

Data mining

et statistique décisionnelle

L'intelligence des données

Stéphane
TUFFÉRY

Analyse discriminante

Arbres de décision

Bootstrap

Classification

Exploration des données

Modèles linéaires

Régression logistique

Réseaux de neurones

Scoring

Text mining

Nouvelle édition
revue et augmentée

$$\text{Logit}(y) = \beta_0 + \sum_{i=1}^p \beta_i x_i$$

Editions TECHNIP

Chapitre 7

Les réseaux de neurones

Il est aujourd’hui difficile de parler du data mining sans parler des réseaux de neurones, qui sont à la base à la fois de certaines techniques descriptives et de certaines techniques prédictives de data mining. Ils se sont largement répandus grâce à leur puissance de modélisation (ils peuvent approcher n’importe quelle fonction suffisamment régulière), qui fait merveille dans une grande variété de problèmes, face à des phénomènes complexes, des formes irrégulières, des données difficiles à appréhender et ne suivant pas de loi probabiliste particulière. Cependant leur utilisation est parfois freinée par les difficultés qu’elle présente : le côté « boîte noire » des réseaux, la délicatesse des réglages à effectuer, la puissance informatique requise et les risques de sur-apprentissage et de convergence vers une solution globalement non optimale.

Si ce chapitre a été placé avant ceux consacrés aux méthodes de classification, de classement et de prédiction, c’est en raison du double emploi des réseaux de neurones en classification (réseaux de Kohonen) et en classement et prédiction (perceptrons, réseaux à fonction radiale de base). Le lecteur non intéressé par ces techniques particulières pourra passer la lecture de ce chapitre.

7.1. Généralités sur les réseaux de neurones

Après la notion initiale de *neurone formel* de Mc Culloch et Pitts en 1943, les *réseaux de neurones* sont apparus en 1958 avec le perceptron de Rosenblatt, ont connu un grand essor dans les années 1980 et une utilisation industrielle depuis les années 1990. Un réseau de neurones (ou réseau neuronal) a une architecture calquée sur celle du cerveau, organisé en neurones et synapses, et se présente comme un ensemble de *nœuds* (ou *neurones formels*, ou *unités*) connectés entre eux, chaque variable continue en entrée correspondant à un nœud d’un premier niveau, appelé *couche d’entrée*, et chaque modalité d’une variable qualitative correspondant également à un nœud de la couche d’entrée. Le cas échéant, lorsque le réseau est utilisé dans une technique prédictive, il y a une ou plusieurs variables à expliquer ; elles correspondent alors chacune à un nœud (ou plusieurs dans le cas des variables qualitatives : voir plus bas) d’un dernier niveau : la *couche de sortie*. Les réseaux prédictifs sont dits « à apprentissage supervisé » et les réseaux descriptifs sont dits « à apprentissage non supervisé ». Entre la couche d’entrée et la couche de sortie sont parfois connectés des nœuds appartenant à un niveau intermédiaire : la *couche cachée*. Il peut exister plusieurs couches cachées.

Un nœud reçoit des valeurs en entrée et renvoie 0 à n valeurs en sortie. Toutes ces valeurs sont normalisées pour être comprises entre 0 et 1 (ou parfois entre -1 et 1, selon les bornes de la fonction de transfert ci-dessous). Une *fonction de combinaison* calcule une première valeur à partir des nœuds connectés en entrée et du poids des connexions. Ainsi, dans les réseaux les plus courants, les perceptrons, il s'agit de la somme pondérée $\sum_i n_i p_i$ des valeurs n_i des nœuds en entrée. Afin de déterminer une valeur en sortie, une seconde fonction, appelée *fonction de transfert* (ou d'*activation*), est appliquée à cette valeur. Les nœuds de la couche d'entrée sont triviaux, dans la mesure où ils ne combinent rien, et ne font que transmettre la valeur de la variable qui leur correspond.

Un nœud de perceptron se présente donc comme suit :

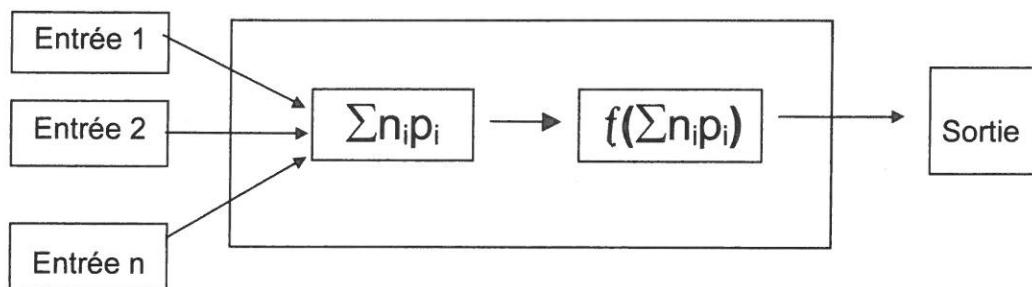


Figure 7.1 – Nœud d'un réseau de neurones

Dans cette figure, on utilise les notations suivantes :

- n_i est la valeur du nœud i du niveau précédent (la sommation sur i correspond à l'ensemble des nœuds du niveau précédent connectés au nœud observé) ;
- p_i est le poids associé à la connexion entre le nœud i et le nœud observé ;
- f est la fonction de transfert associée au nœud observé.

L'apprentissage du réseau de neurones s'effectue à partir d'un échantillon de la population à étudier, les individus de l'échantillon lui permettant d'ajuster le poids des connexions entre les nœuds. Au cours de l'apprentissage, la valeur renvoyée par le nœud en sortie est comparée à la valeur réelle, et les poids p_i de tous les nœuds sont ajustés de façon à améliorer la prédition, par un mécanisme dépendant du type de réseau de neurones. Un mécanisme encore courant est la « rétropropagation du gradient », mais il en existe d'autres, plus récents et plus performants, tels les algorithmes de Levenberg-Marquardt, quasi-Newton, du gradient conjugué, de propagation rapide (« quick propagation ») ou encore les algorithmes génétiques (voir la section 10.12). L'échantillon d'apprentissage est parcouru de nombreuses fois (souvent, plusieurs milliers). L'apprentissage s'achève lorsqu'une solution optimale²¹ a été trouvée et que les poids p_i ne sont plus modifiés significativement, ou lorsqu'un nombre d'itérations fixé *a priori* a été atteint. À l'issue de la phase d'apprentissage, le réseau forme une fonction associant les variables entre elles.

Dans le perceptron, la fonction de transfert peut éventuellement être la fonction linéaire $f(x) = x$, mais il vaut mieux choisir une fonction qui présente un comportement linéaire au

²¹ Optimum global ou seulement local : voir plus loin la section sur le perceptron multicouches.

voisinage de 0 (lorsque les poids des nœuds sont petits), et non linéaire aux extrémités, ce qui permet de modéliser des phénomènes linéaires et non linéaires. On choisit presque toujours une fonction *sigmoïde* et plus particulièrement la sigmoïde logistique $s(x) = 1 / (1 + \exp(-x))$ (que nous reverrons dans la section sur la régression logistique), ou la sigmoïde tangentielle, qui n'est autre que la tangente hyperbolique :

$$\tanh(x) = [\exp(x) - \exp(-x)] / [\exp(x) + \exp(-x)].$$

D'autres sigmoïdes existent, mais les performances sont de toute façon très proches quelle que soit la fonction retenue. Bien que ces fonctions soient toujours croissantes, elles peuvent approcher n'importe quelle fonction continue quand on les combine entre elles (voir la section 7.8.1), c'est-à-dire quand on combine l'activation de plusieurs nœuds.

La prise en compte des relations non linéaires entre les variables est un gros atout des réseaux de neurones.

On voit avec la fonction logistique la nécessité de normaliser les valeurs des données en entrée. Sinon les données avec de grandes valeurs « écraseraient » les autres, et les ajustements des poids seraient sans effet sur la valeur $1/(1 + \exp(-\sum n_i p_i))$, cette valeur ne variant plus beaucoup autour de 0 ou 1 lorsque la valeur absolue de $\sum n_i p_i$ est grande. Ensuite, le fait que toutes les valeurs soient comprises entre 0 et 1 (ou –1 et 1) permet de prendre en entrée d'un nœud la sortie d'un nœud précédent, sans problème de valeurs trop grandes.

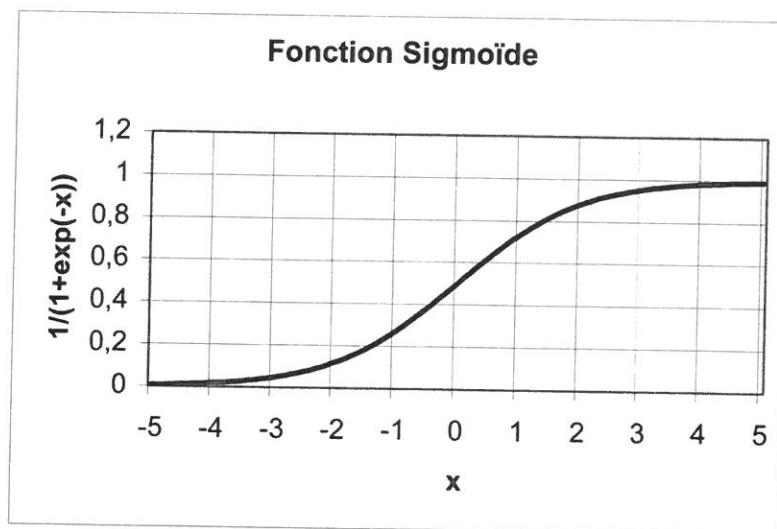


Figure 7.2 – La fonction logistique

De façon générale, les étapes dans la mise en œuvre d'un réseau de neurones pour la prédiction ou le classement sont :

1. l'identification des données en entrée et en sortie,
2. la normalisation de ces données,
3. la constitution d'un réseau avec une structure adaptée,

4. l'apprentissage du réseau,
5. le test du réseau,
6. l'application du modèle généré par l'apprentissage,
7. la dénormalisation des données en sortie.

7.2. Structure d'un réseau de neurones

La structure du réseau de neurones, encore appelée « architecture » ou « topologie » du réseau de neurones, est le nombre de couches et de nœuds, la façon dont sont interconnectés les différents nœuds (choix des fonctions de combinaison et de transfert) et le mécanisme d'ajustement des poids. Le choix de cette structure détermine grandement les résultats qui seront obtenus, et constitue le point délicat dans la mise en œuvre d'un réseau de neurones.

La structure la plus simple est celle où les nœuds sont répartis en deux couches : une couche d'entrée et une couche de sortie. Les nœuds de la couche d'entrée ont une seule entrée et une seule sortie, égale à l'entrée (voir la Figure 7.3). Le(s) nœud(s) de sortie a(ont) en entrée tous les nœuds de la couche d'entrée, avec une fonction de combinaison et une fonction de transfert.

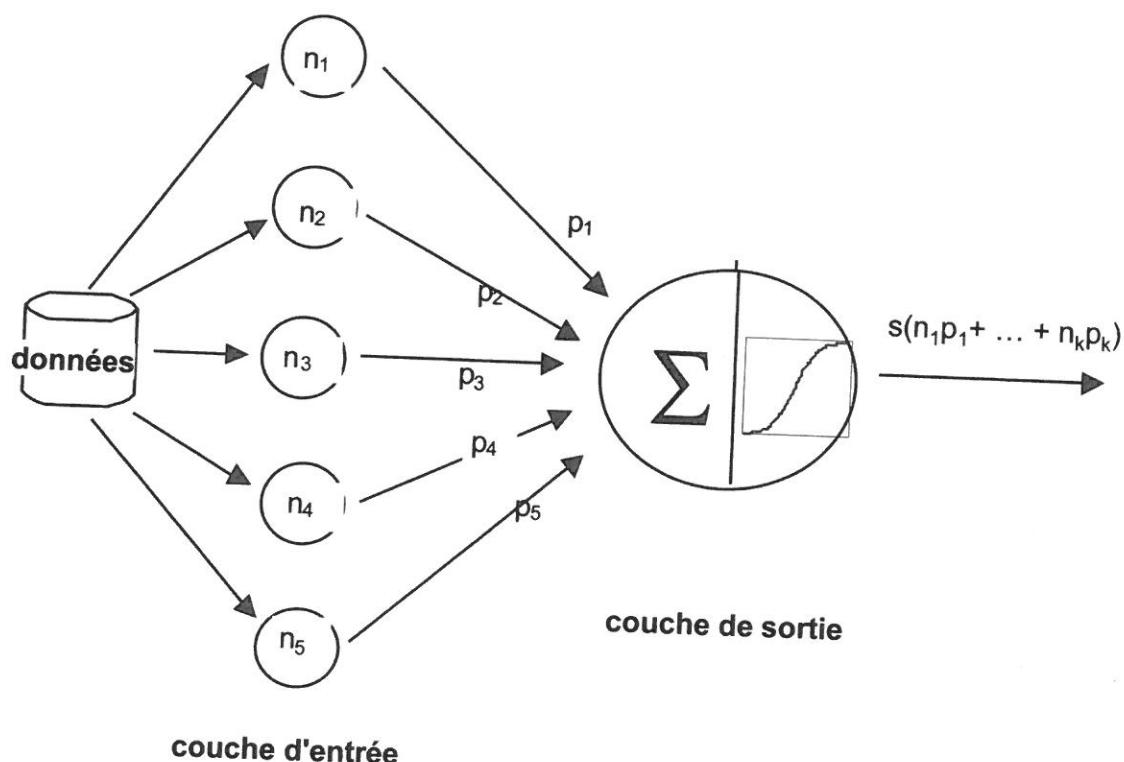


Figure 7.3 – Réseau de neurones sans couche cachée

Dans ce cas, le modèle obtenu est une régression linéaire ou logistique selon que la fonction de transfert est linéaire ou logistique, et les poids du réseau sont les coefficients de la régression.

