une variable aléatoire, égale à $i$ avec probabilité $M_{X_m,i}$. La suite des $X_i$ est un processus de Markov.

e rang de $X_m$, $X_{m+1}$, $X_{m+2}$, ... , $X_{m+k-1}$ parmi les $k$-grammes en ordre lexicographique. Dans la suite, on identifie $[1,n]$ et un alphabet fini, afin de simplifier les notations.

Etant donné un modèle de Markov $A$, on peut évaluer facilement la probabilité de générer un texte $S$ avec ce modèle; avec $[z]$ le rang d'un $k$-gramme $z$ parmi les $k$-grammes en ordre lexicographique,

$$P(S|A) = p([s_0,...,s_{k-1}])\Pi_{w \in S, |w|=k+1} M_{[w_1,...,w_{k-1}],w_k}$$

Pour simplifier cette équation, comme fait dans [Dunning, 1994] on ajoute toujours en tête de texte un $k$-gramme initial, de façon à ce que $p([s_0,...,s_{k-1}]) = 1$; ainsi ce terme peut être supprimé. Nous ferons l'hypothèse que la probabilité *a priori* de chaque classe est égale aux autres, égale à $\mathcal{P}$ (quoique l'on puisse facilement tenir compte d'un a priori). Une chaîne $S$ étant donné, $P(S)$ est égal à 1. Donc

$$P(A|S)P(S) = P(A,S) = P(S|A)P(A) \text{ se simplifie en } P(A|S) = P(S|A)\mathcal{P}$$

Pour trouver $A$ qui maximise $P(A|S)$ parmi différents $A$ possibles, on a juste à calculer tous les $P(S|A)$ et à trouver $A$ qui le maximise. Ainsi, les problèmes majeurs sont les évaluations des paramètres des modèles de Markov d'une classe donnée. On pourrait simplement évaluer $M_{[w_1,...,w_{k-1}],w_k} = \frac{T(w_1,...,w_k)}{T(w_1,...,w_{k-1})}$, avec $T(w)$ le nombre d'occurences de $w$ dans le(s) texte(s) de la langue à modéliser. Le problème est le

risque que certains mot n'apparaissent jamais (par finitude de l'échantillon d'apprentissage), ceci entrainant de mauvais résultats sur des textes comportant des séquences pathologiques, quand bien même ces textes seraient complètement cohérents avec le modèle considéré par ailleurs. [Dunning, 1994] détaille la correction de Laplace des échantillons finis, qui consiste à choisir:

$$M_{[w_1,\ldots,w_{k-1}],w_k} = \frac{T(w_1,\ldots,w_k)+1}{T(w_1,\ldots,w_{k-1})+n} \qquad (n \text{ taille de l'alphabet})$$

D'autres approches plus compliquées sont possibles, mais selon [Dunning, 1994] elles n'améliorent pas significativement les performances.

### 11.1.3 Les méthodes basées sur des outils classiques de classification dans $\mathbb{R}^n$

L'utilisation de la notion de mots ou de $N$-grammes permet aisément de remplacer un texte par une distribution (à chaque $N$-gramme on associe sa fréquence). On remplace ainsi un problème de catégorisation de chaînes par un problème de catégorisation dans $[0,1]^N$, pour lequel de nombreuses approches existent. La difficulté est la très grande dimension. L'utilisation d'**arbres de décision** nécéssite notamment une forte sélection de variables, $C4.5$ par exemple fonctionnant seulement pour des petites dimensions; or certains auteurs considèrent (voir [Joachims, 1998]) que toutes les variables doivent être gardées. La méthode par arbres de décision bénéficie toutefois de l'avantage de la clarté de l'algorithme de classification obtenu. Les réseaux de neurones à rétropropagation, munis d'entrées par listes et non par vecteurs, peuvent par contre traiter toutes les entrées Les **Support Vector Machines** et l'approche **Radial-Basis-Function** (RBF) ayant, pour les noyaux usuels, une complexité linéaire en la dimension et des besoins en mémoire dépendant du nombre d'exemples plutôt que de la dimension, s'avèrent efficaces pour ce problème [Joachims, 1998].

## 11.2 Notre approche

L'encodage dans $\mathbb{R}^n$ (par $N$-grammes ou mots) permet l'utilisation de nombreux algorithmes, en particulier des Support Vector Machines (SVMs, voir une description dans [Vanik, 1995] par exemple). Mais on peut utiliser les SVMs d'une autre façon: on définit $K(T_1,T_2) = exp(-d(T_1,T_2))$ avec $d$ une mesure de dissimilarité. Nous avons choisi la distance du $\chi^2$, sans parvenir à prouver que ce noyau vérifie la condition de Mercer (i.e. nous ne savons pas si $int_u \int_v K(u,v)g(u)g(v)dudv$ st $> 0$ pour tout $g$ $L^2$ non nulle).

De même, on peut aussi utiliser un réseau à base de fonctions radiales (RBF) avec un encodage des textes dans $\mathbb{R}^n$ et la distance euclidienne; mais nous avons estimé plus naturel d'utiliser une distance adaptée à des distributions, par exemple la distance du $\chi^2$. Cette méthode (du moins l'une de ses variantes) est résumée ci-dessous (on trouvera plus d'informations dans [Bishop, 1995]):

- Soit les $(T_i)$ la famille de textes classifiés utilisés pour l'apprentissage et les $T'_i$ les textes à classifier.
- Soit $O$ la matrice telle que $O_{i,j} = 1$ si $T_i$ appartient à la classe $j$, $-1$ sinon.
- Soit $K_{i,j} = exp(-d(T_i,T_J))$
- Soit $K'_{i,j} = exp(-d(T'_i,T_j))$
- Soit $W$ telle que $K1 \times W = O$. $K1$ est la matrice $K$, plus une colonne de 1 à sa droite.
- Soit $O' = K'1 \times W$.
- On assigne à $T'_i$ la classe $argmax_k\ O'(i,k)$.

### 11.2.1 Expérimentations: reconnaissance d'écrivain

Nous utilisons 130 livres en français, écrits par 28 auteurs connus: Balzac, Bloy, Corneille, Diderot, Engels, Flaubert, Fourier, France, Gaberel, Gautier, Gobineau, Hugo, Huysmans, Lamartine, Leibnitz, Maistre, Maupassant, Moliere, Pascal, Racine, Renard, Rostand, Rousseau, Sand, Stendhal, Verne, Voltaire, Zola. Certains sont traduits d'autres langues; ceci et le fait que les textes sont diversement formatés (sans corrélation avec l'auteur) sont considérés comme des difficultés supplémentair s pour l'algorithme; le corpus d'apprentissage représente environ 30 Mo, les longueurs varient de nouvelles de quelques pages à des livres entiers. La plupart des textes proviennent du site http://cedric.cnam.fr/ABU/. Les autres proviennent de la Bibliotheque Nationale de France, http://www.bnf.fr/. Les résultats expérimentaux obtenus avec des 3-grammes sont donnés en table 11.1. Le taux d'erreur est évalué par Leave-One-Out.

| Algorithme | Taux de succès |
|---|---|
| RBF avec noyau $(\chi^2)^p$ pour $p = \frac{1}{2}, \frac{1}{4}, \ldots \frac{1}{32}$ | 87.69 % |
| RBF avec noyau $\chi^2$ pour $p = 1$ | 86.15 % |
| Multiclasse svm avec noyau $\chi^2$ | 86.15 % |
| Multiclasse svm linéaire | 78.462 % |
| SVM avec noyau $\chi^2$ , $C = 1$ | 72.3077 % |
| 1-PPV avec dissimilarité $\chi^2$ | 70.77 % |
| SVM linéaire | 67.69 % |
| 1-PPV av c dissimilarité de Cavnar and Trenkle | 65.38 % |
| 1-PPV avec dissimilarité cosinus | 61.54 % |
| 1- avec dissimilarité Kullbach-Leibler | 52.31 % |

TAB. 11.1 – *Résultats obtenus par Leave-One-Out sur le problème des écrivains.*

Tous ces tests sont réalisés en ! Octave (voir http://www.che.wisc.edu/octave/ pour une description de ce clone gratuit de Matlab). Tous les codes sources peuvent être demandés par mail aux auteurs.

On appelle "SVM multiclasse" une SVM spécialisée dans le multiclasse, comme définie dans [Guermeur et al, 2000]. On peut signaler que dans ce cas précis (très haute dimension, 28 classes) cette SVM est significativement meilleure que la méthode usuelle combinant des SVMs un-contre-tous comme suggéré dans [Vanik, 1995]. On a à la fois SVM multiclasse avec $\chi^2$ significativement meilleur que SVM avec $\chi^2$ et SVM multiclasse linéaire significativement meilleure que la SVM linéaire. Nos e sais avec modèles de Markov, par la méthode donnée par [Dunning, 1994], ont donné de très mauvais résultats, laissant supposer que cette méthode est trop sensible à la présence de séquences pathologiques pour traiter des chaînes longues.

[Yang et al, 1999] conclut en gros que les SVM sont meilleures que les $k$-plus proches voisins eux-mêmes meilleurs que LLSF (voir [Yang et al, 1999]), que les réseaux de neurones multicouches basés sur la rétropropagation, eux mêmes supérieur à l'algorithme Bayésien Naïf (voir [Good, 1965]). Nos essais confirment ces résultats et nous pouvons préciser, avec >> signalant une différence significative avec risque d'au plus 5 %, > une différence significative avec risque d'au plus 10 %, $\geq$ u e différence avec risque d'au plus 15 %:

$$\{ \text{RBF - SVM Mc } (\chi^2) \} >> \text{SVM Mc} \geq \text{SVM } \chi^2 \text{ - SVM - 1-PPV}$$

[Joachims, 1998] explique (partiellement) le bon comportement des SVMs sur la catégorisation de texte par sa capacité à traiter de nombreuses dimensions sans avoir à sélectionner des variables. On peut noter que les RBFs bénéficient de la même caractéristique.

### 11.2.2 Expérimentations: reconnaissance de langue

Après ce premier benchmark, on pourrait conclure (trop vite) que les RBFs avec noyau $\chi^2$ semblent idéales pour la catégorisation de textes. En fait la SVM multiclasse s'est avérée aussi puissante, mais les RBF sont beaucoup plus simples à implémenter et beaucoup plus rapides. Dans nos expérimentations suivantes, nous nous concentrons sur deux algorithmes: les RBFs, pour leurs bonnes performances signalées ci-dessus, et les 1-plus proches voisins, pour leur simplicité d'utilisatio et leur fréquente utilisation dans ce domaine. La suite des essais a été réalisée avec des implémentations Java, basées sur la package Jama (voir http://math.nist.gov/javanumerics). Les codes sources peuvent être demandés par mail aux auteurs. La tâche consiste a reconnaître la langue d'un texte. Nous travaillons sur 5 langages: le français, l'arabe, l'anglais, l'espagnol et l'allemand. La tâche étant facile, nous la compliquons en utilisant des chaînes t! rès courtes pour tester les algorithmes. Des tests préliminaires ont été faits pour comparer les différentes distances utilisées pour des algorithmes de plus proches voisins; nous avons obtenu les meilleurs résultats avec Kullbach-Leibler, suivis de Cavnar & Trenkle, et enfin le $\chi^2$. La différence était forte dans le cas d'échantillons courts à reconnaître, aussi pour l'utilisation par SVM et RBF nous avons conservé le $\chi^2$ néanmoins, en raison de sa formulation intuitivement plus susceptib e de vérifier la condition de Mercer (toujours non démontrée). Nous n'avons pas effectué d'expérimentations avec la méthode de [Cowie et al, 1998].

Nous travaillons maintenant sur 250 échantillons de 100 bytes comme ensemble d'apprentissage, pour étudier l'influence de la segmentation pour les RBFs ou les plus proches voisins; les résultats sont donnés

en table 11.2.

"$m$-regroupés profils" signifie que les échantillons d'apprentissage ont été regroupés $m$ par $m$; "regroupés", que tous les textes d'une même classe sont regroupés. Le paramètre $m$ gère ainsi un compromis entre la qualité des données (plus $m$ est grand, plus les fichiers utilisés en apprentissage sont grands, et donc plus leurs fréquences sont bien définies) et leur quantité (plus $m$ est petit, et plus on a de points disponibles pour la phase d'apprentissage).

On note que la dissimilarité de Kullbach-Leibler semble meilleure que celle du $\chi^2$, notamment pour des petits échantillons en test, mais supporte mal les petits échantillons en apprentissage (comme on s'y attend en examinant le comportement de cette dissimilarité, mal régularisée, pour de petites fréquences). Ceci pourrait toutefois être amélioré en utilisant la divergence de Jensen-Shannon.

| Algorithme | Hyperparamètres | TS (100) | TS (50) | TS (20) |
|---|---|---|---|---|
| RBF | $\sigma^2 = 10$ | 99.2 % | 84.8 % | 31.52 % |
| | $\sigma^2 = 100$ | 98 % | 93.2 % | 71 % |
| RBF (2-regroupés prof.) | $\sigma^2 = 100$ | 97.2 % | 88 % | 68.56 % |
| RBF (5-regroupés prof.) | $\sigma^2 = 1000$ | 98.8 % | 94 % | 80.88 % |
| RBF (10-regroupés prof.) | $\sigma^2 = 1000$ | 99.2 % | 95.2 % | 76.72 % |
| RBF (25 regroupés prof.) | $\sigma^2 = 1000$ | 98.8 % | 94.8 % | 82.4 % |
| RBF (regroupés prof.) | $\sigma^2 = 100$ | 88.4 % | 80.6 % | |
| | $\sigma^2 = 100000$ | 87.6% | 77.4 % | 61.36 % |
| 1-PPV | $\chi^2$ | 99.2 % | 96.6 % | 88.4 % |
| 1-PPV | $KL$ | | | 47.2 % |
| 1-PPV (2-regroupés prof.) | $\chi^2$ | 99.6 % | 96.8 % | 88.8 % |
| 1-PPV (5-regroupés prof.) | $\chi^2$ | 100 % | 97.6 % | 90 % |
| 1-PPV (10-regroupés prof! ..) | $\chi^2$ | 99.2 % | 97.2 % | 88.56 % |
| 1-PPV (10-regroupés prof.) | $KL$ | | | 89.84 % |
| 1-PPV (25-regroupés prof.) | $\chi^2$ | 100 % | 96.8 % | 87.2 % |
| 1-PPV (regroupés prof.) | $\chi^2$ | 100 % | 93 % | 84.56 % |
| 1-PPV (regroupés prof.) | $KL$ | 99.7 % | 97.4 % | 89.4 % |

TAB. 11.2 – *Résultats obtenus en reconnaissance des langues.*

L'hyperparamètre $\sigma^2$ pour l'apprentissage RBF était facilement choisi dans le benchmark précédent (classification des auteurs), car le taux de succès était stable sur une grande plage de valeurs de $\sigma$; mais dans le cas présent (cas de très courtes chaines), l'efficacité variait beaucoup selon $\sigma$, ainsi qu'en fonction du paramètre de regroupement; d'où deux hyperparamètres difficiles.

## 11.3 Conclusion

Sur des données pour lesquels les fréquences sont bien évaluées, on peut finalement résumer nos résultats et ceux de [Yang et al, 1999] et [Joachims, 1998] par:

RBF > SVM Mc $\chi^2$ > SVM Mc > SVM $\chi^2$ > SVM > 1-PPV > LLSF, C4.5, NNets > NB

Avec SVM Mc la SVM multiclasse précédemment définie, SVM la SVM linéaire, LLSF comme décrit en [Yang et al, 1999], NNets des réseaux de neurones autres que SVM et NB l'algorithme bayésien naïf décrit en [Good, 1965]. Notons que RBF > SVM Mc n'est pas significatif en terme de performance; nous le conservons simplement en raison de l'avantage de vitesse et de simplicité.

Dans le cas de fréquences moins bien définies, les plus proches voisins semblent meilleurs que les RBF, en particulier avec de très petits échantillons d'apprentissage. Les résultats de [Dunning, 1994] avec des modèles de Markov, avec deux langages au lieu de 5 ici, suggèrent que les modèles de Markov entraînés avec $50Ko$ de données ont environ les mêmes performances que les 1-plus proches voisins avec $25Ko$. Le taux d'erreur en cas de réponse aléatoire étant de 20% da s le cas de 5 langages et de 50% dans le cas de 2 langages, et en rappelant que les deux langues utilisées par Dunning sont testées sur le même échantillon, nous supposons les $k$-plus proches voisins plus adaptés pour ce problème.

# Bibliographie

[Beesley, 1988] , K. Beesley, Language Identifier: A Computer Program for Automatic Natural Language Identification on On-Line Text, Proceedings of the 29th Annual Conference of the American Translators Association, 47–54, 1988

[Cavnar et al, 1994] , W. Cavnar and J. Trenkl, *N*-Gram Based Text Categorization, Symposium on Document Analysis and Information Retrieval, 1994, Las Vegas

[Cowie et al, 1998] , J. Cowie and E. Ludovik and R. Zacharski, An Autonomous, Web-based, Multilingual Corpus Collection Tool, Proceeding of Natural Language Processing and Industrial Applications, 142–148, 1998, Moncton, Canada,

[Dunning, 1994] , T. Dunning, Statistical Identification of Languages, Computing Research Laboratory, 1994, MCCS 94-273, New Mexico State University, Las Cruces, New Mexico

[Good, 1965] , I. Good, The Estimation of Probabilities: An Essay on Modern Bayesian Methods, MIT Press , 1965

[Guermeur et al, 2000] , Y. Guermeur and A. Elisseeff and H. Paugam-Moisy, A new multiclass SVM based on a uniform convergence result, accepted at IJCNN'2000,2000

[Giguet, 1998] , E. Giguet, Méthode pour l'analyse automatique de structures formelles sur documents multilingues, Université de Caen , 1998, France

[Grefenstette, 1995] , G. Grefenstette, Comparing Two Language Identification Schemes, Proceedings of the 3rd International Conference on the Statistical Analysis of Textual Data (JADT'95), 1995, Rome, Italy

[Joachims, 1998] , T. Joachims, Text Categorization with Support Vector Machines: Learning with Many Relevant Features, Machine Learning: ECML-98, Tenth European Conference on Machine Learning, 137–142, 1998

[Miller et al, 1999] , E. Miller and D. Shen and J. Liu and C.Nicholas, Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System, Journal of Digital Information„ 1999, 1, 5

[Mustonen, 1965] , S. Mustonen, Multiple Disriminant Analysis in Linguistic Problems, Statistical Methods in Linguistics , 1965, Page visitée le 15 juin 2000 à l'adresse http://www.nodali.sics.se/bibliotek/kval/smil

[Newman, 1987] , P. Newman, Foreign Language Identification: First Step in the Translation Process, Proceedings of the 28th Annual Conference of the American Translators Accociation, 509–516 , 1987

[Sahami, 1999] , M. Sahami, Using Machine Learning to Improve Information Access, Computer Science Department, Stanford University , 1999

[Shannon, 1948] , C. Shannon, The Mathematical Theory of Communication, Bell System Technical Journal, 1948, 27, 379–423 and 623–656, Page visitée le 25 juin 2000 à l'adresse http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

[Sibun et al, 1996] , P. Sibun and J. Reynar, Language Identification: Examining the Issues, Symposium on Document Analysis and Information Retrieval, 125–135, 1996, Las Vegas

[Souter et al, 1994] , C. Souter and G. Churcher and J. Hayes and J. Hughes and S. Johnson, Natural Language Identification Using Corpus-Based Models, Hermes Journal of Linguistics, 1994, Faculty of Modern Languages, Aarhus School of Business, Denmark, 13, 183–203

[Bishop, 1995] , C. Bishop, Neural Networks for pattern recognition, Oxford,1995

[Vanik, 1995] , V. Vapnik, The Nature of Statistical Learning, Springer , 1995

[Yang et al, 1999] , Y. Yang and X. Liu„A Re-Examination of Text Categorizarion Methods, SIGIR'99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999, Berkeley, CA, USA
67–88,

# Chapitre 12

# Méthode hybride réseaux neuronaux/modèles de Markov cachés pour la reconnaissance dans le temps

**Résumé**

Nous rappelons dans cette partie différentes méthodes permettant de classer des données comportant une composante dynamique de temps. L'accent sera mis sur une méthode mixte réseaux de neurones/modèles de Markov.

# Table des matières



## 12.1 Introduction

Je m'intéresse dans ce papier, basé sur un mémoire rédigé par moi pendant mon DEA "DIL" (DEA d'informatique de Lyon), aux méthodes de reconnaissance de motifs dans lesquels entre une composante de temps, et en particulier le cas des données pour lesquelles il y a distorsion; sont particulièrement concernées la reconnaissance de l'écriture en ligne et la reconnaissance de la parole. Les modèles de Markov sont une méthode classique, mais souffrant de quelques faiblesses au niveau de la discrimination; les méthodes par réseau neuronaux ont un pouvoir de discrimination beaucoup plus forts, mais sont moins adaptés à des données utilisant une dimension de temps. Je rappelle ici les méthodes les plus couramment utilisées, et notamment des méthodes hybrides, tirant parti à la fois des modèles de Markov cachés et des réseaux neuronaux.

Il est à noter qu'une nouvelle méthode de combinaison des modèles de Markov et des réseaux neuronaux a été proposée récemment, qui semble fort prometteuse ([Trentin, 2001]).

## 12.2 Méthode générale

La méthode générale de reconnaissance est illustrée par la figure 12.1.

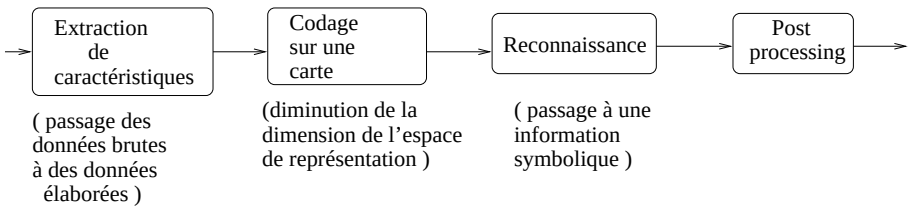



FIG. 12.1 – *Méthode générale de reconnaissance*

## 12.3 Les modèles de Markov cachés

Pour une introduction complète aux modèles de Markov cachés on pourra consulter [Rabiner, 1989]. Un modèle de Markov caché se caractérise par:
• Un ensemble fini d'états $q_1$,...,$q_n$
• Un ensemble de probabilités de transitions entre les états $a_{i,j} = P(X_{t+1} = q_j / X_t = q_i)$
• Un ensemble de lois de probabilités $b_i(y) = P(Y_t = y / X_t = q_i)$
• Un ensemble de probabilités initiales $p_i = P(X_1 = q_i)$

$X_t$ est une variable cachée, alors que $Y_t$ est la variable à laquelle on a accès; le modèle de Markov sert à évaluer la vraisemblance d'une observation $(Y_i)_{i\in[1,N]}$ pour un modèle de Markov donné.

Ainsi on définit:

$$P(Y_1 = y_1,Y_2 = y_2,...,Y_t = y_t/\text{modèle de Markov M}) = \sum_{q_{i_1},...,q_{i_t}} P(X_1 = q_{i_1},...,X_t = q_{i_t},Y_1 = y_1,...Y_t = y_t/M)$$

$$P(Y_1 = y_1,Y_2 = y_2,...,Y_t = y_t/\text{modèle de Markov M}) =$$

$$\sum_{q_{i_1},...,q_{i_t}} P(Y_1 = y_1,...,Y_t = y_t/X_1 = q_{i_1},...,X_t = q_{i_t}/M).(P(X_1 = q_{i_1},...,X_t = q_{i_t}/M)$$

$$P(Y_1 = y_1,Y_2 = y_2,...,Y_t = y_t/\text{modèle de Markov M}) = \sum_{q_{i_1},...,q_{i_t}} p_{i_1}.b_{i_1}(y_1) * \Pi_{k=2..n} a_{i_{k-1},i_k}.b_{i_k}(y_k)$$

On peut utiliser selon la taille du vocabulaire à reconnaître un modèle de Markov par mot, ou un modèle de Markov par lettre/pseuso-lettre/phonème/allophone (selon la taille du vocabulaire à reconnaître et la taille de l'échantillon d'apprentissage auquel on peut avoir accès). Les modèles de Markov peuvent être utilisés à plusieurs échelles; par exemple, utilisation d'un modèle de Markov pour chaque pseudo-lettre, et utilisation d'un modèle de Markov à l'échelle du mot, et enfin utilisation d'un modèle de Markov à l'échelle du langage.

Généralement on n'utilise pas le cas le plus général de modèle de Markov, qui serait trop coûteux; on se limite souvent à une topologie dans laquelle les transitions ne peuvent se faire que d'un état à l'état suivant ou à ce même état, ou bien seulement dans les états proches. Dans [Garcia, 1996] on trouvera quelques exemples de topologies classiques de modèles de Markov.

### 12.3.1 Un lemme bien utile

Pour réaliser des optimisations, on essaiera par la suite de se ramener à des fonctions auxiliaires de la forme suivante:

$\forall ic_i > 0 \rightarrow \sum_i c_i(log(x_i))$ avec pour contrainte $\sum x_i = 1$ admet un maximum unique pour $x_i = \frac{c_i}{\sum_i c_i}$.

On trouvera une preuve par exemple dans [Garcia, 1996].

### 12.3.2 Apprentissage d'un modèle de Markov caché

Le problème est ici de déterminer les probabilités des différentes transitions et les probabilités initiales, pour pouvoir ensuite se baser sur ces données pour faire des reconnaissances.
Les algorithmes suivants, Baum-Welch et Viterbi, sont plus détaillés dans [Jacob, 1995].

• Algorithme de Baum-Welch

On se donne un ensemble d'éléments à reconnaître $w_1$,...,$w_N$. On cherche $\lambda^*$ le modèle de Markov tel que $\Pi_{i\in[1,N]}P(w_i/\lambda^*)$ soit maximal. On utilise une méthode approchée, itérative. On maximise à chaque étape la fonction $Q(\lambda,\lambda') = \sum_i$ chemin de $\lambda_{,n}\, P(i,w_n/\lambda).ln(P(i,w_n/\lambda'))$.

On maximise cette fonction par rapport à $\lambda'$, un modèle $\lambda$ étant donné. Le $\lambda^*$ obtenu est meilleur que $\lambda$ car $\Pi_n P(w_n/\lambda^*) \geq \Pi_n P(w_n/\lambda)$.

On consultera [Baum, 1972] pour une justification approfondie.

• Algorithme de Viterbi

On se donne à nouveau un ensemble d'éléments à reconnaître $w_1$,...,$w_n$.

On définit le modèle $\lambda^*$ comme le modèle qui maximise $\Pi_n P(\tilde{ß}_n,w_n/\lambda^*)$ où $\tilde{ß}_n$ représente le chemin optimal générant $w_n$ par rapport au modèle $\lambda$.

Tout comme dans la méthode précédente on utilise une fonction auxilliaire, cette fois-ci définie par $Q(\lambda,\lambda') = \sum_n P(\tilde{ß},w_n/\lambda).ln(P(\tilde{ß},w_n/\lambda'))$.
Le chemin optimal $\tilde{ß}$ est déterminé par l'algorithme de Viterbi (voir 12.3.3).

### 12.3.3 Reconnaissance par modèle de Markov caché

Les algorithmes suivants, Baum et Viterbi sont extraits de [Jacob, 1995].

• Algorithme de Baum

Soit $Y_1$,...,$Y_T$ une suite d'observations, et $M$ un modèle.
L'objectif de l'algorithme de Baum est d'évaluer la vraisemblance de cette suite d'observations pour ce modèle.
$\alpha_{t,q_j}$ représente la probabilité d'observer les $t$ premières observations et d'être à l'instant $t$ dans l'état $q_j$.
$\beta_{t,q_j}$ représente la probabilité d'observer les $T-t$ dernières observations sachant que l'on est dans l'état $q_j$ à l'instant $t$.

Ces fonctions se calculent par récurrence par les formules suivantes:

$$\forall j \alpha_{1,q_j} = p(j) * b_j(y_1)$$

$$\forall t > 1 \alpha_{t,q_j} = \sum_{q_i} (\alpha_{t-1,q_i}.a_{i,j}.b_j(y_t)$$

$$\forall j \beta_{T,q_j} = 1 \leftrightarrow j \text{ est un état final}$$

$$= 0 \text{ sinon}$$

$$\forall t < T \beta_{t,q_j} = \sum a_{j,i}.b_i(y_{t+1}).\beta_{t+1,q_i}$$

On en déduit la formule de la vraisemblance d'une suite d'observations par rapport à un modèle:

$$P(Y_1 = y_1,...,Y_T = y_T/\text{Modèle } M) = \sum_{q_i} \alpha_{T,q_i}$$

ou bien

$$P(Y_1 = y_1,...,Y_T = y_T/\text{Modèle } M) = \sum_{q_i} b_i(y_1)\beta_{1,q_i}$$

• Algorithme de Viterbi (voir [Fornay, 1973])

Soit $Y_1,...,Y_T$ une suite d'observations. L'objectif de l'algorithme de Viterbi est de déterminer le chemin le plus probable, étant donnée cette observation.

On cherche le chemin $\tilde{ß}$ tel que la probabilité $P(X_1 = q_{\tilde{ß}_1},...,X_T = q_{\tilde{ß}_T},Y_1 = y_1,...,Y_T = y_T/\text{modèle M})$ soit maximale.

On note $I_{t,q_j}$ la probabilité d'émission de la suite $y_1,...,y_t$, pour le chemin le plus probable terminant par $X_t = q_j$.

$$I(1,q_j) = p(j).b_j(y_1)$$

$$I(t,q_j) = max_{q_i}(t-1,q_i).a_{i,j}.b_j(y_t)$$

La probabilité finale est $P(X_1 = q_{\tilde{ß}_1},...,X_T = q_{\tilde{ß}_T},Y_1 = y_1,...,Y_T = y_T/M) = max(I(T,q_j))$. Le chemin final est obtenu en redescendant la récurrence en notant l'argument maximal à chaque fois.

### 12.3.4 Modèles de Markov à l'échelle des phrases

La propriété d'observer une phrase $w_1,...,w_p$ est par définition

$$P(w_1).\Pi_{k=2..p}P(w_k/w_1,..,w_{k-1})$$

• Modèles $n$-grammes: la probabilité $P$ ne dépend que des $n$ termes précédents
• Modèles $n$-classes: la probabilité $P$ ne dépend que de la classe des $n-1$ termes précédents

On consultera notamment [Jacob, 1995, p31].

## 12.4 Les réseaux neuronaux

### 12.4.1 Inspiration biologique

Notre cerveau est intéressant à deux niveaux en ce qui concerne la reconnaissance de données comportant une composante temporelle; d'une part, il y a la réduction de la dimension de l'espace des représentations, souvent utile dans la mesure où des données temporelles sont souvent très volumineuses; on observe dans le cerveau des cartes en dimension 2, régies par des phénomènes d'excitation et d'inhibition, sur lesquelles on peut retrouver les zones correspondantes à différents phonèmes. D'autre part il y a dans le cerveau des mécanismes adaptés au traitement d'informations ayant une composante temporelle.
Le modèle représentant au mieux la réduction de l'espace de représentation est le modèle de Kohonen, fortement inspiré de la réalité biologique; on trouve ainsi dans le néocortex des cartes topographiques - cortex visuel, cortex auditif, cortex somatosensitif... - en dimension 2 représentant des données diverses. Dans le cortex auditif, certaines zones répondent préférentiellement à certains phonèmes, les phonèmes proches étant représentés par des neurones proches; dans le cortex visuel, certaines zones réagissent préférentiellement à certaines orientations, à certaines couleurs, dans le cortex somatosensitif des zones proches correspondent à des excitations proches sur la peau. Les cartes ne sont pas réservées aux zones sensorielles, on en trouve aussi dans les parties motrices; dans le colliculus supérieur une carte code la direction et l'amplitude d'un mouvement de l'oeil.
En ce qui concerne le traitement de séquences de motifs, on constate expérimentalement que les axones

dans le cerveau conduisent les potentiels d'action à une vitesse limitée; en outre souvent les connexions sont beaucoup plus longues donc plus lentes que ne l'impose la distance euclidienne; et des corrélations observées dans le temps montrent que la traitement de séquences temporelles se fait par des délais implémentés dans le cerveau. Le modèle le plus proche de ce qu'il se passe dans le cerveau est ainsi le modèle des réseaux à délais; la méthode d'apprentissage est toutefois largement différente. L'influence du temps se trouve par exemple dans la localisation de la source d'un son, dans la détection du mouvement, dans l'enregistrement de séquences d'actions ordonnées à effectuer. Les réseaux à délais ne sont toutefois pas les seuls types de réseaux pouvant prétendre à une inspiration biologique en terme de traitement de données temporelles; la mémoire récente est constituée d'une boucle de 5 réseaux fortement interconnectés; quelques approches ont été faites dans ce sens, mais les résultats ont été moins concluant que les approches suivantes dans le cadre de la reconnaissance; elles ont plutôt été utilisées pour la prédiction lorsque les données ne subissent pas trop de distorsion dans le temps, ce qui est logique étant donné l'utilisation qui en est faite dans le cerveau.

### 12.4.2 Utilisation pour la réduction de l'espace des représentations

Le schéma 12.1 montre la place de ce module dans le processus global; il est nécessaire lorsque la puissance de calcul dont on dispose est insuffisante pour traiter des données dans un espace de dimension aussi élevée que l'espace d'entrée.

#### Méthode par les cartes de Kohonen

Les cartes auto-adaptatives de Kohonen sont précisément conçues à cet effet. Supposons que l'on souhaite passer d'un espace de dimension $n$ à un espace de dimension $p$. L'ensemble d'apprentissage $x_1$,...,$x_k$,... est inclus dans $R^n$, et on considère un ensemble de neurones placés sur une grille de dimension $p$. Chacun de ces neurones est muni d'un vecteur de poids appartenant à $R^n$. Initialement tous les vecteurs peuvent par exemple être choisis nuls. Les schémas 12.3 (celui de droite est extrait de [Honkela, 1997]) illustrent le principe des cartes de Kohonen. Les cartes de Kohonen s'avèrent aussi performantes, voire plus que les méthodes usuelles, pour un coût en calcul inférieur.

• Apprentissage

On présente des vecteurs $x_i$ de l'ensemble d'apprentissage, par exemple choisis aléatoirement. Pour chaque vecteur $x_i$, on détermine le neurone dont le vecteur de poids est le plus proche de $x_i$. Ensuite, on modifie le vecteur de poids $w_j$ des neurones proches du neurone choisi suivant la formule suivante:

$$w_j \leftarrow w_j + \eta.f(d(j,j^*)).(x_i - w_j)$$

$j^*$ désigne l'indice du neurone choisi, $d$ est une fonction de distance entre deux neurones, et $f$ est une fonction par exemple égale à 1 sur $[0,\theta]$ et à 0 partout ailleurs. D'autres types de fonction $f$ peuvent être choisis. $\eta$ est un réel que l'on fait généralement décroître linéairement avec le temps. La valeur $\theta$ utilisée dans la fonction $f$ est généralement elle aussi décroissante avec le temps. Le schéma 12.2 illustre le fonctionnement de l'apprentissage sur un réseau de Kohonen.

• Reconnaissance

Un vecteur est présenté en entrée, et le vecteur de sortie est tout simplement le vecteur de la carte de Kohonen dont les poids sont les plus proches des poids du vecteur présenté. D'autres systèmes peuvent être employés, par exemple prendre une version pondérée des vecteurs les plus proches en fonction de leur produit scalaire avec le vecteur d'entrée.

#### Autre méthode: le perceptron en diabolo

Une méthode simple à utiliser consiste à effectuer un apprentissage par rétropropagation du gradient, avec en entrée et en sortie les données à traiter; simplement on utilise un réseau multi-couches, et les couches centrales, notamment celle où sera lue la représentation compactée (voir figure 12.4) sont de dimension plus réduites que l'espace d'entrée et l'espace de sortie. Cette méthode présente un avantage: on peut estimer

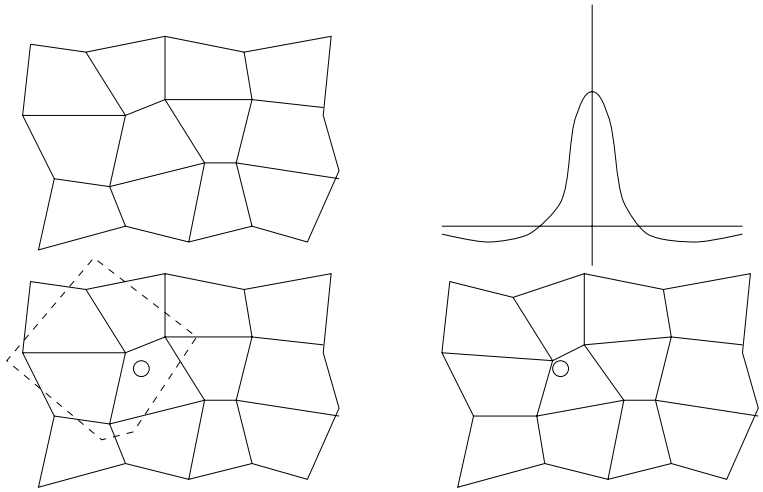


FIG. 12.2 – *En haut à gauche, la carte avant modification. En bas à gauche, encadré en pointillés, la zone influencée par la présentation d'un exemple. En bas à droite, le réseau après modification (on a rapproché les vecteurs proches). En haut à droite un exemple de fonction $f$, dite fonction du chapeau mexicain.*

sur la sortie du réseau la perte d'information que l'on a; ainsi la dimension de l'espace de représentation peut être naturellement adaptée et n'est plus arbitraire.

On trouvera des informations plus précises dans le chapitre "prétraitement et apprentissage non-supervisé", dans la partie "L'apprentissage mis en œuvre".

#### Autres méthodes

Il existe bien sûr les méthodes traditionnelles comme l'analyse en composantes principales (éventuellement étendues comme en analyse en composantes principales à noyau), ou l'analyse en composantes curvilignes.

### 12.4.3 Reconnaissance par réseaux neuronaux

#### Cartes de Kohonen

Les cartes de Kohonen peuvent être encore utilisées à ce niveau; le problème est d'inclure une dimension temporelle. Le seul problème est en fait de définir une norme prenant en compte le temps. On utilise parfois l'intégrale sur la durée des données de la norme sur les données spatiales; de nombreux problèmes se posent au niveau de la différence de durée entre diverses entrée ou au niveau de l'alignement des données dans le temps, mais les résultats sont satisfaisants du moins sur un petit échantillon à apprendre. Pour plus d'informations on consultera [Kangas, 1990].

#### Perceptron multi-couches

Le perceptron multi-couches, muni de l'algorithme de rétro-propagation est l'algorithme le plus classiquement utilisé en réseaux de neurones artificiels. La structure usuellement employée est celle illustrée sur le schéma 12.5, en entrée on place une série de trames (correspondant à une fenêtre temporelle), et on a une sortie pour chaque mot à identifier; la sortie la plus activée révèle le mot reconnu. L'apprentissage se fait par une méthode classique de rétropropagation (par exemple), la littérature à ce sujet est abondante (voir par exemple [Bishop, 1995] pour une liste générale des méthodes couramment employées). Il est clair au vu du schéma que la taille du réseau va très vite augmenter si l'on augmente le nombre de mots à apprendre, en outre le nombre de vecteurs peut être important; la taille du réseau sera donc élevée et le nombre d'exemples nécessaires à un apprentissage efficace sera très grand. Il est donc préférable de s'en tenir à reconnaître par

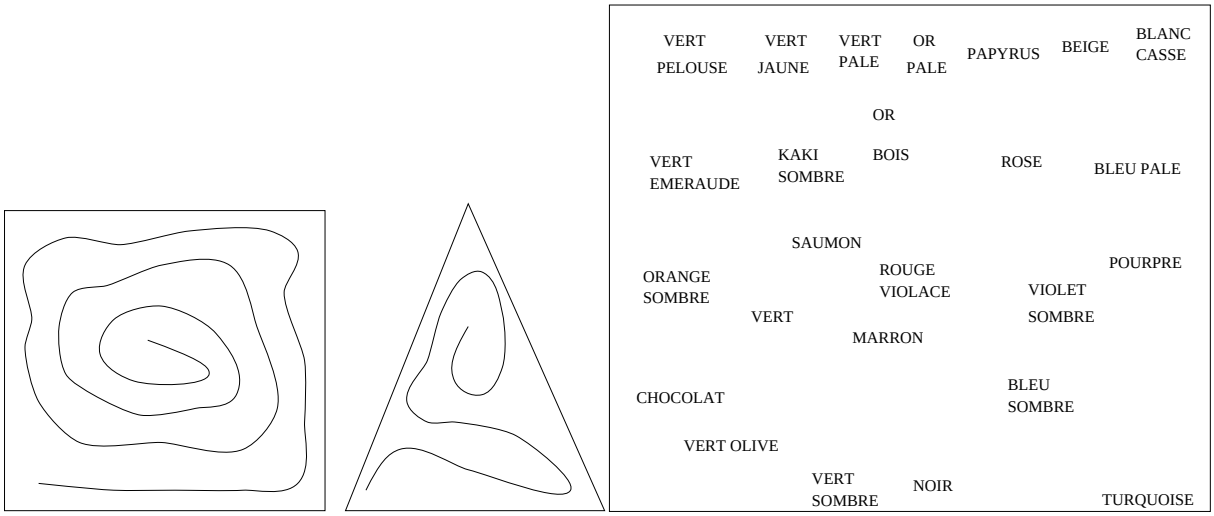



FIG. 12.3 – *Les exemples de gauche montrent le recouvrement d'une forme de dimension 2 par une forme de dimension 1. L'exemple de droite illustre un passage de la dimension 3 (les couleurs sont données en RGB) à la dimension 2.*

exemple des phonèmes ou des lettres. Un défaut majeur de cette nouvelle utilisation est le problème de l'alignement du signal dans le temps, et de sa longueur. Une solution peut être de raisonner en découpant le signal en tranches puis en regroupant les tranches qui se ressemblent jusqu'à obtenir le bon nombre de vecteurs. Par une telle méthode, [Hennebert et al, 1994] annonce des résultats équivalents à ceux obtenus par modèle de Markov caché dans le cas d'un vocabulaire réduit.

**Réseaux à délais**

Le principe de ces réseaux est de munir chaque neurone d'une mémoire. Le nouveau schéma d'un neurone est illustré sur le schéma 12.6, extrait de [Hennebert et al, 1994]. Il est important de noter que *tous* les neurones sont munis de telles mémoires; cela fait que ces réseaux ne sont *pas* équivalents à des réseaux classiques ayant accès à une fenêtre de données.

Très peu d'essais ont encore été faits avec ce type de réseau, mais les premiers résultats semblent très prometteurs.

## 12.5 Méthodes mixtes

Finalement pour combiner le pouvoir de discrimination des réseaux neuronaux et la capacité à traiter des données temporelles des modèles de Markov cachés, des méthodes mixtes sont apparues et se sont rapidement développé.
Je passe sous silence dans la suite les problèmes liés à l'initialisation, ne détaillant que la méthode itérative permettant d'améliorer la reconnaissance; pour plus d'informations on pourra par exemple consulter [Garcia, 1996].

