Cours de

Reconnaissance de Formes

- Résumé -

2004-2005
v 1.4

Benoît Decoux
(benoit.decoux@wanadoo.fr, decoux@efrei.fr)

http://www-rdf.efrei.fr

Sommaire

Introduction

Les différentes parties de ce résumé correspondent à une succession de méthodes, étudiées dans un ordre de difficulté croissante. Toutes ces parties correspondent à des méthodes différentes pouvant être appliquées aux problèmes généraux de classification, sauf la dernière qui porte sur la reconnaissance de parole, qui utilise des méthodes spécifiques, adaptées à la nature temporelle de l'information traitée.

Dans la plupart des méthodes décrites dans ce document, une forme est représentée par un vecteur dont les composantes sont des grandeurs caractéristiques. Le seul cas particulier est celui de la reconnaissance de parole, dans lequel une forme est représentée par un ensemble de vecteurs, la dimension supplémentaire (celle du nombre de vecteurs de cet ensemble) correspondant au temps.

Dans tout ce document, RdF sera synonyme de classification.

Applications industrielles

Parmi les applications industrielles actuelles de la Reconnaissance des Formes (RdF), on peut citer

- la reconnaissance biométrique : empreinte digitale, iris de l'œil, visage (par exemple pour l'accès à des lieux sécurisés) ;
- la reconnaissance de parole (par exemple pour la conversion parole vers texte), de l'écriture (par exemple traitement ou archivage de documents manuscrits) ;
- la classification automatique d'objets divers.

Dans chacune de ces applications, il est assez difficile de connaître les méthodes utilisées, pour des raisons évidentes de secret industriel.

Lien avec autres domaines

La RdF est un domaine très proche de celui de l'analyse de données (utilisé pour la prévision, les études statistiques, etc) : la reconnaissance passe souvent par une analyse de données préalable, d'une subdivision de ces données en groupes d'appartenance distincts, puis d'une prise de décision d'appartenance à l'un de ces groupes.

C'est un domaine également très proche de celui des réseaux de neurones (RdN) : d'une part la RdF est la principale application des RdN, et d'autre part de nombreuses méthodes de RdF peuvent être formulées dans le cadre des RdN.

Principes généraux

La RdF fonctionne souvent par apprentissage, à partir d'exemples de données disponibles. Il existe 2 grandes catégories d'apprentissage : supervisé ou non supervisé. Dans le cas non-supervisé, les paramètres d'un classifieur sont ajustés à partir de données d'apprentissage, sans regarder si la réponse du classifieur est correcte ou non pendant cette modification. Dans le cas supervisé, au contraire, les erreurs commises par le classifieur sont exploitées pour corriger son fonctionnement par l'ajustement de ses paramètres.

Dans la plupart des algorithmes, le nombre de classes des données est initialement indiqué à l'algorithme.

I) Les k plus proches voisins (kPPV)

Principe

La forme à classer est comparée aux autres formes ayant déjà été classées, et on lui affecte la classe la plus représentée parmi les k plus proches d'elle.

Dans le cas particulier k=1, c'est la classe de la forme la plus proche de la forme à classer qui est affectée à cette dernière.

La notion de proximité utilisée est quantifiée par une mesure de similarité. La mesure de similarité la plus utilisée est la distance euclidienne.

Pour que l'algorithme puisse démarrer, il faut utiliser un certain nombre d'exemples dont la classe est connue, auxquels vont être comparés les premiers vecteurs à classer.

Paramètres

Les différents paramètres de l'algorithme, réglables par l'utilisateur, sont :
- le nombre d'exemples dont la classe est connue, utilisés au démarrage de l'algorithme
- la valeur de k
- la mesure de similarité (ex. : distance euclidienne)

Algorithme

- Choisir des exemples initiaux, dont la classe est connue
- Pour chaque vecteur à classer
  - Mesurer la similarité entre le vecteur à classer et tous les vecteurs déjà classés
  - Déterminer les k vecteurs pour lesquels la similarité est la plus grande (=kPPV)
  - Déterminer la classe la plus représentée parmi ces k vecteurs et affecter cette classe au vecteur à classer

Propriétés

- Le coût de calcul augmente avec le nombre de vecteurs déjà classés, ce qui représente un inconvénient.
- Le résultat de classification dépend de l'ordre de présentation des exemples, d'où l'importance de faire une initialisation correcte.
- Le choix de k constitue un compromis : une petite valeur permet de définir des frontières compliquées entre les classes, mais présente une grande sensibilité au bruit ; une grande valeur permet de définir des frontières lisses et de présenter une insensibilité au bruit, mais ne permet pas de traiter le cas de classes possédant un effectif réduit (car dans ce cas il faut beaucoup d'exemples initiaux).
- La règle d'affectation définie dans l'algorithme comporte de nombreux cas indécidables. Par exemple, les cas où il n'y a pas une classe plus représentée que les autres dans les k plus proches voisins (exemple : k=3 et nombre de classes =3 également). Dans de tels cas, on peut soit imposer que la valeur de k respecte quelques contraintes par rapport au nombre de classes (par exemple il ne doit pas être un multiple du nombre de classes), ou choisir une règle de décision supplémentaire, comme par exemple les distances minimales.
- Il peut être démontré, d'une manière théorique, que cette méthode est optimale quand les données sont gaussiennes (voir paragraphe sur classification bayesienne et l'annexe sur les statistiques).