On augmente le pouvoir de prédiction en ajoutant une ou plusieurs couches cachées entre les couches d'entrée et de sortie (Figure 7.4). Bien que le pouvoir de prédiction augmente avec le nombre de couches cachées et de nœuds dans ces couches, ce nombre doit néanmoins être le plus petit possible, afin que le réseau de neurones ne se contente pas de mémoriser l'ensemble d'apprentissage mais puisse le généraliser, en évitant ce que l'on appelle le « sur-apprentissage » (voir la section 10.3.3), qui survient lorsque les poids ne font « qu'apprendre » les particularités de l'ensemble d'apprentissage au lieu d'en découvrir les structures générales. Ceci arrive lorsque la taille de l'ensemble d'apprentissage est trop faible par rapport à la complexité du modèle, c'est-à-dire, ici, la complexité de la topologie du réseau. Quelques indications à ce sujet sont fournies plus bas, dans la section 7.4.

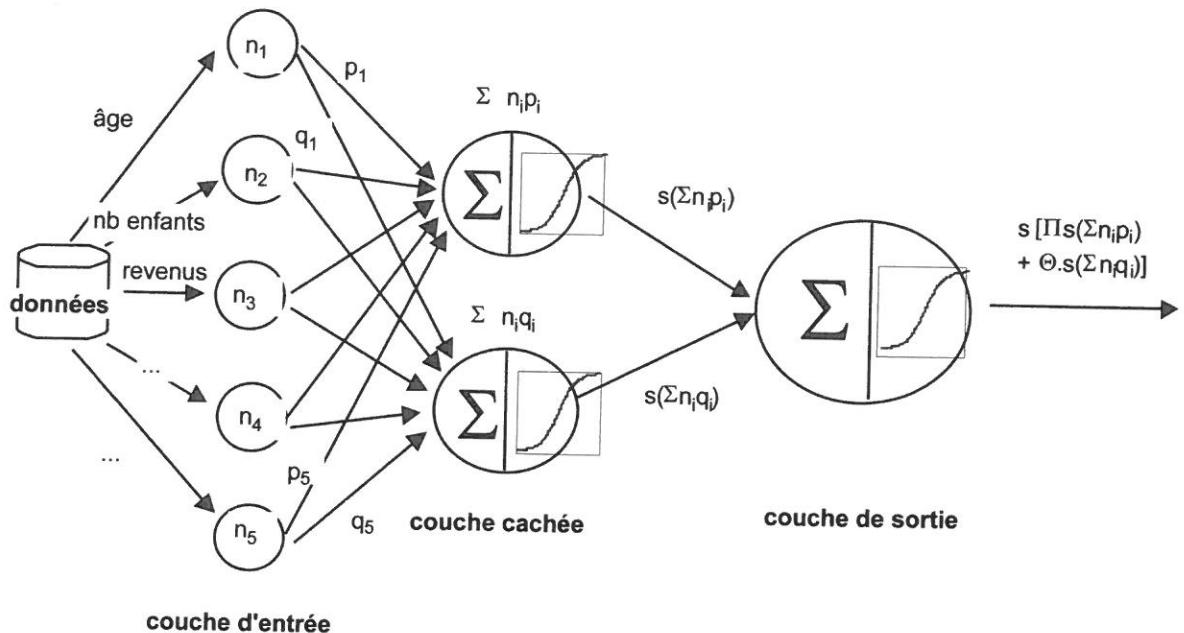


Figure 7.4 – Réseau de neurones avec couche cachée

Indépendamment de l'existence d'une couche cachée, la couche de sortie du réseau peut parfois avoir plusieurs nœuds, lorsqu'il y a plusieurs classes à prédire (Figure 7.5).

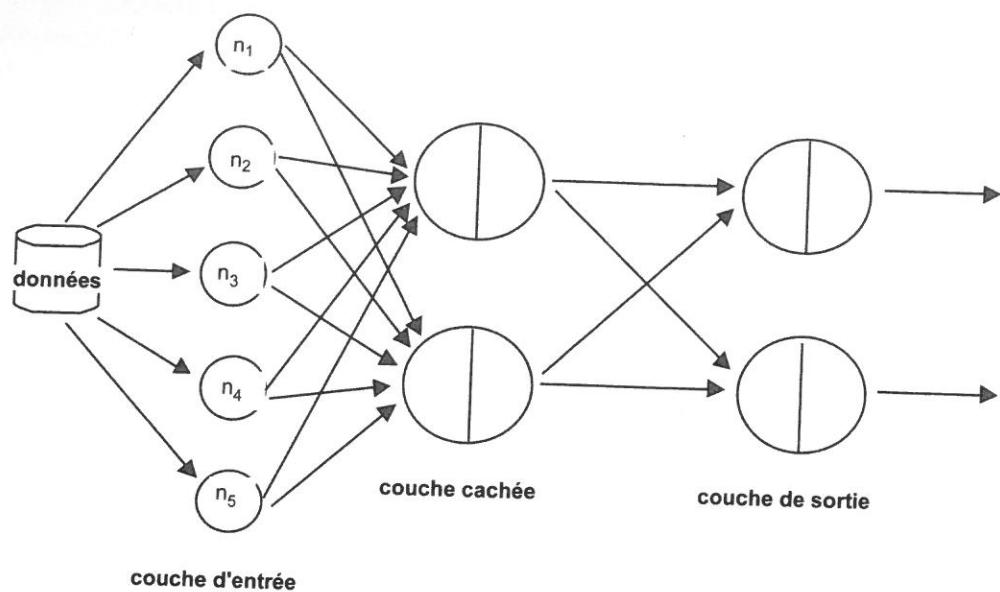


Figure 7.5 – Réseau de neurones avec plusieurs nœuds de sortie

7.3. Choix de l'échantillon d'apprentissage

L'apprentissage du réseau de neurones sera d'autant meilleur qu'il s'effectuera sur un échantillon suffisamment riche pour représenter toutes les valeurs possibles des nœuds de toutes les couches du réseau, c'est-à-dire en particulier toutes les modalités possibles de chaque variable, en entrée ou en sortie. Un réseau n'est capable d'apprendre qu'à partir des configurations qu'il a rencontrées au cours de son apprentissage : si les clients avec plus de 1 000 euros de découvert sont très risqués, mais si l'échantillon d'apprentissage du réseau n'en contenait aucun, le réseau ne saura rien prédire à leur endroit. Cependant, il faut être conscient que le temps d'apprentissage augmente beaucoup avec la taille de l'échantillon, car le réseau de neurones parcourt un grand nombre de fois son échantillon d'apprentissage.

Pour les variables en sortie, l'échantillon d'apprentissage doit contenir toutes les modalités en égales proportions, même si dans la population réelle, certaines modalités sont plus fréquentes (il faut par exemple avoir autant d'événements « négatifs » que d'événements « positifs », même quand les événements « négatifs » sont en réalité plus rares).

7.4. Quelques règles empiriques pour le dimensionnement d'un réseau

Dans un réseau à rétropropagation, il faut au moins 5 à 10 individus pour ajuster chaque poids. Pour la robustesse du réseau, on recommande d'avoir une seule couche cachée pour le réseau à fonction radiale de base, et une ou exceptionnellement deux pour le perceptron multicouches : plutôt que d'ajouter une troisième couche cachée, il vaut mieux

modifier d'autres paramètres, re-tester avec d'autres poids initiaux ou retravailler les données en entrée.

Un réseau à n unités d'entrée, une seule couche cachée, m unités dans la couche cachée et k unités de sortie possède $m(n + k)$ poids. Il faut donc un échantillon d'au moins $5m(n + k)$ individus pour l'apprentissage. Si l'on doit diminuer le nombre de nœuds en entrée, parce que l'échantillon d'apprentissage n'a pas une taille suffisante, il faut diminuer le nombre de variables prédictives. Supposons que l'on veuille passer de 20 à 10 variables prédictives. On testera toutes les combinaisons de 10 variables, en changeant seulement 2 ou 3 à chaque fois. Cette procédure est longue, mais tient compte du fait que certaines variables ne révèlent leur caractère prédictif que combinées avec certaines autres. Une autre procédure rapide et élégante consiste à effectuer une analyse en composantes principales (voir section 6.1) et à substituer les premières composantes principales aux variables en entrée du réseau. Le petit inconvénient de cette technique est qu'elle est par nature linéaire, et peut occulter d'importantes structures non linéaires.

La valeur de m est généralement comprise en $n/2$ et $2n$. Certains auteurs proposent d'aller jusqu'à $3n$; d'autres préconisent $3n/4$; d'autres suggèrent encore une valeur entre $\sqrt{nk}/2$ et $2\sqrt{nk}$. Le lecteur intéressé pourra consulter Ibeling Kaastra, Milton Boyd : *Designing a neural network for forecasting financial and economic time series*, Neurocomputing, 10, pp. 215-236, 1996. Pour un classement, m est généralement au moins égal au nombre de classes à prédire. Le mieux est de procéder à plusieurs essais, en mesurant à chaque fois le taux d'erreur sur l'échantillon de test, et en arrêtant d'augmenter m dès que ce taux a atteint un minimum, afin d'éviter le sur-apprentissage du réseau.

7.5. Normalisation des données

Rappelons que les données utilisées dans un réseau de neurones doivent être numériques et leurs modalités comprises dans l'intervalle $[0,1]$, ce qui implique, quand ce n'est pas le cas, une normalisation des données. Pour que le travail de normalisation décrit ci-dessous soit correct, il faut, bien entendu, que le jeu de données d'apprentissage couvre toutes les valeurs rencontrées dans la population tout entière, et, en particulier, les valeurs extrêmes des variables continues.

7.5.1. Variables continues

Même en les normalisant, les *variables continues* peuvent connaître le problème d'écrasement des valeurs normales par les valeurs extrêmes. Ainsi, la plupart des revenus mensuels se situent entre 0 et 10 000 euros ; mais si un revenu dépasse 100 000 euros, la normalisation standard de la variable « revenu », c'est-à-dire son remplacement par la variable :

$$\frac{\text{revenu} - \text{revenu minimum}}{\text{revenu maximum} - \text{revenu minimum}}$$

rendra presque indiscernable l'écart entre 5 000 et 10 000 euros, et le mettra sur le même plan que l'écart – beaucoup moins significatif – entre 95 000 et 100 000 euros.

Plusieurs moyens existent pour bien normaliser ce type de variables.

On peut discréteriser la variable, et la remplacer par exemple par ses quartiles.

On peut normaliser, non pas la variable, mais le logarithme de cette variable, qui « distend » le début de l'échelle.

On peut normaliser la variable linéairement, comme indiqué ci-dessus, pour ses valeurs comprises entre -3 et $+3$ fois l'écart-type²² σ autour de la moyenne μ , et envoyer les valeurs inférieures à $\mu - 3\sigma$ sur 0, et les valeurs supérieures à $\mu + 3\sigma$ sur 1. Dans cette variante, on peut éventuellement découper en deux l'intervalle $[\mu - 3\sigma, \mu + 3\sigma]$ en envoyant la moyenne μ sur le milieu 0,5 de l'intervalle, et en appliquant linéairement les deux demi-intervalles.