### 12.5.1 Méthode avec un seul perceptron

Je décris ici la méthode mixte modèles de Markov cachés/perceptrons multi-couches; d'autres méthodes hybrides ont été étudiées, utilisant au lieu du perceptron des réseaux à délais par exemple (voir

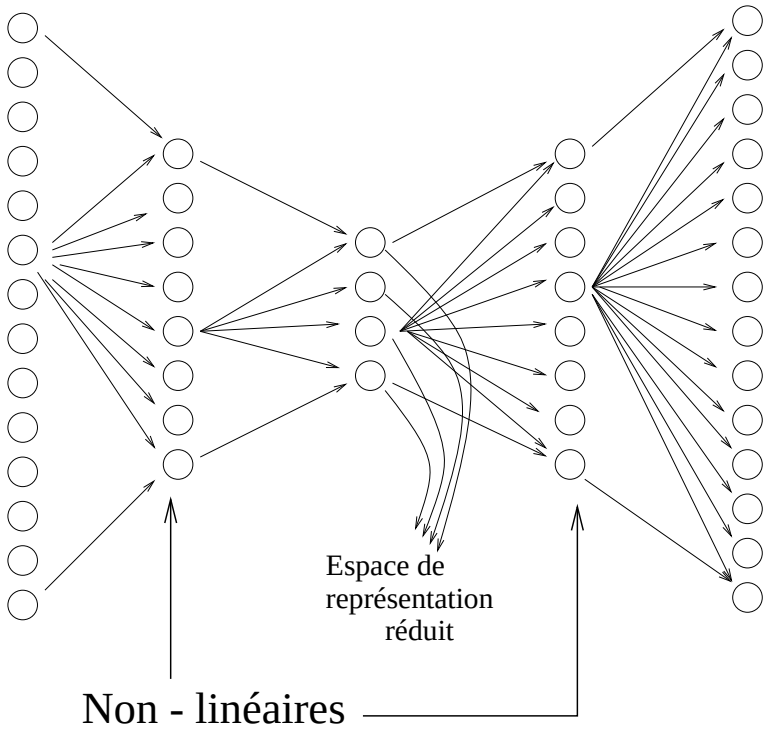



FIG. 12.4 – *Réseau feedforward en diabolo: avec seulement deux couches de poids on obtient la même chose que par une analyse en composantes principales. Certains théorèmes montrent l'intérêt de disposer de 4 couches de poids dont deux dotées de fonctions d'activation non-linéaires.*

[Dugast et al, 1994]).

Comme dans le cas des modèles de Markov, on va considérer que les données observées sont les conséquences d'une suite d'états. L'objectif du perceptron va être l'évaluation de la probabilité *à posteriori* d'un état étant donnée une observation. Le perceptron possède donc une entrée par composante d'un vecteur acoustique (par exemple), et une sortie pour chaque état (correspondant à un état caché du modèle de Markov). L'apprentissage se fait par la méthode classique de rétropropagation, et [Bourlard et al, 1990] montre qu'en effectuant l'apprentissage avec pour sortie un 1 pour l'état correspondant et 0 pour toutes les entrées, sauf arrêt de l'apprentissage dans un minimum local non global, on obtient bien en généralisation pour chaque vecteur acoustique et à chaque sortie la probabilité de l'état correspondant étant donné ce vecteur.

L'apprentissage se fait donc par étapes comme suit:
On dispose d'une matrice de transitions et d'une matrice de poids.
- Pour chaque mot:
  - on détermine pour chaque instant la probabilité de chaque état caché (par le perceptron)
  - on détermine pour le modèle correspondant le chemin le plus probable (par l'algorithme de Viterbi)
- On remet à jour les poids du perceptron (par rétropropagation)
- On remet à jour les probabilités de transition

On répète jusqu'à atteindre une efficacité optimale. Eventuellement le perceptron peut recevoir en entrée une fenêtre temporelle de données (ce qui est un premier pas vers le réseau à délais). Le pouvoir de discrimination est nettement meilleur, ce qui est logique puisque l'apprentissage est basé sur des probabilités à posteriori.

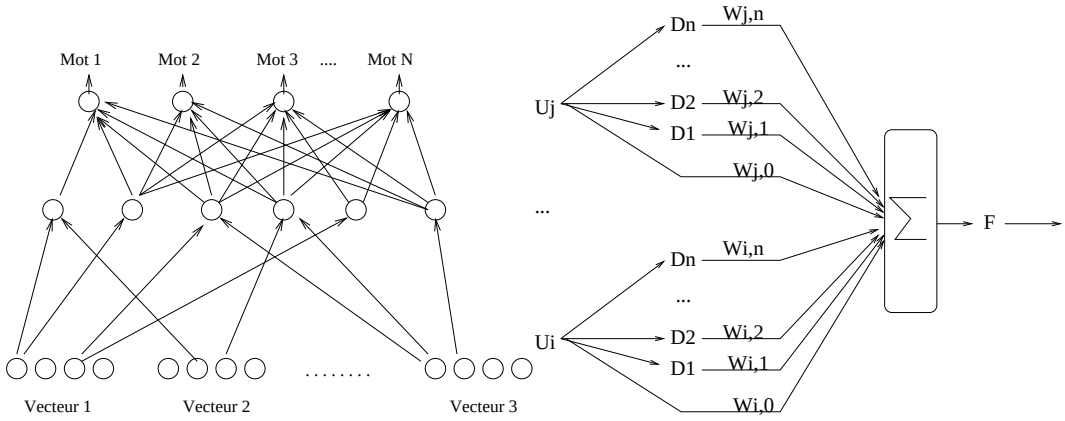



FIG. 12.5 – *Perceptron multi-couches: chaque neurone n'est connecté qu'à des neurones de la couche suivante.*

FIG. 12.6 – *Schéma d'un neurone dans un réseau à délai. $D_i$ est un délai de i étapes. L'apprentissage se fait suivant une rétropropagation du gradient classique.*

### 12.5.2 Méthode avec plusieurs perceptrons

Cette méthode consiste à utiliser un modèle de Markov classique, mais dont chaque état est en fait un perceptron. Chaque perceptron reçoit en entrée l'observation correspondante et son contexte, et renvoit en sortie une autre formulation de l'observation; en comparant la sortie du réseau de neurones à la formulation désirée, par une norme euclidienne, on définit une erreur pour chaque instant pour chaque état. L'algorithme de Viterbi détermine alors la suite d'états optimale. On effectue alors une rétropropagation de l'erreur classique pour remettre à jour le perceptron. Les probabilités de transitions peuvent alors être adaptées. Pour plus d'informations sur cette méthode, on consultera [Garcia, 1996].

## 12.6 Pré-traitement et post-traitement

J'ai supposé ci-dessus que les données étaient fournies sous bonne forme; je vais maintenant détailler un peu plus le prétraitement effectué sur les données pour qu'elles puissent être utilisées dans les algorithmes précédents, dans les deux cas particuliers de la reconnaissance de la parole et de la reconnaissance de l'écrit en-ligne.
Le post-traitement, qui consiste en des vérifications dans le dictionnaire par exemple en utilisant une distance d'éditeur (i.e. la distance entre deux mots est le coût en terme de substitutions/délétions/insertions), voir des vérifications utilisant des modèles de Markov (comme évoqué plus tôt), augmentent très significativement la qualité des résultats.

### 12.6.1 Traitement préliminaire du signal temporel

On peut citer deux catégories de prétraitement importants spécialisés aux signaux temporels (d'autres prétraitements comme la réduction de dimension sont citer plus haut): l'analyse en composantes indépendantes (voir par exemple le proceedings d'Icann 2001), permettant d'isoler une source d'autres sources linéairement combinées à la première, et la segmentation (voir par exemple [Aussem et al, 2001]).

### 12.6.2 Reconnaissance de la parole

La méthode que je donne ici à titre indicatif n'est qu'une méthode parmi d'autres; pour plus d'informations on pourra consulter [Jacob, 1995].

**Filtrage analogique**

Ce filtrage consiste en un filtre passe-bande, éliminant les bruits inintéressants et les erreurs dues aux faiblesses des capteurs. Les données sont ensuite transcrites sous forme digitale.

**Extraction de caractéristiques**

On effectue une transformée de Fourier. On ignore la phase pour ne garder que le module de chaque fréquence. L'aigu est renforcé car il contient beaucoup d'informations par rapport à son amplitude. Selon les cas différents filtres sont appliqués, conduisant généralement à une distribution d'énergie sur les fréquences. On passe ensuite aux coefficients cepstraux en calculant la transformée de Fourier inverse du logarithme des amplitudes.

### 12.6.3 Reconnaissance en-ligne de l'écrit

Des prétraitements sont effectués de manière à ne pas pénaliser la reconnaissance par des problèmes comme:
• L'orientation du tracé
• L'instant de début du tracé

Les données à traiter sont les positions successives du stylo; une fenêtre parcourt le tracé. Les informations que l'on extrait peuvent être par exemple le sinus et le cosinus de l'angle orienté montrant le déplacement dans la fenêtre, l'aire algébrique entre le tracé et le segment reliant le premier et le dernier point de la fenêtre, et les levés de stylo (voir [Garcia, 1996], dont est extrait le schéma 12.7).

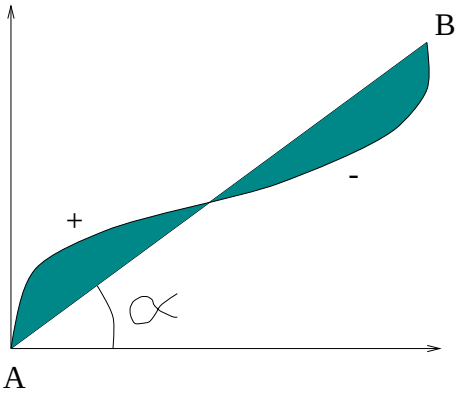



FIG. 12.7 – *A est le premier point de la fenêtre, B le dernier point. On renvoie $cos(\alpha)$, $sin(\alpha)$, l'aire algébrique de la zone grisée et les levers de stylo.*

## 12.7 Conclusion

En conclusion je vais décrire les avantages et les inconvénients des différentes méthodes présentées. Les réseaux neuronaux ont un très bon pouvoir discriminant; par contre ils ne sont pas efficace sur des données temporelles pouvant subir une distorsion (pour des données temporelles parfaitement régulières, ils restent très efficaces, par exemple en matière de prédiction de consommation électrique en fonction de l'heure et du jour et de la consommation des jours précédents). Les modèles de Markov quant à eux ne sont pas gênés par les distorsions dans le temps; par contre leur pouvoir discriminant est assez faible, car les probabilités utilisées sont évaluées à priori. Finalement il semble opportun de choisir une méthode par réseau de neurones lorsqu'il n'y a pas de distorsion dans le temps (par exemple les données issues de systèmes chaotiques, permettant comme le montre le théorème de Takens, un fenêtrage des données sans perte d'information),

et de par contre utiliser un modèle de Markov pour une reconnaissance de classes bien séparées. Lorsque la situation comporte à la fois des distorsions et des classes difficiles à séparer une méthode mixte s'impose; utilisation des réseaux pour l'évaluation à postériori des probabilités, modèles de Markov pour la décision finale. Un prétraitement s'impose en général en raison de la taille des données; en outre dans les deux exemples principaux (reconnaissance de la parole, reconnaissance de l'écriture en ligne), l'utilisation de connaissances d'experts fournit une réduction des données de qualité innateignible par quelque méthode automatique que ce soit (notamment dans le cas de la parole).

# Bibliographie

[Aussem et al, 2001] A. Aussem, M. Fuentes, *Segmentation de série temporelle par une chaîne de Markov Cachée d'experts neuronaux, proceedings of CAP, 2001.*

[Baum, 1972] L.E. Baum, *An Inequality and Associated Maximizations Technique in Statistical Estimation for Probabilistic Functions of a Markov Process, In-equalities,3,1-8*

[Bishop, 1995] C.M. Bishop, *Neural Networks for Pattern Recognition, Clarendon Press, Oxford, 1995*

[Bourlard et al, 1990] Bourlard, C.J. Wellekens, *Links between Markov models and Multi-layer Perceptrons, IEEE trans. Patt. Anal. Machine Intelligence, Vol 12, pp 1167-1178, 1990*

[Dugast et al, 1994] Dugast, L. Devillers, X. Aubert, *Combining TDNN and HMM in a Hybrid System for Improved Continuous Speech Recognition, IEEE transactions on Speech and Audio Processing, 1994, vol 2, No 1, part II, pp 217-223*

[Fornay, 1973] D.R. Fornay, *The Viterbi Algorithm, Proc IEEE, vol 61, ndeg 3, 1973*

[Garcia, 1996] S. Garcia, *Une Approche Neuronale Prédictive pour la Reconnaissance En-ligne de l'Ecriture Cursive, thèse de Doctorat de l'université Paris VI, 1996*

[Hennebert et al, 1994] J. Hennebert, M. Hasler, H. Dedieu, *Neural Networks in Speech Recognition, Proc. 6th Microcomputer School, Prague, Czech Republic, 1994*

[Honkela, 1997] T. Honkela, *Self-organizing Maps in Natural Language Processing, Thesis, University of Technology of Helsinki, 1997*

[Jacob, 1995] B. JACOB, *Un Outil Informatique de Gestion de Modèles de Markov Cachés: Expérimentations en Reconnaissance Automatique de la Parole, thèse de Doctorat del'université Toulouse III, 1995*

[Kangas, 1990] J. Kangas, *Time-delayed Self-organizing Maps, in Proceedings of the International Joint Conference on Neural Networks, 1990, vol. 2,pp 331-336*

[Rabiner, 1989] R. Rabiner, *A Tutorial on Hidden Markov Models and ted Applications in Speech Recognition, proceedings of the IEEE, vol 77, No 2, 1989, pp 257-286*

[Trentin, 2001] E. Trentin, M. Gori, *Continuous Speech Recognition with a Robust Connectionist/Markovian Hybrid Model, in Proceedings of Icann 2001.*

# Quatrième partie

# B: l'apprentissage mis en œuvre - Présentation/étude de quelques algorithmes

Cette sous-partie regroupe des présentations et études de quelques méthodes d'intelligence artificielle; certaines purement numériques, d'autres symboliques.

- L'apprentissage symbolique. Pour de nombreuses applications, les méthodes symboliques restent utiles, pour notamment leur compréhensibilité, même lorsqu'elles sont battues, au niveau des taux de réussite, par des méthodes plus numériques.
- Un texte d'introduction aux méthodes non-supervisées et de prétraitement.
- L'apprentissage Bayésien. Optimal pour une distribution des modèles, l'apprentissage Bayésien, remis au goût du jour par les "Bayes Point Machines", peut bénéficier de bornes théoriques même lorsque l'a priori n'est pas le bon.
- Les Support Vector Machines: outil très classique désormais, amplement vanté dans de très nombreuses publications, les Support Vector Machines ont surtout montré une grande efficacité sur des benchmarks de taille réduite, les résultats obtenus dans des papiers traitant des grandes bases ($> 100.000$) étant plutôt décevant en regard de la prouesse algorithmique. On analyse ici les conséquences de l'approximation consistant à remplacer une fonction de coût binaire par une fonction de coût $L^1$.
- L'implémentation parallèle des réseaux de neurones.
- Une méthode de $k$-plus proches voisins rapides.

# Chapitre 13

# Apprentissage symbolique. Le cas particulier des données numériques et bruitées

**Résumé**

Les aspects théoriques de l'apprentissage, dans lesquels baigne cette thèse, ont été beaucoup plus largement énoncés du côté des méthodes numériques intensives que du côté des méthodes symboliques. Ce manque pourrait sans doute être palié d'autant plus commodément que les méthodes symboliques, manipulant des expressions aisément compréhensibles, travaillent sur des espaces de VC-dimension beaucoup plus réduites. En attendant de tels travaux théoriques, ce chapitre résume différentes techniques symboliques usuelles, sans prétendre, loin de là, à l'exhaustivité.

# Table des matières



## 13.1 Introduction

Je présente dans ce papier différentes méthodes d'apprentissage symbolique bien connues, et plus spécialement le cas des données numériques. Je présente la méthode des arbres de décision, la méthode de l'espace des versions, la méthode de l'étoile, et l'algorithme GSA. Je comparerai en conclusion ces différentes méthodes, ainsi que des algorithmes ultérieurs plus ou moins inspirés de ces algorithmes; je comparerai enfin ces approches aux approches par réseaux neuronaux. Les comparatifs cités sont essentiellement extraits du rapport .

Ce travail est largement basé sur un rapport de recherche rédigé pendant mon DEA, dans le cadre de la notation du cours de D. Zighed.

## 13.2 L'espace des versions (*Mitchell*)

### 13.2.1 Introduction à l'espace des versions

L'algorithme décrit par Mitchell pour l'intégration symbolique dans [Mitchell, 1982] (voir aussi [Smith et al, 1990, Sebag, 1994]) est basé sur l'apprentissage «*par l'action*», dit aussi apprentissage par «*recherche d'explication*».
Les problèmes sur lesquels on travaille sont situés dans un espace, et l'on se ramène d'un problème à un autre par l'application d'un opérateur. Typiquement, si l'on travaille sur l'intégration symbolique, un opérateur (par exemple: l'intégration par parties, ou le remplacement de $\int k.f(x).dx$ par $k \int f(x).dx...$) transforme un problème (par exemple $\int sin(x).cos(x).dx$) en un autre problème.
Ce principe fournit des algorithmes en trois étapes:
•On fournit la solution à un problème (éventuellement la recherche de la solution est automatique). Cette solution est décomposée en une suite d'opérateurs initiaux.
•On génère pour les heuristiques des instances positives (respectivement négatives), c'est à dire des couples $(P,O)$ pour lesquels l'application de l'opérateur $O$ semble transformer le problème $P$ en un problème plus simple (respectivement plus compliqué).
•On modifie les heuristiques pour qu'elles recommandent l'utilisation de l'opérateur $O$ sur le problème $P$ (et sur des problèmes «proches»).
Ce problème est en fait un problème de classification; il faut classifier les problèmes en $k$ classes $K_1$,...,$K_k$, avec $K_i$ l'ensembles des problèmes tels que l'opérateur numéro $i$ soit le plus efficace.

### 13.2.2 Définitions

Une **taxonomie** est un graphe orienté ayant un ensemble de prédicats pour ensemble de sommets tel que:
•Il y a une et une seule racine
•Il n'y a pas de cycle
•($P$ est le fils de $Q$) $\Longrightarrow \forall l\ (P(l) \rightarrow Q(l))$
•$y(l) \wedge y$ a un fils $\Longrightarrow \exists! x/(x$ est fils de $y$ et $x(l)$
Notons qu'un sommet peut tout à fait avoir plusieurs pères.
Par la suite je m'autoriserai lorsque les notations ne sont pas ambigües à identifier un ensemble et le prédicat correspondant (ie. l'ensemble $E$ et le prédicat $P$ tel que $P(x)$ si et seulement si $x \in E$).
Les heuristiques seront décrites à l'aide des taxonomies, c'est à dire que l'on cherchera un ensemble $R_1$,...,$R_i$,... de prédicats de la taxonomie tel que l'opérateur $O$ soit efficace sur le problème $P$ si il existe $i$ tel que $R_i(P)$.
On dit qu'un descripteur ou un prédicat $P$ **couvre** un problème $x$ si et seulement si $P(x)$. Un ensemble de descripteurs couvre un problème si l'un des descripteurs de cet ensemble couvre ce problème. On appelle espace couvert par un descripteur ou un ensemble de descripteurs l'ensemble des problèmes couverts pas le dit descripteur ou ensemble de descripteurs.
On appelle **coût d'un noeud** d'un arbre de recherche le temps CPU qui a été nécessaire pour aller de la racine à ce noeud.
On appelle **noeud ouvert** un noeud sur lequel au moins un opérateur applicable à ce noeud n'a pas encore été appliqué.
On appelle **degré de couverture d'un noeud par une heuristique** la proportion des descriptions de l'heuristique qui couvrent le noeud.

### 13.2.3 Codage d'une heuristique

Une heuristique, comme le montre le principe donné dans l'introduction, doit pouvoir être modifiée de manière incrémentale, c'est à dire que l'on souhaite pouvoir facilement modifier l'heuristique lorsque l'on ajoute une instance positive ou une instance négative.
Une heuristique sera donc codée par deux ensembles $S$ et $G$.
$S$ est un ensemble de descripteurs $D$ tels que tous les fils de $D$ sont des prédicats désignants des problèmes sur lesquels l'heuristique est recommandée; c'est l'ensemble de descripteurs le plus *spécifique*.
$G$ est un ensemble de descripteurs $D$ tels qu'aucune instance négative ne soit couverte par $D$.
Notons que les deux définitions ci-dessus ne sont pas complètes, puisque $S$ et $G$ en sont pas complètement définis. Il suffit ensuite de préciser que pour $G$ on veut l'ensemble de cardinal minimal parmi les ensembles qui couvrent «autant de problèmes que possible» sans inclure d'instance négative, et que pour $S$ on veut l'ensemble aussi petit que possible qui couvre toutes les instances positives.
On voit ici que l'ensemble des ensembles qui peuvent être exactement recouverts dépend de la taxonomie adoptée.
L'état initial dépend du problème traité. On part généralement de $S = \emptyset$; pour $G$ on peut prendre par exemple $G$ couvrant exactement le domaine de définition de l'opérateur.

### 13.2.4 Evolution d'une heuristique

$G$ et $S$ doivent donc être mis à jour à chaque ajout d'une instance positive ou négative.

•Ajout d'une instance positive n'appartenant pas à $G$ La seule solution consiste alors à créer une nouvelle heuristique pour le même opérateur. On peut ainsi avoir un nombre quelconque de paires de la forme ($S$,$G$), avec $S$ et $G$ définis comme précédemment.

•Evolution de $G$ Si un prédicat $P$ est tel que:
•$\exists x_0 / P(x_0) \wedge x_0$ est une instance négative.
•$P \in G$
Alors on supprime $P$ de $G$ et pour tout $Q$ tel que $Q$ est un fils de $P$ et $\neg Q(x_0)$ on ajoute $Q$ à $G$.

•Evolution de $S$ Si $P \notin S$ et si $\forall Q$ ($Q$ est un fils de $P$) on a $Q \in S$ alors on ajoute $P$ à $S$ et on enlève ses fils de $S$.

### 13.2.5 Un exemple

J'illustre le principe de l'espace des versions sur un problème de classification (déterminer les heuristiques revenant à classifier les problèmes suivant l'opérateur à leur appliquer); voir les schémas 13.1, 13.2, 13.3, 13.4, 13.5 et 13.6.

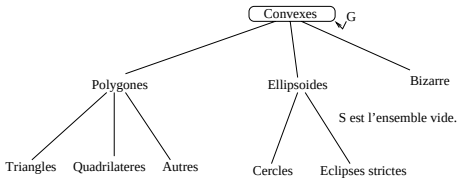



FIG. 13.1 – *Etat initial*

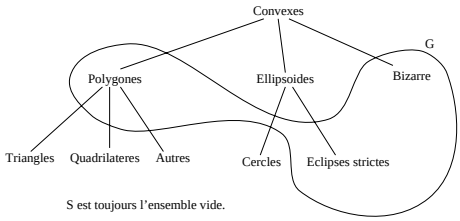



FIG. 13.2 – *Après ajout d'un contre-exemple verifiant «cercle».*

### 13.2.6 Le programme LEX

LEX est un programme d'intégration symbolique utilisant l'espace des versions. Il a été créé par Mitchell, Hutgoff et Banerji.

#### Principe général

LEX est composé de quatre modules distincts (voir le schéma 13.7): un module pour générer les problèmes, un module pour les résoudre, un module pour faire une critique de la résolution, et un module pour faire évoluer les heuristiques.

#### Le module de résolution

L'algorithme est le suivant:
**Répéter**
**Si** aucun noeud ouvert n'est
couvert par une heuristique **alors**
Appliquer un opérateur applicable sur

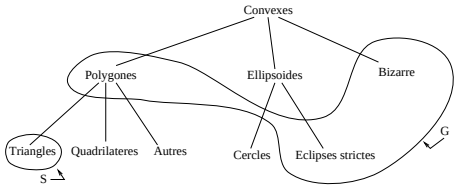



FIG. 13.3 – *Après ajout d'un exemple vérifiant «triangle».*

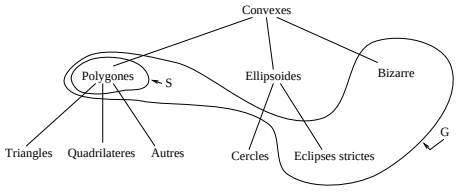



FIG. 13.4 – *Après ajout de deux exemples vérifiant l'un «quadrilatère» et l'autre «autres polygones».*

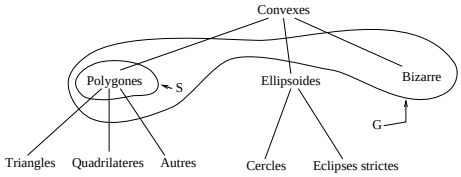



FIG. 13.5 – *Après ajout d'un contre-exemple «ellipse stricte».*

le noeud ouvert le moins coûteux
**sinon**
**Si** une seule heuristique
s'applique à un noeud ouvert **alors**
appliquer l'heuristique au noeud
**Sinon**
Choisir l'heuristique et le noeud qui
donnent le meilleur taux de couverture
**Jusqu'à** ce que le problème soit résolu ou la limite aux ressources atteintes

### Le module de critique

Le module de critique détermine quelles couples (problème,opérateur) seront considérés comme des instances positives ou négatives.
Une **instance positive** est l'application d'un opérateur sur un problème tel que le problème résultant soit à l'intérieur de la solution la plus courte découverte. Une **instance négative** est un couple (opérateur, problème) tel que:
•l'application de l'opérateur sur le problème est telle que l'opérateur résultant ne fait pas partie de la solution la plus courte (découverte par le module de résolution).
•l'application de l'opérateur sur le problème est telle que le noeud obtenu soit sans solution de coût inférieur ou égal à $\alpha$ fois le coût optimal trouvé. Mitchell propose $\alpha = 1.15$.

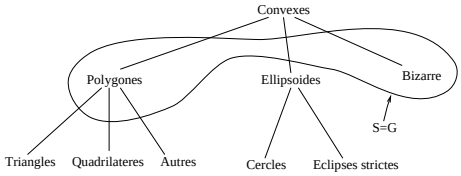



FIG. 13.6 – *Après ajout d'un exemple «bizarre», on a maintenant $S = G$ et donc plus rien ne peut changer.*

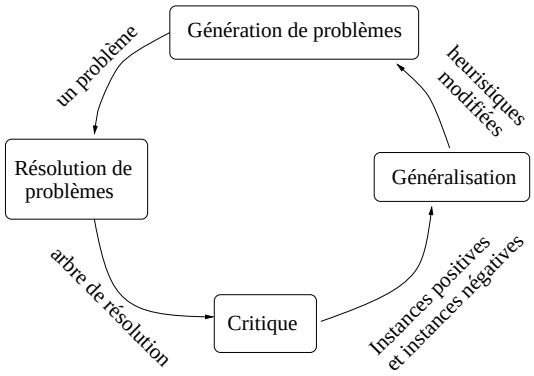



FIG. 13.7 – *Le schéma général*

**Le module de généralisation**

Le module de généralisation propose des heuristiques ou les modifie, comme expliqué dans 13.2.4.

**Le module de génération de problèmes**

Le générateur de problèmes peut générer deux types de problèmes:
•Pour raffiner des heuristiques partiellement apprises, le générateur peut engendrer un problème couvert par certains descripteurs mais pas par d'autres.
•Pour engendrer de nouvelles heuristiques, le générateur choisit un problème sur lequel deux opérateurs distincts peuvent s'appliquer sans que l'on sache lequel utiliser.
Un problème est le choix entre ces deux motivations pour créer un problème. On peut par exemple raffiner les heuristiques jusqu'à ce que ça ne soit plus possible (cas $S = G$), puis passer à la création de nouvelles heuristiques.

### 13.2.7 Améliorations possibles - Implémentation pratique

Les améliorations possibles citées ci-dessous à l'exception de la dernière ont été proposées par Mitchell dans [Mitchell, 1982].
•Créer des problèmes toujours solubles
Méthode: partir d'un état but et appliquer des inverses d'opérateurs.
•Moduler l'allocation de ressources en fonction de l'importance de la résolution d'un problème et de ce que l'on sait de la longueur maximale d'une solution (si l'on a appliqué la méthode ci-dessus).
•Prendre en compte les motivations du générateur: favoriser l'utilisation des heuristiques que celui-ci veut tester.
•Choix entre les deux types de problèmes à générer: vaut-il mieux générer de nouvelles heuristiques ou optimiser les déjà connues?
•Formalisation de la raison pour laquelle une instance est positive
On peut définir le prédicat $Instpos(o,e)$, vrai si $o$ appliqué à $e$ est une instance positive, comme suit:
$Instpos(o,e) = (\neg But(e)) \wedge (But(Appliquer(o,e)) \vee Soluble(Appliquer(o,e)))$
avec $soluble(e) \iff \exists o \; But(Appliquer(o,e)) \vee Soluble(Appliquer(o,e))$ Une méthode efficace pour

l'apprentissage pourrait être:
- Donner l'arbre de preuve de $Instpos(o,e)$
- En déduire une formule $f(e)$ prouvant $Instpos(o,e)$
- La formule $\forall s\ f(s) \Longrightarrow Instpos(o,e)$ est vraie; notons $g$ cette formule
- On écrit une formule aussi proche que possible de $g$ avec la relations de couverture; c'est à dire que l'on se ramène à une formule ne comportant que la fonction *Appliquer*, la relation *couvre*, et les constantes (opérateurs), plus naturellement la quantification sur $s$ le problème.
- On ramène les conditions sur $Appliquer(o,e)$ (avec $o$ une constante) à des conditions sur $e$ en appliquant les opérateurs inversés.
- Ajouter à $S$ les descripteurs des problèmes $s$ vérifiant les conditions.

•Considérer des séquences d'opérateurs

J'ai réalisé une implémentation pratique sur le problème du «saute ou glisse» utilisé par Langley dans [Langley, 1983], problème simple pour lequel il était réalisable d'implémenter certaines des améliorations suggérées ci-dessus.

Je commence par donner quelques défauts de la méthode que j'emploie:
•Le problème est beaucoup plus simple que le problème initialement traité par $LEX$.
•Cette simplicité empêche l'algorithme d'utiliser toutes ses ressources; par exemple pour ce problème (et avec la taxonomie employée) l'algorithme n'a pas besoin de développer plusieurs heuristiques pour un même opérateur
•Le problème ne comporte pas de données numériques
•Je ne donne pas de limite au temps de calcul, et je me limite à des tailles telles que l'algorithme détermine toujours en temps raisonnable si le problème est soluble ou non; cela simplifie notamment la tâche du module de critique. Toutefois il s'agit là d'un cas particulier de l'algorithme initial (cas des ressources infinies et des problèmes toujours soit solubles soit pour lesquels on peut être sûr qu'ils ne sont pas solubles)
Pour vérifier que l'apprentissage n'est pas un simple apprentissage par coeur, on effectue l'apprentissage sur des exemples autres que le jeu d'exemples sur lequel on calcule la vitesse moyenne.
Tout d'abord les résultats qualitatifs sont sensiblement les mêmes que pour $LEX$, comme le montre le graphe 13.8. Je ne donne que les résultats pour une série de tests, mais d'autres tests confirment ces résultats, et l'on retrouve parfois le caractère non-monotone de l'apprentissage.
J'ai tout d'abord cherché à implémenter la méthode consistant à générer les problèmes de l'apprentissage par des applications d'inverses d'opérateurs. L'avantage est d'obtenir exclusivement des problèmes solubles, donc plus instructifs.
Les résultats sont flagrants; et d'autres jeux d'exemples le confirment. Les exemples générés en appliquant des inverses d'opérateurs ne donnent pas un bon apprentissage, même en essayant d'imposer des exemples compliqués (i.e. obtenus avec un grand nombre d'applications d'inverses d'opérateurs). Je ne peux pas expliquer ce phénomène ni garantir qu'il serait aussi vérifié sur des problèmes plus compliqués; puisque les exemples potentiellement générés sont exactement les mêmes, c'est la probabilité pour un exemple d'être généré qui est en cause, et apparemment la méthode aléatoire est ici meilleure (résultat confirmé par plusieurs jeux d'exemples).
J'ai aussi implémenté un algorithme qui favorise les problèmes qui ne sont couverts par le $S$ d'aucune heuristique, et un algorithme qui favorise les problèmes couverts par les $G$ de différentes heuristiques. L'expérience montre que la meilleure méthode parmi celles que j'ai implémenté (au moins dans le cas du problème traité ici) est le tirage au sort d'exemples parmi les problèmes pour lesquelles deux ou plusieurs heuristiques sont potentiellement intéressées (ie leur $G$ couvre le problème mais pas leur $S$).

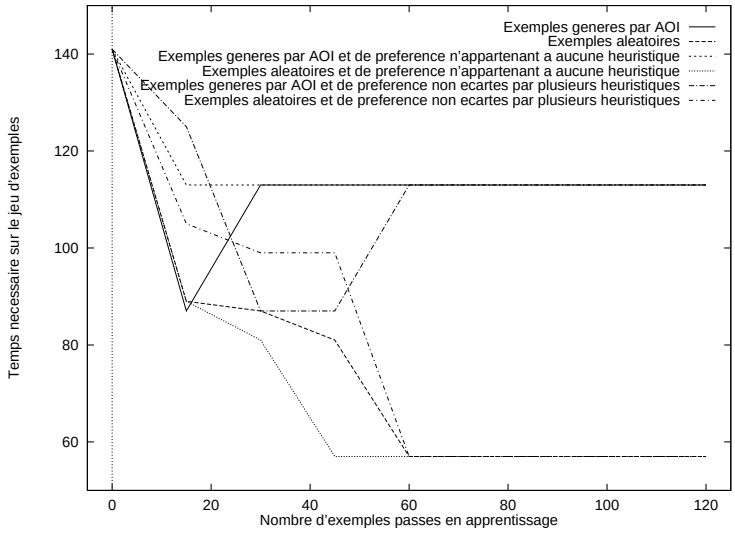



FIG. 13.8 – *Les résultats obtenus sur d'autres jeux d'exemples sont sensiblement les mêmes. AOI signifie «application d'inverses d'opérateurs».*

### 13.2.8 Les failles de l'espace des versions

•Haussler a démontré dans [Haussler, 1988] que le nombre de formules conjonctives dans $G$ peut-être exponentiel en fonction du nombre de descriteurs en logique booléenne. Des améliorations ultérieures (voir [Hirsh, 1992]) réduisent ce problème.
•Les données erronées ont des conséquences très lourdes, car $S$ et $G$ évoluent de manière monotone ($S$ croît et $G$ décroît). Ainsi les données numériques (avec leurs imprécisions) sont elles difficiles à gérer.

## 13.3 Approche descendante : arbres de décision

Je ne donne ici qu'une rapide introduction aux arbres de décision. Pour plus de précisions sur les arbres de décision, on consultera par exemple [Breiman et al, 1984], [Quinlan, 1986] et [Quinlan, 1990].

### 13.3.1 Définitions

On travaille sur un ensemble d'objets ayant des attributs $A_1$,...,$A_n$.
On suppose $A_i$ à valeur dans $\{A_{i,1},...,A_{i,k_i}\}$.
Les objets sont répartis en $p$ classes $E_1$,$E_2$,...,$E_p$.
On suppose qu'il n'existe pas deux objets $x$ et $y$ avec $\forall i\ A_i(x) = A_i(y) \wedge x \in E_a \wedge y \in E_b \wedge a \neq b$.
Un arbre de décision est un arbre tel que chaque noeud non-feuille est étiqueté par un attribut $A_i$, et est relié par des arêtes numérotées $A_{i,1}$,...,$A_{i,k_i}$ à des arbres de décision pour les ensembles d'objets $E_1$,...,$E_{k_i}$ avec $E_j = \{x|A_i(x) = A_{i,j}\}$. L'ensemble des attributs reste le même, c'est à dire que l'on peut avoir deux noeuds pour le même attribut; toutefois on travaille généralement sur des arbres réduits, c'est à dire des

arbres tels qu'un attribut n'est utilisé qu'une fois (au maximum) par chemin.

### 13.3.2 Construction de l'arbre

On procède de manière récursive:
•On choisit un attribut $A_i$
•On a pour noeud racine le noeud étiqueté $A_i$
•Ce noeud a pour fils les $k_i$ arbres de décision correpondant aux ensembles $E_j = \{x|A_i(x) = A_{i,j}$, avec les attributs $A_1$,...,$A_{j-1}$,$A_{j+1}$,...,$A_n$.

Lorsque les données sont vastes on procède comme suit:
Choisir aléatoirement un sous-ensemble des données (ce sous-ensemble est appelé «*fenêtre*»)
Répéter
Construire un arbre de décision pour les données de la fenêtre
Chercher s'il y a des éléments mal classés
Si oui former une nouvelle fenêtre avec les éléments mal classés
Jusqu'à ce qu'il n'y ait plus d'éléments mal classés

### 13.3.3 Choix de l'attribut

Il est bien sûr préférable d'avoir un arbre aussi peu profond que possible.
Une bonne heuristique pour cela est de minimiser le contenu informatif restant après la séparation par le premier attribut.
On mesure la quantité d'informations d'un ensemble réparti en $p$ classes $E_1$,...,$E_p$ par

$$\Sigma_{i\in[1,p]} - p(x \in E_i)log(p(x \in E_i)$$

$p(x \in E_i)$ est le cardinal de $E_i$ divisé par le cardinal de l'union des $E_i$. La quantité totale d'information dans l'ensemble une fois divisé selon l'attribut $A_i$ est donc: $f(A_i) = -\Sigma_{l=1}^{k_i} p(x \in E_i|A_i(x) = A_{i,l})log(p(x \in E_i|A_i = A_{i,l}))$ avec $p(x \in E_i|A_i(x) = A_{i,l})$ la probabilité que $x$ appartienne à $E_i$ sachant que $A_i(x) = A_{i,l}$, c'est donc le cardinal de $E_i \cap \{x|A_i(x) = A_{i,l}\}$ divisé par le cardinal de $\{x|A_i(x) = A_{i,l}\}$.
On choisit donc l'attribut $A_i$ tel que $f(A_i)$ soit minimal.

### 13.3.4 Traitement des données numériques

La méthode ci-dessus est définie pour des variables qualitatives, prenant leurs valeurs dans des ensembles discrets de cardinal fini. Dans les cas continus, il faut discrétiser la variable, en découpant son domaine en un nombre fini d'intervalles.
On pourra consuler [Bouroche et al, 1970], [Fayyad et al, 1993], [Van de Merckt, 1993] et [Botta et al, 1992], dans lesquels on trouvera des méthodes diverses pour réaliser cet objectif.
Une autre solution est l'utilisation d'arbres flous: dans un tel arbre, lorsqu'un noeud correspond à un attribut numérique, il n'y a pas d'embranchement précis à la sortie de ce noeud, de la forme «si $A(x) \in [a,b]$ alors suivre telle branche»; on pondère en fonction de $A(x)$ les branches suivant le noeud, et on examine en

bas de l'arbre les pondérations obtenues pour les différentes classes.

### 13.3.5 Performances

Quinlan (voir [Quinlan, 1986] et [Quinlan, 1990]) souligne que la complexité de cet algorithme est linéaire en la difficulté, la difficulté étant quantifiée comme le produit des 3 quantités:
•Nombre d'exemples
•Nombre d'attributs
•Complexité du concept (égal au nombre de noeuds de l'arbre de décision)

Les arbres de décision ont en outre une bonne résistance aux données bruitées.

### 13.3.6 Faiblesses

Le traitement des données numériques pose des problèmes intéressants: notamment on peut se demander comment segmenter les données. Si l'on veut partager suivant un critère numérique, il faut choisir où couper le segment, et éventuellement en combien de sous-segments le couper.
•Fayyad et Irani proposent une segmentation en intervalles significatifs, les intervalles restants étant regroupés dans une branche particulière (voir [Fayyad et al, 1993]).
•Botta et Giordana travaillent quant à eux sur des arbres de décision flous et utilisent des algorithmes génétiques pour optimiser les choix de branchement (voir [Botta et al, 1992]).
•Heath et al. utilisent des arbres obliques ([Heath et al, 1993]).

## 13.4 L'algorithme étoile *Michalski*

### 13.4.1 Définitions

Une **description de concept** est une structure de données symbolique utilisée pour décrire un concept (c'est à dire une classe d'instances dans le domaine considéré).
On note $D ::> K$ avec $D$ une description de concept et $K$ un prédicat désignant un concept pour l'implication « *Si un objet vérifie D alors il est dans K* ».
On appelle hypothèse inductive un ensemble de la forme $H = \{D_i ::> K_i | i \in I\}$, avec $D_i$ des descriptions de concepts et $K_i$ des noms de concepts (c'est à dire des prédicats décrivant des concepts).
On note $F$ l'ensemble des observations. $F$ est de la forme $\{e_{i,k} ::> K_i\}$, avec $K_i$ un prédicat ou une classe et $e_{i,k}$ un évènement formateur. On suppose que l'on a une et une seule classe par évènement formateur.
On note $A| > B$ l'assertion « $A$ se spécialise en $B$ », c'est à dire « $A$ implique $B$ ». On note $A| < B$ l'assertion « $A$ se généralise en $B$ », c'est à dire « $B$ implique $A$ ».

On souhaite avoir $H| > F$, c'est à dire que l'on souhaite:
•$\forall i (E_i \Longrightarrow D_i)$ avec $E_i$ la disjonction des $e_{i,k}$ (notion de complétude)

•$\forall(i,j) \in I^2 i \neq j \Longrightarrow (D_i \Longrightarrow \neg E_j)$ (notion de cohérence)

On appelle **formule relationnelle** une formule du type $op(t,t')$ avec $t$ et $t'$ des termes quelconques et $op$ l'un des opérateurs $=, \leq, \geq, >, <$.
On appelle **c-expression** une expression formée de quantificateurs (quelconques et en nombre quelconque) sur une formule relationnelle.
On appelle **étoile** de l'évènement $e$ contre les contraintes $E$ l'ensemble de toutes les descriptions possibles non redondantes de l'évènement $e$ qui ne violent pas les contraintes $E$. Pour simplifier on a $e$ un exemple et $E$ un ensemble de contre-exemples. L'étoile de $e$ contre $E$, notée $G(e|E)$, est l'ensemble des c-expressions les plus générales possibles couvrant $e$ et respectant $E$.
On appelle **étoile bornée** $G(e|E,m)$ un ensemble d'au maximum $m$ descriptions inclus dans l'étoile de $e$ contre $E$.

### 13.4.2 Différents types de descriptions

On peut rechercher des descriptions de type très différents suivant l'usage que l'on souhaite en faire. Notons par exemple:
•Les descriptions caractéristiques.
L'objectif est d'avoir une description qui donne toutes les caractéristiques communes à une classe.
Il est indispensable que de telles descriptions soient complètes. L'objectif est qu'elles soient maximales; on cherche une description qui soit une généralisation maximale.
•Les descriptions discriminantes.
Cette fois-ci l'objectif est d'avoir des descriptions qui permette de décider si un objet donné appartient à une classe ou à une autre. Cette fois-ci on cherche à avoir à la fois la complétude et la cohérence, et les meilleures descriptions parmi celles satisfaisant ces deux critères sont les descriptions les plus courtes.

On souhaite donc un algorithme de construction de descriptions qui soit une méthode générale, c'est à dire ayant pour paramètre une fonction d'évaluation des descriptions, qui donne une description aussi «bonne» que possible, «bonne» au sens de la fonction d'évaluation que l'on s'est donné.

### 13.4.3 Différents types d'apprentissage de concepts

#### Apprentissage d'un concept unique

Il s'agit du cas où l'on a une et une seule classe. On a alors deux sous-cas:
•seulement des exemples positifs, c'est à dire qu'on a un ensemble d'éléments appartenant au concept, et pas des exemples d'éléments n'appartenant pas au concept. Dans ce cas la notion de cohérence n'a plus de sens, il n'y a pas de limite à la généralisation de $D_1$. Pour limiter, on peut:
- Chercher la conjonction maximale vérifiant la complétude
- Limiter le pourcentage d'évènements que la description place dans la classe.
•Des exemples positifs et des exemples négatifs
Dans ce cas les deux notions de complétude et de cohérence ont un sens.

**Apprentissage d'un concept multiple**

Dans ce cas les deux notions de complétude et de cohérence ont un sens, mais on peut avoir plusieurs cas:
•Les descriptions $D_i$ doivent être distinctes.
•Les descriptions sont autorisées à s'intersecter.

Dans le deuxième cas la notion de cohérence n'a plus de sens, et il faut donner un critère supplémentaire pour l'évaluation d'une description.

Ces distinctions montrent encore la nécessité d'un algorithme général, permettant de construire une description de qualité, quelle que soit le critère de qualité.

### 13.4.4 Autres critères pour la qualité d'une description. Importance du langage de description

Un avantage majeur de l'apprentissage symbolique sur l'apprentissage par réseaux de neurones est l'intelligibilité des résultats. Pour garder cet avantage on peut s'intéresser à des descriptions facilement compréhensibles, en privilégiant des formules peu profondes, peu larges. On peut aussi considérer qu'en se limitant à certaines catégories de formules logiques on a des résultats plus compréhensibles.