II) Nuées dynamiques

Principe

La principale différence avec les kPPV est que chaque vecteur à classer n'est pas comparé à tous les exemples déjà classés, mais à un seul vecteur représentatif de chaque classe.

Les principales caractéristiques de cette méthode sont les suivantes :

- Chaque classe est représentée par un vecteur : on l'appelle *noyau*, ou *représentant*, ou encore *prototype*.
- Ce représentant est obtenu par combinaison des vecteurs affectés à sa classe par l'algorithme. Souvent, on le prend égal à la moyenne de ces vecteurs.
- Chaque vecteur à classer est comparé à tous les noyaux au moyen d'une mesure de similarité (le plus souvent, la distance euclidienne). Ce vecteur est associé à la classe dont le noyau est le plus proche.
- Le noyau est mis à jour à chaque affectation d'un nouveau vecteur à la classe qu'il représente. L'effet de cette modification est un rapprochement du noyau vers le nouveau vecteur.

  Remarque : Dans le cas où le représentant est constitué par la moyenne des vecteurs affectés à sa classe, il n'est pas nécessaire de re-calculer la moyenne de tous les vecteurs. En effet on peut montrer facilement que la moyenne de n+1 vecteurs peut s'exprimer de manière récursive, en fonction de la moyenne de n vecteurs :

$$m_{n+1} = \frac{n.m_n + x_{n+1}}{n+1}$$

  où x_{n+1} est le $(n+1)^e$ vecteur et m_{n+1} la moyenne de n+1 vecteurs. Cette relation reste valable lorsque x et m sont des vecteurs de dimension quelconque.
- Le représentant initial pour chaque classe peut être un vecteur de cette classe choisi aléatoirement, mais l'algorithme aura plus de chances de s'exécuter efficacement si on utilise plutôt un vecteur représentatif de cette classe, c'est à dire situé aux alentours du centre (par exemple égal à la moyenne de quelques points connus de cette classe).

Paramètres

- représentants initiaux pour chaque classe
- mesure de similarité (ex. : distance euclidienne)

Algorithme

- Choisir le représentant initial de chaque classe (ou le calculer à partir de vecteurs de cette classe)
- Pour chaque vecteur à classer
  - Mesurer la similarité entre le vecteur à classer et les représentants des différentes classes
  - Déterminer le représentant correspondant à la similarité maximale
  - Modifier ce représentant pour prendre en compte le nouveau vecteur affecté à sa classe

Propriétés

- Méthode peu coûteuse en calculs car chaque vecteur à classer n'est comparé qu'à un seul autre vecteur pour chaque classe : le noyau.
- Méthode ayant la propriété de minimiser un critère global : la somme des distances des points au prototype le plus proche d'eux.
- D'un point de vue géométrique, cette propriété correspond à un découpage de l'espace des entrées appelé *pavage* (ou *tessellation*) *de Voronoï*, dans lequel chaque pavé est associé à un vecteur représentant : il est constitué par l'ensemble des points les plus proches de lui que de tous les autres représentants.
- Méthode sensible à l'ordre de présentation des données à classer : si cet ordre est aléatoire, le résultat de la classification sera différent à chaque exécution de l'algorithme.
- Avec la distance euclidienne, méthode ayant tendance à créer des classes hyper-sphériques (des cercles en dimension 2). Donc fonctionne bien avec des classes hypersphériques mais moins avec les formes plus complexes. Pour pallier à cet inconvénient, on peut utiliser plusieurs représentants par classe (voir ci-dessous).

Variantes

Une variante permettant de traiter des classes de formes complexes consiste à utiliser plusieurs représentants par classe. Le principe de l'algorithme reste le même.

III) Apprentissage compétitif

a) Apprentissage compétitif non-supervisé (vector quantization : VQ)

Cette méthode est également appelée quantification vectorielle. Elle désigne le fait de remplacer un vecteur dont les composantes peuvent être quelconques, par un vecteur d'un ensemble discret. Cette opération est analogue à la quantification d'un signal échantillonné : la valeur des échantillons constitue alors une variable mono-dimensionnelle, alors que dans la RdF on raisonne sur des vecteurs multi-dimensionnels.