7.5.2. Variables discrètes

Pour normaliser des *variables discrètes* pour lesquelles la différence entre 0 et 1 est plus grande qu'entre 1 et 2, 2 et 3, etc., on peut effectuer la correspondance suivante :

- $0 \rightarrow 0$
- $1 \rightarrow \frac{1}{2}$
- $2 \rightarrow \frac{1}{2} + \frac{1}{4}$
- ...
- $n \rightarrow \sum_{k=1}^n 2^{-k}$

7.5.3. Variables qualitatives

Quant à la normalisation des *variables qualitatives*, elle pose un problème : elle fait apparaître une relation d'ordre parmi ses modalités qui est souvent artificielle et qui induit le réseau de neurones en erreur.

Un moyen fréquemment utilisé pour obvier à cette difficulté est d'avoir autant de nœuds que de modalités des variables qualitatives, en créant des variables binaires (appelées « indicatrices ») dont la valeur 1 ou 0 signifie que la variable qualitative a ou non cette modalité. L'inconvénient de cette solution est qu'elle entraîne un accroissement du nombre de nœuds, et donc de la complexité et du temps d'apprentissage du réseau, de même que de la taille de l'échantillon nécessaire à l'apprentissage.

Avant d'utiliser un réseau de neurones sur des données qualitatives, il faut donc réduire le plus possible le nombre de modalités de ces données.

²² Rappelons que lorsqu'une variable suit une loi normale de moyenne μ et d'écart-type σ , on trouve 68 % des observations dans l'intervalle $[\mu - \sigma, \mu + \sigma]$, 95 % des observations dans l'intervalle $[\mu - 2\sigma, \mu + 2\sigma]$ et 99,8 % des observations dans l'intervalle $[\mu - 3\sigma, \mu + 3\sigma]$.

7.6. Réseaux de neurones et séries temporelles

Dans un réseau de neurones, certaines variables prédictives peuvent correspondre à des valeurs décalées dans le temps de la variable cible $Y(t)$:

$$X_1(t) := Y(t - 1)$$

$$X_2(t) := Y(t - 2)$$

...

$$X_n(t) := Y(t - n).$$

Ces variables $X_i(t)$ sont introduites pour prédire la valeur de Y à l'instant t , connaissant ses valeurs aux instants antérieurs $t - 1, t - 2, \dots, t - n$. Il s'agit donc de prédire l'évolution d'une série temporelle. On dit que n est la taille de la fenêtre d'observation. Pour simplifier les explications, nous exposons ici le cas de la prédiction de Y au seul instant t , mais on peut aussi chercher à prédire Y aux instants $t, t+1, \dots, t+k$. On dit que $k+1$ est l'horizon de prédiction.

Selon le type d'application, le pas de la série temporelle, c'est-à-dire le décalage entre deux valeurs successives, peut être de 1 jour (prédictions sur les marchés financiers), 1 semaine, 1 mois (score d'attrition), voire 1 trimestre (prédictions macro-économiques).

Fréquemment, les variables $X_i(t) = Y(t - i)$ ne sont pas les seules variables prédictives entrées dans le modèle. Pour prédire l'évolution d'un cours de bourse, on utilisera toute une batterie d'autres indicateurs économiques : valeur du CAC40, taux d'intérêts, taux d'inflation, taux de chômage, taux de change avec certaines monnaies, etc. On peut ainsi être conduit à introduire des variables prédictives :

$$X'_i(t), X''_i(t), \dots$$

avec i variant entre 1 et n et des relations du type :

$$X'_{i+1}(t) = X'_i(t - 1),$$

$$X''_{i+1}(t) = X''_i(t - 1),$$

...

L'architecture d'un réseau utilisé pour traiter des séries temporelles a la forme représentée sur la Figure 7.6.

La variable cible est observée sur n périodes, en même temps que les variables prédictives supplémentaires. À chaque individu sont donc associés n enregistrements dans la base d'analyse. Les noeuds de la couche d'entrée du réseau sont au nombre de $n(m+1)$, où m est le nombre de variables prédictives supplémentaires, et le réseau ajuste les poids des n noeuds correspondant à la variable cible et les poids des $n.m$ noeuds correspondant aux m variables supplémentaires, ainsi que les poids des noeuds des couches cachées, de façon à optimiser en sortie la prédiction de la valeur de la variable cible à la $n+1^{\text{ème}}$ période.

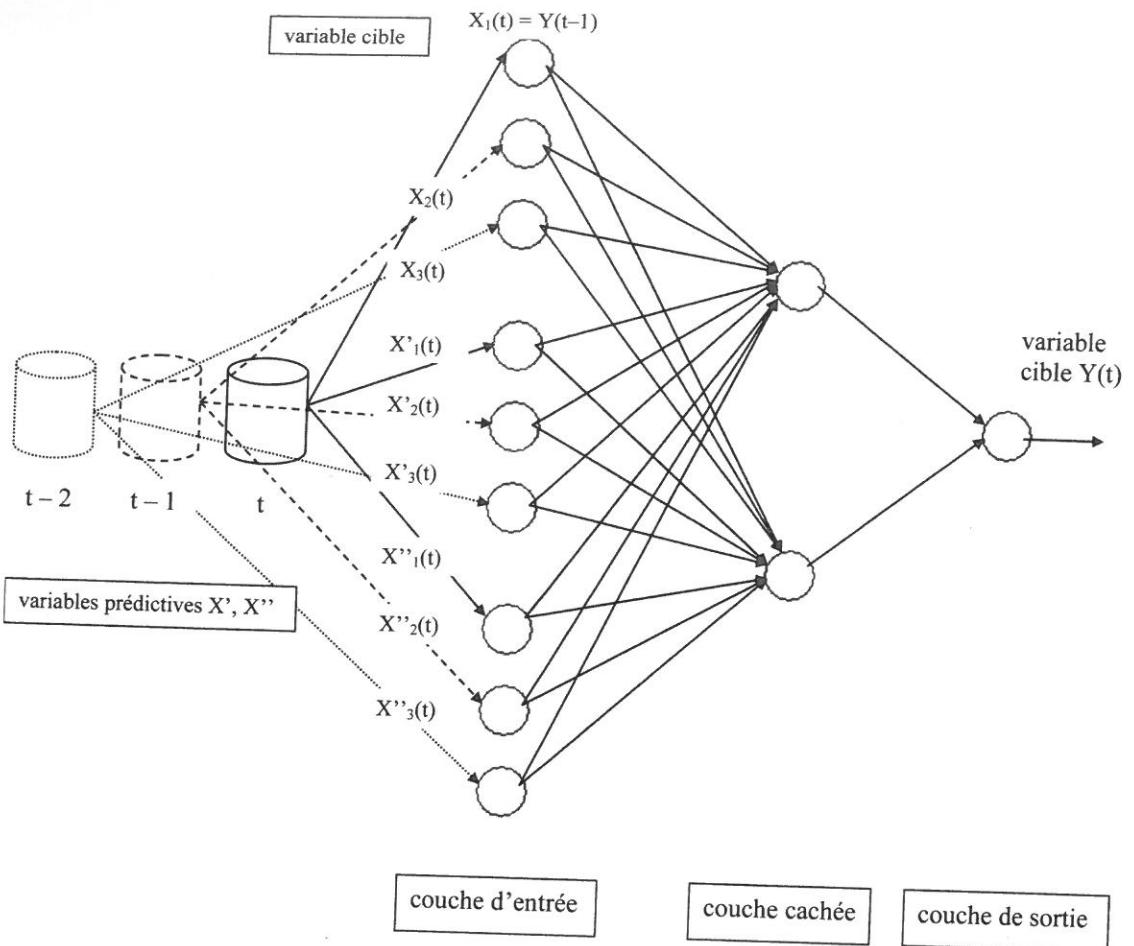


Figure 7.6 – Réseau de neurones pour une série temporelle

7.7. Les algorithmes d'apprentissage

L'algorithme de Levenberg-Marquardt a aujourd'hui souvent la faveur des spécialistes, car il converge plus vite que l'algorithme de rétropropagation du gradient et vers une solution meilleure. Mais il exige une grande capacité de mémoire de l'ordinateur sur lequel il tourne, proportionnelle au carré du nombre de noeuds. De ce fait, il est limité à des petits réseaux, avec peu de variables. Il est aussi limité à un noeud en sortie.

L'algorithme de rétropropagation du gradient est le plus ancien et le plus répandu, surtout sur les grands volumes de données. Mais il manque de fiabilité, en raison de sa sensibilité aux minima locaux.

L'algorithme de la descente du gradient conjugué est un bon compromis, puisque ses performances se rapprochent de celles de Levenberg-Marquardt en termes de convergence, mais qu'il est applicable à des réseaux plus complexes, avec éventuellement plusieurs sorties.

Citons pour finir l'algorithme de quasi-Newton, et les algorithmes génétiques étudiés dans la section 10.12.

7.8. Les principaux réseaux de neurones

Il existe différents modèles de réseaux de neurones. Les principaux, le perceptron multicouches (PMC), le réseau à fonction radiale de base (RBF : *Radial Basis Function*) et le réseau de Kohonen, sont présentés ci-dessous. Plus récents, les réseaux par estimation de densité de Specht (1990) sont utilisés, soit pour le classement (réseau PNN : probabilistic neural networks), soit pour la prédiction (réseau GRNN : general regression neural networks). Il existe aussi des réseaux analogues aux RBF mais basés sur la théorie mathématique des ondelettes.

Le réseau de Kohonen est un réseau à apprentissage non supervisé, utilisé pour la classification automatique, tandis que les autres réseaux cités, PMC, RBF... sont des réseaux à apprentissage supervisé, utilisés avec en sortie une ou plusieurs variables à expliquer.

7.8.1. Le perceptron multicouches

L'archétype des réseaux neuronaux est le perceptron multicouches (« multi-layer perceptron »). Il est particulièrement bien adapté à la découverte de modèles complexes et non linéaires. Sa puissance s'appuie sur la possibilité d'approcher n'importe quelle fonction suffisamment régulière par une somme de sigmoïdes (Figure 7.7). Comme son nom l'indique, ce réseau se décompose en plusieurs couches : les variables en entrée, la ou les variables en sortie, et un ou plusieurs niveaux cachés. Chaque nœud d'un niveau est connecté à l'ensemble des nœuds du niveau précédent.

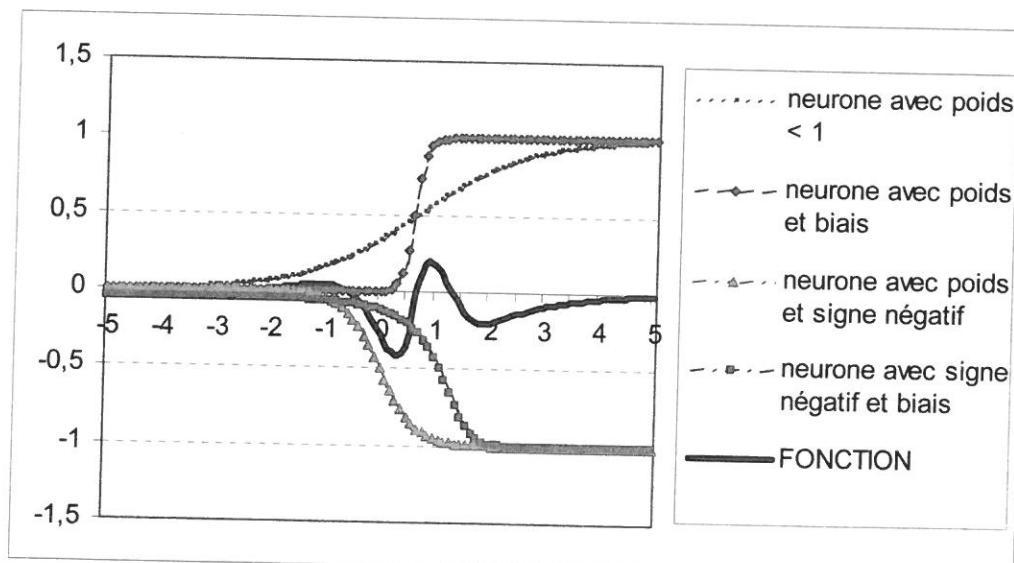


Figure 7.7 – Approximation d'une fonction par une somme de sigmoïdes

Le nombre de nœuds en entrée est toujours égal au nombre de variables du modèle, variables qui sont, le cas échéant, les variables « indicatrices » substituées aux variables

qualitatives d'origine (voir section 7.5.3). Il y a le plus souvent un seul nœud en sortie. Pour le choix du nombre de nœuds de la couche cachée, voir la section 7.4.