### 13.4.5 Principe général

Chercher une hypothèse inductive revient à effectuer une recherche heuristique dans un espace d'états:
•Les états sont les descriptions symboliques
•L'état initial est l'ensemble des observations
•Les opérateurs sont les règles d'inférence, c'est à dire
- Les généralisations
- Les spécialisations
- Les reformulations

$A$ se généralise en $B$ si $A$ implique $B$, $A$ se spécialise en $B$ si $B$ implique $A$, $A$ se reformule en $B$ si $A \iff B$.
•On vise un état qui implique l'état initial et maximise le critère de choix d'une description. Le critère est une fonction d'évaluation appelée $LEF$, dont on pourra consulter la forme précise dans [Michalski, 1983]. On trouvera dans le même article des opérateurs particuliers de généralisation, reformulation, spécialisation.

### 13.4.6 Génération de l'étoile bornée $G(e|NEG,m)$

Dans l'algorithme général, présenté plus bas, on aura besoin de créer des étoiles bornées $G(e|NEG,m)$. On note $LEF_1$ un critère sur les descriptions, qui n'est pas le critère de préférence général; $LEF_1$ est ici

un critère qui commence par minimiser le nombre d'éléments appartenant à $NEG$ qui sont reconnus par cette description, puis qui maximise le nombre d'éléments appartenant à $POS$ qui sont reconnus. $POS$ ne désigne pas seulement $e$ mais bien l'ensemble des exemples positifs.
La méthode est la suivante:
1) On initialise $PS$ à la liste des sélecteurs individuels de $e$; c'est à dire que les éléments de $PS$ sont des généralisations de $e$. Par exemple, si $e$ est de la forme $grand \wedge brun \wedge maigre$, alors $PS$ est la liste ($grand$,$brun$,$maigre$). $grand$ est une généralisation de $grand \wedge brun \wedge maigre$, $brun$ aussi, $maigre$ aussi.
2) On opère des généralisations et des reformulations de $e$ pour générer de nouveaux éléments dans $PS$. Michalski propose de se limiter à certaines règles pour cela (voir [Michalski, 1983]).
3) On trie suivant $LEF_1$ les éléments de $PS$. Quand $|PS| > m$ on ne garde que les $m$ premiers.
4) On met dans la liste SOLUTIONS les descriptions complètes et cohérentes et on les ôte de $PS$. Rappelons qu'une description est complète si et seulement si tous les exemples positifs sont couverts et cohérente si et seulement si aucun exemple négatif n'est couvert. Si $|SOLUTIONS| > \#SOL$ alors on s'arrête. On place dans la liste CONSISTANTS les descriptions cohérentes et incomplètes.
Si $|CONSISTANTS| > \#CONS$ alors on passe à l'étape 6.
5) Chaque expression $E$ de $PS$ est spécialisée de différentes manières en ajoutant un sélecteur de $PS$ de préférence inférieure ou égale à $\%BRANCH$ multiplié par la préférence de $E$. On trie ensuite suivant $LEF_1$ et on ne garde que les $m$ premiers.
Si $|CONSISTANT| < \#CONS$ et si la limite au temps de calcul n'est pas dépassée alors on retourne à l'étape 4.
6) On généralise les expressions de $CONSISTANT$. Pour cette étape Michalski conseille de se limiter à certains types de généralisation, voir [Michalski, 1983].
7) On trie maintenant les généralisations obtenues suivant le critère de préférence $LEF$.

$\#SOL$ est le nombre de descriptions de concept distinctes recherchées. Les paramètre $\#CONS$ et $\%BRANCH$ sont fixés par l'utilisateur. $m$ peut être soit déterminé par l'utilisateur soit estimé par le programme.

### 13.4.7 Algorithme général

La création des étoiles bornées est l'étape la plus complexe du processus. Voici l'algorithme général:
1) Affecter $i \leftarrow 0$
2) Affecter $G \leftarrow G(e|NEG,m)$
3) $D_i$ est le meilleur élément de $G$ suivant le critère $LEF$.
4) Si $POS \subset D_i$ alors on passe à l'étape 8.
5) Affecter $POS \leftarrow POS \cap (\neg D)$
6) Affecter $i \leftarrow i + 1$
7) Aller à l'étape 1.
8) La disjonction des $D_i$ est une description complète et cohérente du concept; on reformule pour simplifier (reformuler, mais pas généraliser ou spécialiser).

### 13.4.8 Efficacité

L'algorithme étoile présente la caractéristique d'être très paramétrable. Cette caractéristique est à la fois une qualité (car elle lui confère une grande flexibilité) et un inconvénient (car elle est beaucoup plus lourde à manipuler que par exemple les arbres de décision).

## 13.5 L'algorithme GSA

La méthode GSA, dîte aussi «par contraintes» est un hybride des méthodes de l'espace des versions et de la méthode de l'étoile.

### 13.5.1 Une nouvelle conception des étoiles

Avec $e$ un exemple positif et $ce_1$,...,$ce_n$ des exemples négatifs, on note $G(e,ce_1,...,ce_n)$ la conjonction des $G(e,ce_i)$ où $G(e,ce_i)$ est la disjonction des formules maximalement générales couvrant $e$ et rejetant $ce_i$.

L'exemple suivant est extrait de l'article [Sebag, 1995]; toutefois il est ici un peu plus dense pour faciliter la compréhension.

| | Forme | Taille | Poids | Ovni? |
|---|---|---|---|---|
| $e$ | circulaire | 16 | ? | OUI |
| $ce_1$ | 2 ailes | 160 | 7 | NON |
| $ce_2$ | circulaire | 3 | 4 | NON |

Je considère ici que «ovale» est la valeur la plus générale généralisant «circulaire» et excluant la valeur « 2 ailes ».

$G(e,ce_1) = [Forme = ovale] \vee [Taille < 160]$
$G(e,ce_2) = [Taille > 3]$
$G(e,ce_1,ce_2) = ([Forme = ovale] \vee [Taille < 160]) \wedge [Taille > 3]$

La complexité de la construction est en $O(N \times P)$, avec $N$ le nombre d'exemples et $P$ le nombre d'attributs.

### 13.5.2 Algorithme général de l'apprentissage de GSA

Dans l'algorithme donné par Michalski pour l'algorithme étoile, on engendre une étoile, puis on travaille sur l'espace dépourvu des exemples couverts par la dite étoile. GSA va lui travailler sur un exemple choisi au hasard, pour définir une première étoile. Puis il va choisir un deuxième exemple, toujours au hasard, et le classifier au vu des étoiles déjà construites. Si l'exemple est bien classé, alors l'algorithme passe à un exemple suivant, sinon il défini une autre étoile par rapport à cet exemple, et ainsi de suite.
On a ainsi une largement meilleure résistance au bruit: un exemple faux ne suffit pas à faire échouer tout l'algorithme.
La complexité de l'apprentissage est en $O(P.N^3)$.

### 13.5.3 Algorithme général d'évaluation d'un exemple par GSA

On compte le nombre d'étoiles de la classe $K$ couvrant $x$; on classe alors $x$ dans la classe pour laquelle $x$ est couvert par le plus grand nombre d'étoiles.
La complexité de la classification est en $O(P.N^2)$.

### 13.5.4 Heuristiques

Les heuristiques qui suivent sont proposées dans [Sebag, 1995].

•Résistance aux erreurs dans les contre-exemples Au lieu de définir $x \in G(e,ce_1,...,ce_n) \iff \forall i x \in G(e,ce_i)$ on définit $x \in G(e,ce_1,...,ce_n)$ si et seulement si $x$ appartient à $(100-\epsilon)\%$ des $G(e,ce_i)$. Ainsi il ne suffit pas d'avoir un contre-exemple faux pour gâcher tout l'apprentissage. $\epsilon$ doit être fixé par l'expert au taux d'erreur estimé dans les données.

•Modification des $G(e,ce_i)$ $G(e,ce)$ est une disjonction de formules très générales; pour éviter que chaque exemple appartienne à toutes les étoiles on impose qu'en fait au moins un nombre minimal des formules soient vérifiées.

### 13.5.5 Atouts et faiblesses de GSA

GSA est très performant par rapport aux autres méthodes considérées ci-dessus, dans le cadre que l'on s'est fixé, c'est à dire l'analyse de données numériques; toutefois il reste plutôt inférieur aux réseaux neuronaux. L'avantage usuel de l'apprentissage symbolique est l'intelligibilité des résultats; dans le cas de GSA les données sont difficilement interprétables.

## 13.6 Conclusion

Le choix d'une méthode d'apprentissage sur des données numériques est très dépendant de l'usage que l'on souhaite faire de la méthode de classement obtenue. Si l'objectif est l'efficacité "brute", ie le taux de succès obtenu ou l'erreur en régression, le mieux est d'utiliser les réseaux de neurones, rapides et efficaces (du moins dans le cas des ondes de Breiman cité dans [Sysmenu, 1995]), surtout avec des architectures adaptées; il faut toutefois noter que vraisemblablement d'autres méthodes numériques "intensives" auraient des résultats similaires (peut-être faut-il préférer des Support Vector Machines si la dimension est grande devant le nombre d'exemples, les réseaux de neurones dans le cas contraire...). Selon des résultats de praticiens, l'augmentation du nombre de paramètres est souvent plus réduite dans le cas des réseaux de neurones, du fait que la fonction est non-linéaire en fonction des paramètres (ce point est plus détaillé dans le chapitre sur l'apprentissage non-indépendant, la stabilisation et le contrôle); toutefois, cette réduction du nombre de paramètres ne conduit pas franchement à une amélioration de l'intelligibilité, même si le nombre de paramètres est souvent retenu comme critère d'intelligibilité; des matrices de poids restent des matrices de poids. La méthode par

contraintes (illustrée ici par GSA) est presque aussi efficace, mais l'information est à peu près aussi difficile à extraire que pour les réseaux neuronaux. Parmi les méthodes efficaces, il reste les tables de décision (voir [Quinqueton, 1983] pour ces outils forts prometteurs et peu détaillés ici), qui ont l'avantage d'être très intelligibles (il s'agit d'élaborer une table qui à l'évaluation de différentes conjonctions de littéraux fait correspondre une classe; les conjonctions de littéraux étant déterminés par des heuristiques basées sur des méthodes statistiques).
Après ce peloton de tête en matière d'efficacité, les méthodes basées sur l'espace des versions, adaptées aux critères numériques, donnent des résultats convenables, moins intelligibles toutefois que les tables de décision. La méthode des monômes vides, avec un même ordre de grandeur d'efficacité, donne des connaissances très très peu intelligibles. La méthode de l'étoile est assez proche en efficacité de l'espace des versions et moins intelligible (l'intelligibilité se mesure généralement à partir de la taille des structures générées, modulées selon que les structures soient purement numériques ou logiques). Enfin encore un peu plus loin les méthodes purement statistiques sont encore un peu moins efficaces (dans le cas où l'on ne suppose pas certaines connaissances statistiques sur les données) et complètement inintelligibles: il s'agit par exemple de la méthode des $k$ plus proches voisins (voir [Covert et al, 1967]), ou de la méthode des noyaux de Parzen (voir [Parzen, 1962]). Enfin en queue de peloton, les arbres de décision, très faciles à interpréter, mais vraiment faibles au niveau efficacité, même dans le cas d'arbres flous (pourtant largement supérieurs aux arbres traditionnels avec données numériques discrétisées).
Les résultats rapidement résumés ici ne sont valables que dans le cadre de données numériques, et certaines méthodes présentent beaucoup d'intérêt pour des données qualitatives. En outre je ne commente pas ici la vitesse de l'apprentissage, qui est un des points forts par exemple des arbres de décision flous (du moins lorsque la dimension est réduite, car la complexité des arbres de décision est souvent quadratique en la dimension - notons qu'un arbre de décision ne tire pas commodément parti du fait qu'une matrice d'exemples soit creuse, contrairement à une rétropropagation ou une Support Vector Machine).

# Chapitre 14

# Prétraitement et apprentissage non-supervisé : présentation de quelques algorithmes

**Résumé**

Ce chapitre traite de quelques plus ou moins usuelles techniques d'apprentissage non-supervisé ou de prétraitement. Ceci inclut le traitement des données manquantes, la réduction de dimension, la classification non-supervisée, la discrétisation. Des résultats théoriques sont illustrés par des applications concrètes sur quelques benchmarks représentatifs de quelques cas usuels. Les implémentations sont faites en octave (clone gratuit de matlab) et peuvent être fournies sur demande email.

# Table des matières



## 14.1 Introduction

Ce travail a bénéficié de discussions avec A. Elisseeff et G. Gavin.

Il y a deux sortes d'apprentissage. Premièrement, l'apprentissage **supervisé**, le plus usuel et intuitif : des couples $(x_i, y_i) \in Input \times Output$ sont fournis, pour $i \in [1, n]$, et, étant donné $x \in Input$, le problème est la recherche de $y \in Output$ tel que $(x, y)$ semble une généralisation naturelle de $(x_i, y_i)_{i\in[1,n]}$. Il y a une sorte de "professeur", qui donne des exemples de $y_i$ comme images des $x_i$, et le problème est d'apprendre les relations sous-jacentes qui donnent $y_i$, $x_i$ étant donné. Si $Output$ est discret (usuellement, fini), alors cet apprentissage est appelé "classification supervisée", alors que si $Output$ est continu (par exemple un sous-ensemble ouvert de $\mathbb{R}^n$) il est qualifié de "régression supervisée".

D'autre part, des techniques **non-supervisées** travaillent sur une famille finie $(x_i)_{i\in[1,n]}$, où $x_i \in Input$, sans information de classe dessus. Comme nous le verrons, on peut distinguer en fait deux sortes d'apprentissage non-supervisé. Le premier est la **réduction de dimension**, le second est la **classification non-supervisée**. La réduction de dimension est la construction d'une application $f$ de $Input$ vers un espace $Output$ (parfois fourni *a priori* et parfois non), tel qu'avec $y_i = f(x_i)$, les $y_i$ ont des propriétés topologiques proches des propriétés équivalentes des $x_i$, et avec $Output$ parfois beaucoup plus simple et facile à manier que $Intput$. Par exemple si $Input$ est un sous-ensemble de $\mathbb{R}^d$, $Output$ pourrait être un sous-ensemble de $\mathbb{R}^{d'}$ avec $d' << d$. La préservation des propriétés topologiques de l'ensemble peut par exemple signifier que pour tous $i$ et $j$, $d(x_i, x_j) \simeq d(y_i, y_j)$. On pourrait par exemple minimiser, comme dans l'algorithme ACC, la somme $\sum_{i,j\in[1,n]^2} (d(x_i, x_j) - d(y_i, y_j))^2$.

La classification non-supervisée est le choix d'une partition $P$ de $[1, n]$, ie une famille finie de sous-ensembles non-vides de $[1, n]$, disjoints, et dont l'union est égale à $[1, n]$. Le choix doit être tel que pour $(i, j) \in E^2$ et $E \in P$, $x_i$ est proche de $x_j$, et pour $i \in E_1$ et $j \in E_2$, $E_1 \neq E_2$, $x_i$ est vraiment différent de $x_j$.

Réduire la dimension est une partie d'un problème plus général : transformer les données en une forme plus maniable. Ceci est appelé **prétraitement**. Une autre forme de prétraitement est le traitement du **bruit**, ce qui est souvent inclus dans la réduction de dimension ; on peut simplement considérer une réduction de dimension $x \mapsto f(x)$, $g$ une pseudo-inverse de $f$, et considérer que la version débruitée de $x$ est $g(f(x))$. Nous ne travaillerons plus sur ceci ; mais on peut noter que dans le cas du diabolo, $f$ est la partie gauche du réseau, et $g$ la partie droite. Dans le cas de Kohonen, $f$ est évidemment la fonction calculée par la carte, et $g$ est la fonction qui associé à un point $y$ dans l'espace de petite dimension, la coordonnée du neurone le plus proche dans l'espace d'entrée ; éventuellement la moyenne pondérée des coordonnées des $k$ plus proches voisins de $y$.

La dernière partie est le traitement des **données manquantes**.

Je résume par le suite quelques techniques non-supervisées. Le lecteur est supposé capable de comprendre quelques techniques usuelles, comme la descente de gradient (voir par exemple [Bishop, 1995, p 263]).

## 14.2 Réduction de dimension / reformulation des données

### 14.2.1 Pourquoi réduire la dimension ?

Comme expliqué dans [Bishop, 1995, p7-9], une haute dimension de l'espace d'entrée implique l'overfitting, pourvu que les exemples ne soient pas très nombreux. Une autre conséquence est un apprentissage lent. Aussi, un prétraitement est requis pour travailler avec des données en plus basse dimension.

### 14.2.2 Les différents algorithmes

#### Les cartes de Kohonen

Une carte de Kohonen est une application d'un ensemble vers une carte topolographique. Le problème est la compréhension de "topographique". Quel est le critère de choix entre différentes applications possibles ?

Par exemple, si les données sont les nombres de 1 à 15, et si on veut les placer sur un segment, une application naturelle serait :

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ceci est "mieux" que :

1 6 3 4 5 2 7 8 14 15 11 12 13 9 10

Le principe de préservation des distances peut justifier cette comparaison.

#### Fondations biologiques

Les cartes de Kohonen sont basées sur des principes biologiques. Le cerveau, plus précisément le cortex cérébral, contient des aires sur lesquelles sont mappées différentes sortes d'entrées : cortex visuel, cortex auditif, cortex somatosensitif. Quand un stimulus visuel est présenté, quelques neurones sont activés dans le cortex visuel ; et si on présente un stimulus $A$, proche d'un stimulus $B$, et regardons les neurones activés $N_A$ et $N_B$, on constate que plus $A$ est proche de $B$, plus $N_A$ est proche de $N_B$.

#### Algorithme basique

• On doit choisir *a priori* la dimension $d$ de la représentation, et un nombre $p$ de **neurones** $N_1$, $N_2$, ..., $N_p$. Les neurones sont connectés par une grille, de dimension $d$. Par exemple, avec $d = 2$, une représentation possible est donnée par la figure 14.2.2 (droite).

• A chaque stade $0 < t < t_{max}$ ($t \in \mathbb{N}$) de l'apprentissage, chaque neurone $N_i$ a une représentation $y_i$ dans *Input*.

• A chaque stade, chaque exemple $x_i$ est présenté une fois au réseau de neurones.

• Alors, on calcule $j \in [1,p]$ tel que $d(x_i, y_j)$ soit minimisé.

• Alors, un voisinage $V$ de $j$ est déterminé parmi $[1,p]$, par rapport à la topologie du réseau. Par exemple, $V$ pourrait être l'ensemble des $k \in [1,p]$ tels que $D(j,k) < \max(1, max_{dist} \times (1 - 4 \times t/t_{max}))$, avec $D$ la distance de Manhattan sur $[1,p]$, équipée d'une grille comme définie précédemment, comme montré sur la figure 14.2.2 (droite). [1]

• Pour chaque $k \in V$, $y_k(t+1) = y_k(t) + \eta(t)(y_j(t) - y_k(t))$, et pour tout $k \notin V$ $y_k(t+1) = y_k(t)$. Par exemple $\eta(t) = \eta \times (t_max - t)/t_max$.

Quand certaines données sont reformulées par une application $f$ donnée par des cartes de Kohonen (ou d'autres techniques) il est intéressant d'être capable de calculer $f(x)$ pour d'autres valeurs de $x$ que celles utilisées pour l'apprentissage. On pourrait simplement considérer que $f(x)$ est égal au numéro du neurone qui a la représentation la plus proche de $x$. Avec une telle méthode, comme le nombre de neurones est fini, la carte de Kohonen peut être considérée comme un algorithme de classification non-supervisée. Quand on peut faire une réduction de dimension ou une reformulation, sans discrétisation, on peut utiliser les algorithmes de $k$-plus proches voisins pour calculer l'image d'un $x$, étant donné une suite finie de $x_i$ et $y_i$.

**Remarques et améliorations possibles**

On peut noter tout d'abord deux inconvénients des cartes de Kohonen :

• L'utilisateur doit spécifier la dimension de *Output* ; et que le coût en temps de calcul est décourageant pour une dimension plus grande que 2 ou 3.

• Considéré comme un outil de classification non-supervisée, la carte de Kohonen a un nombre fixé de classes, choisies *a priori* par l'utilisateur.

La mise-à-jour $\forall k \in V$ $y_k(t+1) = y_k(t) + \eta(t)(y_j(t) - y_k(t))$ et $\forall k \notin V y_k(t+1) = y_k(t)$ pourrait être remplacée par $\forall k y_k(t+1) = y_k(t) + \eta(t) MH(d(j,k)/t)(y_j(t) - y_k(t))$, avec $MH$ la fonction du "chapeau mexicain", comme montré en figure 14.2.2 (gauche). Pour augmenter la vitesse, il est beaucoup mieux d'utiliser une fonction analogue à $MH$, nulle pour $x > x_0$. Intuitivement, les points très proches se rapprochent, les points suivants sont repoussés, et les autres sont à peu près oubliés.

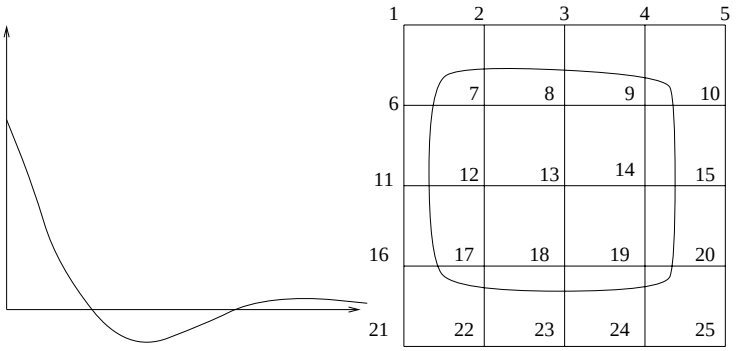



FIG. 14.1 – La fonction du chapeau mexicain (gauche), le voisinage $V$ du neurone 13 de rayon 1 (droite).

**Résultats théoriques**

Les résultats théoriques à propos des cartes de Kohonen sont prouvés seulement sous des conditions restrictives gênantes, pour autant que je sache. On peut lire par exemple [Cottrel et al, 1987, Ritter et al, 1988] et [Benaim et al, 1998].

**L'algorithme ACC**

Pour décrire cet algorithme, Je vais utiliser les notations de [Thiria et al, 1997, p164-170].

[1] On doit prendre garde (ici par la fonction $\max(1,.)$) à ce que le rayon est toujours $\geq 1$ - sinon il n'y a pas d'influence locale.

**L'algorithme**

L'algorithme d'Analyse en Composantes Curvilignes (ACC) minimise l'énergie $E = \sum_{i,j} E_{i,j}$, avec $E_{i,j} = (X_{i,j} - Y_{i,j}) f(Y_{i,j})$, où $X_{i,j} = d(x_i, x_j)$ et $Y_{i,j} = d(y_i, y_i)$. La fonction $f$ pourrait par exemple être constante : il s'agit d'une pondération. Usuellement, $f$ ressemble par exemple à $t \mapsto \frac{1}{1+exp(t-\lambda)}$. Ceci signifie que si $\lambda$ est petit, l'algorithme va seulement se préoccuper des petites distances ; alors que si $\lambda$ est gros, de grosses distances sont influentes.

La minimisation de $E$ sera faite par une descente de gradient. Ceci signifie que l'on va modifier $y_i$ par $y_i(t+1) = y_i(t) - \eta(t) \nabla_{y_i} E$. Afin de clarifier, on va noter $\nabla_i E$ le gradient de $E$ par rapport à $y_i$.

Ainsi on doit calculer $\nabla_i E$. Ceci est fait simplement ; le résultat est

$$\nabla_i(E) = -\sum_{j \neq i} (\frac{X_{i,j}}{Y_{i,j}} - 1) \times (2F(Y_{i,j}) - (X_{i,j} - Y_{i,j}) F'(Y_{i,j})(y_i - y_j)$$

En fait, comme expliqué dans [Thiria et al, 1997, p166], on peut faire mieux que $y_i(t+1) = y_i(t) - \eta(t) \nabla_{y_i} E$, afin d'éviter les minima locaux. Le truc est de modifier $y_j$ pour $j \neq i$ au lieu de $y_i$. La formule est $y_j(t+1) = y_j(t) + \eta(t) \nabla_i E_{i,j}$.

Le paramètre $\lambda$ peut être choisi décroissant linéairement de la plus grande distance vers 0 par exemple, afin que l'algorithme commence par se préoccuper des grandes distances, et alors s'affine pour la représentation sur des petites distances.

**Remarques et améliorations possibles**

On peut noter que l'utilisateur a à choisir la dimension de *Output* lui-même. [Thiria et al, 1997] propose de tracer la courbe des couples $(X_{i,j}, Y_{i,j})$ pour évaluer l'efficacity de l'algorithme, et de conduire le choix de $\lambda$. On pourrait par exemple faire quelques expérimentations différentes et choisir la meilleure dimension en fonction d'un critère combinant la réduction de dimension et la préservation des distances.

**L'apprentissage par kernel-ACP**

Cette partie est basée sur l'article [Scholkopf et al, 1998].

**Principes**

L'ACP à noyau, ou kernel-ACP, est un analogue de l'ACP (Analyse en Composantes Principales), tirant parti des méthodes à noyaux. [2] Un noyau $(x, y) \mapsto K(x, y)$ est une fonction telle que $K(x, y)$ est égal au produit scalaire de $\phi(x)$ et $\phi(y)$ dans un autre espace, appelé **feature** space. On peut noter que l'ACP nécessite seulement le produit scalaire des points ; aussi on peut considérer $K(.,.)$ au lieu d'un produit scalaire usuel, et le résultat va être une ACP dans l'espace des caractéristiques. Dans certains cas (noyaux gaussian à base radiale) la dimension du feature space est infinie ; dans d'autres il est de dimension finie (noyaux linéaires ou polynomiaux).

**L'algorithme**

On considère un ensemble $x_1, \ldots, x_n$ d'exemples. La fonction $K$ peut être par exemple une des suivantes :

$$K(x_1, x_2) = < x_1, x_2 >$$

$$K(x_1, x_2) = exp(-\frac{\| x_1 - x_2 \|}{\sigma^2}) \quad \text{(Noyau RBF gaussien)}$$

$$K(x_1, x_2) = tanh(a < x_1 | x_2 > + b) \quad \text{(noyau sigmoïde)}$$

$$K(x_1, x_2) = (< x_1 | x_2 > + 1)^d \quad \text{(noyau polynomial)}$$

$$K(x_1, x_2) = \Pi_{i \in [1,p]} \beta - spline((\frac{x_1 - x_2}{s})_i, n) \quad \text{(noyau } \beta\text{-spline)}$$

[2] D'autres méthodes à noyaux incluent les Support Vector Machines.

$$K(x_1, x_2) = \pi_{i \in [1,p]} \frac{sin(\frac{(2n+1)(x_1 - x_2)}{s})}{sin(\frac{(x_1 - x_2)}{s})} \quad \text{(Noyau de Dirichlet)}$$

(où $< x|y >$ est le produit scalaire, et $a$, $b$, $s$, $\sigma$ sont des nombres réels, $d$ est un entier $> 0$, $n$ est un entier impair, $p$ tel que $Input = \mathbb{R}^p$)

• Remplissage de la matrice $M$ :

$$M_{i,j} = K(x_i, x_j)$$

(la matrice a taille $(n, n)$, avec $n$ le nombre d'exemples ; comme celle-ci peut être très grande, on pourrait travailler sur un sous-ensemble des exemples - l'algorithme étant capable de reformuler d'autres données tout aussi bien, comme expliqué plus bas)

• Normalisation de $M$ ; $M_{nouveau}(i,j) = M(i,j) - \overline{M(i,.)} - \overline{M(.,j)} + \overline{M(.,.)}$. Par souci de clarté, j'utiliserai $M$ pour $M_{nouveau}$.

• Calcul des vecteurs propres et valeurs propres de $M$.

• Extraction des grandes valeurs propres (les $k$ plus grandes si on veut nécessairement une analyse en dimension $k$ ; sinon l'algorithme peut choisir $k$ lui-même, comme expliqué en partie A). Notons $W_i = \frac{Z_i}{\sqrt{\lambda_i}}$, avec $\lambda_i$ la $i^e$ valeur propre (en ordre décroissant de valeur absolue), et $Z_i$ le vecteur propre associé.

• Soit $P$ la matrice qui a pour lignes les $W_i$, pour $i \in [1, k]$.

• Remplir la matrice $\mathcal{M}$ de même que $M$, pour toutes les données à reformuler (nous avons annoncé que pour $M$ on pouvait considérer seulement un sous-ensemble des données)

$$\mathcal{M}_{i,j} = K(x_i, x_j)$$

• Normalisation de $\mathcal{M}$ ; $\mathcal{M}_{nouveau}(i,j) = \mathcal{M}(i,j) - \overline{\mathcal{M}(i,.)} - \overline{\mathcal{M}(.,j)} + \overline{\mathcal{M}(.,.)}$. Par souci de clarté, j'utilise $\mathcal{M}$ pour $\mathcal{M}_{nouveau}$.

• Construction de $y_i$ :

$$Y = \mathcal{M} \times P$$

Alors, $Y$ a $y_1$ pour première ligne, $y_2$ pour seconde ligne, $y_3$ pour troisième ligne, etc.

**Réseau en diabolo**

Le réseau en diabolo a déjà été évoqué dans le chapitre sur la reconnaissance de données à une composante de temps.

**Le principe**

Depuis son apparition en 1986, la rétropropagation a été largement étudiée. Utiliser un outil si connu et efficace pour la réduction de dimension était tentant. L'idée était qu'une bonne représentation des données étaient une représentations telle que les vieilles données pouvaient être retrouvées à partir de la nouvelle formulation. En quelque sorte, les $y_i$ devaient contenir assez d'information pour "revenir" aux $x_i$. Le principe était d'utiliser un réseau avec une architecture comme celle montrée en figure 14.2. Un apprentissage supervisé est fait avec les $x_i$ pour entrées...et pour sorties. Quand l'apprentissage est bien fait, on peut simplement couper la partie droite du diabolo, après la couche du milieu. Les sorties des neurones de la troisième couche (avec 3 neurones) quand $x_i$ est présenté à la couche d'entrée forment $y_i$. La dimension a été réduite de 5 à 3. Bien sûr, on pourrait faire de même avec toute dimension. Il a été montré dans [Bourlard et al, 1988] que l'on avait à mettre une couche entre la couche d'entrée et la couche du milieu, et entre la couche du milieu et la couche de sortie ; sinon la transformation est une simple ACP.

**Remarques et améliorations possibles**

Comme les réseaux à rétropropagation, le diabolo souffre de quelques inconvénients :

• convergence lente

• minima locaux

Mais il peut être corrigé de la même façon que les réseaux à rétropropagation :

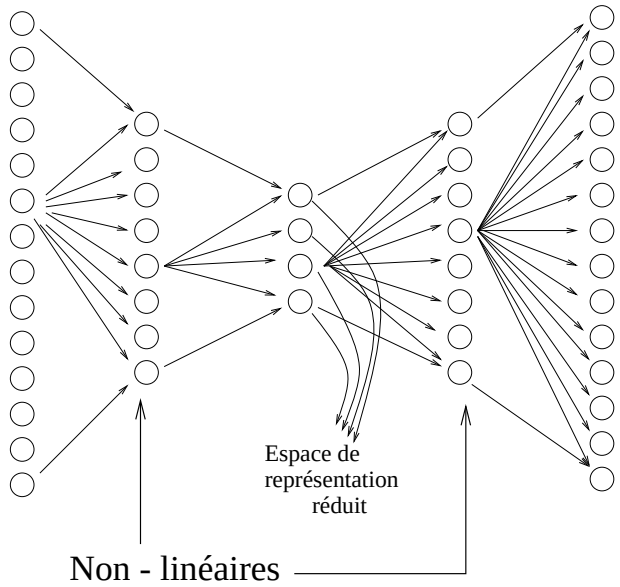



FIG. 14.2 – Chaque couche est entièrement connectée à la suivante (les connections ne sont pas toutes dessinées, par souci de clarté.

• gradient conjugué, et de nombreuses autres améliorations pour la vitesse de convergence. Voir par exemple [Bishop, 1995, p253-292] pour de nombreuses techniques.

• recuit simulé, qui avec probabilité 1 fournit une convergence vers la meilleure solution (mais pas forcément très vite en pratique).

• Algorithmes génétiques, efficaces dans certains cas (très grand nombre de neurones notamment paraît-il).

Un autre inconvénient du diabolo est que l'utilisateur a à définir lui-même la dimension de la représentation. Une solution consisterait en un **pruning** ; on appelle pruning la suppression de certaines unités ou connections. On peut lire [Bishop, 1995] pour des détails sur ces méthodes. Ainsi, on peut entrainer un diabolo avec le même nombre de neurones dans chaque couche, et alors faire un pruning pour réduire la dimension autant que possible.

On peut enfin noter que le diabolo peut aussi se voir comme un outil de débruitage. On peut alors utiliser non pas le réseau réduit à sa moitié gauche, mais le réseau complet ; la version débruitée d'un point donnée est la réponse fournie en sortie.

### 14.2.3 Clustering sur les coordonnées avant la réduction de dimension

Pour augmenter la vitesse d'un algorithme qui a une grande complexité en la dimension, on peut grouper les coordonnées en clusters, et alors travailler sur chaque cluster indépendamment. Ceci peut être fait par un simple algorithme de clustering comme celui décrit en partie 14.3.2, pouru que l'on définisse une distance entre les coordonnées, par exemple $d(i,j) = 1 - |cor((x_.)_i, (x_.)_j)|$ avec $cor(a,b)$ la corrélation entre $a$ et $b$, ou $d(i,j) = 1/|cor((x_.)_i, (x_.)_j)|$.

### 14.2.4 Conclusion

Pour autant que je sache, il n'y a absolument pas consensus à propos de quel algorithme devrait être utilisé pour réduire la dimension. Mon point de vue est que les cartes de Kohonen sont les plus "biologiques" mais souffrent de problèmes concrets pour reformuler en grande dimension, CCA le plus efficace pour préserver les distances (par définition !), l'ACP à noyaux semble prometteuse du point de vue de l'ajout de connaissance à priori (comme une distance pertinente entre les exemples - on peut utiliser le noyau positif défini $K(x,y) =$

$\exp(-d(x,y)^2/\sigma^2)$ dès que $d$ est une distance), et le diabolo semble puissant dans les applications, grâce à l'efficacité algorithmique de la rétropropagation.

## 14.3 Classification non-supervisée

### 14.3.1 Problème

La classification non-supervisée consiste à extraire une partition "pertinente" des données en un nombre (prédéfini ou non) de classes. De nombreux critères formels peuvent être retenus pour traduire cette notion de pertinence : homogénéité, couverture par un nombre minimal de boules...

### 14.3.2 Les différents algorithmes

#### Un algorithme simple et classique de clustering

Cet algorithme est extrait de "Algorithmic Learning", d'Alan Hutchinson, Clarendon Press 1994. Il est intéressant en tant qu'exemple simple d'algorithme de clustering.

**L'algorithme**

On suppose que $n$ exemples, $x_1$, $x_2$, $x_3$, ..., $x_n$ sont donnés.

- Calculer la liste des distances entre les $x_i$.
- Classer cette liste en ordre croissant ; appelons $L$ la liste résultante.
- Création de $n$ clusters, chacun contenant seulement un exemple.
- Pour chaque $l$ élément de $L$, avec $i$ et $j$ tels que $d(x_i, x_j) = l$ :
- • Si $x_i$ et $x_j$ sont dans des clusters différents, unifier ces clusters.
- • Garder en mémoire que cette union a été faite à la distance $l$.
- Si un nombre précis de clusters est requis, on peut arrêter l'algorithme précédent quand le bon nombre de clusters est atteint. Sinon, appelons $T$ le tableau des distances correspondant à des unions successives. Notons $\Delta T$ la différence entre des éléments successifs de $T$, $std$ la déviation standard de $\Delta T$, et $m$ la moyenne de $\Delta T$. Alors on considère le premier indice $k$ de $\Delta T$ tel que $\Delta T(k) > mean + std$. Seuls les $k+1$ premières unions sont faites.

**Remarques**

- La complexité de cet algorithme est :
  - $O(n^2 ln(n))$ en temps
  - $O(n^2)$ en espace

  Ceci peut être prouvé aisément sur l'algorithme donnée en appendice. Pour l'espace, on a seulement à garder en mémoire la liste des distances ($O(n^2)$), la partition (un nuémro de cluster par exemple, $O(n)$), et la liste des éléments de chaque cluster ($O(n)$). Pour le temps, les tris nécessitent $O(n^2 \ln(n^2))$, qui est $O(n^2 \ln(n))$, et les opérations suivantes nécessitent au plus $n^2$ opérations (ceci peut être facilement vérifié).
- Le tri ou même la simple énumération des distances pourrait rapidement devenir trop lente of consommatrice de mémoire.
- Quand les données comporte des valeurs manquantes, on peut utiliser une autre distance ; par exemple, $d(x,y) = \frac{mean(|x_i - y_i|^p)^{1/p}}{sqrt(N_{x,y})}$ avec $N_{x,y}$ le nombre de coordonnées définies à la fois dans $x$ et $y$.
- Il existe des généralisations de cet algorithme ; le nom générique est "classification ascendante hiérarchique" (CAH), et on peut trouver plus de précisions dans [Thiria et al, 1997]. Dans les expérimentations plus loin, l'algorithme simple de classification ascendante hiérarchique est l'algorithme ci-dessus basé sur l'unification à chaque étape des deux classes les plus proches, la distance entre deux classes étant la plus petite distance entre deux points de ces classes, l'algorithme 2 sera celui basé sur la distance du lien maximum (la distance entre deux classes est le max des distances entre deux points de ces classes), et l'algorithme 3 sera celui basé sur l'augmentation de l'inertie (la distance est l'indice de Ward entre les classes).

### Les $k$-means

#### Le principe

Le principe de l'algorithme des $K$-means est que chaque élément d'une partition devrait être plus près du centre de sa classe que de tout autre centre de classe. Ainsi une partition arbitraire est choisie, et ensuite les exemples sont examinés un par un ; dès qu'un exemple est plus près du centre d'une autre classe que de son centre, il est déplacé vers cette classe.

#### L'algorithme

Supposons qu'en veut trouver une partition en $N$ classes $C_1$, ..., $C_N$ de $x_1$, ..., $x_n$.

- Tout d'abord, pour tout $i \in [1, n]$, on met $x_i$ dans la classe $i \ mod \ N + 1$.
- Tant que ((c'est le premier essai) ou (quelque chose a changé dans la partition))
- • Pour chaque exemple $x_i$, $i \in [1, n]$ :
- • • Trouver la classe $C_j$ telle que la moyenne des éléments de $C_j$ est la plus proche de $x_i$.
- • • Si $x_i \notin C_j$, alors met $x_i$ dans $C_j$.
- • Fin de la boucle "pour"
- Fin de la boucle "tant que"

#### Remarques et améliorations possibles

- Cet algorithme est très rapide ; pour autant que je sache il n'y a pas de borne de complexité non-triviale. La complexité en espace est $O(n + N)$.
- L'algorithme génère une séparation qui est un Voronoi. Chaque classe a un centre, et ses points sont ceux dont ce centre est le plus proche. Ainsi par exemple avec une séparation en deux classes, le deux classes choisies sont nécessairement séparées linéairement.
- Le nombre de classes doit être fixé *a priori*.
- Une variante des $k$-means est l'algorithme de **transfert**. La seule différence est que la distance $d(x_i, w_j)$ entre l'exemple $x_i$ et le centre $w_j$ de la classe $j$ est remplacée par $\sqrt{(\frac{Card(j)}{1+Card(j)})} \times d(x_i, w_j)$ avec $Card(j)$ le cardinal de la classe $j$. Avec ceci, les grosses classes sont encouragées à grossir encore.
- L'algorithme peut être utilisé pour des données avec des données manquantes ; la façon la plus simple de le faire est de considérer pour moyenne d'un ensemble de points a données manquantes le vecteur dont chaque coordonnée est la moyenne de la même coordonnée pour les points où elle est définie. Ceci peut être fait de manière incrémentale, comme les $K$-means usuels.
- Lorsque les données ne sont pas iid, il peut être *très* efficace de changer aléatoirement l'ordre des points pour éviter des effets de biais. Ceci est lié à l'implémentation mais réellement efficace parfois.

### Apprentissage non-supervisé à rétropropagation

#### L'algorithme

Pour autant que je sache, ceci est un nouvel algorithme pour l'apprentissage non-supervisé. Comme la rétropropagation a été intensivement étudiée et optimisée, il était tentant de l'utiliser pour une tâche non supervisée. Je propose l'algorithme suivant :

- Initialiser au hasard un réseau avec $d$ entrées (avec $d$ la dimension de $Input$), et le nombre de couches et de neurones que vous voulez. La seule restriction est qu'experimentalement le nombre de sorties doit être suffisamment grand ; une couche avec $5d$ neurones et une couche de sortie avec $5d$ neurones semble correct.
- Répétez
- • Pour chaque exemple $x_i$
- • • Calculer $y_i$, l'image de $x_i$ par le réseau.
- • • Avec $u_i = sign(y_i)$, $(u_i)_j = 1$ si $(y_i)_j \geq 0$, $= -1$ si $(y_i)_j < 0$, faire une rétropropagation de l'erreur avec $u_i$ comme sortie désirée.
- • • Mettre à jour les poids, décroitre le paramètre de vitesse d'apprentissage (comme classiquement en descente de gradient)
- Jusqu'à ce que les $u_i$ ne bougent plus, ou après un nombre donné de stades.

Les exemples sont alors réarrangés en groupes selon leurs images.

**Remarques**

• Il y a des hyperparamètres : le nombre de couches, le nombre de neurones. D'autre part, le résultat semble indépendant de ces paramètres pourvu qu'il y a au moins deux couches et un nombre suffisamment gros de neurones.

• Le nombre de classes est déterminé automatiquement.

• Un algorithme de classification pour des nouvelles entrées est immédiatement déduit ; on peut simpler rentrer $x$ dans le réseau. On pourrait supprimer les neurones de sorties redondant pour simplifier la fonction obtenue.

• Un état stable de l'algorithme d'apprentissage est l'état où chaque neurone répond toujours 1. Il semble donc difficile de trouver des fondations théoriques à cet algorithme. Le fait est, qu'en pratique, on peut faire des centaines de tests sans jamais tomber dans ce cas particulier.

#### Apprentissage Hebbien non-supervisé

**L'algorithme**

L'algorithme est vraiment simple à implémenter. L'architecture est donnée *a priori*. Il peut y avoir autant de couches que l'on veut, et autant de neurones. L'algorithme est le suivant :

• A chaque stade

• • Pour chaque exemple $x_i$

• • • Présentez $x_i$ au réseau.

• • • Calculer toutes les sorties des neurones. Ce sont des neurones à seuil : la sortie est 1 si la somme pondérée des entrées est $\geq 0$, et $-1$ sinon.

• • • Si $w$ est le poids entre le neurone $i$ et le neurone $j$, alors :

• • • • Si $i$ et $j$ ont la même sortie, alors augmenter $w$ (du paramètre d'apprentissage)

• • • • sinon décroitre $w$ (du paramètre d'apprentissage)

• • Décroitre la vitesse d'apprentissage (linéairement vers 0 par exemple)

**Remarques**

• La reconnaissance est un simple processus feedforward. Les mises-à-jour des poids peuvent être faits localement, une fois la sortie de chaque neurone calculée pour un exemple donné.

• Ce processus non supervisé est exactement similaire à $p$ processings nonsupervisés, avec $p$ le nombre de couches. On pourrait en fait traiter les différentes couches successivement.

#### Algorithmes de réduction de dimension

Les cartes de Kohonen peuvent directement être considérées comme un outil de classification non supervisée. Par ailleurs, il peut arriver que la formulation des données soit très compliquée, et qu'une reformulation, utilisant une des techniques de la section 14.2, puisse augmenter largement les performances de l'algorithme (notamment pour les algorithms très biaisés, comme les $K$-means avec $K$-petits).

### 14.3.3 Expérimentations pratiques

#### Sans données manquantes

Les graphes qui suivent montrent des expérimentations avec une distribution en deux classes, qui sont des gaussiennes normalisées, centrées en $(-1, 1)$ et $(1, 1)$. La deuxième gaussienne est quatre fois plus dense que la première.

L'algorithme de clustering simple a choisi lui-même le nombre de classes. On peut noter qu'après quelques expérimentations il est arrivés parfois que l'algorithme mettent tous les points dans un seul cluster. Pour les $K$-means et l'algorithme de transfert le nombre de clusters a été choisi *a priori*, et n'a pas été choisi par l'algorithme. Seuls les résultats des $k$-means sont donnés ici, comme l'algorithme de transfert a donné exactement la même réponse. La rétropropagation et l'algorithme Hebbien ont choisi elles-même le nombre de clusters. Seuls les résultats de l'algorithme Hebbien ont été donnés, comme la rétropropagation a généré exactement la même partition.

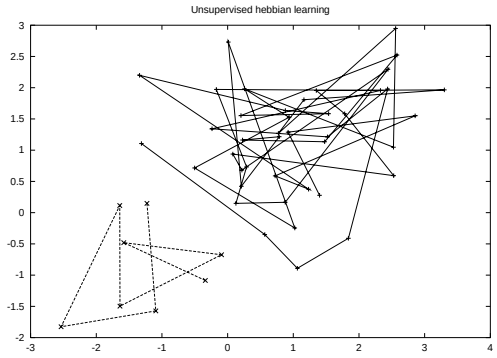
On peut noter deux résultats instructifs (qui sont intuitifs) :

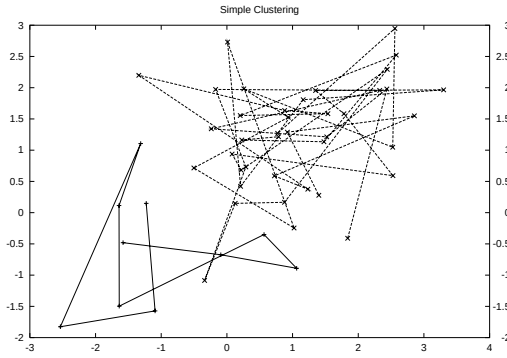
• l'algorithme de clustering simple a un comportement étrange sur un petit nombre de points.

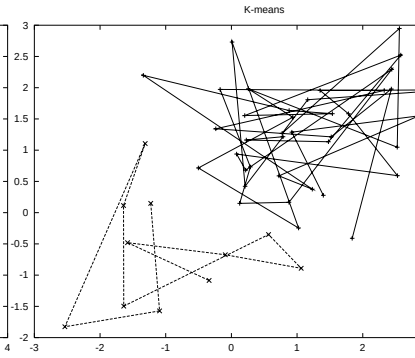
• les $k$-means ou l'algorithme de transfert mettent une séparation exactement à la même distance des moyennes des classes. Ceci n'est pas efficace ici, comme les deux distributions ont des densités très différentes. La rétropropagation ou l'apprentissage hebbien sont beaucoup mieux ici. Un calcul rapide montre que la séparation parfaite serait une ligne à distance $\frac{8-2\times ln(4)}{2\sqrt{8}}$ de $(-1,1)$ sur le segment de $(-1,-1)$ à $(1,1)$, perpendiculaire à celui-ci ; ceci est à peu près le choix de l'algorithme de rétropropagation. Ceci est montré particulièrement sur des grosses expérimentations ; la rétropropagation s'approche de plus en plus de la meilleure séparation, alors que les $k$-means restent sur l'hyperplan médian.

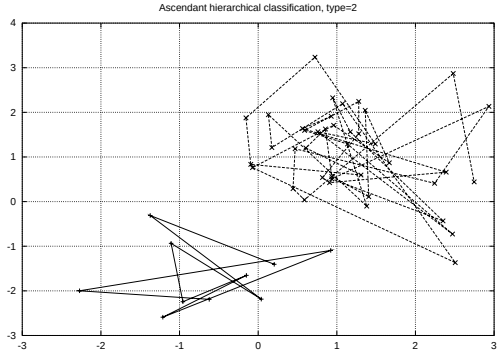
Le second type de classification hiérarchique fonctionne similairement à l'algorithme simple sur cet exemple. Le troisième type fournit des résultats étranges et irréguliers.

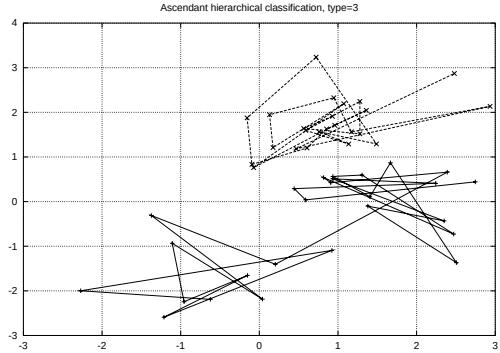
Ceci est montré sur les expérimentations qui suivent :



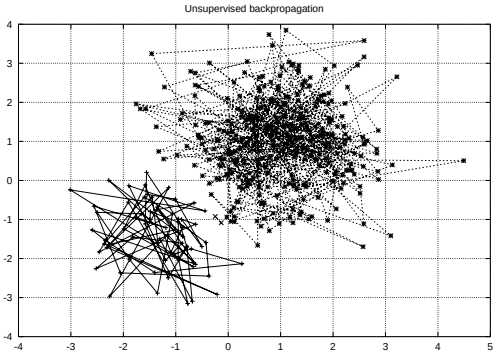



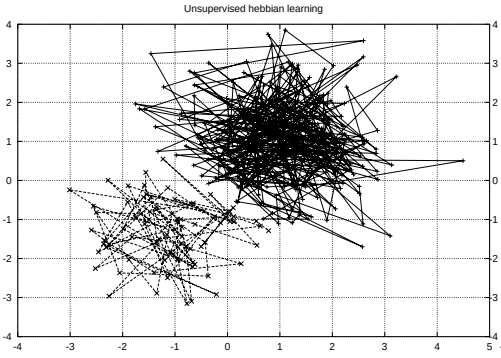



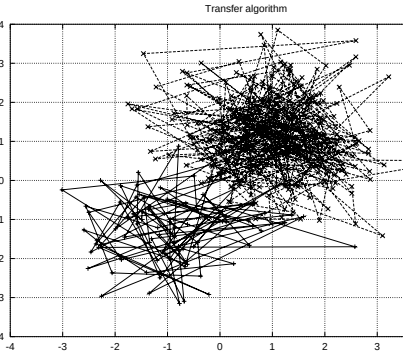



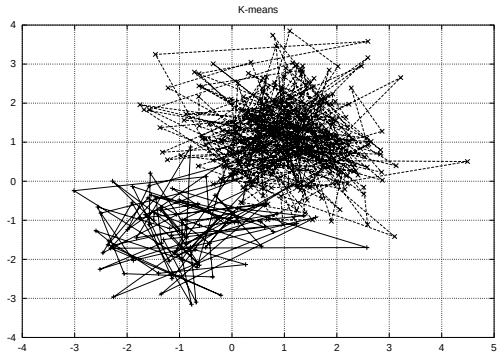



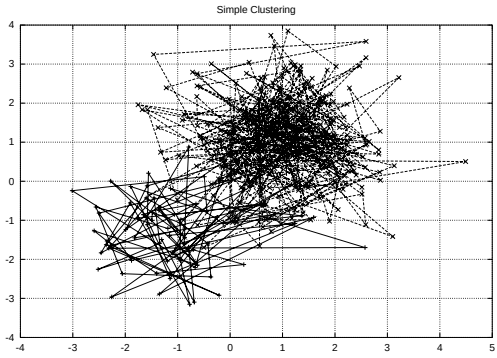

Les courbes suivantes, illustrant le cas d'un grand nombre d'exemples, sont difficiles à lire à cause des images monochromes. Les versions couleur peuvent être obtenues avec le code octave. On peut voir clairement dessus que la séparation choisie par l'algorithme des $k$-means ou l'algorithme de transfert est très proche de zéro alors que la rétropropagation non-supervisée et l'apprentissage hebbien donnent une séparation étonnamment proche de l'optimal théorique, et fournit une séparation très précisé, alors que l'algorithme de clustering simple fournit une limite très floue.



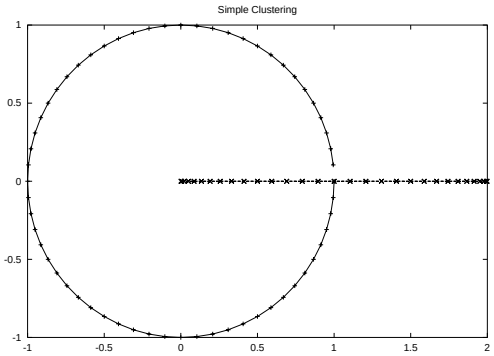



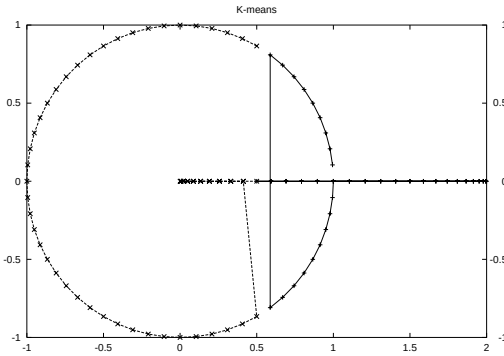



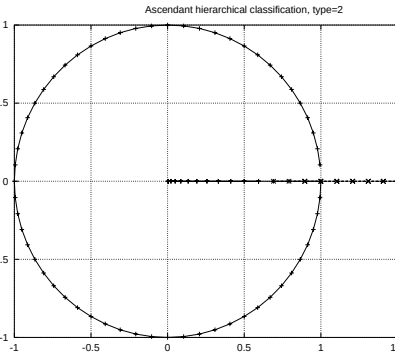


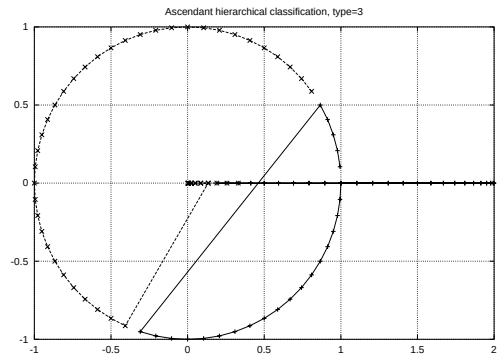




Les courbes suivantes ont été obtenues avec deux anneaux enchainés en dimension 3. L'algorithme de clustering simple est bien sûr très efficace sur ce cas particulier (les autres formes d'ACH sont moins efficaces sur ce cas particulier). Il choisit lui-même le nombre de classes. Les $k$-means trouvent une séparation linéaire, bien sûr, et ainsi ne peuvent trouver la meilleure solution ; je n'ai pas tracé la courbe fournie par l'algorithme de transfert parce qu'elle est égale à celle fournie par les $k$-means. La rétropropagation non-supervisée et l'apprentissage hebbien (avec une couche cachée de 5 neurones et une couche de sortie de 10 neurones) ont mis ensemble tous les exemples dans la même classe. Décroitre le paramètre d'apprentissage peut y remédier, mais la séparation n'a jamais été celle attendue.

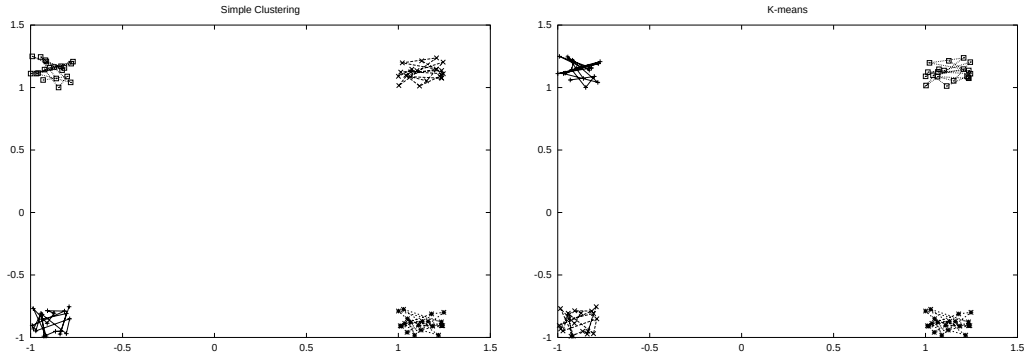



Les résultats qui suivent ont été obtenus avec des données construites en 4 distributions égales en cardinal dans quatre zones distinctes. L'algorithme de clustering simple a trouvé lui-même le nombre de classes, comme la rétropropagation et l'algorithme Hebbien. Les résultats de la rétropropagation, de l'algorithme Hebbien, de l'algorithme de rétropropagation n'ont pas été donnés, car ils étaient égaux aux autres, ie ont été une redécouverte intacte de la partition en 4 classes. L'algorithmes d'CAH l'a retrouvée aussi.



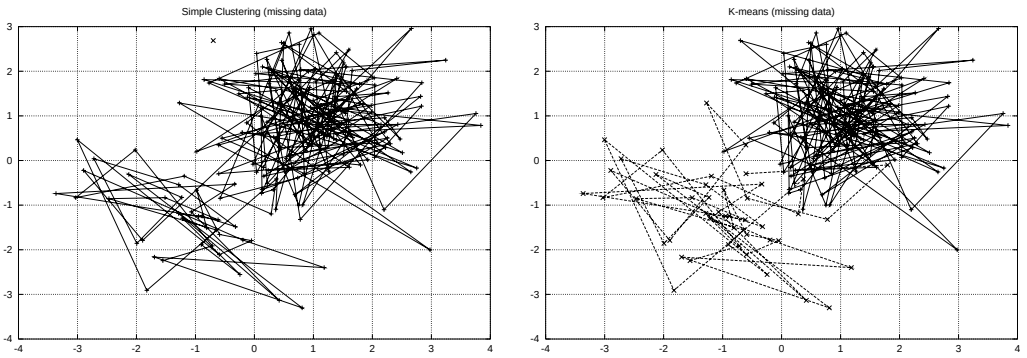

**Avec données manquantes**

Les expérimentations qui suivent ont été faites avec les même données, mais nous avons introduit des données manquantes. Chaque nombre réel a une probabilité 0.1 d'être manquante.

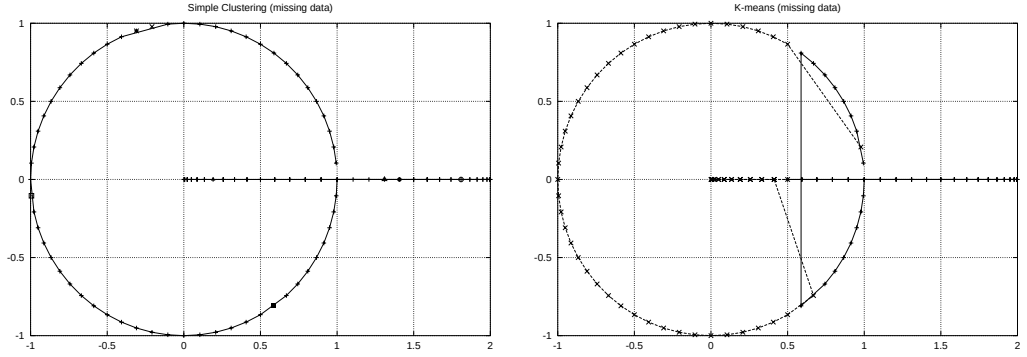
L'algorithme de clustering simple peut aisément être étendu à ce cas : on définit simplement une distance définie même avec des données manquantes. Ceci peut être fait simplement, en utilisant comme distance entre $x_1$ et $x_2$ la racine carrée de la moyenne des $(x_{1,j} - x_{2,j})^2$ pour $j$ tel que $x_{1,j}$ et $x_{2,j}$ sont tous deux définis. Les résultats sont meilleurs si on multiplie par un facteur augmentant avec le nombre de données manquantes.

L'algorithme de $k$-means est aisément traduit, aussi ; on a juste à considérer que la moyenne d'un ensemble fini de points $x_1,\ldots,x_n$ est $x_0$ tel que $x_{0,j}$ est la moyenne des $x_{i,j}$ définis pour $i \in [1,n]$.

Tout d'abord, nos résultats avec les deux gaussiennes :



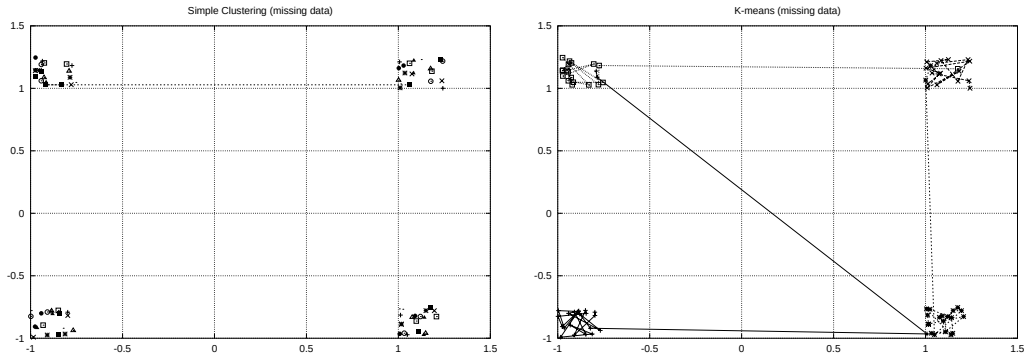

L'algorithme de clustering simple ne peut faire un bon travail. Il y a seulement deux classes et l'une d'elles est réduite à un seul point (qui n'est même pas dans la moins dense des gaussiennes). Les deux autres algorithmes CAH ne sont pas meilleurs (les résultats ne sont pas montrés ici). Les $k$-means sont à peu près stables. Considérons maintenant le problème des deux anneaux emmêlés :



L'algorithme de clustering simple est incapable de distinguer quoi que ce soit. Beaucoup plus sensible aux distances (car seulement basées sur elles !), il ne peut comprendre la structure de l'ensemble de données. Les $k$-means ont le même problème que dans le cas sans données manquantes : la séparation est presque[3] linéaire. Mais les points sont environ aussi bien classés que dans le cas sans données manquantes.

On considère maintenant les résultats avec notre ensemble jouet, avec quatre zones très bien séparées. Le clustering simple construit une partition en 78 clusters, qui n'est pas adéquate. Les trois expérimentations montrent la difficulté d'un clustering basé sur les distances. Il faut noter aussi les avantages de cette méthodes ; un clustering basé sur les distances peut être utilisé pour la réduction de dimension comme expliqué en partie 14.2.3, alors que les $k$-means non.

Sur ces données jouet, l'algorithme des $K$-means place chaque point complet dans le bon cluster, et les points pour lesquels une coordonnée manque sont placés dans l'un des clusters logiques (pour lesquels l'autre coordonnée est cohérente).




### 14.3.4 Conclusion

A nouveau, il n'y a pas de solution unique pour toute tâche non-supervisée. L'algorithme simple semble naïf mais il peut travailler sur de simples distances et il conduit à des résultats très clairs et riches de sens ; par exemple, il peut fournir l'histoire des langues avec leurs sépérations. Les $k$-means sont les plus couramment utilisés, mais certains inconvénients sont flagrants, tels les cas dans lesquels deux classes ont le même centre d'inertie. Les deux algorithmes expérimentaux que je définis ci-dessus (avec des réseaux multi-couches) semblent environ équivalents. Les algorithmes de réduction de dimension ont les mêmes avantages et inconvénients que lorsqu'utilisés en tant que réducteur de dimension.

## 14.4 Remplacement des données manquantes

### 14.4.1 Présentation du problème

Il arrive souvent dans des données réelles que quelques coordonnées de quelques exemples manquent. Par exemple, si les données sont les mesures de température dans cinq points différents de Paris, on peut imaginer que parfois un des capteurs est hors service. Aussi il y aura un ou plusieurs $x_i$ pour lesquels une ou plusieurs coordonnées sont manquantes. La raison pourrait être soit dépendante ou indépendante des points. Je ne traiterai pas le cas dans lequel elle est dépendante. Dans le cas d'indépendance, une solution simple est la suppression des $x_i$ incomplets. Mais il arrive souvent dans les données réelles que chaque exemple a au moins une coordonnée manquante. On va supposer dans la suite que $x_i$ se trouve dans un sous-ensemble de $\mathbb{R}^d$ pour $i \in [1, n]$. La plupart des algorithms ci-dessous peuvent aisément être étendus à d'autres cas : une ou plusieurs des composantes étant discrètes par exemple.

Dans cette partie on va considérer les algorithmes qui remplissent les valeurs manquantes. La partie suivante traitera d'algorithmes apprenant directement sur l'ensemble "à trous".

[3] La condition d'arrêt avant stabilité totale peut donner lieu à des clusterings non-linéaires.

### 14.4.2 Les différents algorithmes

**Remplacement par la moyenne ou par la moyenne des voisins ($K$-plus proches voisins)**

On peut calculer $mean_j = \overline{(x_.)_j}$ et remplacer $(x_i)_j$ par $mean_j$ pour tout $(i,j)$ tel que $(x_i)_j$ est manquant. Une autre solution serait le calcul de $mean_j = \overline{(x_.)_j}$ seulement sur les voisins de $x_i$. Ceci peut être fait en définissant une distance entre $x_i$ et $x_k$ pour tout $k \in [1,n]$. Si les données sont indexées par le temps, on peut par exemple considérer $d(x_i,x_k) = |k-i|$. Sinon, une simple distance euclidienne peut être utilisée. Une fois la distance définie, on peut calculer la moyenne de la $j^e$ coordonnée sur les $K$ plus proches voisins de $x_i$. La moyenne peut être pondérée par une fonction décroissante de la distance. Les $K$ plus proches voisins peuvent être calculés plus efficacement que par l'algorithme trivial, par une représentation spatiale des données ; voir par exemple le chapitre sur les $k$-plus proches voisins rapides. On peut utiliser ces algorithmes avec quelques adaptations, parce qu'un clustering doit être fait avec données manquantes. L'algorithme de KD-tree ne semble pas adaptable à ce cas ; mais l'autre algorithme que je fournis peut être aisément adapté. L'utilisation des $K$-means pour retrouver les plus proches voisins rapidement (adaptés aux données manquantes) semble être une solution rapide et efficace, même si théoriquement les $K$ voisins choisis ne sont pas nécessairement les plus proches.

**Régression**

**Principe et algorithme**

On définit la matrice $M$, par $M_{i,j} = (x_i)_j$. Comme il y a des données manquantes, cette matrice n'est pas définie pour certains $(i,j)$. Considérons un tel $(i,j)$. On cherche $(I,J) \in P([1,n]) \times P([1,d])$ ($P(E)$ est l'ensemble des parties de $E$), maximal pour l'inclusion ou "plus gros" (dans un sens à définir, par exemple on peut considérer que $(I,J)$ a taille $card(I) + card(J)$) qu'une constante donnée, telle que :

- $M_{a,b}$ est défini pour tout $(a,b) \in I \times J \setminus (i,j)$
- $i \in I$ et $j \in J$.

Ce $(I,J)$ peut être construit par exemple par un algorithme glouton [4]. Une heuristique efficace est d'essayer en premier la coordonnée $l$ telle que la covariance de $(x_.)_l$ et $(x_.)_j$ soit aussi grande que possible (en valeur absolute), et les exemples proches de $x_i$ (par exemple $x_{i-1}, x_{i+1}, x_{i-2}, x_{i+2}, x_{i-3}, x_{i+3}, ...$, si les données sont indexées par le temps, ou les plus proches de $x_i$ selon une distance euclidienne).

Dans la suite je noterai $(x_i)_{J\setminus\{j\}}$ la restriction de $x_i$ à ses coordonnées $j_1, j_2, ... ,j_p$ avec $J = \{j, j_1, j_2, ..., j_p\}$. On peut maintenant apprendre les couples $((x_i)_{J\setminus\{j\}}, x_j)$ (avec son algorithme supervisé préféré...) pour $i \in I \setminus \{i\}$. $(x_i)_j$ est maintenant remplacé par l'image de $(x_i)_{J\setminus\{j\}}$ par la fonction résultant de l'apprentissage.

### 14.4.3 Une optimisation possible ; le clustering

Certains des algorithmes précédents sont très lents. Leur complexité peut être largement réduite sans perte de précision par clustering des coordonnées ou des points. Pour clusteriser les points, on peut utiliser une distance définie même sur les exemples incomplets. Ceci n'est pas trivial : calculer une moyenne quadratique entre $x_1$ et $x_2$ seulement parmi les coordonnées sur lesquelles les deux sont définis peut entrainer de grosses erreurs. Mais clusteriser les coordonnées est beaucoup plus facile ; on peut définir aisément la corrélation entre deux coordonnée même avec données manquantes ; on peut seulement supprimer les points non définis pour le calcul de la variance ou covariance. Alors la distance entre deux coordonnées peut être définie comme un moins la valeur absolue de la corrélation. Ceci est particulièrement efficace quand la complexité est due à la dimension. Malheureusement, le clustering basé sur les distances est souvent très inefficace, et ainsi les résultats sont très irréguliers et peut conduire à de longs temps de calcul à cause d'un clustering très déséquilibré.

[4] Ceci signifie que l'on peut essayer à chaque pas d'ajouter un élément $I$ ou un élément à $J$. Si c'est possible sans violter les contraintes, l'éléments est ajoutés. L'ordre dans lequel les éléments sont ajoutés peut être choisis par l'algorithme ; variables corrélées ou anticorrélées d'abord, points proches d'abord.

### 14.4.4 Complexité

Soit $n$ le nombre d'exemples, $d$ la dimension, $M$ le nombre de valeurs manquantes, $m$ le nombre d'exemples incomplets.

• Complexité de l'algorithme des $K$ plus proches voisins :

Avec $k = 1$, $d \times n \times M$. Ceci est le pire cas, ie le cas où on a à trouver le plus proche voisin parmi ceux pour lesquels la valeur manquante est définie une fois par valeur manquante par exemple. En pratique, pourvu qu'il n'y a pas trop de valeurs manquantes, la complexité est plus probablement de l'ordre de $d \times n \times m + M$, simplement en cherchant les $K$ voisins triés et alors en choisissant pour chaque valeur manquante quel exemple à a être choisi.

Avec d'autres valeurs de $k$, la complexité moyenne de l'algorithme trivial est $M \times (n \times d + k)$, comme la complexité moyenne de la recherche des $k$-plus petites distances est $n$ avec un algorithme simple (voir B). Comme ci-dessus, on pourrait supposer que pour un exemple donné, les mêmes voisins pourraient être utilisés pour différentes valeurs manquantes de l'exemple. Ainsi en gros la complexité se réduirait à $m \times (n \times d + k) + M \times k$.

• Complexité des algorithmes de $k$-plus proches voisins, avec clustering sur les coordonnées :

On peut utiliser comme distance entre les coordonnées l'opposée de la valeur absolue de la correlation des coordonnées (ou d'autres distances similaires, comme expliqué en partie 14.2.3, et alors utiliser l'algorithme de clustering simple définie en 14.3.2. La complexité dépend du nombre de clusters et de leurs tailles. Avec $C$ le nombre de clusters, en supposant que les clusters ont environ la même taille, la complexité moyenne est $d^2 log(d) + M \times (n \times d/C + k)$, $d^2 log(d) m \times (n \times d/C + k) + M \times k$ avec l'hypothèse selon laquelle les valeurs manquantes différentes d'un exemple donné peuvent être remplacés à l'aide des mêmes voisins.

• Complexité de l'algorithme des $k$-plus proches voisins, avec clustering sur les points :

On utilise les $k$-means pour clusteriser les points, avec les $k$-means modifiés comme défini en 14.3.2 (cas sans données manquantes). Alors la complexité (en supposant que $C$ est le nombre de classes et que les clusters ont la même taille) est celle des $k$-means (qui sont assez rapides), plus $O(M \times (n \times d/C + k))$, $O(m \times (n \times d/C + k) + M \times k)$ avec l'hypothèse que les différentes valeurs manquantes d'un même échantillon peuvent être remplies en utilisant les mêmes voisins.

• Complexité des $k$-plus-proches-voisins rapides (voir le chapitre sur les $k$-plus proches voisins rapides) :

On suppose que l'arbre construit est bien équilibré, ie qu'à chaque étape les clusters ont la même taille. Alors avec $C$ le nombre de fils de chaque noeud la complexité est la complexité de la construction de l'arbre, plus $O(M \times C \times d \times k \times log_C(n))$. Avec l'hypothèse que les mêmes voisins peuvent être utilisés pour différentes valeurs manquantes d'un échantillon, la complexité devient bornée par $O(m \times C \times d \times k \times log_C(n) + M)$.

• Complexité de la méthode de régression

On va considérer le cas où l'on sélectionne la sous-matrice par algorithme glouton. A chaque stade, on essaie d'ajouter une variable si $c_i|I| < c_j|J|$, et une coordonnée sinon. On utilise par exemple $c_i = c_j = 1$. Pour les coordonnées, les essais sont faits en ordre inverse de la valeur de la corrélation, et pour les points, en ordre inverse de proximité dans la liste. Ceci est seulement efficace dans le cas où les données sont indexées de manière à ce que les points qui ont des indices proches sont proches.

Notons $p$ la probabilité pour une valeur donnée d'être manquante. $p$ peut être évalué par $\frac{m}{nd}$. On considère que $p$ est indépendant de la coordonnée et de l'example. Calculons la complexité du remplissage des valeurs manquantes $i, j$. Tout d'abord, $I$ et $J$ sont initialisés à $\{i\}$ et $\{j\}$ respectivement. Alors l'algorithme doit d'abord trouver une coordonnée sans valeur manquante pour chaque élément de $I$, ce qui est le cas d'une coordonnée avec probabilité $p^{|I|}$. Alors le temps moyen pour trouver un tel élément est $p^{|I|} \sum_{k=1}^{\infty} k(1-p^{|I|})^k = \frac{1-p^{|I|}}{p^{|I|}}$. Sommer pour $|I| = 1, 2, 3, ..., T$ pour un $T$ donné (la taille de la matrice recherchée) on obtient $\frac{1-(\frac{1}{p})^{T+1}}{1-\frac{1}{p}} - T$. Ceci est exponentiel en $T$ si $p$ est constant. On suppose que la complexité de l'algorithme de régression est bornée par $C(T)$. Alors, la complexité final moyenne est $O(M \times C(T) \times (\frac{1-(\frac{1}{p})^{T+1}}{1-\frac{1}{p}} - T))$. Finalement l'algorithme est linéaire en $M$.

## 14.5 Apprendre avec des données manquantes

Dans cette partie, on va considérer un ensemble fini de $x_i$ et $y_i$, supposant que certaines valeurs sont manquantes. On veut effectuer un apprentissage sur cet ensemble.

### 14.5.1 Différents algorithmes

#### Suppression des exemples ou coordonnées incomplets

Bien sûr ceci est l'algorithme le plus simple. La suppression des exemples incomplets suppose que le nombre d'exemples complets est assez grand, et que les points de tests seront complets. La suppression des coordonnées vides ne peut être utilisée si l'on n'est sûr qu'aucune autre coordonnée ne sera manquante parmi les points de test.

#### Considérer les sorties inconnues comme une valeur manquante

Supposons que $A$ soit comme suit :

$$A = \begin{pmatrix} (x_1)_1 & (x_1)_2 & \dots & (x_1)_d & (y_1)_1 & \dots & (y_1)_p \\ (x_2)_1 & (x_2)_2 & \dots & (x_2)_d & (y_2)_1 & \dots & (y_2)_p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (x_n)_1 & (x_n)_2 & \dots & (x_n)_d & (y_n)_1 & \dots & (y_n)_p \\ (ux_1)_1 & (ux_1)_2 & \dots & (ux_1)_d & inconnu & \dots & inconnu \\ (ux_2)_1 & (ux_2)_2 & \dots & (ux_2)_d & inconnu & \dots & inconnu \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (ux_{un})_1 & (ux_{un})_2 & \dots & (ux_{un})_d & inconnu & \dots & inconnu \end{pmatrix}$$

$un$ étant le nombre de points test, certains des $(x_i)_j$ étant manquants. Il apparait clairement sur ceci que le remplacement des valeurs manquantes dans cette matrice complète à la fois les données manquantes parmi les $x_i$ et calcule les sorties inconnues.

#### $k$-plus-proches-voisins

On peut définir une distance sur les points, prenant en compte les valeurs manquantes. Par exemple, $d(x,y) = \frac{mean(|x_i - y_i|^p)^{1/p}}{sqrt(N_{x,y})}$ avec $N_{x,y}$ étant le nombre de coordonnées qui sont à la fois définies en $x$ et $y$. Alors on peut utiliser les algorithmes $k$-plus proches voisins. Quand on calcule la $j^e$ coordonnée de $y_i$, les plus proches voisins d'un point doivent être pris parmi les points pour lesquels cette coordonnée est définie. La solution la plus rapide est de calculer une fois quelques voisins de plus que les $k$ plus proches, et ensuite d'extraire parmi ces voisins les $k$ plus proches pour lesquels la coordonnée requise est définie, pour chaque coordonnée de l'espace de sortie.

### 14.5.2 Autres solutions & conclusion

Le paradigme général EM peut être appliqué ici. Quelques valeurs aléatoires (ou choisies par une heuristique) sont utilisées pour remplir les "trous". Alors l'apprentissage est effectué, et les valeurs manquantes sont évaluées à nouveau, et ainsi de suite jusqu'à stabilisation.

Certains algorithmes peuvent être directement dérivés, tels que l'implémentation d'a priori bayésiens, et des algorithmes usuels peuvent parfois être adaptés, eg simplement en ajoutant une composante par coordonnée potentiellement manquante, égale à 1 ou 0 selon que la valeur soit manquante ou non.

L'algorithme EM est sans doute la solution la plus classique et la plus efficace pour ce genre de problèmes.

## 14.6 Quantification vectorielle

### 14.6.1 Présentation du problème

La discrétisation est le choix, étant donnée une famille finie $DataSet = (x_i, y_i)_{i\in[1,n]}$ (resp. $DataSet = (x_i)_{i\in N}$), d'une sous-famille $(x_i, y_i)_{i\in I}$ (resp. $(x_i)_{i\in I}$) pour $I \subset [1, n]$, qui est aussi représentative de $DataSet$ que possible (l'échantillonnage aléatoire est parfois une méthode tout à fait acceptable, grâce à sa vitesse notamment). On pourrait éventuellement choisir $(x'_i, y'_i)_{i\in I}$ (resp $(x'_i)_{i\in I}$) non inclus dans $DataSet$, mais ayant une structure proche de $DataSet$.

La classification non-supervisée est une façon de faire cela. On peut par exemple extraire les éléments d'une carte de Kohonen, pondérée éventuellement par le nombre d'éléments qui activent chaque neurone. On pourrait utiliser les centres d'une classification $K$-means, éventuellement pondérée par le nombre d'éléments dans chaque classe. La même chose peut être faite avec d'autres techniques de classification. On va seulement donner ici *d'autres techniques*.

Un point important est à noter : on cherche ici à fournir des points représentatifs de la distribution des points sur lesquels on travaille. Si pour une raison ou une autre les points sur lesquels on travaille ne sont pas iid suivant la distribution à laquelle on s'intéresse (eg, il y a un biais en faveur de certains points), alors il est intéressant de les rééchantillonner d'une manière prenant en compte le biais. On peut par exemple pondérer les points dans les $k$-means ou beaucoup d'algorithmes.

### 14.6.2 Quelques autres algorithmes

Les deux algorithmes présentés par la suite sont basés sur une fonction de "fitness", évaluant la pertinence d'un quantification. Le cas détaillé ici est celui d'une fonction de "fitness" correspondant à la qualité de l'apprentissage et utilise donc une information de sortie, mais d'autres cas peuvent être envisagés.

#### Algorithme génétique

**Principe**

On considère à chaque stade un ensemble fini $\{E_1, ..., E_q\}$ de sous-ensembles de $[1, n]$. On évalue alors l'efficacité d'un apprentissage sur chacun des $E_i$. Alors on garde une proportion $p$ d'entre eux (les meilleurs $\lfloor q.p \rfloor$ [5]). On ajoute alors de nouveaux ensembles générés par combinaison de ces meilleurs éléments, perturbés aléatoirement. Ceci nous donne un nouvel ensemble de sous-ensembles de $[1, n]$, avec lequel on peut effectuer un nouveau stade.

**Algorithme**

• Choisir $p$, une proportion d'éléments à conserver d'une génération ; *size*, la taille des sous-ensembles considérés ; *perturbation*, le nombre d'exemples modifiés au hasard ; $q$, le nombre de sous-ensembles étudiés à chaque stade.

• Choisir $G = E_1, ..., E_q$, la première génération (par exemple des sous-ensembles disjoints choisis au hasard).

• Pour un nombre de stades donnés, où quand l'efficacité évaluée ne change pas assez :

• • Evaluer chacun des $E_i$. Ceci est fait en calculant le taux d'erreur, ou l'erreur quadratique moyenne avec un problème de régression. Garder seulement dans $g$ les meilleurs $\lfloor q.p \rfloor$ $E_i$.

• • Pour chaque nouveau sous-ensemble qui doit être construit $(q - \lfloor q.p \rfloor)$ :

• • • Choisir au hasard deux parents $F_1$ et $F_2$ parmi les sous-ensembles conservés.

• • • Avec $A$ les éléments qui sont dans $F_1$ et $F_2$, $B$ les éléments qui sont dans $F_1$ ou dans $F_2$ mais pas dans chacun des deux, $C$ les éléments qui ne sont ni dans $F_1$ ni dans $F_2$ :

• • • Soit $D = A \cup B'$, avec $B'$ un sous-ensemble choisi au hasard dans $B$, avec taille telle que $D$ a taille *size*.

• • • Extraire *perturbation* éléments de $C$, et les mettre dans $D$ au lieu de *perturbation* éléments choisis au hasard dans $D$

• • • Ajouter $D$ dans $G$.

[5] $\lfloor x \rfloor$ étant la partie entière de $x$

• Retourner l'ensemble évalué comme étant le meilleur.

On peut éventuellement considérer le recuit simulé, forme "limite" d'algorithmes génétiques où les mutations sont plus fréquentes au début qu'à la fin de la recherche, et où une génération n'est pas mixée pour engendrer la suivante.

#### Algorithme glouton

**Principe**

Le principe est d'ajouter à chaque stade l'élément qui décroit le plus efficacement le taux d'erreur/l'erreur quadratique moyenne. Autant que possible, le programme doit utiliser l'apprentissage fait sur les apprentissages précédents pour optimiser le nouvel apprentissage. La décroissance de l'erreur peut être utilisé comme critère d'arrêt. Le sous-ensemble initial peut être choisi au hasard, ou par tout quantification vectorielle (dans le cas d'une dichotomie, on peut simplement choisir un exemple par classe). Le nouvel élément peut par exemple être choisi parmi un ensemble fini choisi au hasard parmi les exemples non-vus.

**Commentaires**

Il y a quelques hyperparamètres à choisir :

• Le nombre de nouveaux éléments possibles à tester
• le nombre d'éléments choisis comme ensemble test pour l'évaluation de l'ensemble d'apprentissage
• la taille finale de l'ensemble d'apprentissage

### 14.6.3 Conclusion

Se baser sur un algorithme de classification non-supervisé est sans doute la solution la plus classique, mais on peut exhiber des cas particuliers où l'information supervisée est déterminante, et les deux algorithmes ci-dessus (glouton et génétique) peuvent alors s'avérer beaucoup plus efficaces.

## 14.7 Conclusion

Ce travail est beaucoup plus une introduction qu'un réel travail de recherche. Des résultats supplémentaires pourraient être des comparaisons empiriques ; la validité de telles comparaisons est toujours liées à la puissance de calcul, au type de données utilisées, aux critères (vaste problème dans des sujets comme l'apprentissage non-supervisé et le prétraitement), et donc une telle étude prendrait beaucoup de temps.

Quelques heuristiques simples peuvent être suggérées, au vu des résultats expérimentaux, de calculs de complexité, ou au vu des comparaisons effectuées dans le cadre des rencontres Ecrin concernant les pics de pollution :

- Utiliser les cartes de Kohonen seulement avec petite dimension de sortie.
- Utiliser les algorithmes génétiques seulement pour des fonctions d'un grand nombre de variables ou pour des composantes discrètes.
- Souvent utiliser une transformation non-linéaire des données avant toute classification non-supervisée. Les algorithmes de classification non-supervisée proposés ici semblent tous gérer difficilement les non-linéarités, même si les algorithmes multicouches travaillent dans des espaces de fonctions fortement non-linéaires.
- Utiliser les étiquettes supervisées dès que possibles. L'apprentissage semi-supervisé peut être utilisé dans certains cas (voir [Pedrycz, 1985, Saint-Jean et al, 2001]).
- Réduire la dimension quand elle est grande et qu'on veut utiliser un arbre de décision, ou si elle est grande sans données creuses et qu'on veut utiliser une rétropropagation.
- Les algorithmes EM (espérance-maximisation) sont probablement les meilleures solutions pour apprendre avec des données manquantes et les autres approches proposées ci-dessus semblent moins réalistes.
- L'algorithme de clustering simple semble trop simple pour beaucoup d'applications réelles, mais dans beaucoup de cas, tels le distances entre langues ou l'ADN, ceci est l'algorithme le plus efficace, le plus interprétable, capable de fournir directement des arbres généalogiques.

- L'algorithme de transfert semble un peu plus efficace que les classiques $k$-means.
- Réordonner au hasard les point peut être très efficace avant une classification non supervisée telle les $k$-means ou le transfert, lorsque les points ne sont pas indépendants.
- Les algorithmes multi-couches ont été montrés efficaces comparés à des algorithmes classiques pour la classification non-supervisée, malgré leur manque de justifications théoriques (il existe un minimum global trivial inintéressant).
- La réduction de dimension semble très difficile quand on n'a pas l'opinion d'un expert. Trouver un cas dans lequel réduire la dimension conduise à un meilleur apprentissage qu'un apprentissage régularisé classique n'est pas simple.

L'avant dernier point semble finalement l'apport le plus important de ce chapitre, en dehors de la simple description de méthodes classiques. Les critères semblent déjà rarement clairement établis dans la littérature ; défaut que j'essaie de palier dans ma partie sur la contribution de la théorie de l'apprentissage à l'apprentissage non-supervisé.

# Bibliographie


[Bishop, 1995] C.M. Bishop, *Neural Networks for Pattern Recognition, Oxford, 1995*

[Benaim et al, 1998] M. Benaim, J.-C. Fort, G. Pages, *Almost Sure Convergence of the One-Dimensional Kohonen Algorithm, Advances in Applied Probability, Vol 30, pp 850-869, 1998*

[Bourlard et al, 1988] H. Bourlard, N. Morgan, *Autoassociation by multilayer perceptrons and singular value decomposition. Biological Cybernetics 59, 291-294 (1988)*

[Cottrel et al, 1987] M. Cottrel, J.-D. Fort, *Etude d'un processus d'autoorganisation, Annales de l'institut Henri Poincaré, 23 :1, 1987*

[Pedrycz, 1985] W. Pedrycz, *Algorithms of Fuzzy Clustering with Partial Supervision. Pattern Recognition Letters, 3(1),13-20, 1985.*

[Ritter et al, 1988] H. Ritter, K.-J. Schulten, *Convergence properties of Kohonen's topology conserving maps : Fluctuations, stability and dimension selection. Biological Cybernetics, 60 :59-71, 1988*

[Saint-Jean et al, 2001] C. Saint-Jean, C. Frélicot, *Une méthode paramétrique et robuste de classification semi-supervisée avec rejet, proceedings de CAP 2001.*

[Scholkopf et al, 1998] B. Schölkopf, A. Smola, K.R. Müller, *Nonlinear component analysis as a kernel Eigenvalue problem, Neural computation, 10,5,p1299-1319,1998*

[Thiria et al, 1997] S. Thiria, Y. Lechevallier, O. Gascuel, S. Canu, *Statistique et méthodes neuronales, Dunod, 1997*

# Annexe A

# Extraction of a subset of high values

Let $a_1$,...,$a_n$ a sorted (in decreasing order) list of reals. We want to chose automatically the "high" values of the $a_i$'s. The problem is that we don't know the number of values that we want to keep. A usual technic is to define $\Delta a_i = a_i - a_{i+1}$. Then we consider the sum $S$ of the mean of the $\Delta a_i$'s and of the standard deviation of the $\Delta a_i$'s. We keep the $a_i$'s until the first index such that $\Delta a_i$ is higher than $S$.

# Annexe B

# Algorithm to select the $k$ largests of $n$ elements

The precise problem is the selection of the $k^th$ largest and of the $k$ largests in same time. Let $a_1$, ... ,$a_n$ be considered. One can simply use the following algorithm :

• If $n = 0$ or $n = 1$, the algorithm is finished.

• select an element $a$ of the $a_i$'s (randomly, or the median of a few elements...)