Pour expliquer le fonctionnement du perceptron multicouches, nous nous placerons dans le cas particulier, mais encore fréquent, du PMC à rétropropagation du gradient.

À chaque connexion est associé un poids qui évoluera au cours de l'apprentissage. Le réseau commence son apprentissage en donnant une valeur aléatoire à chacun des poids et en calculant la valeur en sortie à partir d'un ensemble d'enregistrements pour lequel la valeur attendue en sortie est connue : l'échantillon d'apprentissage. Le réseau compare alors la valeur calculée en sortie avec la valeur attendue, et calcule une *fonction d'erreur*, qui peut être la somme des carrés des erreurs commises pour chaque individu de l'échantillon d'apprentissage :

$$\sum_i \sum_j (A_{ij} - O_{ij})^2,$$

où la première somme est sur les individus de l'ensemble d'apprentissage, la seconde est sur les nœuds en sortie, A_{ij} (resp. O_{ij}) est la valeur attendue (resp. obtenue) du $j^{\text{ème}}$ nœud pour le $i^{\text{ème}}$ individu.

Le réseau ajuste ensuite les poids des différents nœuds, en regardant à chaque fois si la fonction d'erreur augmente ou diminue. Il s'agit, comme dans une régression classique, de résoudre un problème de moindres carrés.

S'il y a n connexions dans le réseau, chaque n -uplet (p_1, p_2, \dots, p_n) de poids peut être représenté dans un espace à $n+1$ dimensions, la dernière dimension représentant la fonction d'erreur ϵ . L'ensemble des valeurs $(p_1, p_2, \dots, p_n, \epsilon)$ est une « surface » (ou plutôt une hypersurface) dans un espace de dimension $n+1$, la « surface d'erreur », et l'ajustement des poids afin de minimiser la fonction d'erreur peut être vu comme un déplacement sur la surface d'erreur dont on recherche le point le plus bas. Contrairement aux modèles linéaires dans lesquels la surface d'erreur est un objet mathématique bien défini et bien connu (par exemple, de forme parabolique) dont on peut déterminer le point le plus bas par le calcul, les réseaux de neurones sont des modèles non-linéaires complexes dont la surface d'erreur a un tracé irrégulier, traversé de collines, de vallées, de plateaux, de profonds ravins, etc. Pour trouver le point le plus bas sur cette surface dont on ne possède pas la carte, il faut donc l'explorer. Dans l'algorithme de rétropropagation du gradient, le déplacement sur la surface d'erreur se fait en suivant la ligne de plus grande pente, qui permet d'espérer arriver à un point le plus bas possible. La question se pose de savoir à quelle vitesse il convient de descendre la pente. Si l'on va trop vite, on risque de dépasser le point bas ou de partir dans une mauvaise direction ; si l'on va trop lentement, il faudra de trop nombreuses itérations au réseau pour fournir une solution. Quand on parle d'une itération, il s'agit de la soumission au réseau de l'ensemble tout entier d'apprentissage, de la comparaison des sorties attendues et obtenues, et du calcul de la fonction d'erreur. La fourchette des nombres d'itérations possibles est très large, mais l'ordre de grandeur est de 10 000.

La bonne vitesse est proportionnelle à la pente de la surface et à un paramètre important : le taux d'apprentissage. Ce taux, compris entre 0 et 1, détermine donc l'importance de la modification des poids durant l'apprentissage. Il est intéressant de faire varier ce taux, qui sera important au départ (entre 0,7 et 0,9) pour permettre une exploration rapide de la surface d'erreur et une approche rapide des meilleures solutions (minimums de la surface), puis diminuera en fin d'apprentissage pour permettre d'arriver le plus près possible d'une solution optimale. C'est cette diminution du taux d'apprentissage

qui permet d'éviter que, dans les situations du type de la Figure 7.8, on passe de l'optimum local A à l'optimum local C, avec éventuellement des oscillations entre A et C, sans descendre dans l'optimum global B.

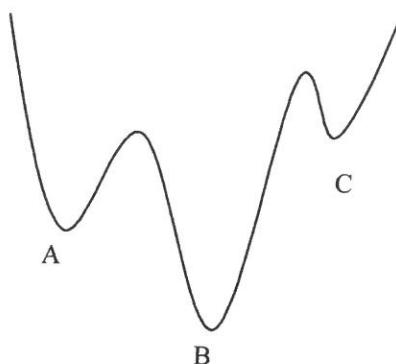


Figure 7.8 – Optimum local et optimum global

Un second paramètre important conditionne les performances d'un perceptron multicouches : le *moment*, qui fait que les poids ont tendance à conserver le même sens d'évolution, en croissant ou en décroissant, car un facteur intègre les ajustements de poids précédents. Le moment limite les oscillations qui pourraient être provoquées par des irrégularités dans les exemples d'apprentissage. Le moment fait que, si l'on se déplace plusieurs fois de suite dans la même direction sur la surface d'erreur, on a tendance à poursuivre sur sa lancée, ne pas se laisser « piéger » par des minimums locaux (comme le point A sur la Figure 7.8) et à les dépasser pour atteindre les minimums globaux (comme le point B sur cette figure). De même que le taux d'apprentissage diminue au fur et à mesure de l'apprentissage, de même le moment augmente souvent au cours de l'apprentissage, pour permettre au réseau d'approcher doucement d'une solution globalement optimale.

En résumé, le taux d'apprentissage contrôle l'importance de la modification des poids durant le processus d'apprentissage ; plus il est élevé, plus l'apprentissage est rapide, mais plus le réseau risque de converger vers une solution globalement non optimale. Le moment agit comme un paramètre d'amortissement en réduisant les oscillations et en aidant à atteindre la convergence ; plus il est faible, plus le réseau « s'adapte au terrain », mais plus l'influence des données extrêmes sur les poids se fait sentir. En quelque sorte, le taux d'apprentissage contrôle la vitesse de déplacement et le moment contrôle la rapidité des changements de direction sur la surface d'erreur ; au début du voyage, on va vite et dans toutes les directions ; à la fin, on ralentit et on vire moins.

On aperçoit le risque majeur d'une modélisation par réseaux de neurones : celui de voir le réseau converger vers une solution localement mais non globalement optimale.

Ce risque a suscité le développement d'outils graphiques de représentation en temps réel du taux d'erreur en apprentissage et en validation, permettant d'interrompre

l'apprentissage dès l'apparition de sur-apprentissage et la hausse du taux d'erreur en validation (Figure 7.9).

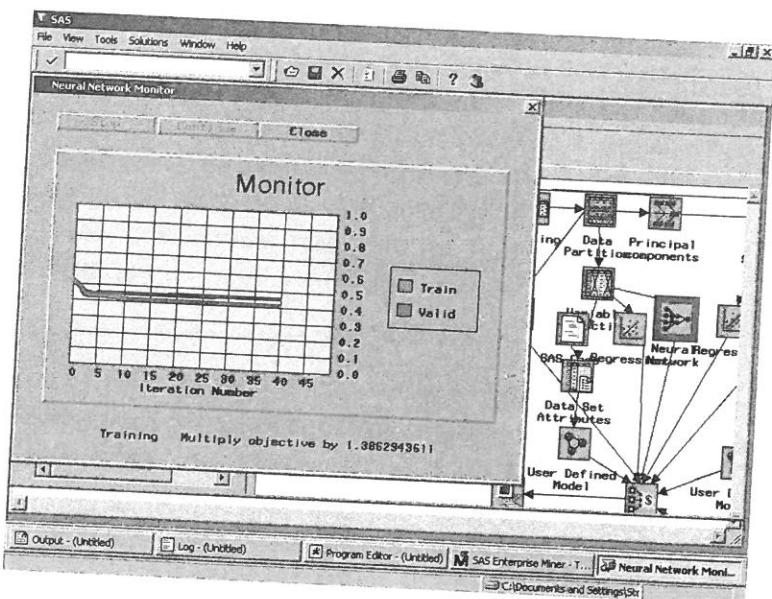


Figure 7.9 – Contrôle graphique d'un réseau de neurones avec SAS Enterprise Miner

7.8.2. Le réseau à fonction radiale de base

Le réseau RBF est à apprentissage supervisé, comme le perceptron multicouches avec lequel il présente des similitudes. Cependant, il travaille avec une seule couche cachée et utilise, pour calculer la valeur de chaque nœud de la couche cachée pour une observation, non pas la somme des valeurs pondérées des nœuds du niveau précédent, mais la distance séparant dans l'espace cette observation du centre du nœud. Contrairement aux poids d'un perceptron multicouches, les centres de la couche cachée d'un réseau RBF ne sont pas ajustés à chaque itération au cours de l'apprentissage (mais on en ajoute parfois si l'espace n'est pas suffisamment découpé). Dans un perceptron, la modification d'un poids synaptique force à réévaluer tous les autres, au contraire d'un réseau RBF où les neurones cachés se partagent l'espace et sont quasiment indépendants les uns des autres. De là vient une plus grande rapidité de convergence des réseaux RBF en phase d'apprentissage, ce qui constitue l'un de leurs avantages.

De même que la surface de réponse (l'ensemble des valeurs) d'un nœud d'une couche cachée d'un perceptron multicouches, avant application de la fonction (généralement non linéaire) de transfert, est un hyperplan $\sum_i p_i X_i = K$, de même, la surface de réponse d'un nœud de la couche cachée d'un réseau RBF est une hypersphère $\sum_i (X_i - \omega_i)^2 = R^2$, et la réponse du nœud à un individu (x_i) est une fonction décroissante Γ de la distance séparant l'individu de cette hypersphère. Cette fonction Γ étant une gaussienne, la surface de réponse du nœud, après application de la fonction de transfert, est une surface gaussienne, c'est-à-dire « en cloche » (Figure 7.10).

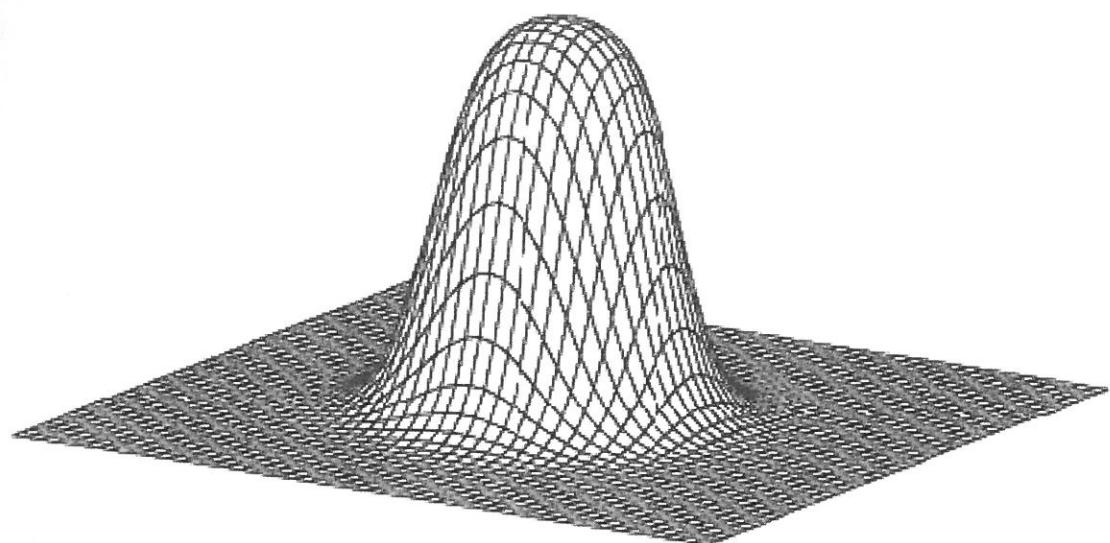


Figure 7.10 – Surface de réponse d'un nœud radial

Si l'on compare les réseaux PMC et RBF, on relève les différences suivantes :

Tableau 7.1 – Comparaison des réseaux PMC et RBF

Réseau →	PMC	RBF
« Poids »	Poids p_i	Centre ω_i
Couche(s) cachée(s)	Fonction combinaison	Produit scalaire $\sum_i p_i x_i$
	Fonction transfert	Logistique $s(X) = 1/(1 + \exp(-X))$
	Nombre de couches cachées	≥ 1
Couche de sortie	Fonction combinaison	Produit scalaire $\sum_k p_k x_k$ Combinaison linéaire de gaussiennes $\sum_k \lambda_k \Gamma_k$ (voir ci-dessous)
	Fonction transfert	Logistique $s(X) = 1/(1 + \exp(-X))$ Fonction linéaire $f(X) = X$
Rapidité	Plus rapide en mode « application du modèle »	Plus rapide en mode « apprentissage du modèle »
Avantage	Meilleure généralisation	Moindre risque de convergence non optimale

Finalement, la réponse globale du réseau à chaque individu (x_i) qui lui est présenté est :

$$\sum_{k=1}^{\text{nb de noeuds cachés}} \lambda_k \exp \left[-\frac{1}{2} \frac{\sum_{i=1}^{\text{nb de noeuds en entrée}} (x_i - \omega_i^k)^2}{\sigma_k^2} \right].$$

Le point délicat du paramétrage d'un réseau RBF concerne le choix du nombre de nœuds de la couche cachée, de leurs centres $\Omega_k = (\omega_i^k)$ et de leurs rayons σ_k .