• Change the order of the $a_i$'s so that the $n'$ elements bigger are at the beginning and the other at the end of the list.

• If $n' > k$, do the same thing, by induction, on the $n'$ first elements. Otherwise, do the same thing on the elements after the $n'$ firsts, with $k' = k - n'$.

At the end of this algorithm, the $k^{th}$ element is the $k^{th}$ largest, and the first $k - 1$ elements are smaller than it. This algorithm has average cost $log(n)$.

# Chapitre 15

# Bornes pour l'inférence Baysésienne et l'algorithme de Gibbs

**Résumé**

Des travaux théoriques récents basés sur les méthodes de l'apprentissage statistique ont souligné l'intérêt de paradigmes anciens tels que l'inférence Bayésienne et l'algorithme de Gibbs. Des bornes de complexité d'échantillon ont été données pour de tels paradigmes dans le cas d'erreurs empiriques nulles. Cette partie étudie le comportement de ces algorithmes lorsqu'on omet cette hypothèse. Les résultats incluent la convergence uniforme de l'algorithme de Gibbs vers l'inférence Bayésienne, la vitesse de convergence de l'erreur empirique vers l'erreur en généralisation, ainsi que la convergence de l'erreur en généralisation vers l'erreur optimale, dans la classe de fonctions sous-jacente.

L'intérêt de cette partie réside surtout dans le fait d'attirer l'attention vers un domaine des statistiques uniformes relativement peu connu des informaticiens, ceux-ci étant plus familiers de la théorie de la VC-dimension, approche non-asymptotique. De même que des travaux montrent que, dans le cas "sans erreur", on peut obtenir des bornes efficaces dans le contexte Bayésien, en utilisant d'autres outils que la VC-théorie (ex. la théorie de l'information), nous proposons ici, pour cette étude, l'utilisation de techniques issues du processus empirique.

Ce travail a donné lieu à deux publications, l'une francophone et l'autre anglophone, avec Hélène Paugam-Moisy ([Teytaud et al, 2001, Teytaud et al, 2001]).

# Table des matières



## 15.1 Introduction

Des travaux récents portant sur la Bayes Point Machine [Herbrich et al, 1999, Herbrich et al, 2000], rapportent de bons résultats sur quelques jeux de données de la littérature, mettant ainsi en avant l'intérêt de paradigmes d'apprentissage assez anciens, comme l'inférence Bayésienne. Haussler, Kearns et Shapire [Haussler et al, 1994] ont donné des résultats sur l'apprentissage Bayésien sous l'hypothèse d'un taux d'erreur empirique nul ; Devroye, Gyorfi et Lugosi [Devroye et al, 1997] rappellent des résultats négatifs pour des paradigmes proches. De même que Haussler, Kearns et Shapire soulignent, comme intérêt principal de leurs résultats, le fait d'utiliser des théorèmes venus d'autres communautés que celle de l'intelligence artificielle (ex. théorie de l'information en plus de la VC-théorie), nous proposons notre article comme un exemple de l'application de résultats issus d'une communauté étudiant la convergence faible du processus empirique.

Parallèlement à la VC-théorie, s'est développée une théorie beaucoup moins connue de la communauté de l'intelligence artificielle et dont les résultats peuvent trouver leurs applications ici. La nécessité de résultats uniformes se fait largement sentir dès les travaux de Solomonoff en 1960 [Solomonoff, 1960]. Plus tôt encore, Doob [Doob, 1949] et surtout Donsker [Donsker, 1951, Donsker, 1952] développent les idées sur l'uniformité : la statistique de Kolmogorov-Smirnov est étudiée de manière détaillée, on généralise les théorèmes centraux à des convergences uniformes, mais contrairement à la VC-théorie, on ne se préoccupe pas de bornes non asymptotiques valables pour toute distribution[1].

En 1961, Kolmogorov et Tikhomirov [Kolmogorov et al, 1961] utilisent les nombres de couverture pour quantifier l'ampleur d'une famille de fonctions. Ces résultats sont en particulier confortés par des bornes sur les nombres de couverture de familles intuitivement intéressantes comme les familles Hölderiennes. Dudley résoud des problèmes importants de mesurabilité dans [Dudley, 1966], et fait avec Strassen [Strassen et al, 1969] un lien entre la continuité du processus gaussien et les nombres de couverture (nombre de boules de rayon $\epsilon$ permettant de recouvrir l'ensemble de fonctions). Dudley étudie l'ensemble des fonctions Hölderiennes [Dudley, 1974], Sudakov poursuit le travail dans [Sudakov, 1969], Bronstein dans [Bronstein, 1976] étudie le cas des ensembles convexes (il étudie leurs nombres de couverture); Dudley dans [Dudley, 1978], Van de Geer dans [Van de Geer, 1991] (fonctions monotones) et Ossiander dans [Ossiander, 1987] utilisent l'*entropie bracketing*, c'est-à-dire que l'on considère non plus un recouvrement par des boules, mais par des "brackets" qui sont des ensembles de fonctions comprises entre deux fonctions extrémales, elles-mêmes à distance $\epsilon$ l'une de l'autre. L'utilisation des brackets se poursuit avec [Andersen et al, 1988] (loi du logarithme itéré).

Efron dans le fameux article [Efron, 1979] définit le Bootstrap, basé sur la notion de classes de Donsker. Une classe de Donsker est l'analogue d'une classe de VC-dimension finie : la convergence y est uniforme, et la précision s'affine en $1/\sqrt{m}$ avec $m$ le nombre d'exemples. On peut regretter que cette convergence

1. Notons que les résultats sur le processus empirique comportent aussi des résultats d'uniformité en la distribution (voir [Van der Vaart et al, 1996, p166-175]). Mais on ne se préoccupe plus ici nécéssairement du pire cas sur *toutes* les distributions, ni de non-asymptotique, ni de considérer des distributions différentes pour chaque nombre d'exemples.

soit asymptotique, mais un gros atout de ce modèle est de fournir un équivalent exact de la précision asymptotique. On évite ainsi les bornes très larges de la VC-théorie. En outre, on peut traiter des familles de fonctions beaucoup plus vastes, toute famille de VC-dimension finie étant Donsker, mais la réciproque étant fausse. Les résultats énoncés dans cette partie, outre quelques résultats classiques en apprentissage comme l'utilisation de l'inégalité de Hoeffding pour des bornes basées sur des nombres de couverture, utilisent ces théorèmes centraux limites généralisés.

On considère des algorithmes d'apprentissage travaillant sur des données indépendantes identiquement distribuées $D = \{(X_1,Y_1),\ldots,(X_m,Y_m)\}$, avec $X_i \in \mathcal{X}$ et $Y_i \in \{0,1\}$. On considère une loi *a priori* $Pr$ sur l'espace $W$ des hypothèses (ceci n'implique *pas*, dans aucun des résultats qui suivent, que cet a priori est supposé correspondre à une distribution selon laquelle une dépendance cachée serait tirée au hasard !). On définit la fonction de coût suivante, pour $w \in W$: $L(Y,X,w) = \ln((1-Y)w(X) + Y(1-w(X)))$. Le coût empirique $L_E(w)$ est la moyenne de $L(Y_i,X_i,w)$ sur l'échantillon d'apprentissage. Le coût en généralisation $L(w)$ est l'espérance de $L(Y,X,w)$. Ceci implique que $w$ n'est pas simplement considéré comme un classifieur, mais que $w(X) = \frac{2}{3}$ quand $Y = 1$ est pire que $w(X) = \frac{9}{10}$ même si un seuillage pourrait conduire à 1 dans les deux cas. Notons qu'un bon comportement de cette fonction implique un bon comportement du taux d'erreur du classifieur associé, qui choisit la classe 1 si et seulement si $f_{D_m}(x) > \frac{1}{2}$. La réciproque n'est pas nécessairement vraie et un point important est le fait que l'inférence Bayésienne peut s'avérer meilleure que tout classifieur de la famille $W$ de fonctions. Dans un souci de clarté, on note $R(Y,X,w) = \ln(1 - \exp L(Y,X,w))$ et $R(w) = E(R(Y,X,w))$ l'espérance de $R(.,.,w)$ pour la loi de $(X,Y)$. On appelle *inférence Bayésienne* l'algorithme qui propose comme classifieur $f_{D_m}(x) = \int_W V(w,D_m)w(x)dPr(w)$ avec

$$V(w,D_m) = \frac{\Pi_{i=1}^m(\exp R(Y_i,X_i,w))}{\int_W \Pi_{i=1}^m(\exp R(Y_i,X_i,w))dPr(w)}$$

vraisemblance normalisée de $w$. Notons que cette fonction n'appartient pas nécessairement à $W$. On appelle *premier algorithme de Gibbs* (voir [Opper et al, 1991]) d'ordre $k$ l'algorithme qui propose comme classifieur $g_{D_m}^k(x) = \frac{1}{k}\sum_{i=1}^k w_i(x)$, les $k$ classifieurs $w_i$ étant tirés au sort suivant la loi de densité $V(w,D_m)Pr(w)$. On appelle *second algorithme de Gibbs* d'ordre $k$ l'algorithme qui propose pour classifieur $g2_{D_m}^k(x) = \frac{\sum_{i=1}^k w_i V(w_i,D_m)}{\sum_{i=1}^k V(w_i,D_m)}$, $w_i$ étant tiré au sort selon $Pr(w)$. Comme les algorithmes de Gibbs sont non déterministes, cet apprentissage peut être vu de deux façons, la première consistant à choisir une fois la fonction de classification, la seconde consistant en tirer de nouveaux $w_i$ à chaque entrée à classifier. Par la suite on considère un choix aléatoire des $w_i$, pendant la phase d'apprentissage. L'espérance du coût en généralisation de cet algorithme est la probabilité d'erreur de l'algorithme non-déterministe équivalent, avant le choix stochastique des $w_i$. $N(\epsilon,\mathcal{F},L_p)$ est le $\epsilon$-nombre de couverture de l'espace $\mathcal{F}$ pour la distance $L^p$.

## 15.2 Prérequis

Les notions suivantes sont utilisées: classes de Donsker, théorèmes centraux limites généralisés sur les classes de Donsker, VC-dimension (la VC-dimension d'une classe de fonctions réelles est ici la VC-dimension de l'ensemble des graphes de ces fonctions), fat-shattering dimension, convergence faible, intégrale d'entropie, nombres de couverture et bracketing-nombres de couverture, $O(.)$ faible. Une introduction à ces notions peut être trouvée dans [Van der Vaart et al, 1996], livre fort clair et fort enrichissant, ou [Radulovic et al, 2000], beaucoup plus restreint et spécialisé, mais consultable sur Internet et dont l'introduction est un rappel très bref et très intuitif de ce qui se fait en la matière. On rappelle qu'une classe $\mathcal{F}$ de fonctions est Donsker pour une distribution d'exemples $X_i$ si $\frac{1}{\sqrt{m}}\sum_{i=1}^m X_i$ converge faiblement avec $m$ vers une limite "tight" dans $l^\infty(\mathcal{F})$. Ceci implique la convergence faible uniforme des moyennes empiriques vers leurs espérances, en $O(1/\sqrt{m})$. La différence principale avec la VC-théorie est la non-uniformité (en la distribution) et le caractère asymptotique des convergences. Les équations (15.1) et (15.2) utilisées plus bas illustrent sans doute mieux ce point qu'un long discours (bien qu'elles ne fassent pas ressortir la possibilité de tirer parti de l'a priori sur la distribution des exemples).

### 15.2.1 Théorèmes généraux

Les théorèmes suivants mettent en évidence le fait que les résultats basés sur les classes de Donsker sont plus généraux que ceux basés sur la VC-dimension (mais, bien sûr, il y a une perte au niveau du caractère asymptotique des résultats). $P$ est la distribution des exemples, $\mathcal{F}$ est la famille des fonctions de coût (on peut travailler, pour un grand nombre de fonctions de coût, indifféremment sur la famille de fonctions utilisées en régression et la famille de fonctions de coût, en utilisant le caractère Lipschitzien de la fonction de coût).

**Théorème 15.1 (Condition suffisante basée sur l'entropie)** *Soit $F$ enveloppe de $\mathcal{F}$, famille mesurable de fonctions. Si*

$$\int_0^\infty \sqrt{sup_Q \log N(\epsilon \parallel F \parallel_{Q,2}, \mathcal{F}, L_2(Q))} d\epsilon < \infty$$

*alors $\mathcal{F}$ est $P$-Donsker pour tout $P$ tel que l'espérance de $F$ pour la distribution des exemples est finie.*

L'entropie bracketing est une forme d'entropie basée sur les "brackets" ; au lieu de recouvrir les espaces de fonctions par des boules, on les recouvre par des segments $[g_1, g_2] = \{f / g_1 \leq f \leq g_2\}$. Les bracketing-nombres de couverture ainsi obtenus sont plus importants en général que les nombres de couverture classiques, mais ils permettent certains résultats supplémentaires et permettent de prendre en compte la distribution marginale (sur l'espace d'entrée) des exemples. Le résultat suivant en rend compte:

**Théorème 15.2 (Condition suffisante basée sur l'entropie-bracketing)** *Si*

$$\int_0^\infty \sqrt{log\ N_{[\,]}(\epsilon, \mathcal{F}, L_2(P))} d\epsilon < \infty$$

*alors $F$ est $P$-Donsker.*

Des preuves de ces résultats peuvent être trouvées dans [Van der Vaart et al, 1996]. Notons que les théorèmes 1 et 2 ne s'incluent pas l'un l'autre : le second est plus efficace dans certains cas car il permet d'utiliser la distribution marginale des exemples, mais le premier a l'avantage de recouvrir la famille de fonctions par des boules et non par des brackets.

Les théorèmes 7 et 11 illustreront à nouveau cette plus grande généralité des classes de Donsker : la fermeture pour la topologie de la convergence point à point de l'enveloppe convexe d'une famille de VC-dimension finie est Donsker. Fondamentalement, les résultats qui suivent tireront parti de ce fait : l'inférence Bayésienne, comme l'algorithme de Gibbs, travaillent sur l'enveloppe convexe de la famille de fonctions utilisées.

Pour qu'une classe soit de Donsker, il suffit que le logarithme des nombres de couverture pour $L^2$ soit en $O(1/e^c)$ pour un certain $c < 2$, alors qu'en VC-théorie les nombres de couverture sont polynomiaux. Un atout de ces résultats, outre le fait que l'on peut donc traiter des familles de fonctions plus vastes, est la plus grande précision obtenue : les bornes de VC-théorie sont larges car elles sont estimées au pire cas sur la distribution - la distribution étant choisie la pire possible, pour chaque nombre d'exemples donné. Ici, on connait la distribution limite des écarts multipliés par la racine du nombre d'exemples entre les moyennes empiriques et les espérances. Néanmoins, la distribution limite est difficile à évaluer en dehors du facteur en racine du nombre d'exemples : la distribution est entièrement définie par les espérances et covariances des écarts, mais la valeur numérique du sup des écarts est difficile à évaluer. A notre connaissance, il n'existe pas encore de théorème général bornant les constantes en jeu, en fonction de l'évolution du nombre de couverture.

### 15.2.2 Pour aller plus loin

On peut citer aussi l'existence de résultats passant outre l'hypothèse de données indépendantes identiquement distribuées ; des résultats similaires existent en VC-théorie, l'inégalité de Hoeffding permettant des bornes sans nécessiter une distribution identique. L'indépendance est plus difficile à traiter en VC-théorie (quoique des extensions existent), mais sur ce point des théorèmes comme le suivant (et d'autres, par exemple dans le cas markovien) nous apparaissent plus forts :

**Théorème 15.3** *Considérons, pour tout $n \in \mathbb{N}$, $(X_{n,1}, \ldots, X_{n,n})$ des variables aléatoires indépendantes. Pour tout $n$ et $f,g \in \mathcal{F}^2$, considérons $d_n(f,g)^2 = \frac{1}{n}\sum_{i=1}^n (f(X_{n,i}) - g(X_{n,i}))^2$. Supposons que $\sup_{d(f,g)<\delta_n} \frac{1}{n}\sum_{i=1}^n (Z_{n,i} -$*

$EZ_{n,i})^2 \to 0$ *pour toute suite* $\delta_n$ *décroissant vers* 0. *Supposons enfin que*

$$\int_0^{\delta_n} \sqrt{\log N(\epsilon,\mathcal{F},d_n)}d\epsilon \to 0$$

*faiblement, pour toute suite* $\delta_n$ *décroissant vers* 0.

*Alors la suite des* $\frac{1}{n}\sum f(X_{n,i}) - Ef(X_{n,i})$ *converge en distribution dans* $l^\infty(\mathcal{F})$ *si la suite des fonctions de covariance converge simplement sur* $\mathcal{F} \times \mathcal{F}$.

On notera que des améliorations de ce résultat permettent en outre d'envisager des théorèmes de la même forme que le théorème de minimisation du risque structurel (dit aussi théorème de minimisation structurelle du risque). On pourra, pour plus d'informations, consulter [Van der Vaart et al, 1996, p220-221], dont le théorème ci-dessous est extrait (et simplifié) pour donner un avant-goût :

**Théorème 15.4 (Classes de fonctions changeant avec $n$)** *On suppose que* $\mathcal{F}$ *est l'ensemble des* $f_{n,t}$ *(supposées mesurables) pour* $n \in \mathbb{N}$ *et* $t \in T$. $\mathcal{F}_n$ *est l'ensemble des* $f_{n,t}$ *pour un* $n$ *donné. On suppose les enveloppes des* $\mathcal{F}_n$ *toutes bornées par* 1. *On suppose en outre que* $\int_P (f_{n,s_n} - f_{n,t_n})^2$ *tend vers* 0 *pour* $n \to \infty$ *pour tous* $d(s_n,t_n) \to 0$. *Dans ces conditions, si, pour toute suite* $\delta_n$ *décroissant vers* 0,

$$\int_0^{\delta_n} \sqrt{N_{[\ ]}(\epsilon,\mathcal{F}_n,L_2(P))}d\epsilon \to 0$$

*alors la suite des évaluations empiriques converge uniformément en* $O(1/\sqrt{n})$ *faible vers un processus gaussien "tight" pourvu que les covariances convergent simplement sur* $\mathcal{F} \times \mathcal{F}$.

Une application possible de ce résultat est la suivante:

1. Considérons une famille $\mathcal{F}'_1, \ldots, \mathcal{F}'_n, \ldots$ de familles de fonctions de coût.
2. Notons $N'(\epsilon,n)$ le bracketing-$\epsilon$-nombre de couverture de $\mathcal{F}'_n$.
3. Avec $\mathcal{F}_n = \mathcal{F}'_n$, les hypothèses du théorème 4 ne seront pas nécessairement vérifiées. On va donc définir une suite $\alpha_n$, et considérer $\mathcal{F}_n = \alpha_n \times \mathcal{F}'_n$.
4. Définissons $N(\epsilon,n)$ le bracketing-$\epsilon$-nombre de couverture de $\mathcal{F}_n$.
5. On a $N(\epsilon,n) = N'(\epsilon/\alpha_n,n)$.
6. On fixe alors $\alpha_0 = 1$ et $\alpha_n \leq \alpha_{n-1}$ aussi grand que possible tel que $N(\epsilon,n) = N(\epsilon,1)$. Cela est toujours possible.

On peut alors appliquer le théorème 4, si les fonctions vérifient en outre certaines hypothèses : par exemple, on peut envisager les fonctions Hölderiennes, avec une norme Hölderienne croissante avec $n$. La convergence n'est plus en $O(1/\sqrt{m})$ ; on a un facteur $1/\alpha_n$ qui apparait. D'où la nécessité de choisir $\alpha_n$ décroissant lentement. A notre connaissance il n'existe pas encore d'étude des possibilités de cette méthode.

Un autre point intéressant concernant les classes de Donsker est la nature du processus gaussien limite. En fait on peut le définir entièrement. En effet, un processus gaussien centré "tight" est entièrement défini par ses covariances : ici les covariances sont simplement les covariances marginales. Ceci ne permet pas d'exhiber, à notre connaissance, une constante pour le sup des écarts à l'espérance, c'est-à-dire un $K$ tel que $P(\frac{1}{\sqrt{m}} \sup_f |\sum f(X_i) - mEf| > K) \to \delta$ pour un $\delta$ donné. Ceci ne serait pas équivalent à la théorie de la VC-dimension : il s'agit là de probabilités asymptotiques. Ceci permettrait par contre de déterminer le rapport entre une borne de VC-théorie et la borne asymptotique. Il serait par exemple intéressant de déterminer s'il existe un rapport minimal indépassable entre une borne de VC-théorie et une borne issue des classes de Donsker (qui permettent aussi de travailler, éventuellement, sur des pires-cas sur toutes les distributions, comme le montre le théorème 1 - mais sans considérer une distribution différente pour chaque valeur de $m$). Formellement, l'idée serait d'élucider la différence entre les termes ci-dessous :

$$\sup_P \lim_{m\to\infty} \frac{1}{\sqrt{m}} \sup_f \{\epsilon / P(|\sum f(X_i) - mEf| > \epsilon) = \delta\} \tag{15.1}$$

$$\lim_{m\to\infty} \sup_P \frac{1}{\sqrt{m}} \sup_f \{\epsilon / P(|\sum f(X_i) - mEf| > \epsilon) = \delta\} \tag{15.2}$$

Les classes de Donsker permettent d'étudier l'équation (15.1) et la VC-théorie l'équation (15.2). L'expression (15.1) étant finie, même pour certains nombres de couverture exponentiels, on peut s'attendre à ce que, dans le cas d'une VC-dimension finie (et donc, par le lemme de Sauer, de coefficients de pulvérisations polynomiaux, donc de nombres de couverture polynomiaux), la différence entre les deux termes soit forte. Des résultats comme ceux de [Van der Vaart et al, 1996, p101 et annexe A.2] pourraient permettre de traiter ces problèmes.

Enfin, il est intéressant de noter que [Van der Vaart et al, 1996, p166] traite l'uniformité en la distribution sous-jacente d'exemples, soit pour toute distribution, soit pour des classes de distributions. Ces résultats peuvent être comparés à [Natarajan, 1988] ou [Alon et al, 1997], qui fournissent des résultats non-asymptotiques. Le cas de fonctions indicatrices se ramène à la finitude de la VC-dimension, si l'on considère toutes les distributions possibles. Grossièrement, on peut résumer les résultats en disant que la loi uniforme des grands nombres nécessite le fait que le logarithme des nombres de couverture pour une distribution discrète répartie en probabilités multiples entières de $1/n$ soit un $o(n)$ uniforme en la distribution, quel que soit $\epsilon$, et que la VC-dimension assure en fait que ce nombre est fini. Il y a donc un écart important. Le théorème central limite uniforme est, lui, valable lorsque les conditions sont vérifiées uniformément. L'élaboration exacte de ces conditions est technique et les résultats de [Alon et al, 1997] sont certainement plus lisibles, plus maniables, et nécessaires et suffisants au pire cas sur toutes les distributions : pour des classes plus restreintes, ces théorèmes présentent une importance réelle. Le lecteur intéressé peut se reporter à [Van der Vaart et al, 1996, p166].

### 15.2.3 Exemples de classes de Donsker

Les théorèmes suivants fournissent d'intéressants exemples de classes de Donsker. Tout d'abord, les classes suivantes, dont les nombres de couverture ont été bornés dans [Kolmogorov et al, 1961] puis [Lorentz, 1966], [Birman et al, 1967], [Dudley, 1984],avec une formulation plus lisible dans [Van der Vaart et al, 1996] :

**Théorème 15.5** *Soit $\mathcal{F}_{B,k,d}$, où $k > 0$ réel, la famille des ensembles de la forme $\{(x,t)/f(x) < t\}$ pour $f$ de $[0,1]^{d-1}$ vers $[0,1]$ tel que $|||f|||_k$ est bien défini et borné par $B$, avec*

$$|||f|||_k = \max_{\sum k_i \leq \lfloor \alpha \rfloor} \sup_x \left| \frac{\partial^{\sum k_i} f(x)}{\partial x_1^{k_1} \partial x_2^{k_2} \dots \partial x_d^{k_d}} \right| \tag{15.3}$$

$$+ \max_{\sum k_i = \lfloor \alpha \rfloor} \sup_{x \neq y} \frac{\left| \frac{\partial^{\sum k_i} f(x)}{\partial x_1^{k_1} \partial x_2^{k_2} \dots \partial x_d^{k_d}} - \frac{\partial^{\sum k_i} f(y)}{\partial x_1^{k_1} \partial x_2^{k_2} \dots \partial x_d^{k_d}} \right|}{|x-y|^{\alpha - \lfloor \alpha \rfloor}} \tag{15.4}$$

*où $\lfloor \alpha \rfloor$ est le plus grand entier* strictement plus petit *que $\alpha$ (égal à $\alpha - 1$ si $\alpha$ est entier). Cet espace de fonctions est appelé espace de* Hölder. *$\mathcal{F}_{B,\alpha,d}$ vérifie :*

$$\log N_{[\,]}(\epsilon, \mathcal{F}_{1,\alpha,d}) \leq K(\alpha,d) \left(\frac{1}{\epsilon}\right)^{2(d-1)/\alpha}$$

Notons que cette famille de fonctions a une VC-dimension infinie. On pourra consulter [Devroye et al, 1997, p479 et suivantes] pour plus d'informations en utilisant les nombres de couverture pour $L_\infty$ au lieu de $L_1$. Le résultat suivant provient quand à lui de [Birman et al, 1967], [Dudley, 1966].

**Théorème 15.6** *Pour $\mathcal{F}$ l'ensemble des convexes de $[0,1]^d$ où $d \geq 2$, $\log N_{[\,]}(\epsilon, \mathcal{F}) \leq K(\frac{1}{\epsilon})^{(d-1)}$*

Il n'est donc intéressant, pour une convergence en $O(1/\sqrt{m})$, qu'en petite dimension. Toutefois, même en dimension 2, la VC-dimension est infinie.

Enfin le théorème 11 fournira un outil pour construire de nouvelles classes de Donsker. En outre, le corollaire 8 fournira des bornes explicites sur les nombres de couverture de fermetures d'enveloppes convexes pour la convergence point à point. Citons enfin quelques autres classes de Donsker : les familles dénombrables de fonctions $L^2$ dont la somme des variances est finie. Toute famille dénombrable de fonctions $f_i$ non corrélées telle que $\sum L^2(f_i)$ est finie vérifie $\{\sum c_i f_i | c_i \in \mathbb{R}, \sum c_i^2 \leq 1\}$ est Donsker. [Van der Vaart et al, 1996, chap 2.13] fournit de nombreux autres exemples.

### 15.2.4 Résultats préliminaires et plan

Pour $\phi$ lipschitzienne, on rappelle que $\{\phi \circ f / f \in \mathcal{F}\}$ est Donsker pourvu que $f$ soit Donsker. Ceci permet de montrer le caractère Donsker de familles de fonctions de coût (en particulier, dans le cas de notre fonction de coût, qui est lipschitizenne pourvu que $w$ et $1-w$ soient minorés par $c > 0$) dans des cas où la famille de fonctions utilisée pour la prédiction est Donsker.

Quelques résultats simples peuvent être énoncés. Les $\epsilon$-nombres de couverture pour $L_1$ de la classe de fonctions parmi laquelle $g^k_{D_m}$ est choisi sont bornés par $N(\epsilon,W,L_1)^k$. Si la VC-dimension $V$ de $W$ est finie, alors la VC-dimension de cette classe de fonctions est bornée par $k \times V$. Les coefficients de Lipschitz en inférence Bayésienne ou lors d'algorithme de Gibbs sont préservés, ce qui permet des bornes en termes de fat-shattering dimension (pour des exemples de résultats basés sur la fat-shattering dimension, le lecteur est renvoyé à [Bartlett, 1998] et [Alon et al, 1997]). Le moyennage bayésien, dans les Bayes Point Machines par exemple, consiste à utiliser comme classifieur l'espérance de l'ensemble des $w \in W$ tels que tous les points empiriques soient bien classés par $w$. Cet algorithme n'est pas équivalent à l'inférence Bayésienne. Des exemples [Teytaud, 2000] de cas où l'inférence Bayésienne est asymptotiquement significativement meilleure que le moyennage Bayésien peuvent le montrer. Des résultats dans [Rujan, 1997] expliquent comment calculer le moyennage bayésien efficacement.

La partie 15.3 donne des bornes sur la différence entre le coût empirique et le coût en généralisation. La partie 15.4 donne des bornes sur la vitesse de convergence de l'inférence Bayésienne vers l'erreur optimale dans $W$. La partie 15.5 donne une borne sur la convergence de l'algorithme de Gibbs vers l'inférence bayésienne. La conclusion ouvre une question sur la deuxième version de l'algorithme de Gibbs.

## 15.3 Inférence Bayésienne: erreur en généralisation

Supposons que $W$ est inclus dans un espace dans lequel des intégrales de fonctions de $W$ dans $\mathbb{R}$ ou de $W$ dans $W$ peuvent être calculées par des sommes de Riemann. En particulier, $\exists w_{k,i} / i \leq k, k \in \mathbb{N}$

$$f_{D_m}(x) = \frac{\lim_{k\to\infty} \sum_{i=1}^k \theta_{k,i} \Pi_{j=1}^m \exp R(Y_j,X_j,w_{k,i}) w_{k,i}}{\lim_{k\to\infty} \sum_{i=1}^k \theta_{k,i} \Pi_{j=1}^m \exp R(Y_j,X_j,w_{k,i})} = \lim_{k\to\infty} \sum_{i=1}^k \alpha_{k,i,D_m} w_{k,i}$$

avec $\alpha_{k,i,D_m} = \frac{\theta_{k,i} \times \Pi_{j=1}^m \exp R(Y_j,X_j,w_{k,i})}{\sum_{l=1}^k \theta_{k,l} \Pi_{j=1}^m \exp R(Y_j,X_j,w_{k,l})}$, qui conduit à $\sum_{i=1}^k \alpha_{k,i,D_m} = 1$.

Cette hypothèse implique que $f_{D_m}$ est dans la fermeture $W'$ de l'enveloppe convexe de $W$ pour la convergence point à point. Le résultat suivant en découle.

### 15.3.1 Cas 1 : $W$ a une VC-dimension finie

Si la VC-dimension de $W$ est finie, alors les logarithmes des nombres de couverture de $\overline{conv}\ W$ sont bornés par $O(1/\epsilon^p)$ avec $p < 2$. Cette condition est vérifiée dans le cas d'une VC-dimension finie $V+1$, mais une borne sur les nombres de couverture dépendant de la distribution est suffisante.

**Théorème 15.7 (*voir par exemple [Van der Vaart et al, 1996, p142]*)** *Si $W$ vérifie $N(\epsilon,W,L_2) = O((\frac{1}{\epsilon})^V)$, alors $\log N(\epsilon,\overline{conv}\ W,L_2) = O((\frac{1}{\epsilon})^{\frac{2V}{V+2}})$*

**Corollaire 15.8** *Si $W$ a une VC-dimension finie $V$, alors le logarithme des $\epsilon$-nombres de couverture de $\overline{conv}\ W$ pour $L^2$ est en $O(\epsilon^{-\frac{2V+2}{V+3}})$.*

Un fait remarquable est que l'exposant est plus petit que 2. Ceci implique que l'intégrale d'entropie converge, d'où la nature Donsker de $\overline{conv}\ W$. En outre, des résultats à propos des nombres de couverture (dans [Vidyasagar, 1997] notamment) conduisent à la borne non-asymptotique du théorème 9 (qui n'est *pas* indépendante de la distribution), pourvu que $w$ et $1-w$ soient tous deux minorés par $c$ (cette condition peut être supprimée si l'on considère $\exp L$ au lieu de $L$; en fait, toute fonction de coût Lipschitzienne peut être utilisée, ainsi que quelques autres). Ce résultat découle de [Vidyasagar, 1997, p188], où l'auteur considère directement la minimisation du risque empirique. Les nombres de couverture pour $L_1(\mu)$ nécessitent une connaissance a priori sur la distribution des exemples. Par exemple, les $\epsilon$-nombres de couverture pour $L_1(\mu)$,

pour une densité $\mu$ bornée par $K$, sont bornés par les $\frac{\epsilon}{K}$-nombres de couverture pour $L_1$. Aucun a priori sur la distribution des $Y$ n'est requis ; la seule hypothèse est la loi de la distribution marginale de $X$. Ceci mène au corollaire 10. La dépendance en $\epsilon$ est plus faible que $1/\epsilon^4$. Cette convergence est uniforme sur l'ensemble des distributions vérifiant l'hypothèse sur la densité. Le résultat suivant, non-uniforme, donne une meilleur borne, même si $W$ a une VC-dimension infinie, pourvu que $W$ soit Donsker.

**Théorème 15.9 (Bornes non-asymptotiques issues des nombres de couverture)** *Soit les $x_i$ tirés suivant une densité $\mu$. Soit $N$ le $(\epsilon \times c)$-nombre de couverture de $\overline{conv}\ W$ pour $L_1(\mu)$. Alors, avec probabilité $\geq 1-\delta$, $|L(f_{D_m}) - L_E(f_{D_m})| \leq \epsilon$ si $m \geq \frac{\ln(2N/\delta)}{2c^2\epsilon^2}$, avec $f_{D_m}$ choisi au sein d'une $\epsilon$-couverture (ceci implique une discrétisation).*

**Corollaire 15.10** *Si $W$ a une VC-dimension $V$ finie, alors il existe une constante universelle $M$ telle que, si la densité des exemples est bornée par $K$, alors $|L(f_{D_m}) - L_E(f_{D_m})| \leq \epsilon$ avec probabilité $\geq 1-\delta$ si $m \geq \frac{K^2}{\epsilon^2} \times (M(\frac{K}{\epsilon})^{\frac{2V+2}{V+3}} - 8\log(\delta))$.*

### 15.3.2 Cas 2 : $W$ est Donsker

Le (double) théorème suivant a des corollaires intéressants dans le cas de l'inférence Bayésienne.

**Théorème 15.11 (*voir par exemple [Van der Vaart et al, 1996, p190]*)** *Si $W$ est Donsker, alors l'enveloppe convexe de $W$ est Donsker et la fermeture point-à-point de $W$ est Donsker.*

**Corollaire 15.12** *$\overline{conv}\ W$ est Donsker dès que $W$ est Donsker pour la distribution sous-jacente.*

*conv* $W$ est Donsker car $W$ est Donsker. $\overline{conv}\ W$ est Donsker car *conv* $W$ est Donsker. Ceci implique la convergence faible du coût empirique vers le coût en généralisation en $O(1/\sqrt{m})$. Ce résultat est énoncé dans le corollaire suivant :

**Corollaire 15.13** *Si $W$ est Donsker et si les hypothèses d'intégrabilité (au sens Riemannien) sont vérifiées, alors la convergence faible de la différence entre l'erreur empirique et l'erreur en généralisation vers 0 en $O(1/\sqrt{m})$ a lieu pour l'inférence Bayésienne.*

## 15.4 Inférence Bayésienne : vitesse de convergence

Supposons désormais que $Pr(GC(\epsilon)) \geq K'\epsilon^d$, pour $\epsilon$ suffisamment petit [2], où $GC(\epsilon)$ est l'ensemble des classifieurs $w$ tels que $R(w) \geq R^* - \epsilon$, avec $R^* = \sup_{w \in W} R(w)$. En outre, nous supposerons que

$$\forall (w,x) \in W \times \mathcal{X} \quad 0 < c \leq w(x) \leq 1 - c' < 1$$

Les moyennes empiriques $\frac{1}{m}\sum_{i=1}^{m}(R(Y_i,X_i,w))$ convergent uniformément vers $R(w)$, à vitesse $O(K/\sqrt{m})$ pourvu que $W$ soit Donsker (e.g., $W$ a une VC-dimension finie). L'expression $\inf_{w \in GC(\epsilon)} \Pi_{i=1}^{m} \exp R(Y_i,X_i,w)$ est alors minorée par $\exp(m(R^* - \epsilon)) \times \exp(K\sqrt{m})$, d'où l'on tire l'inégalité (15.5).
D'autre part, avec $\neg GC(\epsilon)$ le complémentaire de $GC(\epsilon)$ dans $W$, l'inégalité (15.6) est vérifiée, ce qui conduit à (15.7), qui est le quotient entre le "poids", dans l'intégrale définissant $f_{D_m}$, des "bons classifieurs" (ceux de $GC(\epsilon)$), et des "mauvais classifieurs" (ceux de $\neg GC(\epsilon)$). Cette relation signifie que les "bons classifieurs" sont beaucoup plus vraisemblables.
$A(\epsilon,m)$ est borné par $\epsilon$ si $\exp(-\epsilon/2)^m \exp(2K\sqrt{m}) \leq \frac{K'}{x^d}\epsilon^{d+1}$. Le logarithme de ceci, divisé par $m$, permet

2. Cette condition peut être relâchée à $Pr(GC(\epsilon)) = \exp(-o(\frac{1}{\epsilon}))$, sans perte significative sur la vitesse de convergence en $O(1/\sqrt{m})$.

d'obtenir l'inégalité 15.8, et donc le théorème 14.

$$\int_{GC(\epsilon)} \Pi_{i=1}^{m} \exp R(Y_i,X_i,w) dPr(w) \geq K'(\exp(R^* - \epsilon/2))^m \exp(-K\sqrt{m})(\frac{\epsilon}{2})^d \tag{15.5}$$

$$\int_{\neg GC(\epsilon)} \Pi_{i=1}^{m} \exp R(Y_i,X_i,w) dPr(w) \leq (\exp(R^* - \epsilon))^m \exp(K\sqrt{m}) \tag{15.6}$$

$$\frac{\int_{\neg GC(\epsilon)} V(w) dPr(w)}{\int_{GC(\epsilon)} V(w) dPr(w)} \leq A(\epsilon,m) = \frac{\exp(\frac{\epsilon}{2})^m \exp(2K\sqrt{m})}{K'(\epsilon/2)^d} \tag{15.7}$$

$$\epsilon/2 \geq \frac{\ln \frac{2^d}{K'}}{m} + \frac{(d+1)\ln\left(\frac{1}{\epsilon}\right)}{m} + 2K/\sqrt{m} \tag{15.8}$$

**Théorème 15.14 (Vitesse de convergence)** *Sous l'hypothèse ci-dessus, l'erreur en généralisation converge vers l'erreur optimale dans $W$ à vitesse $O(1/\sqrt{m})$. Le résultat est asymptotique si $W$ est Donsker. Le résultat est non-asymptotique et indépendant de la distribution si $W$ a une VC-dimension finie.*

## 15.5 Approximation de l'inférence Bayésienne par les algorithmes de Gibbs

On peut considérer les algorithmes de Gibbs comme des approximations de l'inférence Bayésienne par des algorithmes de Monte-Carlo. Nous étudions l'efficacité de ces algorithmes à partir de ce point de vue.

Considérons le premier algorithme de Gibbs:

$$g_{D_m}^k(x) = \frac{1}{k}\sum_{i=1}^{k} w_i(x)$$

Si l'ensemble $W^\perp = \{w \mapsto w(x)/x \in \mathcal{X}\}$ est Donsker, alors la vitesse de convergence est en $O(1/\sqrt{m})$ . La convergence est uniforme en la distribution de $w_i$ pourvu que $W^\perp$ ait une VC-dimension finie. Ceci signifie que $W$ a une VC-codimension finie, ce qui est vrai en particulier si $W$ a une VC-dimension finie[3]. Ceci implique que, sous cette hypothèse de VC-dimension finie, choisir $k = \Theta(m)$ conduit à une convergence aussi rapide que celle de la fonction de coût.

**Théorème 15.15 (Dépendance en $k$ de l'algorithme de Gibbs)** *La convergence de $g_{D_m}^k$ vers $f_{D_m}$ est en $O(1/\sqrt{k})$ et indépendante de la distribution, dès que $W$ a une VC-codimension finie. Ceci est vrai même si $\overline{conv}W$ a une VC-dimension infinie.*

Une conclusion similaire n'a pas pu être établie dans le cas du second algorithme de Gibbs. Si un tel résultat était vrai, son grand intérêt serait la justification formelle d'un algorithme disposant des avantages suivants :

- Algorithme facilement implémentable.
- Complexité en temps : les résultats ci-dessus suggèrent $k = \Theta(m)$, conduisant à $\Theta(m^2)$ évaluations de $w(x_i)$, assurant ainsi une précision $O(1/\sqrt{m})$ sur le coût.
- Convergence asymptotique vers le coût optimal en $O(1/\sqrt{m})$ (ceci n'est pas vérifié par les algorithmes usuels, pour lesquels la borne théorique à minimiser est modifiée dans l'intérêt du temps de calcul, comme c'est le cas pour les Support Vector Machines).

## 15.6 Conclusion et perspectives

Cette partie propose des résultats généraux positifs pour l'inférence Bayésienne : contrôle de la différence entre le coût en généralisation et le coût empirique, bonnes propriétés asymptotiques (convergence vers le

3. On rappelle que la VC-codimension est bornée par $2^V$, lorsque $V$ est la VC-dimension.

coût optimal), bonnes propriétés non-asymptotiques, qui sont indépendantes de la distribution lorsque la VC-dimension est finie, approximation efficace par l'algorithme de Gibbs sous certaines hypothèses raisonnables sur le choix de $k$. Plus généralement, la structure de $\overline{conv}\ W$, la famille de fonctions utilisée par l'inférence Bayésienne, est exprimée comme une classe de Donsker (ce qui permet d'établir des résultats en dehors de l'hypothèse d'indépendance et de distribution identique), dès que $W$ est Donsker. En outre, son $\epsilon$-entropie est explicitement bornée dans le cas d'une VC-dimension finie. Des améliorations possibles pourraient inclure des bornes non-asymptotiques sur certaines classes de distributions, dans l'esprit de [Natarajan, 1988] ou de quelques résultats de [Van der Vaart et al, 1996].

Il aurait été intéressant de pouvoir établir un résultat similaire dans le cas du second algorithme de Gibbs, mais ce problème reste ouvert. En vertu des résultats généraux sur la difficulté de l'apprentissage, on peut craindre qu'un tel résultat soit faux, malheureusement. Néanmoins, cette question est très motivante car le second algorihtme présente le grand avantage d'être plus facile à mettre en pratique, puisqu'il requiert seulement des tirages aléatoires des $w_i$'s selon $Pr$. Un tel algorithme minimiserait réellement la fonction de coût souhaitée, alors que les algorithmes usuels, tels que les Support Vector Machines ou les Bayes Point Machines, contiennent des approximations qui peuvent mener à des comportements indésirables[4].

Une autre forme d'extension à notre connaissance inexistante des résultats positifs ci-dessus pourrait être l'utilisation du théorème 4 et des remarques qui le suivent en vue de l'extension du paradigme bayésien au cas d'espaces de fonctions variant avec le nombre d'exemples (afin d'atteindre la consistance universelle).

Des travaux récents comme [Radulovic et al, 2000] permettent d'étudier de nouvelles propriétés des processus empiriques en tirant parti de différentes hypothèses de régularité sur la distribution des exemples. Il serait certainement intéressant de recommencer l'étude ci-dessus en adoptant ce point de vue. Le principe est d'utiliser, comme approximation de $P$, non pas la somme des masses de Dirac en les $X_i$, mais $\frac{1}{nh}\sum_{i=1}^{n} K(\frac{x-X_i}{h})$ avec $K$ tel que $\int xK(x)dx = 0$, $\int x^2K(x)dx$ et $\int K^2(x)$ soient finis. $h$ est choisi tel que $nh^2 \to \infty$ et $nh^4 \to 0$. Ceci peut se reformuler comme une convolution. Dans le cas général, sans régularité, Yukich [Yukich, 1992] et Van der Vaart [Van der Vaart, 1994] ont montré que cet estimateur était à peu près aussi bon que l'estimateur initial (par masses ponctuelles), en passant, en fait, par des bornes sur la différence entre ces deux estimateurs. Mais Radulovic et Wegkamp [Radulovic et al, 2000] vont plus loin, en montrant que cet estimateur est meilleur dans certains cas, car il permet des résultats positifs dans le cas de familles prégaussiennes, non nécessairement Donsker. En revanche, des conditions plus fortes sont imposées sur la régularité ; dans ce cadre, ils démontrent l'aspect nécessaire et suffisant du caractère prégaussien pour des fonctions indicatrices. En outre, ils soulignent que l'estimateur classique peut échouer sur des familles prégaussiennes non-Donsker. Cet estimateur est donc *supérieur*, et non simplement équivalent, à l'estimateur usuel.

L'hypothèse d'intégrabilité au sens de Riemann peut être considérée comme gênante. Il semble possible de passer outre ces restrictions ou tout du moins d'alléger significativement cette hypothèse.

On peut imaginer aisément trois directions de recherche pour des informaticiens cherchant à tirer profit des classes de Donsker :

- Alors que des efforts importants commencent à être faits pour développer des algorithmes polynomiaux ou à complexité raisonnable permettant d'implémenter des paradigmes issus de la VC-théorie[5], les méthodes du type processus empiriques ont peu influencé l'algorithmique. On pourrait imaginer (comme dans cette partie) la recherche de fonctions de coût différentes de la simple fonction de coût 1 ou 0, concrètement intéressantes, qui aient de bonnes propriétés au niveau des nombres de couverture (exposant du log plus petit que 2), et que l'on puisse optimiser efficacement (en temps polynomial) et/ou avec certitude (convexité stricte de la fonction objectif sur un domaine convexe).
- Etudier expérimentalement l'intérêt de ces résultats, puisque leur nature asymptotique a pour conséquence que rien ne garantit qu'ils soient utilisables aux ordres de grandeur usuels.

4. Les Support Vector Machines sont biaisées par la distance des points à l'hyperplan séparateur, puisque le nombre d'erreurs est remplacé, pour préserver la rapidité du calcul, par une somme distance, et les Bayes Point Machines, pour des raisons similaires, travaillent sur un *moyennage* bayésien au lieu d'une réelle inférence Bayésienne - $f_{D_m}$ est choisi comme étant le centre de l'espace des versions par un algorithme développé dans [Rujan, 1997].

5. les SVMs visant à implémenter la minimisation structurelle du risque reposent sur des fonctions-objectifs strictement convexes sur des domaines convexes, dans la plupart des cas ; la Lp-machine ramène la minimisation à un problème de simplexe, certes exponentiel au pire cas, mais pour lequel des algorithmes efficaces existent ; les réseaux de neurones, intimement liés à la VC-théorie depuis de longues années, ont vu se développer de très nombreux algorithmes optimisant la rétropropagation

– Etudier, dans ce contexte, des extensions classiques en apprentissage : apprentissage avec bruit (points aberrants), cas d'une petite erreur minimale (peut-on alors obtenir des bornes en $1/m$ au lieu de $1/\sqrt{m}$?).

De manière générale, une difficulté récurrente liée à l'application de résultats basés sur les classes de Donsker est l'absence de méthode générale bornant les constantes de la décroissance faible, en $O(1/\sqrt{m})$, du sup des écarts. Etant simples débutants intéressés par l'étude de ces résultats nous ne pouvons que supposer que de tels théorèmes n'existent pas encore. Il semble possible d'aborder ce problème par des inégalités maximales (lemme 2.2.8, dans [Van der Vaart et al, 1996]), ou la loi du logarithme itéré.

Une critique possible des travaux présentés dans cette partie est le fait que nos résultats sont essentiellement asymptotiques. On peut répondre à cette remarque par les considérations suivantes : si la VC-théorie et les bornes non-asymptotiques présentent un grand intérêt pour la validation, avec des applications à l'apprentissage universellement consistant[6], en revanche le choix d'une fonction pour un nombre fixé d'exemples ne peut pas être résolu de manière universelle (pour plus de détails, voir [Wolpert, 1996] ou bien les résultats négatifs de [Devroye et al, 1997]). A moins d'inclure un a priori, aucun algorithme ne l'emportera définitivement. Il nous semble donc raisonable d'utiliser un algorithme quelconque, éventuellement basé sur des résultats asymptotiques, en essayant de rechercher la consistance universelle et de bonnes vitesses de convergence, tout en utilisant des bornes non-asymptotiques pour la validation. En outre, le paradigme Bayésien est optimal pour une distribution connue des éléments sous-jacents. Une telle optimalité n'est pas un atout négligeable.

6. On peut utiliser les bornes non-asymptotiques pour exhiber des algorithmes demandant des exemples supplémentaires jusqu'à temps de fournir un classifieur ou une fonction de régression probablement approximativement corrects

# Bibliographie


[Alon et al, 1997] Alon N., Ben-David S., Cesa-Bianci N., Haussler D. (1997), Scale-sensitive dimensions, uniform convergence and learnability. Journal of the ACM, 44(4):615-631.

[Andersen et al, 1988] Andersen N.T., Giné E., Ossiander M., Zinn J. (1988), The central limit theorem and the law of iterated logarithm for empirical processes under local conditions. Probability theory and related fields 77.

[Bartlett, 1998] Bartlett P.L. (1988), The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE transactions on Information Theory, 44:525-536.

[Birman et al, 1967] Birman M.S., Solomjak M.Z. (1967), Piecewise-polynomial approximation of functions of the classes $W_p$. Mathematics of the USSR Sbornik 73, 295-317.

[Bronstein, 1976] Bronstein E.M. (1976), Epsilon-entropy of convex sets and functions. Siberian Mathematics Journal 17,393-398.

[Devroye et al, 1997] Devroye L., Györfi L., Lugosi G. (1997), A Probabilistic Theory of Pattern Recognition, Springer.

[Donsker, 1951] Donsker M.D. (1951), An invariance principle for certain probability limit theorems. Memoirs of the American Mathematical Society 6.

[Donsker, 1952] Donsker M.D. (1952), Justification and extension of Doob's heuristic approach to the Kolmogorov-Smirnov theorems. Annals of mathematics 23.

[Doob, 1949] Doob J.L. (1949), Heuristic approach to the Kolmogorov-Smirnov theorems. Annals of Mathematical Statistics 20.

[Dudley, 1966] Dudley R.M. (1966), Weak convergence of measures on nonseparable metric spacces and empirical measures on Euclidean spaces. Illinois Journal of Mathematics 10.

[Dudley, 1974] Dudley R.M. (1974), Metric entropy of some classes of sets with differentiable boundaries. Journal of approximation theory 10,(227-236). Correction : Journal of approximation Theory 26 (1979), 192-193.

[Dudley, 1978] Dudley R.M. (1978), Central limit theorems for empirical measures. Annals of probability 6. Correction : Annals of probability 7, (1978) : Théorème centraux limite généralisés basé sur l'entropie uniforme. Equivalents avec la bracketing entropie.

[Dudley, 1984] Dudley, R.M. (1984), A course on empirical processes (Ecole d'été de Probabilités de Saint-Flour XII-1982). Lecture Notes in Mathematics 1097, 2-141 (ed P.L. Hennequin), Springer-Verlag, New-York.

[Efron, 1979] Efron B. (1979) Bootstrap methods: another look at the jackknife. Annals of Stat.,7.

[Van de Geer, 1991] Van de Geer S. (1991), The entropy bound for monotone functions. Report 91-10. University of Leiden.

[Haussler et al, 1994] Haussler D., Kearns M., Schapire R.E. (1994), Bounds on the Sample Complexity of Bayesian Learning Using Information Theory and the VC Dimension. Machine Learning, 14(1):83-113.

[Herbrich et al, 1999] Herbrich R., Graepel T., Campbell C. (1999), Bayes Point Machines: Estimating the Bayes Point in Kernel Space. In *Proc. of IJCAI Workshop on Support Vector Machines, 23-27.*

[Herbrich et al, 2000] Herbrich R., Graepel T., Campbell C. (2000), Robusts Bayes Point Machines. In *Proc. of ESANN 2000, 49-54.*

[Kolmogorov et al, 1961] Kolmogorov A.N., Tikhomirov V.M. (1961), $\epsilon$-entropy and $\epsilon$-capacity of sets in functional spaces. Amer. Math. Soc. Transl. 17, 277-364.

[Lorentz, 1966] Lorentz G.G. (1966), Approximations of functions, Holt, Rhinehart, Winston.

[Natarajan, 1988] Natarajan B.K. (1988), Learning over classes of distributions. In *Proc. of the 1988 Workshop on Computational Learning Theory, pp 408-409, San Mateo, CA, Morgan Kaufmann.*

[Opper et al, 1991] Opper M., Haussler D. (1991), Calculation of the learning curve of Bayes optimal classification algorithm for learning a perceptron with noise. In Computational Learning Theory: Proceedings of the Fourth Annual Workshop, p75-87. Morgan Kaufmann.

[Ossiander, 1987] Ossiander M. (1987) A central limit theorem under metric entropy with $L_2$ bracketing. Annals of probability 15.

[Radulovic et al, 2000] Radulovic D., Weckamp M. (2000), Weak convergence of smoothed empirical processes: beyond Donsker classes. High Dimensional Probability II, E. Gine, D. Mason and J. Wellner, Editors. Birkhauser.

[Rujan, 1997] Ruján P. (1997) Playing Billiard in version space. Neural Computation.

[Teytaud, 2000] Teytaud O. (2000), Bayesian learning/Structural Risk Minimization. Research Report RR-2005, Eric, http://eric.univ-lyon2.fr.

[Solomonoff, 1960] Solomonoff R.-J. (1960), A preliminary report on general theory of inductive inference. Tec. Rep. ZTB-138, Zator Company, Cambridge, MA.

[Strassen et al, 1969] Strassen V., Dudley R.M. (1969), The central limit theorem and $\epsilon$-entropy. Lecture Notes in Mathematics 89, 224-231. Springer-Verlag, New York.

[Sudakov, 1969] Sudakov V.N. (1969), Gauss and Cauchy measures and $\epsilon$-entropy. Doklady Akademii Nauk SSSR 185, 51-53.

[Teytaud et al, 2001] O. Teytaud, H. Paugam-Moisy, Bornes pour l'inférence bayésienne et l'algorithme de Gibbs, actes de CAP 2001.

[Teytaud et al, 2001] O. Teytaud, H. Paugam-Moisy, Bounds on the generalization ability of Bayesian Inference and Gibbs algorithms, Proceedings of Icann 2001.

[Van der Vaart, 1994] Van der Vaart A.W. (1994), Weak convergence of smoothed empirical processes. Scandinavian Journal of Statistics, 21, 501-504.

[Van der Vaart et al, 1996] Van der Vaart A.W., Wellner J.A. (1996), Weak Convergence and Empirical Processes, Springer.

[Vidyasagar, 1997] Vidyasagar M. (1997), A theory of learning and generalization, Springer.

[Wolpert, 1996] Wolpert D.H. (1996) The lack of a priori distinctions between learning algorithms. Neural Computation, 8(7):1341-1390.

[Yukich, 1992] Yukich, J.E. (1992), Weak convergence of smoothed empirical processes. Scandinavian Journal of Statistics, 19, 271-279.

# Chapitre 16

# Comportement asymptotique des support vector machines

**Résumé**

Nous étudions le comportement asymptotique des support vector machines, en exhibant notamment des cas où faire tendre vers $\infty$ le nombre d'exemples n'est pas suffisant pour que le taux d'erreur soit aussi bas que le permettrait l'architecture, et des cas où le nombre de support vectors tend vers $+\infty$. Les méthodes employées sont le passage à la limite d'un problème discret à un problème continu. Differents cas sont distingués: pénalisation des erreurs constante ou variable; noyau polynomial, linéaire ou RBF.

# Table des matières



## 16.1 Introduction

Ce travail a été réalisé en collaboration avec J. Droniou, et est une version étendue d'un papier publié à CAP 1999.

Les support vector machines sont des systèmes d'apprentissage dont on trouvera une description complète dans [Vapnik, 1995] et [Vapnik, 1997]. Le réseau est constitué d'une architecture à deux couches de poids. L'idée générale est de passer à une dimension supérieure de manière à obtenir une représentation interne linéairement séparable, sur laquelle on puisse définir des hyperplans séparateurs "optimaux" (en un sens bien précis) de manière à avoir de bonnes capacités en généralisation. L'article [Osuna et al, 1997] propose des solutions permettant de travailler sur des grandes bases de données, mais admet n'obtenir que des résultats inférieurs à ceux d'une simple rétropropagation dans ces cas-là. Nous nous intéressons au comportement asymptotique des support vector machines et fournissons des éléments qui peuvent expliquer cette infériorité dans le cas d'un grand nombre d'exemples. [Smola, 1996] fournit une extension de l'algorithme SMO défini par Platt au cas de la régression et est aisé à implémenter.

Lorsque le nombre d'exemples est important les support vector machines tendent à développer des réseaux à très grand nombre de neurones. En outre, elles n'approchent pas le taux d'erreur bayésien, lorsque l'on fait tendre le nombre d'exemples vers l'infini. On cherche à préciser ces phénomènes. On soulignera notamment l'effet d'une constante de pénalisation finie; son avantage est que dans le cas de données séparables, lorsque le nombre d'exemples est très grand, elle n'empêche pas la mise en place d'une fonction séparant les données. En outre, elle permet que le problème soit toujours soluble. Par contre, elle a le défaut de ne pas faire converger la fonction générée vers la fonction minimisant le taux d'erreur.

On se préoccupe ici de classification en deux classes; on étudiera le cas des support vector machines linéaires, polynômiales et à noyau RBF. On ne travaille pas dans le cas d'un noyau sigmoïde; il n'existe toujours pas de preuve que l'application $(x,y) \mapsto K(x,y) = tanh(s({}^t x.y) + c)$ vérifie la condition de Mercer pour certaines valeurs de $c$ et $s$.

Introduisons quelques définitions utiles pour la suite de cette étude: un couple $(x,y)$ est dit **bien classé** si $w.x+\theta > 0$ et $y = 1$, ou si $w.x+\theta < 0$ et $y = 0$, et **mal classé** sinon. Un couple $(x,y)$ est dit **strictement bien classé** si $w.x+\theta \geq 1$ et $y = 1$, ou si $w.x+\theta \leq -1$ et $y = 0$. Il est dit **semi-mal classé** si $w.x+\theta \leq 0.5$ et $y = 1$, ou $w.x+\theta \geq -0.5$ et $y = 0$. Enfin l'expression de l'énergie à minimiser en séparée en deux termes:

$$E = \underbrace{\frac{1}{2}w^2}_{\text{Energie basique}} + \underbrace{C\sum_{i=1}^{m}\epsilon_i}_{\text{Pénalisation des erreurs}}$$

Les contraintes sont $wx_i + b \geq 1 - \epsilon_i$ si $y_i = 1$ et $wx_i + b \leq -1 + \epsilon_i$ si $y_i = 0$ et $\forall i \epsilon_i \geq 0$.

On dira que les données sont **séparables avec une marge** $> 0$ s'il existe une fonction $f$ calculée par la support vector machine et un réel $M > 0$ telle que $f(x) > M$ pour toute donnée $x$ de la classe 1 et $f(x) < -M$ pour toute donnée $x$ de la classe $-1$.

On dira que les données sont **séparables avec une marge** 0 s'il existe une fonction $f$ calculée par la support vector machine telle que $f(x) \geq 0$ pour toute donnée $x$ de la classe 1 et $f(x) \leq 0$ pour toute donnée $x$ de la classe $-1$.

On appellera **partie positive** d'une fonction $f$ la fonction $f^+ : x \mapsto max(f(x),0)$.

On rappelle enfin que la fonction $x \mapsto w.x$, dépendant du noyau choisi, s'exprime comme sous la forme $w.x = \sum_{i \in I} \lambda_i y_i K(x_i,x)$ avec les $\lambda_i$ des réels $> 0$, les $x_i$ étant des **support vectors**, extraits de l'ensemble des exemples. On a $w^2 = \sum_{(i,j) \in I^2} \lambda_i \lambda_j y_i y_j K(x_i,x_j)$ (voir [Vapnik, 1995]).

On notera $< .|. >$ le produit scalaire euclidien usuel et $|.|$ la norme usuelle associée.

## 16.2 Dans le cas du noyau polynômial ou linéaire

### 16.2.1 Cas $C$ infini

Ce cas est le cas où l'on enlève le terme de pénalisation des erreurs mais où l'on ajoute la contrainte que pour tout $i$ $\epsilon_i = 0$.

#### Cas où les données peuvent être séparées par une courbe de degré égal au degré du polynôme avec une marge $> 0$

Deux résultats à noter:
a) Si les données peuvent être séparées et si la distribution des exemples est telle que les données soient en position générale dans le "feature space" avec une probabilité 1, le nombre de support vectors est borné; en effet la probabilité est nulle pour que l'on ait plus de $C(n+d,d)$ (voir le livre [Comtet, 1970, vol 1, p 39]) points sur une courbe polynomiale de degré $d$ dans un espace de dimension $n$; le nombre de support vectors est donc majoré par $C(n+d,d)+1$. On supposera par la suite que l'on est dans ce cas-ci.

b) Il existe une fonction séparant les données avec une marge $> 0$, par hypothèse. On peut donc en déduire une majoration de l'énergie minimale $E$, indépendamment du nombre d'exemples. On peut alors constater que la somme des $|\lambda_i|$ est bornée; or le gradient de la somme des $\lambda_i.K(x,x_i)$ est lui-même majoré en norme par cette somme multipliée par une constante. Notons $B$ sa borne; on fixe $\epsilon = 1/B$. Avec une probabilité 1, à partir d'un certain nombre d'exemples, il y a un vecteur exemple dans chaque boule de rayon $\epsilon$ centrée sur un point de la classe 1 ou de la classe 0. Alors, tous les points sont bien classés par la machine, en vertu du théorème des accroissements finis. Donc à partir d'un certain rang, le taux d'erreur est nul en généralisation.

#### Cas où la marge est nulle

Si la distribution des exemples est telle qu'avec une probabilité 1 les données sont en position générale dans le feature space, par les arguments vus au a) ci-dessus, le nombre de support vectors ne peut tendre vers l'infini. De par le lemme annoncé par Vapnik dans [Vapnik, 1995, p 135] ( $E[P_{error}] \leq \frac{E[\text{ nombre de support vectors }]}{\text{nombre de training vectors}-1}$), on peut conclure que l'espérance du taux d'erreur en généralisation va tendre vers 0 (l'espérance est majorée par $\frac{C(n+d,d)+1}{\text{nombre d'exemples}-1}$).

#### Cas où les données ne peuvent pas être séparées par une courbe de degré le degré du polynôme

On montre alors que l'algorithme ne va plus converger à partir d'un certain nombre d'exemples, avec une probabilité égale à 1. En effet, sinon, par le lemme donné par Vapnik cité ci-dessus et puisque le nombre de support vectors est borné pour les mêmes raisons que précédemment, on pourrait conclure que le taux d'erreur en généralisation tend vers 0, ce qui est impossible par hypothèse.

### 16.2.2 Cas $C$ fini

On va travailler avec une SVM linéaire, sans perte de généralité puisque la SVM polynômiale équivaut à une SVM linéaire dans le feature space.

On considère le problème continu, c'est-à-dire (par définition) que l'on considère le problème de la minimisation de l'énergie

$$E_c(C,w) = \frac{w^2}{2} + C. \parallel w \parallel . \int d^+(x)dx^+ + C. \parallel w \parallel . \int d^-(x)dx^-$$

avec $d^+(x) = max(\frac{1-\theta-w.x}{\parallel w \parallel},0)$ et $d^-(x) = max(\frac{w.x+\theta+1}{\parallel w \parallel},0)$. $dx^+$ désigne la densité des exemples de classe 1, $dx^-$ désigne la densité des exemples de classe $-1$.

Ce problème servira à étudier le problème dit problème discret défini comme suit:

$$E_d(C,w,n) = \frac{w^2}{2} + \frac{C. \parallel w \parallel}{n}. \sum_{i=1..n, y_i=1} d^+(x_i) + \frac{C. \parallel w \parallel}{n}. \sum_{i=1..n, y_i=-1} d^-(x_i)$$

Il faut bien noter la division de $C$ par $n$; il ne s'agit pas du problème réel (correspondant à l'algorithme d'apprentissage des support vector machines) qui est:

$$E_r(C,w,n) = \frac{w^2}{2} + C. \parallel w \parallel . \sum_{i=1..n, y_i=1} d^+(x_i) + C. \parallel w \parallel . \sum_{i=1..n, y_i=-1} d^-(x_i)$$

On rappelle tout d'abord le théorème de Glivenko-Cantelli:

**Théorème 16.1** *Soit $(X_n)$ une suite de variables aléatoires indépendantes identiquement distribuées, de loi $P$, $\frac{1n}{\sum}{}^{n}_{i=1} \delta_{X_i}$ ($\delta_{X_i}$ étant un dirac en $X_i$) converge presque sûrement vers $P$ pour la topologie faible-$*$.*

Une preuve peut être trouvée dans [Billingsley, 1986]. Des extensions uniformes de convergence de la moyenne des $f(X_i)$ vers l'espérance de $f$ existent pour des familles données de $f$: en particulier, $f \in F$ avec $F$ de VC-dimension finie, incluant notamment les séparations linéaires dans des espaces de de dimension finie.

Le résultat suivant donne un premier intérêt à ces considérations:

**Proposition 16.2** *Toute suite extraite convergente des suites de solutions aux problèmes discrets $E_d(C,w,n)$ tend vers une solution du problème continu $E_c(C,w)$.*

Il suffit de voir pour cela que

$$E_d(C,w,n) - E_c(C,w) \to 0 \text{ quand } n \to +\infty$$

et que l'on peut se restreindre à un compact pour conclure.□

On définit $E'_c(w) = \parallel w \parallel \int d^+(x)dx^+ + \parallel w \parallel \int d^-(x)dx^-$ [1].

**Lemme 16.3** *Si $w$ dans les problèmes réels est borné, alors les limites de suites extraites convergentes des solutions aux problèmes réels $E_r(C,w,n)$ sont des minima de $E'_c(w)$. Si $w$ dans les problèmes continus est borné, alors les limites de suites extraites des solutions aux problèmes continus $E_c(nC,w)$ sont des minimas de $E'_c(w)$.*

La minimisation de $E_r(C,w,n)$ revient à la minimisation de

$$\frac{w^2}{2} + C \parallel w \parallel \sum_{i=1..n, y_i=1} d^+(x_i) + C \parallel w \parallel \sum_{i=1..n, y_i=-1} d^-(x_i)$$

donc de

$$E'_r(C,w,n) = \frac{\parallel w \parallel}{n} \sum_{i=1..n, y_i=1} d^+(x_i) + \frac{\parallel w \parallel}{n} \sum_{i=1..n, y_i=-1} d^-(x_i) + \frac{w^2}{2Cn}$$

1. Rappelons que $d^+$ et $d^-$ dépendent de $w$.

alors que la minimisation de $E_c(Cn,w)$ revient à celle de

$$E'_c(Cn,w,n) = w\int d^+(x)dx^+ + w\int d^-(x)dx^- + \frac{w^2}{2Cn} = wE'_c(w) + \frac{w^2}{2Cn}$$

Si l'énergie est bornée on peut se restreindre à $w$ dans un ensemble compact ne contenant pas 0; on considère pour la suite que nos énergies ne sont définies que sur cet ensemble.

$$\| E'_r(C,w,n) - E'_c(Cn,w) \| \to 0$$

quand $n \to +\infty$ et

$$\| E'_c(Cn,w) - wE'_c(w) \| \to 0$$

comme $n \to +\infty$ avec $E'_c(w) = \int d^+(x)dx^+ + \int d^-(x)dx^-$ et donc les suites extraites convergentes de la suite $w_n$ des solutions optimales de la minimisation de $E_r(C,w,n)$ ainsi que les suites extraites de la suite $w'_n$ des solutions optimales de la minimisation de $E_c(nC,w)$ tendent vers des solutions optimales de $E'_c(w)$.

Considérons maintenant le cas où $w \to \infty$.

On appelle **séparation** associée à $w$ et $\theta$ le couple $(\frac{w}{\|w\|}, -\frac{\theta}{\|w\|})$.

Par extension on appellera séparation associée à une minimisation en $w$ une séparation associée à un minimum de la fonction à minimiser.

Etant donnée une séparation $s = (e,L)$ on note $E_s = \int max(L- < x|e > ,0)dx^+ + \int max(< x|e > -L,0)dx^-$.

On peut remarquer que dans le lemme suivant on pourrait se restreindre à des sous-suites convergentes au lieu de suites convergentes.

**Lemme 16.4** *Si la norme de $w$ tend vers $+\infty$ pour les problèmes continus $E_c(Cn,w)$ et si la séparation associée à $E_c(Cn,w)$ converge vers une limite donnée pour $n \to \infty$, alors la suite des séparations pour les problèmes continus $E_c(Cn,w)$ tend vers un minimum de $E_s$. Si la norme de $w$ tend vers $+\infty$ pour les problèmes réels $E_r(C,w,n)$ et si la séparation associée à $E_r(C,w,n)$ converge vers une limite donnée pour $n \to \infty$, alors la suite des séparations pour les problèmes réels $E_r(C,w,n)$ tend vers un minimum de $E_s$.*

On se donne $w_n$ une suite de solutions des problèmes réels $E_r(C,w,n)$.

On se donne $w'_n$ une suite des solutions des problèmes continus $E_c(Cn,w)$.

Soit $e$ limite des $\frac{w_n}{\|w_n\|}$.

Alors l'énergie des problèmes réels est égale à

$$\frac{\| w \|^2}{2}$$

plus un équivalent de

$$C\sum_{i=1..n, y_i=1} d^+(x_i) + C\sum_{i=1..n, y_i=-1} d^-(x_i)$$

et donc elle est égale à

$$\frac{\| w \|^2}{2}$$

plus un équivalent de

$$Cn\int d^+(x)dx^+ + Cn\int d^-(x)dx^-$$

(avec $d^+(x) = max(\frac{1-\theta-\|w\|.<e|x>}{\|w\|},0)$ et $d^-(x) = max(\frac{\|w\|.<e|x>+\theta+1}{\|w\|},0)$)

L'énergie des problèmes continus vérifie exactement la même propriété.

On peut réécrire $d^+(x)$ comme la partie positive de $\frac{1}{\|w\|} + L- < e|x >$, avec $L = -\frac{\theta}{\|w\|}$ et $u$ le vecteur directeur de $w$. Par hypothèse, $L$ converge vers un certain $L$. $\| w \|$ tendant vers $+\infty$, $d^+(x)$ tend donc vers la partie positive de $L- < e|x >$; on peut procéder de même avec $d^-(x)$.

Si $L$ et $e$ ne tendent pas vers un minimum de $\int max(L- < x|e > ,0)dx^+ + \int max(< x|e > -L,0)dx^-$, alors pour $w$ assez grand l'énergie serait moindre en changeant $L$ et $e$; donc $L$ et $e$ doivent tendre vers un minimum de cette fonction.

**Cas où les données peuvent être séparées par un polynôme du degré considéré avec une marge $> 0$**

Dans ce cas on n'a pas besoin des considérations ci-dessus, et on peut considérer directement le problème sans que la constante soit divisée par $n$.

Alors on constate les points suivants: • L'énergie est bornée (car il existe une solution indépendante de $n$ pour laquelle tout vecteur est strictement bien classé).

• Le gradient de $x \mapsto w.x$ est donc borné, et donc il suffit qu'un vecteur ne soit pas semi-mal classé pour que tout vecteur à moins d'une certaine distance $\epsilon$ soit bien classé.

• Les exemples sont denses parmi les classes (avec une probabilité 1).

• A partir d'un certain rang $N$ les boules centrées sur les exemples $x_1$, ... ,$x_N$ et de rayon $\epsilon$ couvrent les deux classes, à part peut-être un ensemble de mesure nulle, avec une probabilité 1.

• Si $x_i$, avec $i \leq N$, reste mal classé indéfiniment, alors dans un voisinage autour de $x_i$ on aura par la suite une infinité d'exemples qui seront à leur tour semi-mal classés, et donc l'énergie tendra vers l'infini; d'où contradiction.

On conclut donc qu'à partir d'un certain rang, le taux d'erreur en généralisation est nul. On peut voir aussi que si l'on utilise une constante divisée par $n$, on ne peut garantir que l'on va tendre vers la solution optimale (on peut même montrer qu'éventuellement on ne tendra pas vers cette solution).

**Cas où les données peuvent être séparées par un polynôme du degré considéré avec une marge 0 seulement**

Il semblerait que dans ce cas la marge tend vers 0, et que donc le taux d'erreur tend vers 0, mais ce n'est certain; ci-dessous une preuve, seulement dans le cas où l'on a deux densités uniformes (au moins au voisinage de la séparation): on considère deux classes, l'une sur $[0,\frac{1}{2}] \times [0,1]^q$, l'autre sur $[\frac{1}{2},1] \times [0,1]^q$.

$$E = \frac{2}{M^2} + t.\frac{C}{M}.(d_1.\int_{x-M}^{\frac{1}{2}} u - (x-M).du + d_2.\int_{\frac{1}{2}}^{x+M} x + M - u.du)$$

On dérive par rapport à $M$ et par rapport à $x$, on écrit que les dérivées sont nulles à moins qu'on soit au bord, on fait tendre $t \to +\infty$, on obtient $x = \frac{1}{2}$.

**Cas où les données ne peuvent pas être séparées par une courbe de degré le degré du polynôme**

Il est alors clair que le nombre de support vectors va tendre vers $+\infty$, car tout vecteur mal classé est support vector.

**◇ Cas sans recouvrement de classes**

Il est intéressant de noter que le critère à minimiser n'est pas le taux d'erreur, comme le montre la figure 16.1. Les résultats expérimentaux confirment ces prédictions théoriques (les valeurs sont ici $I = 1$ et $\mathcal{J} = 5$, donc on attend une convergence vers 0.1), comme le montre le tableau 16.1.

| $C$ | Nombre d'exemples | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 40 | 80 | 160 | 320 | 640 | 1280 | 2560 | 5120 |
| 0.4 | 1.0 | 1.0 | 0.18 | 0.16 | 0.24 | | 0.19 | 0.14 | 0.1084 | 0.1030 |
| 3.2 | 1.0 | 1.0 | 0.36 | 0.15 | 0.23 | 0.25 | 0.19 | 0.13 | 0.1064 | 0.1060 |
| 25.6 | 0.16 | 0.34 | 0.17 | 0.15 | 0.23 | 0.25 | 0.19 | 0.13 | 0.1064 | |

TAB. 16.1 – *$x$ en fonction du nombre d'exemples et de $C$, avec les données décrites en figure 16.1.*

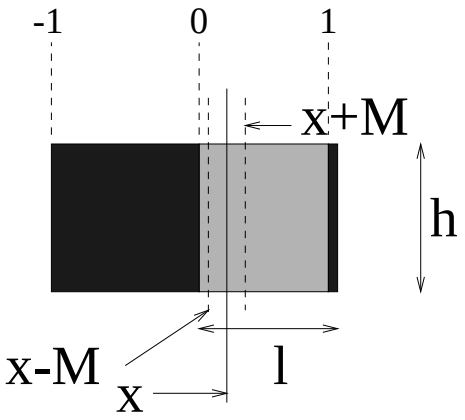



FIG. 16.1 – *Un cas où l'on ne va pas minimiser le taux d'erreur. Les deux larges zones sont supposées uniformes, comprises respectivement entre* $-1$ *et* $0$ *et* $0$ *et* $1$*, avec la même densité l'une que l'autre, égale à* $\mathcal{J}/h$*; la zone à droite, de largeur* $t$*, a elle une densité* $I/(ht)$*.* $t$ *est supposé petit, et* $h = 1$*. Si* $x \leq M$ *l'énergie est égale à* $\frac{2}{M^2} + \frac{Ct}{M}(I.(l - x + M) + \mathcal{J}\int_0^{M-x} u.du + \mathcal{J}\int_0^{x+M}(x + M - u).du)$*, donc après calcul à* $\frac{2}{M^2} + \frac{Ct}{M}(I.(l-x+M)+\mathcal{J}.M^2+\mathcal{J}.x^2)$*. Si* $x \geq M$ *l'énergie est égale à* $\frac{2}{M^2} + \frac{Ct}{M}.[I.(l-x+M)+\mathcal{J}\int_0^{x+M} t.dt]$*, donc à* $\frac{2}{M^2} + \frac{Ct}{M}.(I.(l-x+M)+\mathcal{J}.\frac{x^2}{2}+\mathcal{J}.\frac{M^2}{2}+\mathcal{J}.M.x)$*. Dans le second cas* $M = x$ *en fait, comme on pourra s'en convaincre en consultant la dérivée de l'énergie par rapport à* $M$*; donc soit on est dans le premier cas, soit on est dans le cas tangent* $x = M$*. La dérivée de* $E$ *en fonction de* $M$ *pour la première expression étant négative pour* $M = x$*, on est en fait dans le premier cas. On a donc* $x = \frac{I}{2.\mathcal{J}}$*. Le taux d'erreur ne tend donc pas vers le taux d'erreur minimal possible. Il est à noter que ce résultat est indépendant de* $C$ *et demande seulement que le nombre d'exemples tende vers l'infini.*

◇ **Cas avec recouvrement de classes**

On a montré, sur la figure 16.1, qu'une SVM linéaire est loin d'approximer le taux d'erreur minimal que puisse obtenir un séparateur lorsque la meilleure séparation n'est pas linéaire. La figure 16.4 en donne un nouvel exemple pour lequel, pourtant, la meilleure séparation possible est bel et bien une séparation linéaire. On peut construire de manière similaire un problème en dimension quelconque et pour un polynôme de degré quelconque, en considérant deux classes distribuées avec symétrie sphérique, de manière à ce que l'intégrale de la première distribution soit égale à $\frac{1}{2}$ sur une sphère de rayon $\leq 1$ et 0 ensuite, et à ce que l'intégrale de la deuxième distribution soit égale à 1 sur une sphère de rayon $\leq \frac{1}{2}$ et 0 ensuite. Par symétrie la solution sera nécéssairement une séparation sphérique, donc de degré deux quel que soit le degré de la support vector machine, et on sera dans la même situation que sur la figure 16.4; la séparation sera une sphère trop petite par rapport à la séparation bayésienne.

**Cas où $C = o(1/n)$, indépendamment des positions des classes**

Comme l'explique la figure 16.2, on va dans ce cas, avec une probabilité 1, au bout d'un nombre fini d'exemples, avoir un apprentissage consistant simplement à toujours renvoyer la classe la plus nombreuse.

## 16.3 Dans le cas du noyau RBF

### 16.3.1 Cas $C$ infini

On se place dans le cas où la constante de pénalisation des erreurs est infinie.

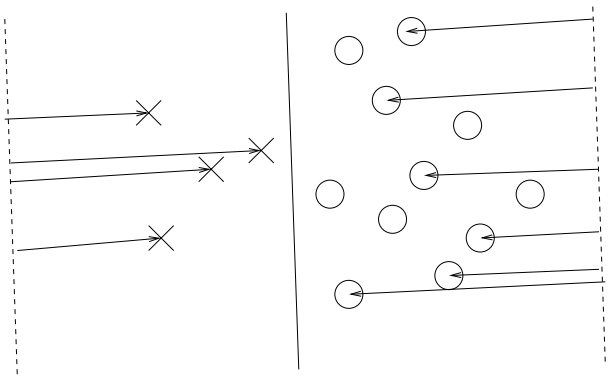


FIG. 16.2 – *La terme de pénalisation des erreurs va être de plus en plus petit à mesure que C décroit, et donc la marge va tendre vers l'infini. On constate alors que la dérivée de l'énergie en fonction de la distance entre l'hyperplan et l'origine du repère (quelle que soit cette origine) est $I - J$ avec $I$ le cardinal de la classe 1 et $J$ le cardinal de la classe $-1$, si l'hyperplan séparateur ne laisse pas les deux classes dans le même demi-espace. Donc la machine va tendre à classer toutes les données dans la classe majoritaire. Différents résultats pratiques confirment cette prédiction théorique; voir par exemple le tableau 16.2 et le tableau 16.1.*

**Classes telles que l'on ait une solution aux contraintes avec une marge $> 0$**

Tout d'abord on se demande comment évolue le taux d'erreur en généralisation. La marge étant toujours $> 0$, l'énergie est bornée, donc la fonction calculée par la support vector machine a une dérivée bornée. Lorsqu'un vecteur $x$ est strictement bien classé, on en déduit donc que dans un rayon $\epsilon$, tous les vecteurs sont bien classés. Or puisque la constante de pénalisation est infinie et puisqu'un ensemble fini est strictement séparable avec une probabilité 1, à chaque étape tous les exemples sont strictement bien classés, et puisqu'à partir d'un certain rang, avec une probabilité 1, les boules de rayon $\epsilon$ et centrées sur les exemples couvrent les classes, à partir d'un certain rang le taux d'erreur est nul.

On va maintenant se préoccuper de l'évolution du nombre de support vectors.

On note $\mathcal{M}(X)$ l'ensemble des mesures réelles finies sur un compact $X$; le théorème de représentation de Riesz permet d'identifier $\mathcal{M}(X)$ au dual topologique de $\mathcal{C}(X)$, espace des fonctions continues de $X$ dans $\mathbb{R}$, muni de la norme sup, définie par

$$\| f \|_\infty = sup_{x \in X} |f(x)|$$

L'ensemble des mesures positives sur $X$ est noté $\mathcal{M}_+(X)$.
Si $Y$ est un fermé de $X$, alors une mesure finie sur $Y$ s'étend naturellement en une mesure finie sur $X$ (en complétant par zéro). Moyennant ce prolongement usuel, on note, lorsque $F$ et $G$ sont deux parties de $X$,

$$\mathcal{M}_{F,G} = \{\lambda = \lambda_F - \lambda_G | \lambda_F \in \mathcal{M}_+(F), \lambda_G \in \mathcal{M}_+(G)\}$$

On se donne $d$ entier $> 0$, et on note $D = [-1,1]^d$. $A$ et $B$ sont des fermés disjoints de $D$. On se donne aussi $\sigma$ appartenant à $\mathbb{R}_*^+$.
Pour $\lambda$ appartenant à $\mathcal{M}(D)$ on définit la fonction $g_\lambda$ de $D$ dans $\mathbb{R}$ par

$$g_\lambda(x) = \int_D e^{-\sigma.|x-y|^2}.d\lambda(y)$$

On se donne une énergie $E$ (qui sera à minimiser) définie sur $\mathcal{M}(D)$ par

$$E(\lambda) = \int_D g_\lambda(x).d\lambda(x)$$

On note $\mathcal{M}^{\delta}_{A,B}$ l'ensemble des éléments de $\mathcal{M}_{A,B}$ qui sont des combinaisons linéaires de mesures de Dirac; tout élément de $\mathcal{M}^{\delta}_{A,B}$ s'écrit donc $\lambda = \sum_{i=1}^{k} \lambda_i.\delta_{x_i}$, avec $\delta_{x_i}$ la mesure de Dirac en $x_i$. Les $x_i$ appartiennent à $A \cup B$, et $\lambda_i$ est positif si $x_i \in A$ et négatif si $x_i \in B$.
On se donne une suite $x_i$ de points dans $A \cup B$. On définit alors $\mathcal{M}^{\delta}_{A,B,g}(p)$ comme l'ensemble des mesures $\lambda$ de $\mathcal{M}^{\delta}_{A,B}$ dont le support est inclus dans $X_p = \{x_1,...,x_p\}$ et telles qu'il existe $\theta$ tel que $x \in X_p \cap A \Rightarrow g_\lambda(x) \geq -\theta + 1$ et $x \in X_p \cap B \Rightarrow g_\lambda(x) \leq -\theta - 1$.

On se pose les trois questions suivantes, que l'on appellera *problème discret*:
• $\mathcal{M}^{\delta}_{A,B,g}(p)$ est-il non vide?
• Si oui, existe-t-il $\lambda \in \mathcal{M}^{\delta}_{A,B,g}(p)$ qui minimise $E$ sur $\mathcal{M}^{\delta}_{A,B,g}(p)$.
• Si oui, ce $\lambda$ est-il unique?
• Si oui, le nombre de support vectors, c'est à dire de $\lambda_i$ non nuls, tend-il vers l'infini?
La première question a une réponse affirmative, car [Cover, 1965] montre que $n$ points en dimension $n$ sont toujours linéairement séparables (il suffit donc de considérer un espace de dimension $n$ contenant tous les points dans le feature space). La deuxième et la troisième questions auront aussi des réponses affirmatives, comme on le verra plus loin.

Pour répondre à ces questions, on va introduire un autre problème, comme cas limite, pour $p$ tendant vers l'infini, du problème discret.
Le *problème continu* consiste à minimiser $E$ sur

$$\mathcal{M}_{A,B,g} = \{\lambda \in \mathcal{M}_{A,B} \mid \theta\,,\, g_\lambda \geq -\theta + 1 \text{ sur } A\,,\, g_\lambda \leq -\theta - 1 \text{ sur } B\}$$

On se pose ici aussi les trois questions:
• $\mathcal{M}_{A,B,g}$ est il non vide?
• Si oui, existe-t-il $\lambda \in \mathcal{M}_{A,B,g}$ qui minimise $E$?
• Si oui, est-il unique?

Nous verrons que, dans le cas où la première réponse est oui, alors il existe effectivement un unique minimum de $E$ sur $\mathcal{M}_{A,B,g}$. Cependant, le fait que $\mathcal{M}_{A,B,g}$ soit non vide n'est pas, dans le cas général, évident.
Nous étudierons enfin l'evolution du nombre de support vectors dans un cas particulier (où le problème continu est non vide et ou ce nombre de support vectors tend vers l'infini).

Nous allons maintenant montrer quelques lemmes préliminaires.

**Lemme 16.5** *Si $\lambda \in \mathcal{M}(D)$, alors*

$$E(\lambda) = \int_{D\times D} e^{-\sigma|x-y|^2}\, d\lambda(x)\, d\lambda(y) \tag{16.1}$$

$$= \sum_{n\geq 0} \frac{(2\sigma)^n}{n!} \int_D \int_D \left(\sum_{i=1}^{d} x_i y_i\right)^n e^{-\sigma|x|^2} e^{-\sigma|y|^2}\, d\lambda(x)\, d\lambda(y) \tag{16.2}$$

*En particulier, si $\lambda \neq 0$, $E(\lambda) > 0$ et $E$ est strictement convexe sur $\mathcal{M}(D)$.*

**Démonstration:**
La première formule pour $E$ résulte d'une simple application du théorème de Fubini sur les définitions de $E$ et $g_\lambda$.
On écrit ensuite le développement en série entière de l'exponentielle:

$$e^{-\sigma|x-y|^2} = e^{-\sigma|x|^2} e^{-\sigma|y|^2} e^{2\sigma\langle x,y\rangle} = \sum_{n\geq 0} \frac{(2\sigma)^n}{n!} \left(\sum_{i=1}^{d} x_i y_i\right)^n e^{-\sigma|x|^2} e^{-\sigma|y|^2}$$

et la convergence est uniforme en $(x,y) \in D^2$.

On peut donc intervertir série et intégrale dans la définition de $E$ pour trouver

$$E(\lambda) = \sum_{n\geq 0} \frac{(2\sigma)^n}{n!} \int_D \int_D \left( \sum_{i=1}^d x_i y_i \right)^n e^{-\sigma|x|^2} e^{-\sigma|y|^2} \, d\lambda(x) \, d\lambda(y) \tag{16.3}$$

Mais, par la formule du multinôme,

$$\left( \sum_{i=1}^d x_i y_i \right)^n = \sum_{k_1+\cdots+k_d=n} \frac{n!}{k_1! \cdots k_d!} x^k y^k$$

où on a noté, pour un $d$-uplet $k = (k_1, \ldots, k_d)$ et un point $x \in \mathbb{R}^d$, $x^k = x_1^{k_1} \cdots x_d^{k_d}$. Cela nous donne, associé à (16.3),

$$\begin{aligned} E(\lambda) &= \sum_{n\geq 0} \frac{(2\sigma)^n}{n!} \left( \sum_{k_1+\cdots+k_d=n} \frac{n!}{k_1! \cdots k_d!} \int_D \int_D x^k e^{-\sigma|x|^2} y^k e^{-\sigma|y|^2} \, d\lambda(x) \, d\lambda(y) \right) \\ &= \sum_{n\geq 0} \frac{(2\sigma)^n}{n!} \left( \sum_{k_1+\cdots+k_d=n} \frac{n!}{k_1! \cdots k_d!} \left( \int_D x^k e^{-\sigma|x|^2} \, d\lambda(x) \right)^2 \right), \end{aligned}$$

c'est à dire la deuxième formule pour $E$.