Le nombre de nœuds est généralement choisi par l'utilisateur, même si le réseau peut en créer d'autres pour améliorer la précision des résultats. Il faut en prévoir un nombre suffisamment élevé, généralement plus que dans un perceptron multicouches, pour bien modéliser la structure des données. Un nombre de quelques centaines n'est pas exceptionnel. En effet, la décroissance rapide de la gaussienne entraîne une capacité d'extrapolation moindre du réseau RBF quand on s'éloigne des centres des nœuds de la couche cachée. Ceux-ci doivent donc être suffisamment nombreux. On peut y voir une faiblesse du réseau RBF par rapport au perceptron multicouches, mais on peut aussi se dire que l'on est plus à l'abri de certaines extrapolations hasardeuses d'un perceptron multicouches.

Une fois choisi le nombre de nœuds, il faut se préoccuper de leurs centres. Certains réseaux les positionnent aléatoirement. D'autres ont recours à la méthode des centres mobiles (voir section 8.9.1) ou aux réseaux de Kohonen (voir ci-dessous) pour découper l'espace en classes (en partitions) conformes à la distribution des données. De la sorte, si les données sont distribuées en paquets, les centres de ces paquets seront choisis pour être les centres du réseau RBF. De plus, on positionnera plus de centres dans les zones de forte densité en observations (adaptation à l'entrée), ou dans les zones où le résultat à prédire varie plus rapidement (adaptation à la sortie). Le résultat est, bien entendu, supérieur à celui obtenu par un simple positionnement aléatoire.

Le dernier point du paramétrage concerne les rayons des nœuds de la couche cachée, qui sont l'écart-type des gaussiennes. Une solution simple consiste à choisir des rayons égaux à deux fois la distance moyenne entre centres. S'ils sont trop grands, le réseau manque des détails de structure et perd en précision. S'ils sont trop petits, l'espace est mal couvert par les gaussiennes, et le réseau est obligé d'interpoler entre ces surfaces, d'où une moindre capacité de généralisation des résultats obtenus en phase d'apprentissage. De même que pour les centres, on choisira des rayons plus faibles dans les zones de forte densité en observations, ou dans les zones où le résultat à prédire varie plus rapidement. Plusieurs méthodes existent pour déterminer au mieux les rayons : une méthode intéressante relève des k -plus proches voisins (voir section 10.2), et consiste à regarder, pour chaque centre de nœud, où se trouvent ses k -plus proches voisins (k étant à choisir convenablement par l'utilisateur), et à retenir comme rayon la distance moyenne à ces k -plus proches voisins. Cette méthode a le mérite de s'adapter à la structure des données.

Le réseau RBF, comparé au réseau PMC, présente le gros avantage de ne nécessiter qu'une seule couche cachée, et d'utiliser le plus souvent (hormis quelques variantes sophistiquées) des fonctions de combinaison et de transfert *linéaires* dans la couche de

sortie (voir le Tableau 7.1). Il en résulte une bien plus grande rapidité d'apprentissage du réseau, et beaucoup moins de soucis de réglages compliqués de paramètres pour l'utilisateur. On évite aussi le risque, propre au mécanisme de rétropropagation du perceptron multicouches, de convergence vers une solution localement mais non globalement optimale.

Le point faible du réseau à fonction radiale de base, par rapport au perceptron multicouches, est qu'il peut nécessiter un grand nombre de noeuds dans sa couche cachée, ce qui augmente le temps d'exécution du réseau en phase d'application (alors que le réseau RBF est plus rapide en phase d'apprentissage), sans malgré tout parvenir toujours à une parfaite modélisation des structures complexes et des données irrégulières. Le réseau RBF peut aussi être gêné par un nombre un peu trop important de variables en entrée. Enfin, peut-être plus encore que le perceptron multicouches, le réseau RBF requiert un ensemble d'apprentissage couvrant bien toutes les configurations et toutes les modalités de variables susceptibles de survenir lors de son application à l'ensemble de la population étudiée. Les avantages et inconvénients du réseau RBF sont un peu ceux des réseaux à estimation de densité de probabilité. Le réseau PMC est celui qui offre la meilleure capacité de généralisation, notamment sur données bruitées.

7.8.3. Le réseau de Kohonen

Le réseau de Kohonen est le plus utilisé des réseaux à apprentissage non supervisé. On parle aussi de réseau auto-adaptatif ou auto-organisé, car le réseau « s'auto-organise » autour des données. Autres synonymes : carte de Kohonen, SOM (Self Organizing Map).

Comme tout réseau de neurones, il se décompose en couches de noeuds et en connexions entre ces noeuds. La grande différence avec les réseaux précédemment décrits est qu'il n'y a ici pas de variable à prédire. Son but est « d'apprendre » la structure des données pour pouvoir y distinguer des classes (des « clusters »).

Le réseau de Kohonen se compose de deux niveaux (Figure 7.11) :

- la couche d'entrée, avec un noeud pour chacune des n variables utilisées dans la classification ;
- une couche de sortie, dont les noeuds sont disposés sous forme d'une grille généralement carrée ou rectangulaire (parfois hexagonale) de $l \times m$ noeuds (on peut avoir $l \neq m$), chacun de ces $l \times m$ noeuds étant connecté à chacun des n noeuds de la couche d'entrée, la connexion ayant un certain poids p_{ijk} ($i \in [1, l]$, $j \in [1, m]$, $k \in [1, n]$).

Les noeuds de la couche de sortie ne sont pas connectés entre eux mais une distance est définie entre eux, de sorte que l'on peut parler du « voisinage » d'un noeud.

Les noeuds de la couche d'entrée correspondent aux individus à classer et cette couche sert à la présentation des individus ; les états de ses noeuds sont les valeurs des variables caractérisant les individus à classer. C'est pour cela que cette couche contient n noeuds, où n est le nombre de variables utilisées dans la classification.

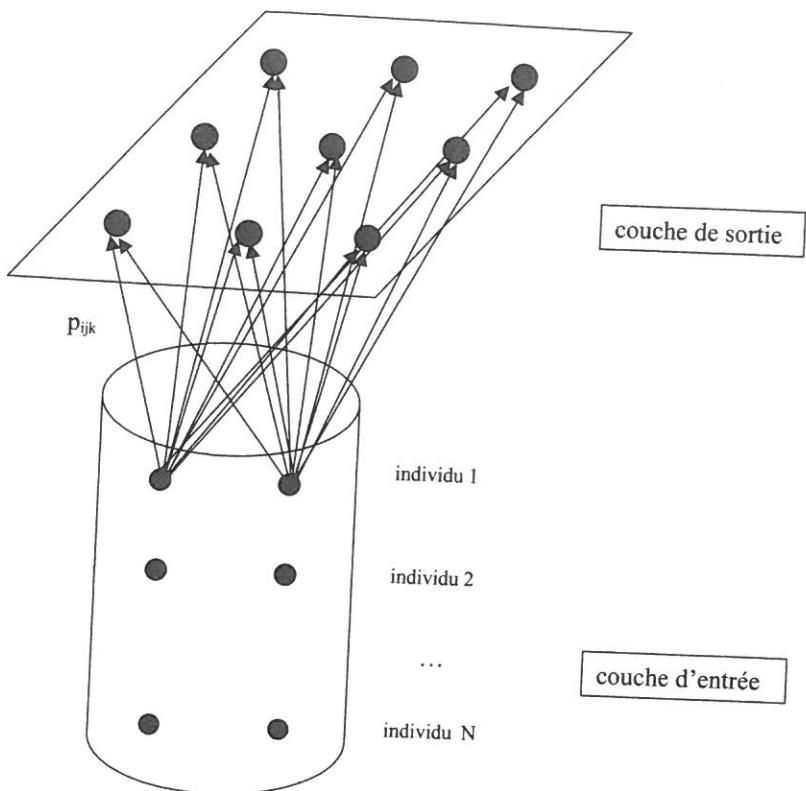


Figure 7.11 – Réseau de Kohonen

La grille sur laquelle sont disposés les nœuds de la couche de sortie est appelée « carte topologique ». La forme et la taille de cette grille sont généralement choisies par l'utilisateur, mais elles peuvent aussi évoluer au cours de l'apprentissage. À chaque nœud (i,j) de sortie étant associé un vecteur de poids $(p_{ijk})_{k \in [1,n]}$, la réponse de ce nœud à un individu $(x_k)_{k \in [1,n]}$ est, par définition, la distance euclidienne :

$$d_{ij}(x) = \sum_{k=1}^n (x_k - p_{ijk})^2.$$

Ceci dit, comment se fait l'apprentissage d'un réseau de Kohonen ?

Tout d'abord, les poids p_{ijk} sont initialisés aléatoirement. Ensuite, pour chaque individu (x_k) de l'échantillon d'apprentissage sont calculées les réponses des $l \times m$ nœuds de la couche de sortie.

Le nœud retenu pour représenter (x_k) est le nœud (i,j) pour lequel $d_{ij}(x)$ est minimum. On dit que ce nœud est « activé ». Ce nœud et tous les nœuds voisins voient leurs poids ajustés afin de les rapprocher de l'individu en entrée. Les nœuds voisins de (i,j) sont par exemple les huit nœuds $(i-1,j)$, $(i+1,j)$, $(i,j-1)$, $(i,j+1)$, $(i+1,j+1)$, $(i+1,j-1)$, $(i-1,j+1)$, $(i-1,j-1)$. La taille du voisinage diminue généralement au cours de l'apprentissage : au début, le voisinage peut être la grille tout entière ; à la fin, le voisinage peut être réduit au nœud lui-même. Ces réglages font partie des paramètres du réseau.

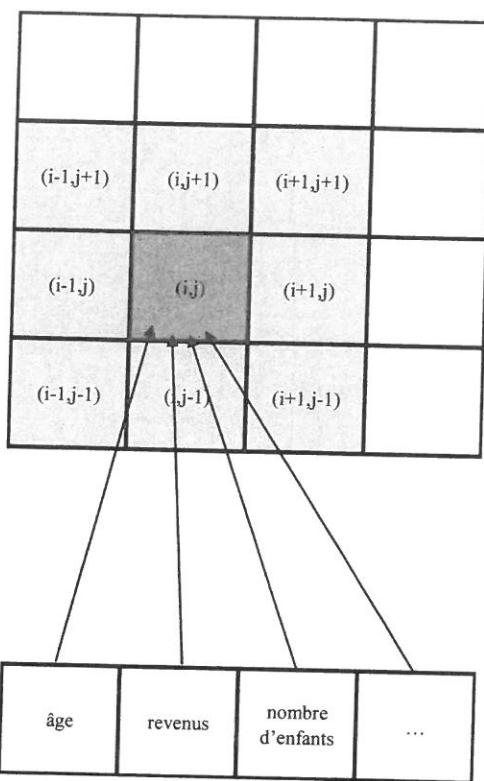


Figure 7.12 – Activation d'un nœud d'un réseau de Kohonen

Les nouveaux poids d'un voisin (I,J) du « gagnant » (i,j) sont :

$$p_{IJK} + \Theta \cdot f(i,j; I, J) \cdot (x_k - p_{IJK}) \text{ pour tout } k \in [1, n],$$

où $f(i,j; I, J)$ est une fonction décroissante de la distance entre les nœuds (i,j) et (I,J) , telle que $f(i,j,i,j) = 1$. Il peut ainsi s'agir d'une gaussienne : $\exp(-\text{distance}(i,j; I, J)^2/2\sigma^2)$.