Cette dernière écriture nous permet de voir que $E(\lambda)$, en tant que série de termes positifs ($\sigma$ est strictement positif), est positif et ne peut être nul que si chacun des termes

$$\sum_{k_1+\cdots+k_d=n} \frac{n!}{k_1! \cdots k_d!} \left( \int_D x^k e^{-\sigma|x|^2} \, d\lambda(x) \right)^2$$

est nul, i.e. si pour tout multi-indice $k \in \mathbb{N}^d$ on a

$$\int_D x^k e^{-\sigma|x|^2} \, d\lambda(x) = 0$$

Dans ce cas, cela signifie que la mesure $e^{-\sigma|.|^2}\lambda$ est nulle (en tant qu'élément de $(\mathcal{C}(D))'$) sur tout polynôme; les polynômes formant un ensemble dense dans $\mathcal{C}(D)$ (théorème de Stone-Weierstrass), on en déduit que $e^{-\sigma|.|^2}\lambda = 0$, i.e. que $\lambda = 0$.

Pour voir enfin que $E$ est strictement convexe, c'est assez simple: on prend $(\lambda,\mu) \in \mathcal{M}(D)$ différentes et $t \in ]0,1[$, puis on cherche à voir si (définition de la stricte convexite):

$$E(t\lambda + (1-t)\mu) < tE(\lambda) + (1-t)E(\mu) \tag{16.4}$$

En notant $b(\lambda,\mu) = \int_{D\times D} e^{-\sigma|x-y|^2} \, d\lambda(x) \, d\mu(y)$ la forme bilinéaire symétrique dont $E$ est la forme quadratique, montrer (16.4) revient à voir que

$$tE(\lambda) + (1-t)E(\mu) - (t^2E(\lambda) + 2t(1-t)b(\lambda,\mu) + (1-t)^2E(\mu)) > 0$$

i.e. que

$$t(1-t)(E(\lambda) - 2b(\lambda,\mu) + E(\mu)) > 0$$

Mais $E(\lambda) - 2b(\lambda,\mu) + E(\mu) = E(\lambda - \mu) > 0$ par le caratère défini positif de $E$, et (16.4) est donc prouvé. □

**Corollaire 16.6** *Il existe $C > 0$ tel que, pour tout $\lambda \in \mathcal{M}_{A,B}$, $E(\lambda) \geq C||\lambda||^2$.*

**Démonstration:**

Nous démontrons ce résultat par l'absurde, en faisant l'hypothèse que, pour tout $n \geq 1$, il existe $\lambda^{(n)} \in \mathcal{M}_c$ telle $E(\lambda_n) < \frac{1}{n}||\lambda_n||^2$.

Considérons alors $\mu^{(n)} = \frac{1}{||\lambda_n||}\lambda_n \in \mathcal{M}_c$; comme $(\mu^{(n)})_{n\geq 1}$ est bornée dans $\mathcal{M}(D)$, on peut supposer, quitte à extraire une suite, que $\mu^{(n)} \to \mu$ pour la topologie faible-$*$ de $\mathcal{M}(D)$.

$E$ étant homogène de degré 2 (cf. (16.1)), on a $E(\mu^{(n)}) < 1/n$ et $E(\mu^{(n)})$ tend donc vers 0 lorsque $n \to \infty$; la formule (16.2) nous permet de voir que, pour tout $k \in \mathbb{N}^d$, il existe $\alpha_k > 0$ tel que

$$\forall \lambda \in \mathcal{M}(D)\,,\, E(\lambda) \geq \alpha_k \left( \int_D x^k e^{-\sigma|x|^2}\, d\lambda \right)^2 .$$

Cette inégalité, associée au fait que $E(\mu^{(n)})$ tend vers 0 lorsque $n \to \infty$, nous donne

$$\langle \mu^{(n)}, e^{-\sigma|x|^2} x^k \rangle_{\mathcal{M}(D),\mathcal{C}(D)} \to 0 \text{ lorsque } n \to \infty.$$

$\mu^{(n)}$ tendant vers $\mu$ dans $\mathcal{M}(D)$-faible-$*$, on en déduit que la mesure $e^{-\sigma|.|^2}\mu$ s'annule sur toutes les fonctions de la forme $x \to x^k$, donc sur tous les polynômes et sur toutes les fonctions continues sur $D$ (par densité des polynômes dans $\mathcal{C}(D)$), i.e. que $\mu$ est la fonction nulle.

Prenons maintenant $f \in \mathcal{C}(D)$ qui vaut 1 sur $A$ et $-1$ sur $B$ (c'est possible car $A$ et $B$ sont des fermés disjoints de $D$). Comme $\mu^{(n)} \in \mathcal{M}_c$ et $\mu^{(n)}$ tend vers 0 dans $\mathcal{M}(D)$-faible-$*$, on a (avec les notations introduites lors de la définition de $\mathcal{M}_c$)

$$\langle \mu^{(n)}, f \rangle_{\mathcal{M}(D),\mathcal{C}(D)} = \mu_A^{(n)}(A) + \mu_B^{(n)}(B) = ||\mu^{(n)}||_{\mathcal{M}(D)} \to 0,$$

ce qui est une contradiction puisque les $\mu^{(n)}$ sont de norme 1.□

**Lemme 16.7** *Si $\lambda^{(n)} \to \lambda$ dans $\mathcal{M}(D)$ pour la topologie faible-$*$, alors $g_{\lambda^{(n)}} \to g_\lambda$ dans $\mathcal{C}(D)$, pour la topologie forte. En particulier, $E(\lambda^{(n)}) \to E(\lambda)$.*

**Démonstration:** Comme $(\lambda^{(n)})_{n\geq 1}$ converge pour la topologie faible-$*$ de $\mathcal{M}(D)$, cette suite est bornée (par un certain $M > 0$), et on sait donc ($e^{-\sigma|.|^2}$ étant $2\sigma\sqrt{2d}$-lipschitzienne sur $D$) que, pour tout $n \geq 1$, $g_{\lambda^{(n)}}$ est $2\sigma\sqrt{d}M$-lipschitzienne sur $D$; le théorème d'Ascoli-Arzela nous permet alors de voir que la suite $(g_{\lambda^{(n)}})_{n\geq 1}$ est relativement compacte dans $\mathcal{C}(D)$.

Mais, pour tout $x \in D$,

$$g_{\lambda^{(n)}}(x) = \langle \lambda^{(n)}, e^{-\sigma|x-.|^2} \rangle_{\mathcal{M}(D),\mathcal{C}(D)} \to \langle \lambda, e^{-\sigma|x-.|^2} \rangle_{\mathcal{M}(D),\mathcal{C}(D)} = g_\lambda(x)$$

(par convergence faible de $\lambda^{(n)}$ vers $\lambda$); ainsi, la suite $(g_{\lambda^{(n)}})_{n\geq 1}$ relativement compacte a une seule valeur d'adhérence, à savoir $g_\lambda$, et converge donc vers cette valeur d'adhérence.

Pour la convergence des énergies, on écrit simplement, puisque $g_{\lambda^{(n)}} \to g_\Lambda$ dans $\mathcal{C}(D)$ et $\lambda^{(n)} \to \Lambda$ dans $\mathcal{M}(D)$-faible-$*$,

$$E(\lambda^{(n)}) = \langle \lambda^{(n)}, g_{\lambda^{(n)}} \rangle_{\mathcal{M}(D),\mathcal{C}(D)} \to \langle \Lambda, g_\Lambda \rangle_{\mathcal{M}(D),\mathcal{C}(D)} = E(\Lambda).$$

□

**Proposition 16.8 (Approximation de mesures par des masses de Dirac bien réparties)** *Soit $K$ un compact de $\mathbb{R}^d$ et $\mathcal{D}$ un ensemble dénombrable dense dans $K$; on note*

$$\mathcal{M}_{\mathcal{D}} = \{ \sum_{i=1}^{p} \alpha_i \delta_{x_i}\,,\, p \geq 1\,,\, (\alpha_1,\ldots,\alpha_p) \in \mathbb{R}^+\,,\, (x_1,\ldots,x_p) \in \mathcal{D} \}$$

*Pour tout $\mu \in \mathcal{M}_+(K)$, il existe une suite $(\mu^{(n)})_{n\geq 1} \in \mathcal{M}_{\mathcal{D}}$ qui converge vers $\mu$ pour la topologie faible-$*$ de $\mathcal{M}(K)$.*

**Démonstration:** Durant cette démonstration, on raisonne avec la norme sup sur $\mathbb{R}^d$, i.e. $||x|| = \sup_{i\in[1,d]} |x_i|$.

On se donne $n \geq 1$, et on découpe $K$ en un nombre fini de cubes (dans $K$) disjoints de cotés inférieur à $1/n$, par exemple en écrivant

$$K = \bigcup_{k \in \mathbb{N}^d} K \cap \prod_{i=1}^{d} \left[ \frac{k_i}{n}, \frac{k_i+1}{n} \right[$$

et en ne conservant que les cubes qui rencontrent $K$ (en nombres finis puisque $K$ est compact); on a donc

$$K = \bigcup_{l=1}^{L_n} K_{l,n}$$

avec $\mathrm{diam}(K_{l,n}) \leq \frac{1}{n}$.

$\mathcal{D}$ étant dense dans $K$ et $K_{l,n} \subset K$ étant non vide, il existe, pour tout $l \in [1,L_n]$, un point $x_{l,n} \in \mathcal{D}$ situé à une distance inférieure à $\frac{1}{n}$ de $K_{l,n}$.

On pose alors

$$\mu^{(n)} = \sum_{l=1}^{L_n} \mu(K_{l,n}) \delta_{x_{l,n}} \in \mathcal{M}_{\mathcal{D}}$$

Nous allons voir que $\mu^{(n)} \to \mu$ dans $\mathcal{M}(K)$ pour la topologie faible-$*$.

Pour cela, on se donne $\phi \in \mathcal{C}(K)$; par uniforme continuité de $\phi$, il existe, pour tout $\varepsilon > 0$, un $\eta_\varepsilon > 0$ tel que

$$\forall (x,y) \in K^2 \, , \, ||x-y|| \leq \eta_\varepsilon \Longrightarrow |\phi(x) - \phi(y)| \leq \varepsilon$$

On prend alors $n_0 \geq 2/\eta_\varepsilon$ et on remarque que, pour tout $n \geq n_0$,

$$\forall l \in [1,L_n] \, , \, \forall x \in K_{l,n} \, , \, ||x - x_{x_{l,n}}|| \leq \frac{2}{n} \leq \eta_\varepsilon$$

ce qui nous permet d'écrire

$$\begin{aligned} \left| \int_{K_{l,n}} \varphi(x) \, d\mu(x) - \varphi(x_{l,n}) \mu(K_{l,n}) \right| &= \left| \int_{K_{l,n}} (\varphi(x) - \varphi(x_{l,n})) \, d\mu(x) \right| \\ &\leq \int_{K_{l,n}} |\varphi(x) - \varphi(x_{l,n})| \, d\mu(x) \\ &\leq \varepsilon \, \mu(K_{l,n}) \end{aligned}$$

On en déduit donc, les $(K_{l,n})_{l \in [1,L_n]}$ formant une partition de $K$,

$$\begin{aligned} \left| \int_K \varphi(x) \, d\mu(x) - \int_K \varphi(x) \, d\mu(n)(x) \right| &= \left| \sum_{l=1}^{L_n} \int_{K_{l,n}} \varphi(x) \, d\mu(x) - \sum_{l=1}^{L_n} \varphi(x_{l,n}) \mu(K_{l,n}) \right| \\ &\leq \sum_{l=1}^{L_n} \varepsilon \mu(K_{l,n}) = \varepsilon \mu(K) \end{aligned}$$

ce qui conclut cette démonstration. □

**Théorème 16.9 (Existence et Unicité des Solutions aux problèmes discrets et continus)** *Soient $F$ et $G$ deux fermés disjoints non vides de $D$ tels que $\mathcal{M}_{F,G,g} \neq \emptyset$. Il existe alors un unique $\Lambda \in \mathcal{M}_{F,G,g}$ qui minimise $E$ sur $\mathcal{M}_{F,G,g}$.*

**Démonstration:**

L'unicité vient du fait que l'on minimise $E$ strictement convexe (lemme 5) sur l'ensemble convexe $\mathcal{M}_{F,G,g}$.

Pour l'existence, la technique employée est très standard (méthode directe). On prend une suite $\lambda^{(n)} \in \mathcal{M}_{F,G,g}$ telle que $E(\lambda^{(n)}) \to \inf_{\mathcal{M}_{F,G,g}} E$; par le lemme 6, la suite $(\lambda^{(n)})_{n\geq 1}$ est bornée dans $\mathcal{M}(D)$ et, quitte à en extraire une suite, on peut supposer que $\lambda^{(n)} \to \Lambda$ dans $\mathcal{M}(D)$ pour la topologie faible-$*$.

On voit alors simplement que $\Lambda$ est dans $\mathcal{M}_{F,G}$ (il suffit de constater que la convergence faible-$*$ de $\lambda^{(n)}$ dans $\mathcal{M}(D)$ est équivalente à la convergence faible-$*$ de $\lambda_F^{(n)}$ dans $\mathcal{M}(F)$ et de $\lambda_G^{(n)}$ dans $\mathcal{M}(G)$).

Par le lemme 7, on a $E(\lambda^{(n)}) \to E(\Lambda)$; il suffit donc, $E(\lambda^{(n)})$ tendant vers $\inf_{\mathcal{M}_{F,G,g}} E$, de voir que $\lambda \in \mathcal{M}_{F,G,g}$ pour en déduire que $\Lambda$ est bien un minimum de $E$ sur $\mathcal{M}_{F,G,g}$.

Pour cela, on prend $\theta_n$ correspondant au fait que $\lambda^{(n)} \in \mathcal{M}_{F,G,g}$, i.e. tels que

$$\theta_n \geq 1 - g_{\lambda^{(n)}} \text{ sur } F\,,\ \theta_n \leq -1 - g_{\lambda^{(n)}} \text{ sur } G. \tag{16.5}$$

On constate de plus que $g_{\lambda^{(n)}}$ est bornée dans $\mathcal{C}(D)$ (elle converge uniformément vers $g_\Lambda$); ainsi, $F$ et $G$ étant non vides, on en déduit que $(\theta_n)_{n\geq 1}$ est bornée dans $\mathbb{R}$, et converge donc à une sous-suite près vers $\theta \in \mathbb{R}$. Un simple passage à la limite dans (16.5) nous permet donc de voir que $\Lambda \in \mathcal{M}_{F,G,g}$ car

$$\theta \geq 1 - g_\Lambda \text{ sur } F\,,\ \theta \leq -1 - g_\Lambda \text{ sur } G.$$

□

Ce résultat nous permet déjà de répondre aux trois premières questions du problème discret. En effet, avec les notations précédentes, on a $\mathcal{M}^\delta_{A,B,g}(p) = M_{A\cap X_p, B\cap X_p, g}$ avec $A \cap X_p$ et $B \cap X_p$ fermés disjoints de $D$; les résultats de [Cover, 1965] nous disent que cet ensemble est non-vide, et qu'il existe donc un unique $\Lambda^{(p)}$ qui minimise $E$ sur cet ensemble.

On voit aussi que, dans le cas où $\mathcal{M}_{A,B,g}$ est non vide, il existe aussi un unique minimum de $E$ sur cet ensemble, minimum que nous noterons $\Lambda^\infty$.

**Proposition 16.10 (Le problème continu comme limite du problème discret)** *Supposons que $M_{A,B,g}$ soit non vide et que $(x_i)_{i\geq 1}$ soit dense dans $A \cup B$. Alors, avec les notations précédentes, $\Lambda^{(p)}$ tend vers $\Lambda^\infty$ dans $\mathcal{M}(D)$-faible-$*$.*

**Démonstration:** Comme $(x_i)_{i\geq 1}$ est dense dans $A \cup B$, et puisque $A$ et $B$ sont à distance strictement positive, $\{x_i\,,\ i \geq 1\} \cap A$ est dense dans $A$ et $\{x_i\,,\ i \geq 1\} \cap B$ est dense dans $B$. En appliquant la proposition 8 aux deux compacts $A$ et $B$ et aux mesure positives $\Lambda_A^\infty$ et $\Lambda_B^\infty$, on constate qu'il existe une suite $(\mu^{(n)})_{n\geq 1} \in \mathcal{M}^\delta_{A,B}$, dont le support est inclus dans $\{x_i\,,\ i \geq 1\}$, qui converge vers $\Lambda^\infty$ dans $\mathcal{M}(D)$ faible-$*$.

On sait alors que $g_{\mu^{(n)}} \to g_{\Lambda^\infty}$ uniformément sur $D$; ainsi, $a_n = \inf_A g_{\mu^{(n)}} \to a = \inf_A g_{\Lambda^\infty}$ et $b_n = \sup_B g_{\mu^{(n)}} \to b = \sup_B g_{\Lambda^\infty}$ (où $\theta$ est un réel qui traduit que $\Lambda^\infty \in \mathcal{M}_{A,B,g}$).

Pour $n$ assez grand, $a_n - b_n > 0$ (car $a - b \geq 2$, $\Lambda^\infty$ étant dans $\mathcal{M}_{A,B,g}$); on peut définir

$$\nu = \frac{a-b}{a_n - b_n}\mu^{(n)}.$$

$(\nu^{(n)})_{n\geq 1}$ tend alors vers $\Lambda^\infty$ dans $\mathcal{M}(D)$ faible-$*$ lorsque $n \to \infty$ ($\frac{a-b}{a_n-b_n} \to 1$ lorsque $n \to \infty$); notons $k_n$ un entier tel que $\nu^{(n)}$ ait son support dans $X_{k_n} = \{x_1, \ldots, x_{k_n}\}$. On a

$$\inf_A g_{\nu^{(n)}} - \sup_B g_{\nu^{(n)}} = \frac{a-b}{a_n - b_n}(a_n - b_n) = a - b \geq 2;$$

en prenant $\theta = 1 - a$, on en déduit que $g_{\nu^{(n)}} \geq -\theta + 1$ sur $A$ (donc sur $A \cap X_p$ pour tout $p \geq k_n$) et $g_{\nu^{(n)}} \leq -\theta - 1$ sur $B$ (donc sur $B \cap X_p$ pour tout $p \geq k_n$).

On a donc trouvé $\nu^{(n)} \in \mathcal{M}^\delta_{A,B,g}(p)$ $(\forall p \geq k)$ qui converge vers $\Lambda^\infty$ dans $\mathcal{M}(D)$-faible-$*$, ce qui donne entre autres

$$E(\nu^{(n)}) \to E(\Lambda^\infty). \tag{16.6}$$

Or, par définition de $\Lambda^{(p)}$,

$$E(\nu^{(n)}) \geq E(\lambda^{(p)}) \tag{16.7}$$

pour tout $p \geq k_n$, donc $(E(\Lambda^{(p)}))_{p\geq 1}$ est une suite bornée et, comme $E(.) \geq C||.||^2_{\mathcal{M}(D)}$ sur $\mathcal{M}_{A,B}$, on en déduit que $(\Lambda^{(p)})_{p\geq 1}$ est une suite bornée de $\mathcal{M}(D)$.

On peut donc supposer, quitte à extraire une suite, que $\Lambda^{(p)} \to \lambda$ dans $\mathcal{M}(D)$ faible-$*$; il n'est alors pas difficile de voir que $\lambda \in \mathcal{M}_{A,B}$.

Notons $\theta_p$ un réel tel que

$$g_{\Lambda^{(p)}} \geq -\theta_p + 1 \text{ sur } A \cap X_p \,,\, g_{\Lambda^{(p)}} \leq -\theta_p - 1 \text{ sur } B \cap X_p. \tag{16.8}$$

Comme $(g_{\Lambda^{(p)}})_{p\geq 1}$ est bornée dans $\mathcal{C}(D)$ (elle converge uniformément vers $g_\lambda$), on en déduit que $(\theta_p)_{p\geq 1}$ est bornée dans $\mathbb{R}$ et converge donc, à une sous-suite près, vers $\theta$. En passant à la limite $p \to \infty$ dans (16.8), on voit que $g_\lambda(x_i) \geq -\theta + 1$ pour tout $i \in \mathbb{N}$ tel que $x_i \in A$ et $g_\lambda(x_i) \leq -\theta - 1$ pour tout $i \in \mathbb{N}$ tel que $x_i \in B$; $g_\lambda$ étant une fonction continue sur $D$ et $\{x_i \,,\, i \geq 1\} \cap A$ et $\{x_i \,,\, i \geq 1\} \cap B$ étant respectivement denses dans $A$ et $B$, cela nous permet de voir que $g_\lambda \geq -\theta + 1$ sur $A$ et $g_\lambda \leq -\theta - 1$ sur $B$. $\lambda$ est donc dans $\mathcal{M}_{A,B,g}$.

Passer à la limite $p \to \infty$ dans (16.7) nous donne $E(\nu^{(n)}) \geq E(\lambda)$, et passer à la limite $n \to \infty$ dans (16.6) nous donne donc $E(\Lambda^\infty) \geq E(\lambda)$; comme $\Lambda^\infty$ est l'unique minimum de $E$ sur $\mathcal{M}_{A,B,g}$ et comme $\lambda \in \mathcal{M}_{A,B,g}$, on en déduit $\Lambda^\infty = \lambda$.

On a donc montré qu'une sous-suite de $(\Lambda^{(p)})_{p\geq 1}$ converge vers $\Lambda^\infty$ dans $\mathcal{M}(D)$-faible-$*$. Le raisonnement ci-dessus pouvant être effectué en partant de n'importe quelle sous-suite de $(\Lambda^{(p)})_{p\geq 1}$, on en déduit le résultat du théorème.□

**◇ Le cas particulier boule/couronne**

On considère maintenant deux fermés $A$ et $B$ bien particuliers; avec $d \geq 2$ et $a \in D$ et $0 < r_0 < r_1 < r_2 < d(a,D^c)$, on définit:

$$A = \{x/d(x,a) \leq r_0\}$$
$$B = \{x/d(x,a) \in [r_1,r_2]\}$$

On va travailler sur le cas continu tout d'abord. On peut énoncer:
• $\mathcal{M}_c$ contient une fonction satisfaisant les contraintes (considérer simplement une gaussienne centrée en $a$, multipliée par un coefficient suffisant)
• Les hypothèses de la partie précédente sont vérifiées
• Le minimum de $E$ parmi les solutions est atteint et est unique
• La solution en question, étant unique, est invariante par rotation autour de $a$

Les résultats de la partie précédente permettent en outre d'assurer que la solution du problème discret tend vers la solution du problème continu pour $p \to +\infty$, du moment que l'on a choisi les points $\{x_i \,,\, i \in \mathbb{N}\}$ bien répartis dans $A \cup B$; on se demande donc maintenant si le nombre de support vectors peut ne pas tendre vers l'infini. Dans ce cas la solution en l'infini ne comporte qu'un nombre fini de diracs; or vu l'invariance par rotation, dans ce cas il n'y a qu'un dirac en l'infini, situé en $a$, ce qui est impossible.
Dans le cas qui nous occupe, le nombre de support vectors tend donc vers l'infini.□

On remarque en outre que la même méthode s'applique à d'autres cas où l'on a les mêmes propriétés d'existence d'une infinité de symétries (cylindres...).

### Cas où les données ne peuvent être séparées

**◇ Cas sans recouvrement de classes**

On ne sait en fait pas si ce cas peut se présenter. Il suffirait pour montrer que non de montrer que pour tout $\sigma$ l'ensemble des fonctions de la forme suivante:

$$f(x) = \sum_{i\in[1,n]} \lambda_i e^{-\sigma\|x-x_i\|}$$

pour $n \in \mathbb{N}$ et $x_i$ dans $D$

est dense pour la topologie forte dans l'ensemble des fonctions continues de $D$ dans $\mathbb{R}$. Les résultats de densité sur ce genre de cas utilisent des gaussiennes à largeur variable, ou bien se contentent de convergence dans $\mathcal{L}^p$ pour tout $p$ fini, ce qui ne permet pas de conclure ici.

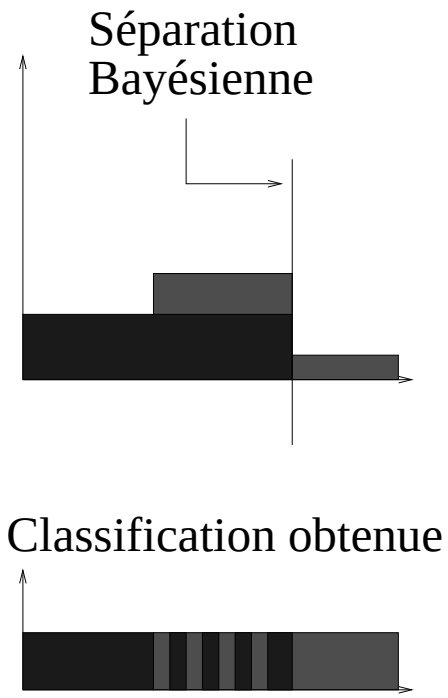



FIG. 16.3 – *En haut, le problème de classification; il y a deux classes A et B, dont les densités sont représentées par les hauteurs des pavés correspondants. La séparation la meilleure est simplement la séparation linéaire, mais la support vector machine trace une frontière compliquée, rendant compte de chaque exemple. La constante est ici infinie, ce qui est bien sûr une mauvaise idée dans les cas pratiques.*

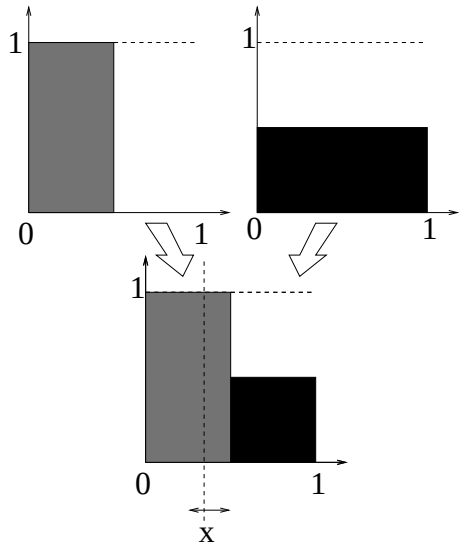



FIG. 16.4 – *L'énergie est égale à* $\frac{2}{M^2} + t.\frac{C}{M}.(\int_{x-M}^{\frac{1}{2}} u - (x-M).du + \int_0^{x+M} \frac{x+M-u}{2}.du)$. *On écrit que la dérivée par rapport à $M$ est nulle, que la dérivée par rapport à $x$ est nulle, on fait tendre $t \to +\infty$, et on obtient que $x = \frac{5}{12}$, soit environ 0.4; donc le taux d'erreur ne tend pas vers le minimum. Il faut noter que quelle que soit la valeur de $C$ on ne peut tendre vers $x = \frac{1}{2}$.*

◇ **Cas avec recouvrement de classes**

Tout ensemble de points peut être explosé, et la constante $C$ étant infinie, la SVM va toujours séparer les examples. Sur la figure 16.3 on a représenté un problème avec sa meilleure séparation, la séparation bayésienne; la support vector machine, avec un nombre fini d'exemples, ne peut pas renvoyer cette séparation (du moins elle a une probabilité 1 de ne plus jamais la renvoyer à partir d'un certain nombre d'exemples), mais plutôt une séparation qui satisfasse chaque exemple, lorsque $C$ est infini. Par contre, il est possible que la mesure de l'ensemble des vecteurs qui ne sont pas envoyés dans la classe majoritaire tende vers 0 lorsque le nombre d'exemples tend vers l'infini (simple hypothèse, vraisemblable au vu des résultats pratiques).

### 16.3.2 Cas $C$ fini

**Cas où les données peuvent être séparées avec une marge $> 0$**

L'énergie est alors bornée. Le gradient de la fonction calculée par la support vector machine est donc borné; donc il suffit qu'un vecteur $x$ soit bien classé avec $d^+(x) < 0.5$ si $x$ est dans la classe 1 et $d^-(x) < 0.5$ si $x$ est dans la classe $-1$ pour que dans un rayon $\epsilon$ autour de $x$ les vecteurs soient tous bien classés. Or

pour $n$ assez grand, avec une probabilité 1, les boules de rayon $\epsilon$ situées autour des $n$ premiers exemples recouvrent les classes. Et si les vecteurs en question ne verifient jamais la propriété demandée sur les $d^+$ et $d^-$, alors l'énergie tend vers $+\infty$, car avec une probabilité 1 il y aura un nombre arbitrairement grand de vecteurs suffisamment près de $x$ pour avoir un $d^+$ ou un $d^-$ supérieur à 0.4.
Donc à partir d'un certain rang le taux d'erreur en généralisation est 0 avec une probabilité 1.

**Cas où les données ne peuvent être séparées**

N'ayant pu faire mieux, on va ici travailler sur un problème très simplifié, simplement en guise de premier pas. On se donne une petite zone où les classes se recouvrent, pour considérer ce qu'il se passe sur cette zone. On suppose qu'un vecteur dans cette zone a une probabilité $p$ d'être dans la classe 1 et une probabilité $1-p$ d'être dans la classe $-1$. On va considérer le problème continu, et donc travailler sur $\lambda$, mesure, et sur $g$, sa convoluée.
On suppose en outre que la zone est suffisamment petite pour que les effets de bords soient négligeables; ainsi $\lambda$ et $g_\lambda = g$ sont constants.

$$E = \int g(x)\lambda(x)dx + C.p.(-\theta + 1 - g(x)) + C.(1-p).(g(x) + \theta + 1)$$

$$= \int g(x)\lambda(x)dx - 2.C.p.g(x) + C.g(x) + C - \theta.C.(2.p - 1)$$

avec la contrainte que $g \leq -\theta + 1$ et $g \geq -\theta - 1$.
On a alors, si aucune des contraintes sur $\theta$ n'est active, une dérivée en $\theta$ dépendant du signe du $p - \frac{1}{2}$; donc $-\theta$ est égal à $g+1$ ou $g-1$, de manière à ce que tous les exemples soient classés dans la classe majoritaire.

Les résultats expérimentaux de la table 16.2 ont été obtenus avec une distribution en deux classes, présentes l'une à 70 %, et l'autre à 30 %. En abscisse on a le nombre d'exemple, et en ordonnées la constante de pénalisation des erreurs. Le pourcentage donné représente la surface que la support vector machine attribue à la classe majoritaire (donc plus l'on est proche de 100 et plus la support vector machine est efficace en généralisation).

| | 10 | 20 | 40 | 80 | 160 | 320 | 640 | 1000 | 1280 | 2560 |
|---|---|---|---|---|---|---|---|---|---|---|
| C=0.001 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| C=0.01 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| C=0.1 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| C=1 | 93.5 | 88.2 | 93.4 | 100 | 99.8 | 99.9 | 100 | 100 | 100 | 100 |
| C=10 | 87.4 | 87.3 | 81.3 | 89.2 | 95.7 | 97.1 | 100 | 100 | 100 | 100 |
| C=100 | 87.4 | 82.7 | 74.2 | 80.4 | 86.5 | 97.2 | 99.9 | 100 | 100 | 100 |
| C=1000 | 87.4 | 80.0 | 62.3 | 66.9 | 87.1 | 97.6 | 99.0 | 100 | 100 | 100 |
| C=10000 | 87.4 | 78.5 | 58.5 | 62.6 | 80.1 | 97.3 | 98.7 | 100 | 100 | 100 |

TAB. 16.2 – *Pourcentage de vecteurs classés dans la classe majoritaire.*

## 16.4 Conclusion

On a donc montré que dans le cas polynômial pour le nombre d'exemples tendant vers l'infini:
• les support vectors machines n'approximent pas nécéssairement le taux d'erreur minimal lorsque les données ne sont pas séparables par le modèle considéré, indépendamment de $C$.
• par contre elles l'atteignent lorsque les données peuvent être séparées par le modèle avec marge $> 0$.
• si elles sont séparables avec une marge nulle, alors le taux d'erreur tend vers 0 si la constante est infinie;

si la constante de pénalisation des erreurs est finie, un résultat partiel suggère que c'est aussi le cas (pas de preuve dans le cas général).

Et dans le cas RBF pour le nombre d'exemples tendant vers l'infini:
• si les classes sont séparables avec une marge $> 0$, alors on atteint le taux d'erreur minimal (c'est à dire 0), quel que soit $C$
• si les classes ne sont pas séparables et si $C$ est infini, alors on n'atteint jamais le taux d'erreur minimal, par contre il semblerait qu'on l'approxime; mais ce n'est qu'une conjecture
• si les classes ne sont pas séparables et si $C$ est fini, il semblerait qu'on approxime le taux d'erreur minimal (pour peu que des conditions minimales de régularité soient vérifiées), et même qu'on l'atteigne si la distance entre la zone où la densité de la classe 1 est strictement supérieure à la densité de la classe $-1$ et la zone où la densité de la classe $-1$ est strictement supérieure à la densité de la classe 1 est strictement positive. Un bien petit résultat théorique moyennant des hypothèses trop fortes semble corroborer cette conjecture.

En outre on montre que dans le cas où le problème vérifie certaines contraintes de symétrie, alors le nombre de support vectors tend vers l'infini, pour peu que la constante de pénalisation des erreurs soit infinie. Dans le cas d'une constante finie, il semblerait en fait que le nombre de support vectors est en fait plus élevé.

Le choix de la constante $C$ est un problème intéressant.
• Choisir $C$ infini est avantageux au niveau de l'architecture générée, mais coûteux en temps de calcul dans le cas du kernel RBF et conduisant à des contraintes non satisfiables lorsque les classes ne sont pas séparables dans le cas du kernel polynômial.
• Choisir $C$ constant (ou ne décroissant pas au moins en $1/n$) est avantageux au niveau du temps de calcul, mais lorsque le nombre d'exemples tend vers l'infini, le résultat est indépendant de $C$, et n'est pas forcément le meilleur par rapport au taux d'erreur en généralisation.
• Remplacer $C$ par $C/n$ a des avantages; on converge alors vers la solution du problème continu, et le coût en temps de calcul est bon. Par contre il se peut que dans certains cas, alors que l'algorithme aurait convergé vers la solution optimale avec $C$ constant, indépendamment de la valeur de $C$, on a maintenant le problème du choix de $C$, et l'on peut avoir gâté le résultat par rapport à $C$ constant.
• Remplacer $C$ par un $o(1/n)$ conduit la support vector machine à classer toutes les données dans la classe majoritaire.
Nb: lorsqu'on parle de convergence vers un taux d'erreur nul, ou d'atteindre un taux d'erreur nul, il s'agit du cas général parmi les distributions des exemples, c'est à dire que cela arrive dans un cas de mesure 1; il est certes des tirages d'exemples (par exemple indéfiniment le même exemple...) qui ne permettent pas d'avoir ce résultat; simplement ce cas est de mesure nulle. Atteindre un taux d'erreur nul, signifie que si on utilise suffisamment d'exemples, on obtient un taux d'erreur nul en généralisation.

# Bibliographie

[Billingsley, 1986] P. Billingsley, *Probability and Measure. Wiley and Sons, New York, second edition, 1986.*
[Comtet, 1970] L. Comtet, *Analyse combinatoire, Presses universitaires de France, 1970*
[Cover, 1965] T. M. Cover, *Geometrical and Statistical Properties of Systems of Linear Inequalities in Pattern Recognition, IEEE transactions on electronic computers, 1965*
[Osuna et al, 1997] E. Osuna, R. Freund, F. Girosi, *An Improved Training Algorithm for Support Vector Machines, proc of IEEE NNSP'97, Amelia Island, FL, 1997*
[Smola, 1996] A. Smola, *Regression Estimation with Support Vector Learning Machine, Master's Thesis, Technische Universität München, 1996*
[Vapnik, 1995] V.N. Vapnik, *The Nature of Statistical Learning, Springer, 1995*
[Vapnik, 1997] V.N. Vapnik, *Support Vector Learning machines. Tutorial at NIPS*97, Denver (CO), 1997*

# Chapitre 17

# Plus proches voisins rapides

**Résumé**

Ce chapitre traite d'algorithmes de $k$ plus proches voisins. Ce paradigme est très ancien, mais a toujours des avantages forts qui le rendent très important:

- Consistance universelle forte dans $\mathbb{R}^d$ pourvu qu'une astuce est mise en œuvre dans le cas d'égalités parmi les distances (voir [Devroye et al, 1997] pour plus d'informations).
- Simplicité.
- Possibilité de travailler avec seulement des distances.

Un nouvel algorithme approximé rapide est proposé, et des (trop peu de) résultats pratiques sont donnés.

# Table des matières



## 17.1 Algorithmes de $k$-plus proches voisins

Etant donné un ensemble $x_1$, ... , $x_n$ d'exemples dans $\mathbb{R}^d$, et un élément $x$ dans le même espace, on cherche les $k$ éléments $x_{n_1}$, $x_{n_2}$, ..., $x_{n_k}$ qui sont les plus proches de $x$ parmi les $x_i$. Les $x_{n_i}$ n'ont pas besoin d'être ordonnés.

### 17.1.1 L'algorithme naïf

Une solution simple est la suivante:
• Pour tout $i \in [1,n]$
• • $dist(i) = \| x_i - x \|$
• Trouver par l'algorithme de la section A la liste des $k$ plus petits éléments de $dist$.
La complexité de cet algorithme est en $O(n)$. Comme l'algorithme a souvent à être utilisé avec un grand nombre de $x$ différents, ceci est beaucoup trop lent, et de meilleurs algorithmes ont été trouvés.

### 17.1.2 L'algorithme des arbres KD

#### Le principe

Cet algorithme est le plus usuel pour cette tâche. Le principe est le suivant:
• Tout d'abord, construisez l'arbre dans lequel sont placés les $x_i$ (méthode décrite plus bas).
• Alors, chercher le plus proche voisin de $x$ dans cet arbre.

#### La structure de l'arbre KD

Un arbre KD est un arbre binaire. Le noeud principal d'un arbre binaire représentant les $x_i$ a les éléments suivants:
• Un point chef, $x_{i_0}$, choisi parmi les $x_i$.
• Une direction privilégiée $i \in [1,d]$, avec $d$ la dimension de l'espace dans lequel sont les $x_i$'s.
• Deux arbres KD, un pour les $x_i$ qui ont une $d^e$ composante plus grande que $x_{i_0}$, et un pour les autres.

• $max_i x_{ij}$ et $min_i x_{ij}$ pour $j \in [1,d]$ (ceci signifie, un hyperrectangle dans lequel sont tous les $x_i$).
• Eventuellement, les coordonnées d'une sphère dans laquelle se trouvent les $x_i$. Nous ne détaillerons pas ceci ici.

**La construction de l'arbre KD.**

La structure fournit immédiatement un algorithme récursif pour construire un arbre KD. Le seul problème est le choix d'une dimension privilégiée, et d'un point chef.

Le choix d'un point chef est facile; comme on souhaite équilibrer l'arbre autant que possible, on va choisir l'élément médian des $x_i$ par rapport à la composante $j^e$, avec $j$ la direction privilégiée. Le choix de la direction privilégiée peut être fait de deux façons usuelles:
• [Omohundro, 1987] suggère qu'une bonne dimension est celle qui maximise la variance.
• [Moore, 1991] suggère qu'une autre bonne solution est celle qui maximise l'étendue des $x_i$.

Pour notre propos, la seconde solution sera considérée comme la meilleure, comme il fournit des régions plus "carrées", ce qui va accroître la vitesse de la recherche des plus proches voisins.

**La recherche des $k$-plus proches voisins**

[Moore, 1991] fournit seulement un algorithme pour le plus proche voisin. Nous suggérons ici une extension au cas des $k$ plus proches voisins.
• Soit $Dist$ égal à $\infty$.
• Soit $L$ la liste réduite au noeud chef.
• Tant que $L$ est non vide
• • Extrait le dernier élément $l$ de $L$
• • Si $l$ contient moins de $k$ points,
• • • ajouter ces points à la liste $F$,
• • Sinon
• • • Notons $KD1$ le fils de $l$ qui contient $x$, $KD2$ l'autre fils.
• • • Ajoutez $KD2$ et $KD1$ dans $L$, si leurs rectangels ont une intersection non-vide avec la sphère de centre $x$ et de rayon $Dist$, dans cet ordre (afin que le prochain élément extrait soit $KD1$)
• • Si $F$ a plus de $k$ éléments, garder seulement dedans les $k$ éléments les plus proches de $x$, et soit $Dist$ la distance entre $x$ et le $k^e$ plus proche élément de $x$ parmi les éléments de $F$.

A la fin de cet algorithme, $F$ doit contenir les $k$ éléments les plus proches de $x$.

**Résultats théoriques**

Il y a différents papiers donnant des résultats sur la complexité de cet algorithme, mais il n'y a pas de borne générale rigoureuse. Un survol de quelques résultats peut être trouvé dans [Omohundro, 1987]. On peut noter que les temps de calcul sont supposés augmenter très vite comme $d$ augmente.

### 17.1.3 Nanabozo: un nouvel algorithme basé sur le clustering

**Principes**

Le principe, comme dans les arbres KD, sera l'utilisation de structures spatiales, de manière à ce que les points soient organisés en un arbre. Cet algorithme ne sera pas exact; les $k$ voisins choisis ne sont pas nécesairement les $k$ plus proches. Les expérimentations pratiques montrent que les résultats ne sont pas nécessairement moins bons que ceux des "rigoureux" $k$-plus proches voisins.

La structure est un arbre, dont les noeuds sont de la forme suivant, pour un ensemble $x_i$:
• Un chef $x$, parmi les $x_i$.
• $N$ arbres, chacun correspondant à un sous-ensemble des $x_i$.

Les éléments des différents arbres sont sélectionnés par n'importe quel algorithme de clustering.

Le nombre $N$ d'arbres peut être prédéterminé, par exemple avec l'algorithme des $k$-means (partie C), ou choisi, par exemple par un simple clustering comme en partie B (cet algorithme pourrait être utilisé même sans données manquantes).

**L'algorithme**

Premièrement, l'algorithme utilisé pour construire l'arbre:

• Si le cardinal des $x_i$ est plus petit qu'une constante donnée (par exemple $k$ lui-même),

• • Retourner l'arbre avec les $x_i$ comme feuilles, et les plus centrés parmi eux comme chef.

• sinon

• • Construire une partition $E_1$, $E_2$, ... ,$E_N$ des $x_i$.

• • Construisez $T_1$, ... ,$T_N$ les arbres correspondant à $E_1$, ... ,$E_N$ respectivement.

• • Soit $C_1$, ... , $C_N$ les chefs respectifs de ces arbres.

• • Retourner l'arbre avec les $T_i$ comme fils et le plus centré des $C_i$ comme chef.

Et maintenant, l'algorithme utilisé pour trouver $k$ proches voisins (qui ne sont pas nécessairement optimaux!); nous supposons que $w$ est le point dont nous cherchons les voisins.

• Soit $L$ la liste réduite au chef de l'arbre, soit $List$ la liste vide, et soit $Dist$ $\infty$.

• Tant que $L$ est non vide,

• • Considérons $l$ le dernier élément de $L$.

• • Si $l$ n'a pas d'arbre comme fils,

• • • alors il a quelques $x_i$ pour fils; placer certains des $x_i$ qui sont à distance $< Dist$ de $w$ dans $List$.

• • Sinon, avec $T_1$, ... ,$T_n$ les fils,

• • • Placer dans $L$ la liste des $T_i$ dont les chefs sont à distance $< Dist$ de $w$.

• • Si $L$ a taille $> k$ alors garder seulement dans $L$ les $k$ arbres qui ont les chefs les plus proches de $w$.

• Retourner les $k$ éléments de $List$ qui sont les plus proches de $w$.

**Borne sur la complexité théorique de la recherche**

Avec $C$ le nombre de fils de chaque noeud, supposant que l'arbre est équilibré (hypothèse certes hasardeuse!), la complexité est bornée par $C \times k \times d \times log_C(n)$.

## 17.2 Expérimentations pratiques

Par la suite, on appelle "algorithme Nanabozo" l'algorithme défini précédemment. Ce nom a été choisi en l'honneur de Oumpah-Pah; voir [Uderzo et al, 1958] pour plus de détails.

### 17.2.1 Sur app3e4

Cette base de donnée (simulée) a été fourni par Elf Antar France; il s'agit d'une tâche de classification, avec 15174 points pour apprendre et 15174 points pour tester (choisis au hasard parmi les 30348 points de l'ensemble). Il se trouve en dimension 4, avec deux classes.