Le paramètre $\Theta \in [0,1]$ est un taux d'apprentissage qui, comme dans le cas du perceptron multicouches, évolue au fil de l'apprentissage en décroissant, de façon linéaire ou exponentielle.

C'est l'extension de l'ajustement des poids à tout le voisinage du nœud « gagnant », qui rapproche les nœuds voisins de (i,j) de l'individu (x_k) en entrée, et qui permet à des individus proches dans l'espace des variables d'être représentés par des nœuds identiques ou voisins de la couche, de même qu'à des stimuli proches répondent des neurones voisins dans le cortex. Tout se passe comme si la grille d'un réseau de Kohonen était en caoutchouc et si on la déformait pour lui faire traverser le nuage des individus en s'approchant au plus près des individus (Figure 7.13). Contrairement à un plan factoriel (voir section 6.1), il s'agit d'une projection non-linéaire.

Une fois que tous les individus de l'échantillonnage d'apprentissage ont été présentés au réseau et que tous les poids ont été ajustés, l'apprentissage est achevé.

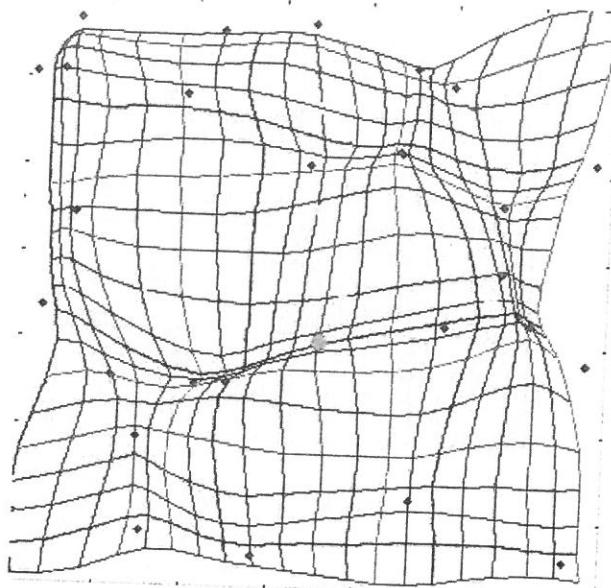


Figure 7.13 – Représentation d'un réseau de Kohonen

En résumé, au cours de l'apprentissage du réseau :

- pour chaque individu, un seul nœud de sortie est activé (« le gagnant ») ;
- le gagnant et ses voisins voient leurs poids ajustés ;
- l'ajustement est tel qu'à deux individus proches correspondent deux nœuds proches en sortie ;
- des groupes (clusters) de nœuds se forment en sortie.

En phase d'application, le réseau de Kohonen fonctionne en représentant chaque individu en entrée par le nœud du réseau qui lui est le plus proche au sens de la distance définie ci-dessus. Ce nœud sera la classe de l'individu.

Cet algorithme fait penser aux méthodes des centres mobiles et des *k-means* (voir la section 8.9.1). Il présente toutefois une différence importante. Dans la méthode des *k-means*, l'introduction d'un nouvel individu dans un groupe ne conduit qu'au recalcul du barycentre (centre de gravité) du groupe, sans répercussion sur les autres barycentres. En revanche, l'introduction d'un nouvel individu dans un réseau de Kohonen conduit à l'ajustement, non seulement du nœud le plus proche de l'individu, mais aussi des nœuds voisins. On s'intéresse au voisinage du nœud « gagnant », tandis que l'on ne s'intéresse pas au voisinage du barycentre « gagnant ».

Une autre grande différence entre les réseaux de Kohonen et la méthode des centres mobiles ou celle des *k-means* est que, contrairement à ces dernières méthodes, la classification de Kohonen se fait en opérant une réduction du nombre de dimensions de l'espace des variables, comme en analyse factorielle, le nouvel espace de travail étant très généralement de dimension 2, comme dans notre exposé, ou exceptionnellement de dimension 3 ou 1.

10.9.8. Le modèle additif généralisé

Le modèle additif généralisé de Hastie et Tibshirani (1990) est encore plus général que le modèle linéaire généralisé, puisque l'on écrit ici :

$$g(E(Y/X=x)) = \beta_0 + \sum_i f_i(x_i),$$

en utilisant toujours une fonction de lien g , mais en remplaçant le produit des coefficients β_i par une fonction f_i générale (spline, par exemple). Le modèle est dit additif en raison de la sommation sur les i . Cette modélisation est puissante, mais, comme avec les réseaux de neurones, il faut prendre garde au sur-apprentissage et à l'interprétabilité des résultats.

Il est implémenté de façon très complète dans le logiciel R (package *mgcv*), et aussi à titre expérimental dans SAS 9.

10.10. Le classement et la prédiction par réseaux de neurones

Les réseaux neuronaux à apprentissage supervisé, et surtout le perceptron multicouches et le réseau à fonction radiale de base, sont utilisés à la fois pour le classement et la prédiction.

Le classement peut s'opérer de deux manières.

Il faut se souvenir que les nœuds de la couche de sortie prennent des valeurs continues entre 0 et 1.

On peut créer dans la couche de sortie un nœud par classe à prédire, et considérer que la classe d'un individu est celle dont le nœud renvoie la plus grande valeur (la plus proche de 1). On est parfois plus restrictif : on n'attribue une classe à un individu que si le nœud correspondant renvoie une valeur supérieure à un certain seuil d'acceptation (par exemple : 0,5), et si les autres nœuds renvoient tous des valeurs inférieures à un certain seuil de rejet (par exemple : 0,5). Si ces conditions ne sont pas remplies (si l'on a, par exemple, 0,8 et 0,6), la classe de l'individu est décrétée imprévisible. L'apprentissage de ce type de réseau se fait en associant, à chaque individu de l'ensemble d'apprentissage, des valeurs de nœuds de sortie égales à 1 pour le nœud correspondant à la classe (connue) de l'individu, et 0 pour les autres nœuds.

La seconde façon de procéder n'est réellement utilisable qu'en présence de deux classes à prédire. Elle consiste à n'avoir qu'un seul nœud dans la couche de sortie, et à considérer que la valeur 0 du nœud correspond à une classe et la valeur 1 à l'autre classe. En apprentissage, le nœud ne prend que les valeurs 0 et 1, mais en phase d'application toutes les valeurs comprises entre 0 et 1 peuvent être prises. En phase de test, avec un échantillon distinct de celui d'apprentissage, nous choisirons donc une valeur seuil séparant les classes. Cela peut se faire en cherchant sur la courbe ROC ou dans l'ensemble des matrices de confusion un compromis entre la sensibilité du réseau (aptitude à détecter les événements) et sa spécificité (aptitude à ne pas détecter à tort de faux événements).

10.10.1. Avantages des réseaux de neurones

Les réseaux de neurones présentent certains avantages spécifiques.

Tout d'abord, les réseaux de neurones prennent bien en compte les relations non linéaires et des interactions complexes entre les variables, du moins si l'on y met le prix en termes de nombre de nœuds dans la ou les couches cachées. La contrepartie de cette complexité et du nombre éventuellement important de nœuds du réseau, est le risque de sur-apprentissage et le fait qu'il n'est pas facile d'extraire, de l'ensemble de toutes les variables prédictives potentielles, le sous-ensemble des variables les plus pertinentes.

Deuxième avantage : la technique des réseaux de neurones est *non-paramétrique*, ce qui signifie qu'elle ne suppose pas que les variables explicatives suivent des lois probabilistes particulières.

Troisième avantage : certains réseaux, le perceptron multicouches plus que le RBF, résistent mieux que d'autres techniques aux données défectueuses. Si une variable en entrée d'un perceptron multicouches est vraiment trop bruitée, le nœud correspondant verra son poids diminuer jusqu'à zéro.

Enfin, les réseaux de neurones sont aptes à modéliser des problèmes très variés : classification, classement, prédiction, séries temporelles (prévisions économiques), reconnaissance de caractères optiques (signatures) et lecture automatique de l'écriture manuscrite sur les enveloppes (tri postal³⁸) et les chèques (réseau de neurones utilisé par AT&T à 7 couches), analyse linguistique (text mining), reconnaissance et synthèse de la parole, reconnaissance des visages, reconnaissance des objets en fonction de leur forme ou de leur signal dans les domaines militaires et industriels (scanner des images vidéos d'une station de métro afin de déterminer automatiquement son encombrement, détecter un dysfonctionnement sur une machine en analysant ses vibrations), pilotes automatiques d'avions, gestion des machines dans un processus industriel de production, traitement du signal dans le domaine médical (diagnostic d'un signal cardiaque, estimation de la taille d'une tumeur), prévisions météorologiques, jeu d'échecs (un module d'apprentissage à base de réseaux de neurones a permis à l'ordinateur *Deep Blue* de vaincre en 1997 le champion du monde Garry Kasparov)...

10.10.2. Inconvénients des réseaux de neurones

L'utilisation des réseaux de neurones présente plusieurs inconvénients sérieux, déjà mentionnés au sujet de la classification neuronale :

- la convergence vers la meilleure solution globale n'est pas toujours assurée ;
- le risque important de sur-apprentissage, si le nombre de cas est trop faible par rapport au nombre de nœuds (voir la section 7.4) ;
- l'impossibilité de traiter un trop grand nombre de variables ;
- les résultats non explicites, ce qui est rédhibitoire pour certaines applications comme le diagnostic médical ou les pilotes automatiques d'avions (dès qu'il existe une couche cachée, il faut effectuer une analyse *a posteriori* pour connaître le poids des différentes variables entrant dans le calcul du score) ;

³⁸ La reconnaissance automatique des codes postaux manuscrits est l'un des grands classiques des réseaux de neurones, en raison de ses enjeux économiques, des vastes bases de données existantes (Service Postal des États-Unis) et des intéressants problèmes qu'elle soulève : avant même de chercher à déchiffrer chaque caractère, en tenant compte de sa taille, de son orientation et de son style, il faut déjà parvenir à les séparer les uns des autres, tâche plus ardue qu'il n'y paraît.

- la difficulté d'utilisation correcte, les paramètres (nombre de couches cachées, nombre de noeuds, taux d'apprentissage, moment...) étant nombreux et délicats à régler ;
- les réseaux de neurones ne s'appliquent naturellement qu'aux variables continues dans l'intervalle [0,1], ce qui provoque une multiplication du nombre de noeuds pour les variables qualitatives.