($k$-PP=$k$ plus proches voisins, la backpropagation est une backpropagation simple sans weight decay)

| Algorithme | Paramètres | Temps d'éxécution | Taux de succès |
|---|---|---|---|
| $k$-pp, algo. naïf | $K = 1$ | 112836.69 | 0.976540 |
| $k$-pp avec arbre KD | $K = 1$ | 72293.73 | 0.976540 |
| $k$-pp, Nanabozo | $K = 1$,$N = 1$ | 4184.05 | 0.794003 |
| $k$-pp, Nanabozo | $K = 1$,$N = 5$ | 4485.12 | 0.934959 |
| $k$-pp, Nanabozo | $K = 1$,$N = 11$ | 6486.37 | 0.940494 |
| $k$-pp, Nanabozo | $K = 1$,$N = 21$ | 11094.08 | 0.965404 |
| $k$-pp, Nanabozo | $K = 1$,$N = 31$ | 13388.26 | 0.968567 |
| $k$-pp, algo. naïf | $K = 3$ | 110753.19 | 0.978649 |
| $k$-pp avec arbre KD | $K = 3$ | 108328.92 | 0.978649 |
| $k$-pp, Nanabozo | $K = 3$,$N = 1$ | 11462.32 | 0.947611 |
| $k$-pp, algo. naïf | $K = 5$ | 116482.48 | 0.980033 |
| $k$-pp, Nanabozo | $K = 5$,$N = 1$ | 17849.30 | 0.955255 |
| $k$-pp, Nanabozo | $K = 5$,$N = 31$ | 21253.76 | 0.970280 |
| $k$-pp, Nanabozo | $K = 7$,$N = 31$ | 26394.00 | 0.972521 |
| Backpropagation (une couche cachée) | 15 neurones sur la couche cachée | 23970.75 | 0.976804 |
| Backpropagation (une couche cachée) | 25 neurones sur la couche cachée | 32455.66 | 0.976474 |

Le résultat est tracé sur le graphique 17.1.

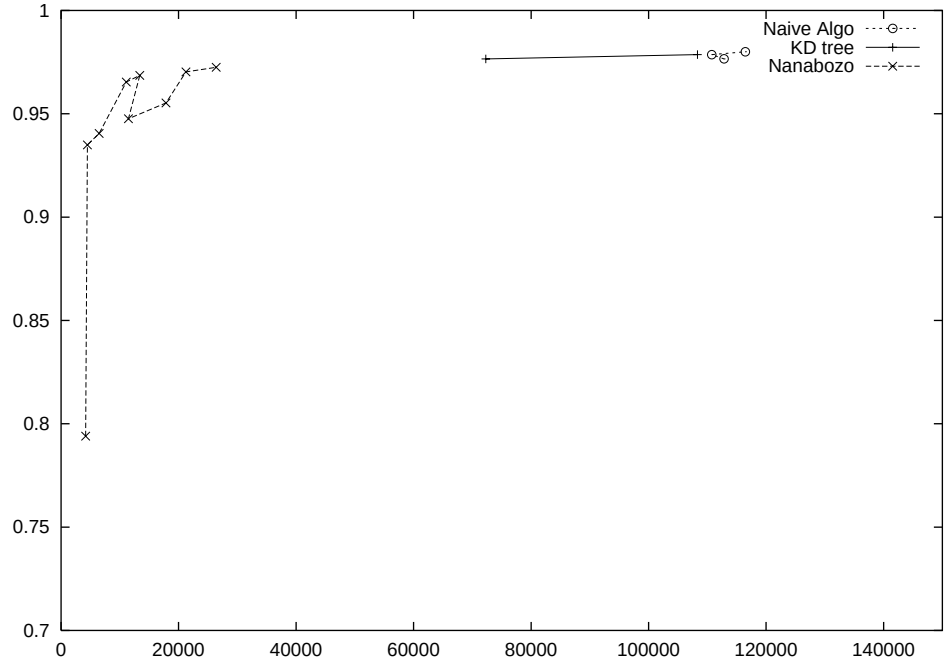



FIG. 17.1 – *Comparaison entre les différents plus proches voisins sur app3e4. Abscisse: le temps d'éxécution, ordonnée le taux de succès.*

### 17.2.2 Sur Isolet

Cette base de donnée peut être trouvée à l'URL `http://axon.cs.byu.edu/~martinez/classes/470/MLDB/`; le titre complet est Isolated Letter Speech Recognition. Il y a 26 classes, 6238 instances dans l'ensemble d'apprentissage, 1559 dans l'ensemble test. La dimension est 617.

Les créateurs sont Ron Cole et Mark Fanty, Department of Computer Science and Engineering, Oregon Graduate Institute, Beaverton, OR 97006, cole@cse.ogi.edu, fanty@cse.ogi.edu.

On peut trouver à l'URL ci-dessus des liens vers des études sur ce problème. La table suivante est extraite de la même URL et donne des résultats dessus:

| Algorithme et configuration | erreurs | % erreur | % correct |
|---|---|---|---|
| Opt 30-bit ECOC | 51 | 3.27 | 96.73 |
| Opt 62-bit ECOC | 63 | 4.04 | 95.96 |
| Opt OPC | 65 | 4.17 | 95.83 |
| C4.5 107-bit ECOC soft pruned | 103 | 6.61 | 93.39 |
| C4.5 92-bit ECOC soft pruned | 107 | 6.86 | 93.14 |
| C4.5 45-bit ECOC soft pruned | 109 | 6.99 | 93.01 |
| C4.5 107-bit ECOC soft raw | 116 | 7.44 | 92.56 |
| C4.5 92-bit ECOC soft raw | 118 | 7.57 | 92.43 |
| C4.5 107-bit ECOC hard pruned | 126 | 8.08 | 91.91 |
| C4.5 92-bit ECOC hard pruned | 127 | 8.15 | 91.85 |
| C4.5 62-bit ECOC soft pruned | 131 | 8.40 | 91.60 |
| C4.5 30-bit ECOC soft pruned | 134 | 8.60 | 91.40 |
| C4.5 62-bit ECOC soft raw | 134 | 8.60 | 91.40 |
| C4.5 77-bit ECOC hard pruned | 138 | 8.85 | 91.15 |
| C4.5 45-bit ECOC soft raw | 145 | 9.30 | 90.70 |
| C4.5 30-bit ECOC soft raw | 175 | 11.23 | 88.77 |
| C4.5 30-bit ECOC hard pruned | 185 | 11.87 | 88.13 |
| C4.5 multiclass soft pruned | 239 | 15.33 | 84.67 |
| C4.5 multiclass soft raw | 248 | 15.91 | 84.09 |
| C4.5 multiclass hard pruned | 254 | 16.29 | 83.71 |
| C4.5 15-bit ECOC soft pruned | 259 | 16.61 | 83.39 |
| C4.5 multiclass hard raw | 264 | 16.93 | 83.07 |
| C4.5 OPC soft pruned | 296 | 18.99 | 81.01 |
| C4.5 15-bit ECOC soft raw | 321 | 20.59 | 79.41 |
| C4.5 107-bit ECOC hard raw | 334 | 21.42 | 78.58 |
| C4.5 92-bit ECOC hard raw | 349 | 22.39 | 77.61 |
| C4.5 OPC soft raw | 379 | 24.31 | 75.69 |
| C4.5 15-bit ECOC hard pruned | 383 | 24.57 | 75.43 |
| C4.5 77-bit ECOC hard raw | 424 | 27.20 | 72.80 |
| C4.5 OPC hard pruned | 437 | 28.03 | 71.97 |
| C4.5 62-bit ECOC hard raw | 463 | 29.70 | 70.30 |
| C4.5 OPC hard raw | 519 | 33.29 | 66.71 |
| C4.5 45-bit ECOC hard raw | 568 | 36.43 | 63.57 |
| C4.5 30-bit ECOC hard raw | 617 | 43.04 | 56.96 |
| C4.5 15-bit ECOC hard raw | 991 | 63.57 | 36.43 |

Légende:

- OPT = conjugate-gradient implementation of backprop.
- C4.5 = Quinlan's C4.5 system, Release 1.
- OPC = one-per-class representation
- ECOC = error-correcting output code
- raw = unpruned decision trees
- pruned = pruned decision trees (CF=0.25)
- hard = default trees
- soft = trees with softened thresholds.
- multiclass = one tree to do all 26-way classifications.

Avec l'algo. $k$-pp, on a les résultats suivants:

| $k$ | Taux de succès | Temps écoulé |
|---|---|---|
| 21 | 0.921103 | 124349.29 |

Et avec l'algorithme Nanabozo:

| $k$ | $N$ | Taux de succès | Temps écoulé |
|---|---|---|---|
| 21 | 15 | 0.915972 | 19849.02 |
| 21 | 25 | 0.918538 | 23589.11 |
| 21 | 35 | 0.910199 | 23442.59 |
| 25 | 15 | 0.908916 | 24368.33 |
| 25 | 25 | 0.912123 | 24707.26 |
| 25 | 35 | 0.911482 | 25909.77 |
| 29 | 15 | 0.914047 | 27642.34 |
| 29 | 25 | 0.917896 | 28163.32 |
| 29 | 35 | 0.910199 | 28655.31 |
| 33 | 15 | 0.915972 | 33966.73 |
| 33 | 25 | 0.916613 | 32532.81 |
| 33 | 35 | 0.909557 | 31543.34 |
| 37 | 15 | 0.911482 | 45170.00 |
| 37 | 25 | 0.910840 | 37946.31 |
| 37 | 35 | 0.910199 | 30765.82 |
| 41 | 15 | 0.913406 | 207016.14 |
| 41 | 25 | 0.914047 | 45414.75 |

## 17.3 Conclusion

Nous avons présenté un algorithme alternatif de $k$-plus proches voisins. Il est rapide mais approché. Dans certains cas théoriques (exemples jouets construits exprès), on peut montrer qu'en fait la convergence (de l'erreur en généralisation vers l'erreur bayésienne) est meilleure avec un tel algorithme qu'avec les plus proches voisins traditionnels, du moins pour des choix de $k$ donnés; une supériorité de cet algorithme sur l'algorithme traditionnel est toutefois improbable sur des données réalistes, comme montré sur les exemples traités. Le réel apport au niveau de la vitesse rend néanmoins l'approche non-inintéressante.

# Bibliographie

[Devroye et al, 1997] L. Devroye, L. Györfi, G. Lugosi, A probabilistic Theory of Pattern Recognition, Springer, 1997.
[Moore, 1991] A. Moore, *PhD. Thesis: Efficient Memory-based Learning for Robot Control, technical report No 209, Computer Laboratory, University of Cambridge, 1991*
[Omohundro, 1987] S.M. Omohundro, *Efficient algorithms with neural network behaviour. Journal of complex systems, 1(2):273-347,1987*
[Uderzo et al, 1958] A. Uderzo, R. Goscinny, *Oumpah-Pah le peau-rouge, 1958*

# Annexe A

# Algorithm to select the $k$ largests of $n$ elements

The precise problem is the selection of the $k^th$ largest and of the $k$ largests in same time. Let $a_1$, ... ,$a_n$ be considered. One can simply use the following algorithm:

- If $n = 0$ or $n = 1$, the algorithm is finished.
- select an element $a$ of the $a_i$'s (randomly, or the median of a few elements...)
- Change the order of the $a_i$'s so that the $n'$ elements bigger are at the beginning and the other at the end of the list.
- If $n' > k$, do the same thing, by induction, on the $n'$ first elements. Otherwise, do the same thing on the elements after the $n'$ firsts, with $k' = k - n'$.

At the end of this algorithm, the $k^{th}$ element is the $k^{th}$ largest, and the first $k - 1$ elements are smaller than it. This algorithm has average cost $log(n)$.

# Annexe B

# A classical simple clustering algorithm

This algorithm is extracted from "Algorithmic Learning", from Alan Hutchinson, Clarendon Press 1994. It is interesting as a simple example of clustering algorithm.

## B.1 The algorithm

We suppose that $n$ examples, $x_1$, $x_2$, $x_3$, ..., $x_n$ are given.
• Compute the list of distances between the $x_i$'s.
• Sort this list in increasing order; let's call $L$ the resulting list.
• Creation of $n$ clusters, each one including only one example.
• For each $l$ element of $L$, with $i$ and $j$ such that $d(x_i,x_j) = l$:
• • If $x_i$ and $x_j$ are in different clusters, unify their clusters.
• • Keep in memory that this union as been made for distance $l$.
• If a precise number of clusters was required, one can stop the previous algorithm when the right number of clusters is reached. Otherwise else, let's call $T$ the array of the distances corresponding to the successive unions. Let's note $\Delta T$ the differences between successive elements of $T$, $std$ the standard deviation of $\Delta T$, and $m$ the mean ot $\Delta T$. Then one considers the first indice $k$ of $\Delta T$ such that $\Delta T(k) > mean + std$. Only the first $k+1$ unions are considered.

## B.2 Remarks

• The complexity of this algorithm is:
- $O(n^2 ln(n))$ in time
- $O(n^2)$ in space

This can be proved easily on the algorithm given in appendix. For the space, one only has to keep in memory the list of distances ($O(n^2)$), the partition (one cluster number per example, $O(n)$), and the list of elements of each cluster ($O(n)$). For the time, the sorting needs $O(n^2 ln(n^2))$, which is $O(n^2 ln(n))$, and the following operations need at most $n^2$ operations (this can easily be checked).

• The sorting or even simple enumeration of the distances might quicly become prohibitively slow or memory-greedy.

# Annexe C

# The $k$-means

## C.1 The principle

The principle of the $k$-means algorithm is that each element of a partition should be closer to the center of its class than to any other class center. So an arbitrary partition is chosen, and then the examples are examined once at a time; when an example is closer to another class center than its own one, it is moved to this class.

## C.2 The algorithm

Let's assume that we want to find a partition in $N$ classes $C_1$, ... , $C_N$ of $x_1$, ..., $x_n$.
• First of all, for any $i \in [1,n]$, we put $x_i$ in the class $i \ mod \ N + 1$.
• While ((it's the first attempt) or (something has been changed in the partition))
• • For each example $x_i$, $i \in [1,n]$:
• • • Find the class $C_j$ such that the mean of the elements of $C_j$ is the closest from $x_i$.
• • • If $x_i \notin C_j$, then put $x_i$ in $C_j$.
• • End of "for" loop.
• End of "while"

## C.3 Comments and possible improvements

• This algorithm is very fast; but as far as I know there's no known non-trivial complexity bound in time. The complexity in space is $O(n + N)$.

• The algorithm generates a separation which is a Voronoi. Each class has one center, and its points are the closer to it. So for example with a separation in two classes, the two chosen classes are necessarily linearly separable.

• The number of classes must be fixed *a priori*.

• A variant of the $k$-means algorithms is the so-called **transfer algorithm**. The only difference is that the distance $d(x_i,w_j)$ between the example $x_i$ and the center $w_j$ of the $j$-th class is replaced by $\sqrt{(\frac{Card(j)}{1+Card(j)})} \times d(x_i,w_j)$ with $Card(j)$ the cardinal of the $j^{th}$ class. With this, big classes are encouraged to get bigger yet.

• The algorithm can be used for data with missing points; the simplest way of doing this is to consider as the mean of a set of points with missing data the vector which has for coordonates the mean of this coordonates among the points for which it is defined. This can be done incrementally, as usual $K$-means.

# Chapitre 18

# Implémentation parallèle des réseaux neuronaux

**Résumé**

Nous résumons ici différentes techniques de parallélisations correspondant à diverses architectures et divers types de réseaux neuronaux.

# Table des matières



## 18.1 Introduction

Cette partie a été écrite en tant que "travail de recherche" en tant qu'examen pour un cours de parallélisme (dans le cas général du parallélisme, et non seulement dans le cas des réseaux neuronaux) de B. Tourancheau alors que j'étais étudiant au DIL (DEA d'Informatique de Lyon), puis complété pendant ma thèse.

Les réseaux de neurones artificiels ont montré leur efficacité théorique; mais ils ont maintenant à prouver leur intérêt pratique par des architectures efficaces et bon marché pour des utilisations pratiques. Beaucoup de simulations ont été faites sur des machines séquentielles ou des machines parallèles à passages de messages, montrant la puissance de cet outil pour une vaste plage d'applications; mais les modèles connectionnistes sont très intensifs au niveau communication et ils bénéficieraient d'architectures plus spécialisées, qui pourraient mettre en valeur le caractère parallèle des modèles neuronaux.

Dans l'article [Cesa-Bianchi, 1990], N. Cesa-Bianchi et G. Mauri ont résumé en 1989 un survol des implémentations pratiques des réseaux neuronaux sur des machines parallèles. Beaucoup de choses ont été faites à propos des réseaux neuronaux depuis 1989, de nouveaux outils très prometteurs ont été proposés, et les réseaux neuronaux commencent à être utilisés massivement dans l'industrie. Je vais résumer différentes formes de parallélisme (parallélisme pipeline, paralélisation de l'ensemble d'apprentissage, paralélisation neuronale, paralélisation des sommes linéaires pondérées), différentes architectures parallèles (du design direct de circuits spécialisés de calcul neuronal - numérique, optique, à temps discret et données continues, à temps et données continus - aux structures en tableaux - tableaux linéaires, rectangulaires - et neuro-ordinateurs généralistes et différents algorithmes de simulations parallèles (BLAS niveau 3, snake, duplication d'information, découpage vertical...), selon le type de réseaux neuronaux (rétropropagation, réseau Hopfield, réseaux neuronaux d'inspiration biologique...) et les nécessités pratiques (analyse temps-réel, résistance aux pannes, précision, tolérance aux erreurs, bas prix, forte connectique, capacité à apprendre ou poids fixés, fiabilité, partitionnement & rééchelonage, reconfigurabilité, amour, vitesse...). Par la suite, CPS signifie "connections par seconde" et CUPS signifie "mise-à-jours des connexions par seconde"; ces unités sont utiles pour exprimer la vitesse des algorithmes neuronaux.

## 18.2 L'algorithme de rétropropagation

Nous nous intéressons à explorer des réseaux multi-couches, entièrement connectés. La forme général est celle montrée dans la figure 18.1. Les différentes sortes de parallélisme dans un tel réseau sont (voir [Arrigo et al, 1990]):

• Parallélisme neuronale: différents neurones d'une couche donnée sont calculés sur différents processeurs
• Parallélisme synaptique: ce parallélisme à grain fin se préoccupe du calcul parallèle de la somme pondérée. Ceci peut être fait efficacement sur des unités de calcul vectoriel, voir [Rolando, 1989].
• Parallélisme de l'ensemble d'apprentissage: on peut utiliser le parallélisme pour calculer l'erreur pour différents exemples de l'ensemble d'apprentissage en même temps.
• Parallélisme de pipeline: ce parallélisme peut signifier différentes choses. Quand le réseau est utilisé comme réseau feedforward, sans apprentissage, atteindre un speed-up linéaire avec un pipeline parallèle est simple. Mais dans la phase d'apprentissage, le problème est le moment de la mise à jour des poids; on peut continuer à remplir le réseau alors que l'erreur est propagée, mais alors il ne s'agit pas précisément d'une rétropropagation basée sur la descente de gradient. Théoriquement le calcul de l'erreur doit être fait sur tous les exemples, et alors l'erreur doit être rétropropagée, et alors les poids sont mis-à-jour, et juste alors de nouveaux exemples peuvent être utilisés. Dans les cas pratiques, l'erreur est souvent calculée sur un sous-ensemble de l'échantillon d'apprentissage; beaucoup de compromis peuvent être faits, pipeline ou pas, utiliser du parallélisme sur l'ensemble d'apprentissage ou un sous-ensemble de l'ensemble d'apprentissage.

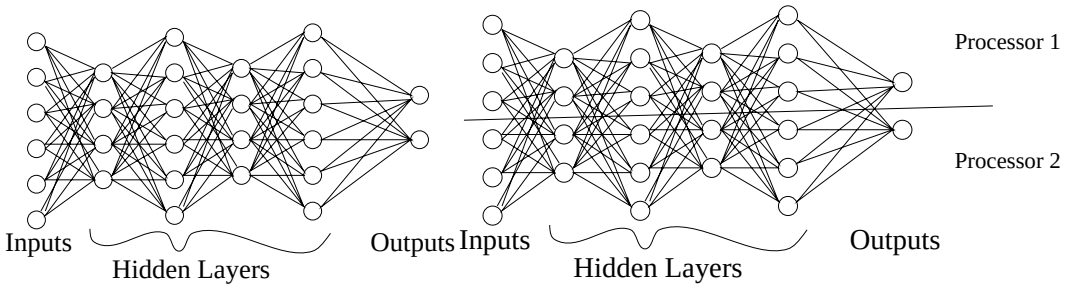



FIG. 18.1 – *A gauche: le nombre de neurones dans chaque couche peut être n'importe quel entier. Chaque entrée d'un neurone (à part les neurones de la couche d'entrée) est la somme pondérée des sorties du neurone de la couche précédente. La sortie d'un neurone est l'image de son entrée par $f$, une fonction d'activation donnée. A droite: cet exemple de "section verticale" peut aisément être généralisé à un grand nombre de processeurs: le principe est juste de donner le même nombre de neurones d'une couche donnée à chaque processeur.*

### 18.2.1 Parallélisme neuronal sur des machines à passage de messages

Une technique usuelle est le "découpage vertical" (voir par exemple [Beynon et al, 1987] et [Baiardi, 1990] - desquels sont extraits les résultats suivants - et la figure 18.1). Les figures 18.2 montrent le principe des messages point-à-point et broadcast sur une topologie thoroïdale.

#### Phase de reconnaissance

L'algorithme est le suivant:
• $i \leftarrow 1$
• *Début de boucle:*
• Chaque processeur calcule les sorties des neurones de la couche 1 qui lui sont alloués
• Il y a une communication tous-à-tous, ainsi chaque processeur reçoit toutes les sorties de tous les neurones de la couche $i$
• si $i \leq$ *nombre de boucles*, va à *Début de boucle*
Pour un nombre de couches donné, le nombre optimal de processeurs est linéaire en le nombre de neurones par couche. Le speed-up est linéaire en le nombre de neurones, aussi.

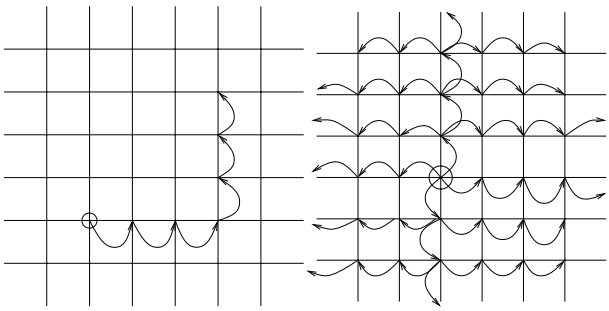


FIG. 18.2 – *Pour la communication point-à-point le message est envoyé à la colonne appropriée, processeur après processeur, et alors le message est envoyé au processeur approprié de la colonne, processeur après processeur. Le nombre de pas est donc la différence horizontale plus la différence verticale. Pour le broadcast le message est envoyé au processeur au-dessus et en-dessous, et chaque fois qu'un processeur de la colonne reçoit le message il l'envoit à ses voisins de droite et gauche, qui transmettent le message.*

**Phase d'apprentissage**

Pour la phase d'apprentissage, une méthode relativement efficace est la suivante:
• La reconnaissance en feedforward est comme en 18.2.1
• L'évaluation des erreurs sur la dernière couche; chaque processeur calcule l'erreur sur les neurones de la dernière couche qui lui sont alloués.
• $i \leftarrow$ *nombre de couches*
• *Début de boucle:*
• Les erreurs sont broadcastées sur tous les processeurs
• Chaque processeur calcule l'erreur rétropropagée sur les neurones qui lui sont alloués; pour cela, ils ont besoin des erreurs de la couche suivante (qu'ils ont juste reçues par le broadcast) et des poids de la connection (qui doivent ainsi être stockées sur chaque neurone, celui avant la connection et celui après)
• Si $(i > 1)$ va à *Début de boucle*
• Chaque processeur met à jour les poids des connections afférentes à ses neurones
• Les nouveaux poids sont broadcastés afin que chaque processeur connaissent les poids des connexions afférentes et efférentes.
Une fois encore, pour un nombre de couche donné, le nombre optimal de processeurs est linéaire en le nombre de neurones par couche. Le speed-up optimal est linéaire en le nombre de neurones par couche.
L'efficacité de chaque transputer dans une telle architecture, pour le speed-up optimal, est d'environ 50% dans un réseau à 3 couches, ce qui montre que les modèles connexionnistes sont probablement trop intensifs en communication pour des architectures dans lesquelles les communications sont considérées comme des évènements occasionnels (voir [Ernoult, 1987]).

### 18.2.2 Blas niveau 3: tableaux systoliques, machines à passage parallèle de messages, unités de calcul vectoriel

On peut écrire les équations de la rétropropagation en utilisant une forme matricielle. Les avantages sont le grand nombre d'architectures sur lesquelles existent des produits de matrices parallélisés efficaces implémentées en Blas niveau 3. Cette formalisation est extraite de [Arrigo et al, 1990].
Phase d'apprentissage:
• Pour $i = 1..N-1$ faire
• $\quad [I^{i+1}] = [W^i] * [O^i]$
• $\quad [O^{i+1}] = f([I^{i+1}])$
• $[\Delta^N] = ([t_0] - [0^N]) * f'([net^N])$
• Pour $i = N-1$ downto 2
• $\quad [\Delta^i] = [sW^i]^T * [\Delta^{i+1}] * f'([net^i])$

• Pour $i = 1..N-1$
• $[\Delta W^i](t) = \eta(\Delta^{i+1}) * [O^i]^T + \alpha[\Delta W^i](t-1)$
On peut voir dans [Vogl, 1988] des discussions à propos d'$\eta$ et $\alpha$.
$W^i$ est la matrice des poids entre la couche $i$ et la couche $i+1$, $I^i$ est la matrice des entrées de la couche $i$ (c'est un vecteur si on utilise les exemples un par un), $O^i$ les sorties correspondantes, $\Delta W^i(t)$ si la différence entre $\Delta W^i(t)$ et $\Delta W^i(t+1)$. $sM$ est la matrice $M$ sans la dernière colonne (correspondant au biais, ie la connection du neurone dont la sortie est constante), et $M^T$ est la transposée de la matrice $M$.
Cette technique, mûre, peut être utilisée sur presque toute architecture, elle est peu coûteuse et efficace.

### 18.2.3 Parallélisme pipeline et parallélisme neuronal sur des tableaux systoliques

Pour plus de détails on peut lire l'article [Kung et al, 1993] duquel sont extraites les figures et notations suivantes.
Cette section est spécialisée au cas d'un réseau qui contient une couche d'entrée, une couche cachée et une couche de sortie. Dans les figures qui suivent, la couche d'entrée est supposée large de 7 neurones, la couche cachée de 5 neurones, et la couche de sortie de 3 neurones. Ceci peut aisément être étendu au cas de nombres variés de neurones; dans le cas de tableaux rectangulaires, la méthode peut être utilisée dans le cas d'un autre nombre de couches; pour le tableau linéaire, la méthode est un peu différente mais peut être déduite aisément.

Avec $\underline{W}$ la matrice des poids entre le couche d'entrée et la couche cachée, et avec $\overline{W}$ la matrice des poids entre la couche cachée et la couche de sortie, $x$ est l'entrée, $\underline{\theta}$ et $\overline{\theta}$ sont des biais, $F$ est la fonction d'activation appliquée à un vecteur, $y$ est la sortie, $t$ est la sortie désirée, $a$ est l'entrée de la couche de sortie, on a les équations suivantes:
$\underline{u} = \underline{W}x + \underline{\theta}$, $a = F[\underline{u}]$, $\overline{u} = \overline{W}a + \overline{\theta}$, $y = F[\overline{u}]$ pour la phase feedforward.
$g_i = f'(f^{-1}(y_i)).(t_i - y_i)$, $\underline{\delta} = \overline{W}^T g$, $h_j = f'(f^{-1}(a_j))\underline{\delta}_j$, $\Delta\overline{W} = \eta g a^T$, $\Delta\underline{W} = \eta h x^T$ pour la phase d'apprentissage.

#### Tableaux systoliques linéaires.

La figure 18.3. L'article [Kung et al, 1993] donne une méthode générale pour mapper les réseaux neuronaux sur des tableaux systoliques. Dans ce cas, les poids sont adaptés après chaque exemple (alors que pour l'algorithme théorique, l'erreur globale doit être prise en compte, mais même sur des machines séquentielles en général on utilise les exemples un par un).

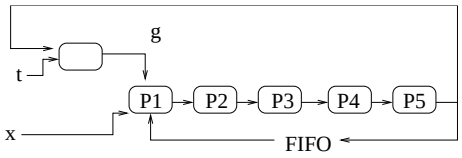



FIG. 18.3 – *Chaque PO stock une ligne de $\underline{W}$ et une colonne de $\overline{W}$.*

#### Tableaux systoliques rectangulaires

Voir la figure 18.4. L'article [Kung et al, 1993] donne une méthode générale pour mapper les réseaux de neurones sur des tableaux systoliques. Dans ce cas, les poids sont adaptés après un ensemble d'exemples à cause de la méthode pipeline; les temps de mise-à-jour soulignent le fait que ceci est une approximation du vrai cas. Le taux d'utilisation est un peu plus bas que dans le tableau linéaire; mais il est toujours très haut, et le speed up est bien meilleur.

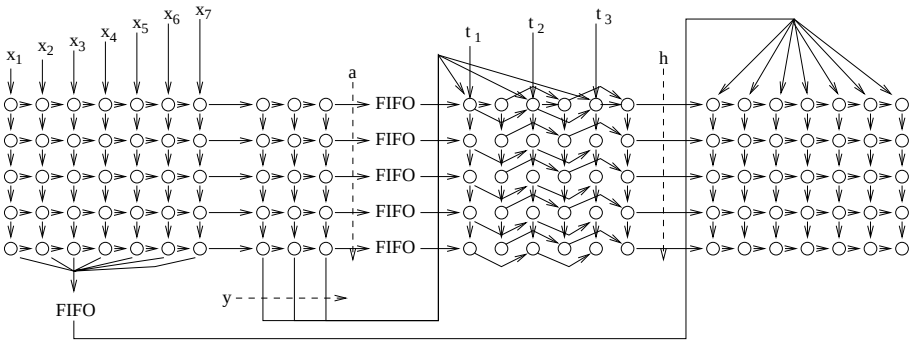



FIG. 18.4 – *Les FIFO sont utilisées de manière à retarder la transmission.*

**Implementation sur des machines parallèles avec un nombre de processeurs plus petit que le nombre de neurones**

On peut aisément se convaincre qu'un tableau de $n$ processeurs peut être simulé par un tableau plus petit de $p$ processeurs avec un temps de calcul multiplié par $n/p$ (à part dans le cas de problèmes de cache ou de mémoire). Pour une preuve formale, et les détails de l'implémentation, on peut lire [Misra, 1992].

## 18.3 Réseaux de Hopfield (et quelques autres réseaux neuronaux récurrents)

Les implémentations suivantes se préoccupent juste de reconnaissance par réseaux de Hopfield; les poids sont fixés; il n'y a pas de phase d'apprentissage.

### 18.3.1 Implémentation sur une architecture systolique

Le mapping suivant sur un tableau systolique 2D est extrait de [Misra, 1992]. Le réseau est un réseau de $n$ neurones et va être mappé sur une architecture 2D de $n \times n$ processeurs; $P_{i,j}$ est le processeur à la $i^e$ ligne et la $j^e$ colonne. $P_{i,i}$ stocke l'activation du $i^e$ neurone. Le poids $W_{i,j}$ est stocké sur le processeur $P_{i,j}$.
Dans un réseau de Hopfield, l'activation du $i^e$ neurone à l'instant $k$ est calculée par $a_i^{k+1} = f_i(\sum_{j=1}^{n} W_{i,j} a_j^k)$.
Cette équation peut être écrite $a_i^{k+1} = f(x^k)$ et $x^k = W.a^k$ avec $W$ la matrice de connections. L'algorithme est le suivant:
Premier pas: la valeur $a_j$ est communiquée à tous les processeurs de la $j^e$ colonne.
Second pas: $\sum W_{i,j} a_j$ est calculé pour chaque ligne $i$
Troisièle pas: la fonction $f$ est appliquée à chaque noeud $W_{i,i}$.
La complexité dépend de l'architecture, et est la somme des complexités suivantes:
• Pour le premier pas:
Avec un bus pour un broadcast par colonne, $O(1)$; pour une architecture avec communications à 4 voisins, $O(n)$.
• Pour le second pas:
Multiplications:$O(1)$ si chaque processeur a un multiplieur. $O(n)$ sinon, avec un multiplieur par colonne.
Addition:$O(n)$ sur la plupart des architectures.
• Troisième pas: $O(1)$ si chaque processeur de la diagonale peut calculer la fonction $f$.

### 18.3.2 Réseaux récurrents à connexions creuses

Le mapping suivant sur un tableau systolique 2D est extrait de [Misra, 1992]. Le réseau est supposé composé de $n$ neurones interconnectés par $e$ connections non-nulles. Un tableau de $\sqrt{n+e} \times \sqrt{n+e}$ processeurs

est requis. Le processeur $P_{i,j}$ a un index donné par $(i-1)*n+j$. Pour $i \in$ [1,$n$] (en ordre croissant), l'activation $a_i$ est mappée sur le processeur de plus petit index qui n'a pas une activation déjà mappée dessus. Alors, les poids sont mappés sur le tableau en serpent par plus grand numéro de colonne d'abord (voir figure 18.5).

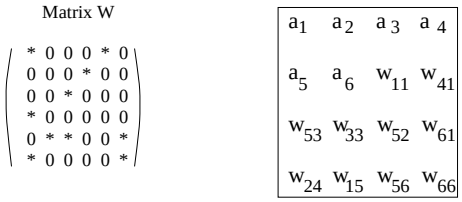



FIG. 18.5 – *Un exemple de mapping de connexions non nulles en serpent par ordre de colonne de plus grand numéro d'abord.*

L'ordre en serpent par colonne de grand index d'abord est défini formellement comme suit: si $i$ est impair, l'index attribué à $P_{i,j}$ est défini par $(i-1)*n+j$; sinon, il est défini par $(i-1)*n+n+1-j$. Pour $j=1..n$, l'entrée non-nulle de la $j^e$ colonne de la matrice qui a le plus petit indice parmi ceux qui ne sont pas encore attribués est assigné au processeur de plus petit indice (indice comme défini plus tôt) qui n'a pas encore un poids qui lui est assigné.
Avec ce mapping, les valeurs non-nulles d'une colonne sont assignées à une région connexe.
Une itération est définie comme suit: $a_i^{k+1} = f_i(\sum_{j=1}^{n} w_{i,j} a_j^k)$ pour $i \in$ [1,$n$]. L'algorithme d'une itération est le suivant:
• Pour $j \in$ [1,$n$], $a_j$ est envoyé à tous les processeurs qui contiennent un poids de la $j^e$ colonne.
• Calcul de $w_{i,j}.a_j$
• Le résultat du calcul précédent est envoyé à un processeur tel que le produit correspondant à une ligne forme une région connectée; cela signifie qu'ils sont envoyés aux processeurs déterminés en serpent par colonne de grand index d'abord.
• La somme des produits est calculée sur chacune de ces régions.
• Cette somme est envoyée au processeur qui stocke les valeurs d'activation.
• La fonction d'activation est appliquée.
Une routine efficace ($O(\sqrt{n+e})$ est détaillée dans [Misra, 1992].

## 18.4 Quelques autres techniques prometteuses

### 18.4.1 Algorithmes génétiques appliqués à la phase d'apprentissage

Les algorithmes génétiques peuvent être très efficaces pour apprendre les poids, par exemple plus que la rétropropagation quand le nombre de couches est très grand. La méthode usuelle de parallélisation d'algorithmes génétiques est alors possible; voir [Buckles, 1993] pour plus de détails à propos des algorithmes génétiques.

Un regain d'intérêt particulier pour les algorithmes génétiques pour les réseaux neuronaux est dû à l'efficacité des algorithmes génétiques poru apprendre sans ensemble d'apprentissage, eg les problèmes de contrôle, où une fonction de fitness peut être fournie mais pas un ensemble d'apprentissage.

### 18.4.2 Implémentations optiques

Un avantage des implémentations optiques est la très haute connectique; un inconvénient est la difficulté de l'adaptation des poids. Voir figure 18.6 pour un exemple dans le cas d'un réseau de Hopfield; d'autres implémentations sont possibles, par exemple pour les résultats multicouches. Pour plus de détails sur les implémentations optiques, on peut lire [Keller et al, 1993].

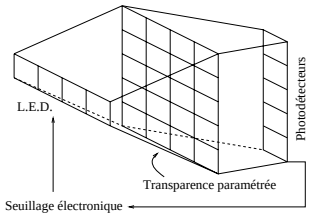



FIG. 18.6 – *Implémentations optiques du réseau de Hopfield: une itération est* $O(1)$

### 18.4.3 Puces analogues ou hybride analogues/numériques

Les systèmes analogues sont moins précis que les numériques pour le stockage à long terme; ils sont plus sensibles au bruit. Mais les amplificateurs opérationnels sont aisément construits à partir de simples transistors, et peuvent effectuer très vite des fonctions de transfert sigmoïdes. Les sommations pondérées sont très naturelles et rapides, aussi. Finalement, ils offrent des puces très compactes et rapides, avec une faible dissipation d'énergie. A présent, ces puces ne sont usuellement pas capable d'apprendre; mais ils peuvent être très efficaces par exemple pour des algorithmes de vision primitive. Certaines puces sont capables de charger des poids (par exemple, la puce Intel ETANN, qui contient 64 neurones et 10240 synapses, et effectue 2 GCPS par seconde), mais pas vraiment d'apprentissage à la volée. Des architectures hybrides sont très puissantes, aussi (par exemple, la puce ANNA, des laboratoires AT&T, avec 4096 synpases, 256 entrées et 8 sorties, effectue 2 GCPS - jusqu'à 5 GCPS en crête).

### 18.4.4 Quelques autres paradigmes

#### Inférence Bayésienne

Un regain d'intérêt pour l'inférence Bayésienne est dû à la naissance de la Bayes Point Machine, présentée comme une optimisation des Support Vector Machines, ellesm-même considérées comme des approximations d'inférence Bayésienne. La phase d'apprentissage de l'inférence Bayésienne peut être calculée par un billard (comme dans les Bayes Point Machine), qui peut être parallélisé. L'efficacité de cette parallélisation n'a jamais été discutée, pour autant que nous le sachions.

Une autre solution pour l'inférence Bayésienne consiste en les algorithmes de Gibbs, qui peuvent être parallélisées aussi avec un speed up linéaire pendant la phase d'apprentissage.

#### Algorithmes rapides avec résistance au bruit

Un développement récent de théorie de l'apprentissage consiste en la "théorie de l'apprentissage avec bruit", où une proportion $\eta$ des exemples est perturbée par du bruit. Les points bruités sont, dans certains cas, remplacés par le même point avec une autre étiquette ou par un point choisi par un adversaire sans limite de puissance de calcul.

Un algorithme classiqeu proposé par Keans et Li avec des adaptations de Decatur consiste en séparer l'ensemble d'apprentissage en $n+1$ sous-ensembles, et alors apprendre indépendamment sur chacun des $n$ premiers sous-ensembles, et choisir le meilleur classifieur par comparaison sur le $n+1^e$ sous-ensemble. Pour un choix ad hoc de $n$ ceci conduit à une résistance presqu'optimale au bruit et à une meilleur complexité en temps que l'algorithme initial, quand bien même il n'y a pas de parallélisation. Bien sûr, la parallélisation peut améliorer la complexité linéairement en $n$ avec $n$ processeurs.

#### Moyennage après apprentissages indépendants

Une solution simple pour paralléliser consiste en séparer l'ensemble d'apprentissage en différents sous-ensembles, et apprendre sur chacun. Dans le paragraphe ci-dessus, les classifieurs étaient alors comparés sur un autre sous-ensemble et le meilleur était utilisé. Une autre solution consiste en utiliser la moyenne de ces

classifieurs. Ceci peut aisément être parallélisé, et dans nos (rapides) expérimentations le classifieur résultant est meilleur que celui résultant d'un apprentissage global (et coûteux en temps de calcul).

#### L'informatique quantique

On a pu voir dans des conférences récentes l'apprentissage cité comme un des problèmes pouvant être liés à l'informatique quantique. Malheureusement, les problèmes les plus intéressants en apprentissage sont NP-complets, or pour le moment il n'existe pas de problème NP-complet traité par parallélisme quantique. Les applications éventuelles restent donc à prouver.

## 18.5 Conclusion

Ce chapitre est un peu court pour résumer toutes les techniques utilisées pour paralléliser les réseaux neuronaux. J'ai détaillé le cas de la rétropropagation et le cas des réseaux de Hopfield; les idées principales utilisées dans ces techniques peuvent être utilisées pour beaucoup d'autres réseaux neuronaux. Un survol intéressant de hardware neuronal est [Heemskerk, 1995]. L'implémentation d'algorithmes neuronaux sur des machines à passage de messages n'est pas très efficace; le réel avantage de cette architecture est son bas coût et sa reconfigurabilité aisée pour la recherche; mais les modèles connectionnistes sont très consommateurs de communication pour ces architectures. Les systèmes analogiques sont très efficaces pour les applications qui n'impliquent pas de stockage à long terme et peuvent supporter du bruit, par exemple les systèmes sensoriels. Les tableaux systoliques semblent être un bon compromis entre le bas coût et l'efficacité et la configurabilité pour beaucoup d'applications. Les techniques optiques sont très prometteuses mais pas assez mûres encore. L'expression en BLAS niveau 3 du calcul neuronale est détaillée en [Kung et al, 1989] et fournit une façon simple de programmer les réseaux de neurones efficacement et simplement.

# Bibliographie

[Arrigo et al, 1990] P. Arrigo, A. Corana, F. Giuliano, L. Marconi, M. Morando, S. Ridella, C. Rolando, F. Scalia, *Neural Networks: Computer Simulations and Biomedical Applications, in Second Italian Workshop on Parallel Architectures and Neural Networks, World Scientific publishing Co, 1990*

[Beynon et al, 1987] T. Beynon and N. Dodd, *The Implementation of Multi-Layer Perceptrons on Transputer Networks, Proc. of the* $7^{th}$ *Occam User Group, Grenoble, 1987*

[Baiardi, 1990] F. Baiardi, R. Mussardo, R. Serra, G. Valastro, *Feedforward Layered Networks on Message Passing Parallel Computers, in Second Italian Workshop on Parallel Architectures and Neural Networks, World Scientific publishing Co, 1990*

[Buckles, 1993] Bill P. Buckles, F.E. Petry, *Genetic Algorithms, Electronica Books Ltd., Middlesex (UK), 1993*

[Cesa-Bianchi, 1990] N. Cesa-Bianchi, G. Mauri, *Implementing Neural Networks, in Second Italian Workshop on parallel architectures and Neural Networks, ed. R. Caianiello, World Scientific Publishing Co., 1990*

[Ernoult, 1987] C. Ernoult, *Performance of Back-Propagation on a Transputer based machine, proceedings of tenth IJCAI, Morgan Kaufmann Publ. Sans Francisco CA, 1 (1987) 323-326.*

[Heemskerk, 1995] J.N.H. Heemskerk, *Neurocomputers for Brain-style processing. Design, Implementation and Application, PhD thesis, Leiden University, ftp.mrc-apu.cam.ac.uk/pub/nn, 1995*

[Kung et al, 1993] S.Y. Kung and W.H. Chou, *Mapping Neural Networks onto VLSI array processors, in Parallel Digital Implementations of Neural Networks, 1993, Prentice-Hall*

[Keller et al, 1993] P.E. Keller, A.F. Gmitro, *Computer-generated holograms for optical neural networks: on-axis versus off-axis geometry, Applied Optics 32, 1304-1310, 1993*

[Kung et al, 1989] S.Y. Kung, J.N. Hwang, *A unified modeling of connectionnist neural networks, in International Joint Conference on Neural Networks, 1989*

[Misra, 1992] M. Misra, *Implementation of Neural Networks on Parallel Architectures, Faculty of the Graduate School University of Southern California in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy (Computer Engineering), 1992*

[Rolando, 1989] C. Rolando, S. Corana and S. Ridella, *Neural Network Simulator with high performance on FPS M64 Series Minisupercomputers, in "Fuse at the Falcon", proc. of the 1989 conference, Stratford upon Avon*

[Vogl, 1988] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink and D.L. Alkon, *Accelerating the Convergence of Back-Propagation Method, Biological Cybernetics 59, 257-263, 1988*