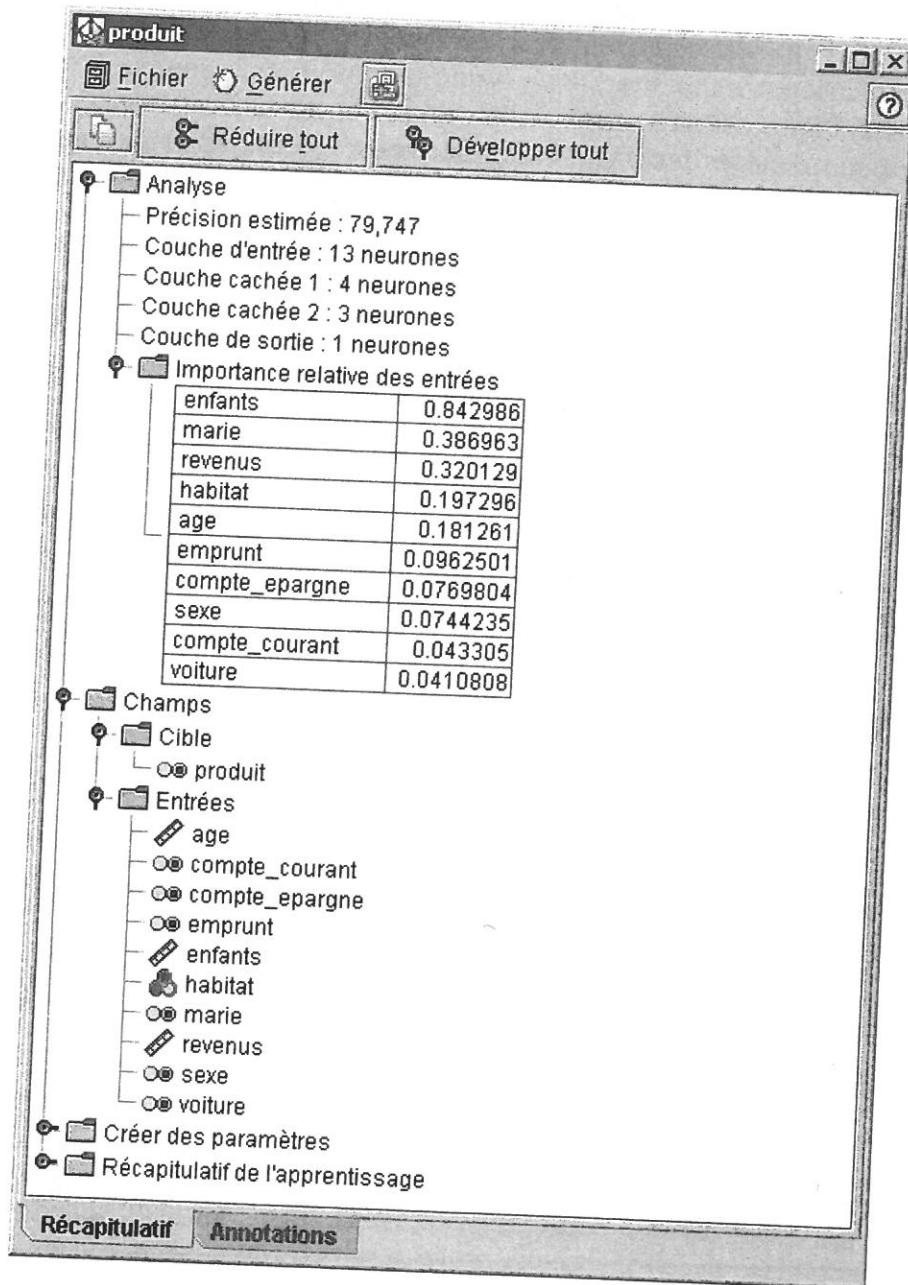


Figure 10.82 – Résultat de l'apprentissage d'un réseau de neurones par Clementine®

Comme signalé dans le chapitre consacré aux réseaux de neurones, la convergence d'un réseau vers une bonne solution n'est pas toujours assurée, car un réseau considère qu'il a trouvé une bonne solution lorsque la fonction d'erreur, qui est à minimiser par

l'ajustement des poids, rencontre un minimum et ne diminue plus lorsque les poids continuent d'être ajustés. Mais ce minimum peut n'être que local et non global. Il faut donc éviter au réseau d'être « piégé » dans un minimum local. Pour cela, on commence l'apprentissage avec une forte vitesse d'ajustement des poids, puis on diminue cette vitesse au cours de l'apprentissage, lorsque le réseau se rapproche d'une bonne solution. On règle aussi le *moment* du réseau en sorte de limiter les oscillations du réseau autour des solutions ; le moment est ce qui pousse les poids à continuer d'évoluer dans le sens de leur évolution.

Au sujet des résultats non explicites, rappelons (section 7.4) qu'un réseau à n unités d'entrée, une seule couche cachée, m unités dans la couche cachée et 1 unité de sortie possède $n \cdot m + m$ poids. Le nombre de paramètres d'un réseau de neurones peut donc rapidement devenir important. De plus, des jeux de poids différents peuvent conduire à des prédictions similaires. L'ensemble des poids d'un réseau ne permet donc pas de comprendre les résultats fournis par ce réseau, contrairement à ce qui se passe avec un arbre de décision et ses feuilles. En revanche, la plupart des logiciels savent classer les variables du modèle par ordre de pertinence.

À titre d'exemple (Figure 10.82), après l'apprentissage, Clementine® itère sur chacune des variables en entrée et calcule les performances du réseau en l'absence de signal en entrée sur cette variable. Ce calcul donne un poids à chaque variable et permet donc de les classer par ordre décroissant d'impact sur la précision.

En conclusion, nous n'utiliserons les réseaux de neurones qu'à la double condition de disposer d'un échantillon d'apprentissage suffisamment important et d'avoir constaté que les méthodes classiques ne donnaient pas satisfaction, par exemple en raison de relations hautement non linéaires entre les variables.

10.11. Le classement par « support vector machines » (SVM)

10.11.1. Introduction aux SVM

Basée sur les travaux de Vladimir Vapnik en théorie de l'apprentissage, cette méthode récente de classement est une sorte d'analyse discriminante généralisée, effectuée dans un espace de dimension assez grande pour qu'existe une séparation linéaire. Elle se déroule en deux étapes. Dans la première, une transformation non linéaire Φ fait passer de l'espace d'origine dans un espace de dimension plus grande (voire infinie) mais doté d'un produit scalaire. Seconde étape : dans cet espace, on cherche un séparateur linéaire $f(x) = a \cdot x + b$ (exemple : fonction discriminante de Fisher), qui est un hyperplan remplissant simultanément deux conditions :

- il sépare bien les groupes (précision du modèle), au sens où $f(x) > 0 \Rightarrow$ classe A et $f(x) \leq 0 \Rightarrow$ classe B ;
- il est le plus loin possible de toutes les observations (robustesse du modèle), sachant que la distance d'une observation x à l'hyperplan est $|a \cdot x + b| / \|a\|$.

L'exemple de transformation Φ de la Figure 10.83 montre bien que c'est Φ qui permet de prendre en compte la non-linéarité.

Input Space: $\vec{x} = (x_1, x_2)$ (2 Attributes)

Feature Space: $\Phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$ (6 Attributes)

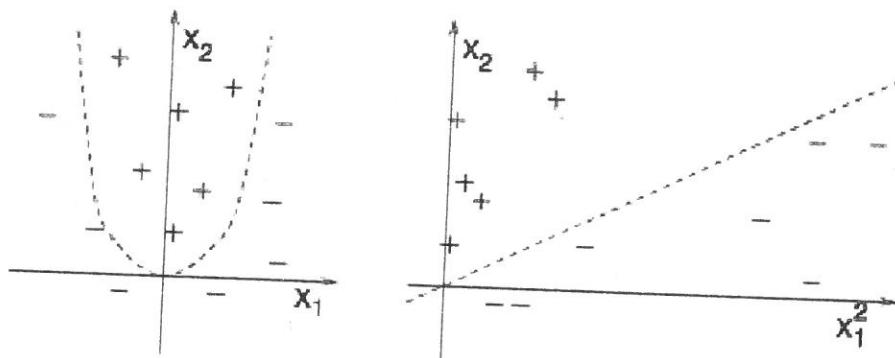


Figure 10.83 – Exemple de transformation dans un SVM

La Figure 10.84 (réf : Tan, Steinbach, Kumar) montre comment se traduit visuellement la seconde condition : le choix de l'hyperplan B_1 est supérieur à celui de l'hyperplan B_2 , car c'est celui qui maximise la marge.

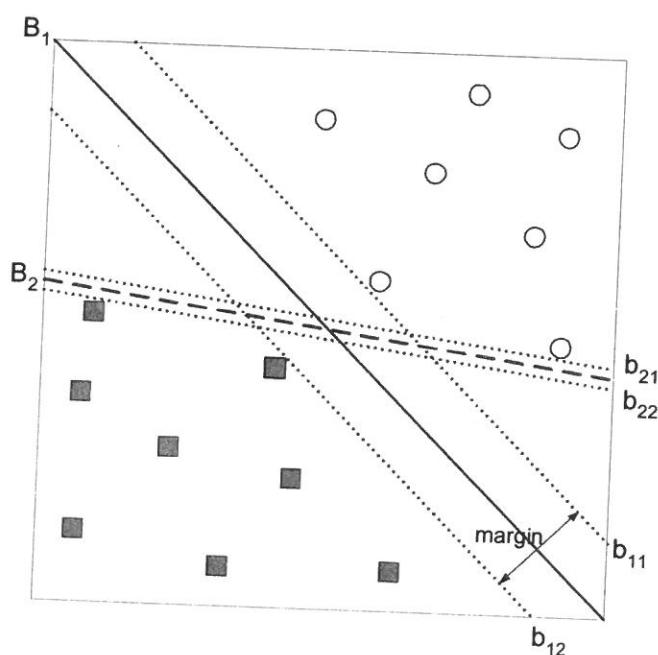


Figure 10.84 – Séparation correcte (B2) et séparation optimale (B1)

Le mécanisme des SVM justifie la traduction française « séparateur à vaste marge », la marge étant $2/\|a\|^2$.

Étant donnés les points (x_i, y_i) , avec $y_i = 1$ si x_i est dans A et $y_i = -1$ si x_i est dans B, trouver le séparateur linéaire $f(x) = a \cdot x + b$ équivaut à trouver un couple (a, b) satisfaisant simultanément les deux conditions :

- pour tout i, $y_i(a \cdot x_i + b) \geq 1$ (bonne séparation) ;
- $\|a\|^2$ est minimum (marge maximale).

En pratique, un terme doit être ajouté à chacune de ces deux expressions lorsque les deux populations à discriminer ne sont pas parfaitement séparées mais se recouvrent, mais cela ne change pas le principe du problème d'optimisation.

La solution $f(x)$ s'exprime donc en fonction de produits scalaires $x \cdot x'$. Après transformation Φ , elle s'exprime en fonction de produits scalaires $\Phi(x) \cdot \Phi(x')$. La quantité $k(x, x') = \Phi(x) \cdot \Phi(x')$ est appelée *noyau*. Dans l'algorithme, c'est le noyau k et non Φ qui est choisi, et en s'y prenant bien, on peut calculer $k(x, x')$ sans faire apparaître Φ . Tout le sel des SVM est de réussir à exprimer $f(\Phi(x))$ en fonction de x sans faire intervenir explicitement Φ . Les calculs sont alors faits dans l'espace de départ, et deviennent beaucoup plus simples et plus rapides. Pour cette raison, on parle de machine à noyau ou *kernel machine*.

Quelques exemples de noyaux :

- linéaire $k(x, x') = x \cdot x'$;
- polynomial $k(x, x') = (x \cdot x')^d$; si $d = 2$, $x = (x_1, x_2)$ et $\Phi(x) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$, alors $\Phi(x) \cdot \Phi(x') = (x_1 x_1 + x_2 x_2)^2 = (x \cdot x')^2$;
- gaussien (RBF) $k(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$; l'un des plus couramment employés ;
- sigmoïdal $k(x, x') = \tanh \{\kappa(x \cdot x') + \theta\}$, où κ est le *gain* et θ le *seuil*.

Selon le choix du noyau, le temps de calcul sera plus ou moins important, mais ce choix permet aussi de modéliser des problèmes variés avec une qualité des résultats qui fait des SVM sans doute une technique d'avenir et certainement une technique en vogue. Après les SVM pour le classement (SVC) introduits par Corinna Cortes et Vladimir Vapnik en 1995 (Support-vector network, *Machine Learning*, 20, 1–25), une variante est apparue en 1996 pour le cas où la variable à expliquer est continue, avec la Support Vector Regression (SVR) de Vapnik, Harris Drucker, Chris Burges, Linda Kaufman et Alex Smola.

10.11.2. Exemple

Les SVM sont aujourd'hui implémentés dans le logiciel commercial KSVM de l'éditeur KXEN, et, à titre expérimental, dans SAS *Entreprise Miner*. Par ailleurs, le package *kernlab* du logiciel R contient la fonction *ksvm* implémentant plusieurs algorithmes de SVM. Voici comment l'interface graphique Rattle de R permet d'utiliser la fonction *ksvm*.

On choisit les SVM parmi d'autres techniques de modélisation dans l'onglet « Model » de Rattle (Figure 10.85).

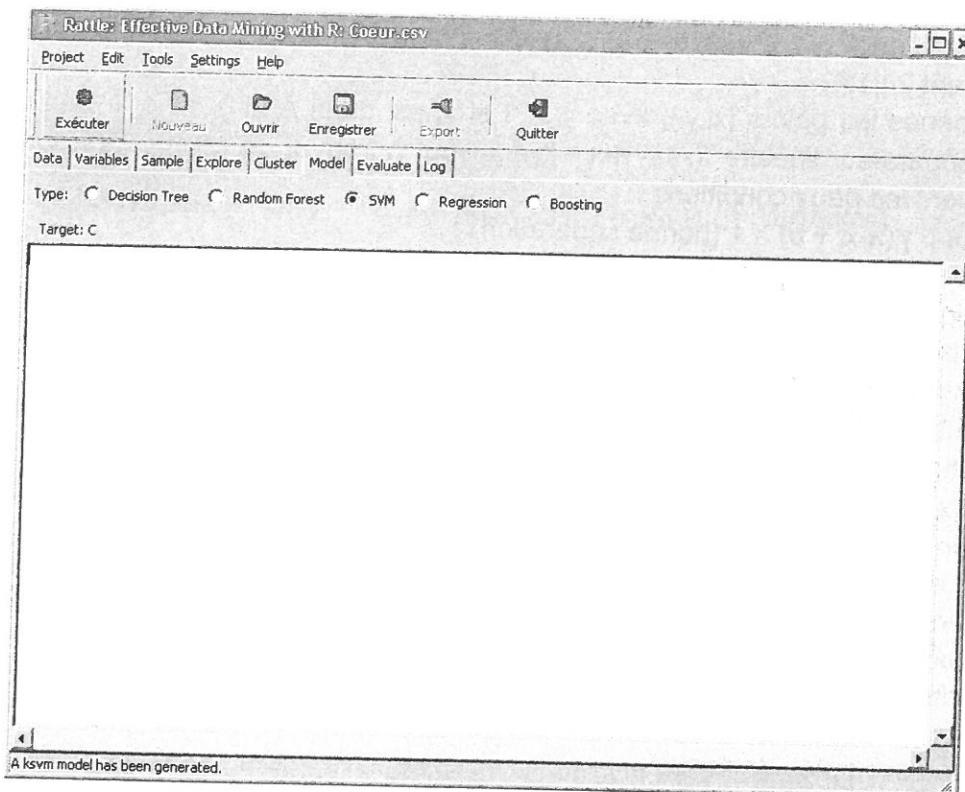


Figure 10.85 – SVM dans la fenêtre de modélisation de Rattle

```

Rattle: Effective Data Mining with R: Coeur.csv
Project Edit Tools Settings Help
Exécuter Nouveau Ouvrir Enregistrer Export Quitter
Data Variables Sample Explore Cluster Model Evaluate Log
## Generate textual output of the rpart model.
print(crs$rpart)
printcp(crs$rpart)
## List the rules from the tree using a Rattle support function.
list.rules.rpart(crs$rpart)
#####
## Timestamp: 2007-03-07 22:44:36
## Build a logistic regression model using glm.
crs$glm <- glm(C ~ ., data=crs$dataset[crs$sample,c(2,4:10)], family=binomial(logit))
## Summary of the resulting GLM model
summary(crs$glm)
#####
## Timestamp: 2007-03-07 22:49:18
## The kernlab package supplies the ksvm function.
require(kernlab, quietly=TRUE)
## Build a support vector machine model.
crs$ksvm <- ksvm(as.factor(C) ~ ., data=crs$dataset[crs$sample,c(2,4:10)], prob.model=TRUE)
## Generate textual output of the svm model.
dcs$ksvm

```

Figure 10.86 – Syntaxe SVM dans la fenêtre « log » de Rattle

Après avoir exécuté l'algorithme, on va sélectionner dans la fenêtre « Log » la syntaxe générée (Figure 10.86), que l'on colle ensuite dans la console R.

On transmet la commande et on obtient le résultat comme dans la Figure 10.87.

```

RGui
Fichier Edition Msc Packages Fenêtres Aide
R Console
> ## The kernlab package supplies the ksvm function.
>
> require(kernlab, quietly=TRUE)
[1] TRUE
Warning message:
le package 'kernlab' a été compilé avec la version R 2.4.1
>
> ## Build a support vector machine model.
>
> crs$ksvm <- ksvm(as.factor(C) ~ ., data=crs$dataset[crs$sample,c(2,4:10)], pr$Using automatic sigma estimation (sigest) for RBF or laplace kernel
>
> ## Generate textual output of the svm model.
>
> crs$ksvm
Support Vector Machine object of class "ksvm"
SV type: C-svc (classification)
parameter : cost C = 1
Gaussian Radial Basis kernel function.
Hyperparameter : Sigma = 0.162770240673064
Number of Support Vectors : 40
Objective Function Value : -22.3352
Training error : 0.084507
Probability model included.
> 
```

Rattle 2007-03-07 22:43:28 Stéphane

Decision Tree Coeur.csv \$ C

INSYS > - < 21.6

1
44 cases
88.6%

Figure 10.87 – Exécution de la fonction ksvm dans la console R

Si l'on veut évaluer la performance du modèle SVM et la comparer à celle d'autres modèles, on revient dans l'interface de RATTLE, sur l'onglet « Evaluate ». Ainsi, dans la Figure 10.88, on a coché les trois modèles à évaluer : « Decision Tree », « SVM » et « Regression », l'arbre de décision et la régression logistique ayant été préalablement construits. On a aussi coché « ROC » pour obtenir la courbe ROC de ces modèles (voir la section 10.16.5).

Les courbes ROC des trois modèles s'affichent dans la fenêtre graphique de la console R (Figure 10.89).

On constate ici que les SVM fournissent le meilleur modèle, suivis par la régression logistique, le modèle d'arbre (avec seulement deux nœuds) étant le moins performant.

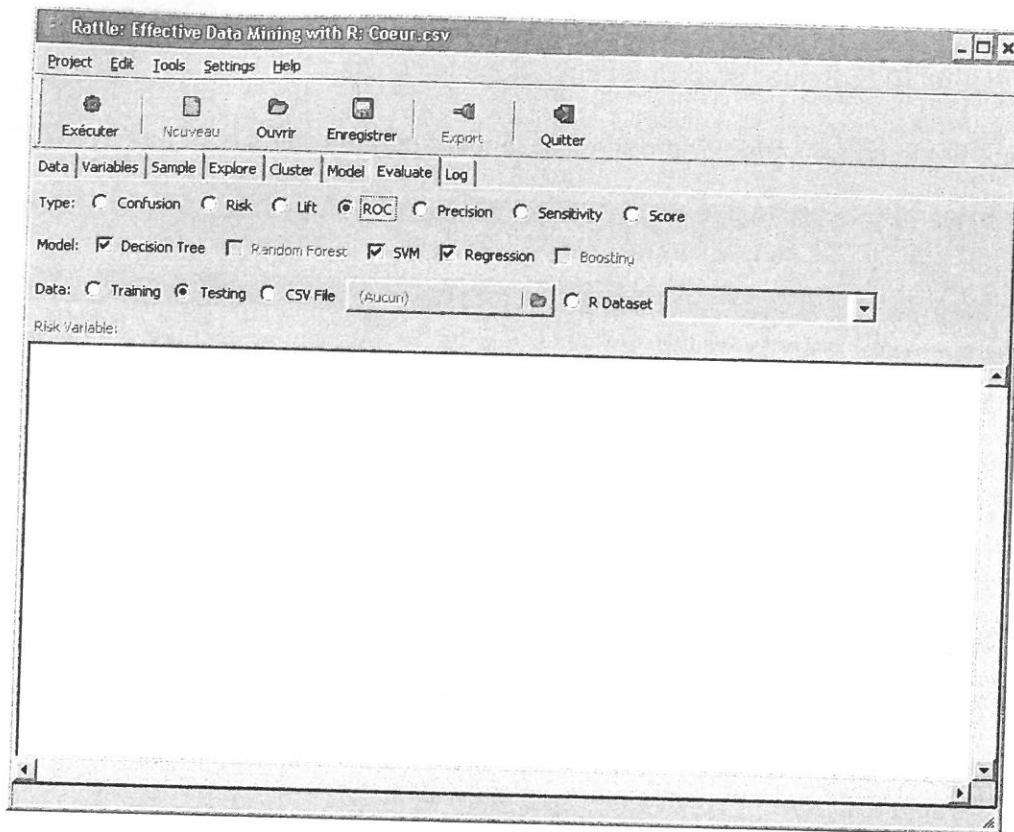


Figure 10.88 – Fenêtre d'évaluation de modèles de Rattle

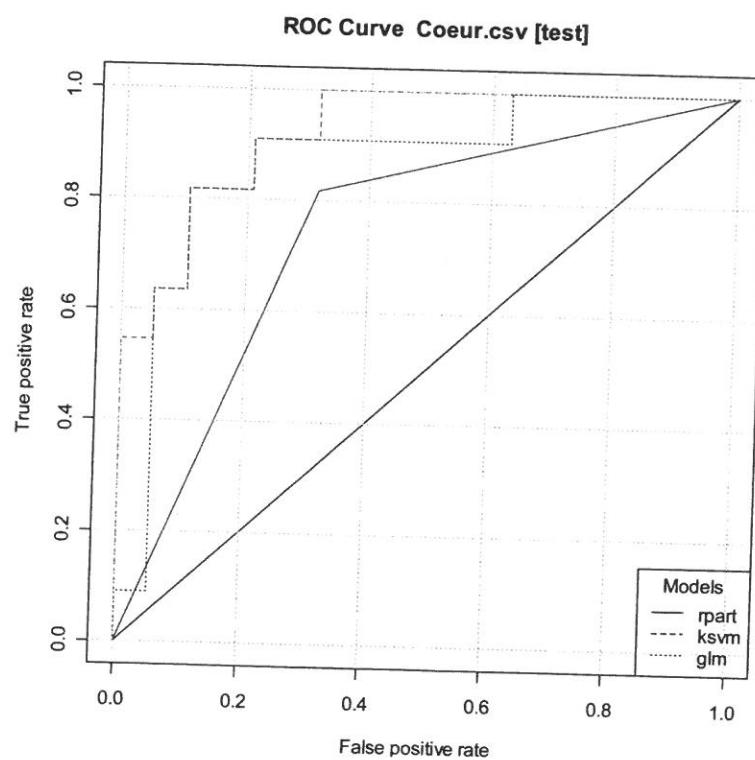


Figure 10.89 – Courbes ROC des modèles SVM, logit et CART

10.11.3. Avantages des SVM

Les avantages des SVM sont :

- leur capacité à modéliser les phénomènes non linéaires ;
- la précision dans certains cas de leurs prédictions.

10.11.4. Inconvénients des SVM

Les inconvénients des SVM sont :

- l'opacité des modèles (que l'on peut chercher à expliquer par un arbre de décision, mais on risque de perdre en précision, c'est le même problème que pour les réseaux de neurones) ;
- la sensibilité au choix des paramètres du noyau (σ pour le RBF, le degré pour le noyau polynomial, etc.) et la difficulté de bien les choisir, qui peut obliger à tester un grand nombre de valeurs possibles ;
- les temps de calcul un peu longs ;
- le risque de sur-apprentissage ;
- les logiciels implémentant les SVM sont rares et ceux qui le font ne sont pas nécessairement bon marché (outre R, on peut citer toutefois le logiciel MySVM).

10.12. La prédiction par algorithmes génétiques

Selon la théorie de l'évolution, la sélection naturelle permet aux individus les mieux adaptés à leur environnement de transmettre leur matériel génétique à leurs descendants.

Par analogie, les algorithmes génétiques, développés par l'école de John Holland dès le début des années 1970, permettent aux règles les plus appropriées à la résolution d'un problème (prédiction ou classement) d'être sélectionnées pour transmettre leur « matériel génétique » (c'est-à-dire leurs variables et leurs modalités) à des règles « filles ». Ce que l'on appelle ici « règle » est un ensemble de modalités de variables ; par exemple : client entre 36 et 50 ans, détenteur d'un patrimoine financier $< 20 \text{ k€}$ et de revenus mensuels $> 2 \text{ k€}$. Une règle est l'équivalent d'une branche d'un arbre de décision. Elle est ici l'analogue d'un *gène*.

Les algorithmes génétiques cherchent donc à reproduire les mécanismes de la sélection naturelle, en *sélectionnant* les règles les mieux adaptées à la prédiction (ou le classement), et en les *croisant* et en les *mutant* jusqu'à obtention d'un modèle suffisamment prédictif. Ils sont, avec les réseaux de neurones, le second type d'algorithmes mimant des mécanismes qui sont naturels pour expliquer des phénomènes qui ne le sont pas forcément.

Le déroulement d'un algorithme génétique comporte trois phases :

- 1) génération aléatoire des règles initiales,
- 2) sélection des meilleures règles,
- 3) génération de nouvelles règles par mutation ou croisement,
la phase 3 bouclant sur la phase 2 jusqu'à la fin du déroulement de l'algorithme.