Principe

La méthode de quantification vectorielle est très proche des nuées dynamiques. Mais plutôt que d'utiliser la moyenne des vecteurs affectés à une classe pour représenter celle-ci, les vecteurs représentants sont adaptés par une règle d'apprentissage, à chaque classification d'un nouveau vecteur.

De plus cette méthode est souvent formulée dans le cadre des Réseaux de Neurones, qui fait que les termes employés sont différents de ceux de la méthode des nuées dynamiques, tout en désignant des choses très proches. (voir l'introduction aux Réseaux de Neurones en annexe). Les prototypes sont alors codés dans les vecteurs poids des neurones. Dans la version "classique", on recherche par exemple une distance minimale comme critère de similarité maximale, mais dans la version neuronale on recherche un produit scalaire

maximal, entre un vecteur à classer et les vecteurs poids des neurones. On peut montrer que ces deux critères sont équivalents, dans certaines conditions (voir annexe sur calcul matriciel).

L'apprentissage est de type non-supervisé, c'est à dire que l'information d'appartenance des vecteurs d'exemple à une classe n'est par utilisée pendant l'apprentissage, sauf pour ceux qui sont utilisés pour initialiser l'algorithme (comme dans le cas des nuées dynamiques).

Paramètres

- coefficient d'apprentissage
- représentants initiaux pour chaque classe
- mesure de similarité (ex. : distance euclidienne)

Algorithme

- Choisir les représentants initiaux pour chaque classe
- Initialiser les poids de chaque neurone à un exemple représentatif de chaque classe (1)
- Répéter un certain nombre de fois (2) :
  - Appliquer un vecteur en entrée du réseau
  - Sélectionner le neurone dont le vecteur poids est le plus proche du vecteur d'entrée (neurone "vainqueur") (3)
  - Modifier les poids de ce neurone par la règle d'apprentissage (4) :

$$\Delta w_{ij} = \alpha.(x_i - w_{ij}) y_j$$

(1) ou à des valeurs aléatoires, mais cela peut poser le problème des neurones "morts".

(2) En général, on présente une fois tous les vecteurs de la base d'apprentissage, puis on recommence. On peut fixer un nombre de présentations de manière arbitraire et empirique, ou définir un critère de convergence. Par exemple, utiliser la base de test après chaque présentation de la base d'apprentissage ; le critère d'arrêt est alors atteint quand le taux de reconnaissance sur la base de test passe au dessus d'un seuil.

(3) La mesure de similarité utilisée peut être basée sur le calcul d'une distance euclidienne, de Hamming, ou d'un produit de corrélation.

(4) La sortie du neurone vainqueur est fixée à 1 et celle des autres à 0 (c'est le principe de l'apprentissage compétitif). Par application de la règle d'apprentissage, seuls les poids du neurone vainqueur sont modifiés. C'est équivalent à appliquer la règle

$$\Delta w_{ij} = \alpha.(x_i - w_{ij})$$

uniquement au neurone ayant répondu le plus fortement.

L'effet de cette règle d'apprentissage est de rapprocher le vecteur poids du neurone vainqueur, du vecteur à classer comme dans les nuées dynamiques.

Variante

Une variante consiste à utiliser plusieurs vecteurs représentants par classe. Elle permet de traiter le cas des classes dont la forme est plus complexe que simplement hypersphérique.

Comme dans les nuées dynamiques, la structure générale de l'algorithme reste la même.

D'un point de vue neuronal, dans le cas où il y a plusieurs représentants par classe, cela correspond à avoir 2 couches de neurones, dont la 2^e (neurones de codage des classes) implémentent des fonctions OU.

b) Apprentissage compétitif supervisé (learning vector quantization : LVQ)

LVQ (pour Learning Vector Quantization) est une version supervisée de l'algorithme de quantification vectorielle. Comme dans toutes les méthodes supervisées, pendant l'apprentissage, on utilise l'information d'appartenance des exemples d'apprentissage à une classe.

Il existe plusieurs versions de cet algorithme, appelées : LVQ1, LVQ2, LVQ2.1, LVQ3, OLVQ1. Le package LVQ_PAK (disponible sur Internet) regroupe les sources de ces programmes pour toutes les versions, sauf LVQ2.

Principe

Pour un vecteur d'entrée à classer, si le vecteur poids du neurone vainqueur est un prototype de la classe correcte, il est modifié de la même façon qu'avec la méthode VQ (version non-supervisée de l'algorithme) ; il est donc rapproché du vecteur d'entrée. A la différence de VQ, si le vecteur poids du neurone vainqueur n'est pas un prototype de la classe correcte, il est éloigné du vecteur d'entrée.

On peut avoir plusieurs représentants par classe. En effet, comme dans le cas des Nuées Dynamiques, avec un seul représentant par classe la méthode de bons résultats pour les classes hypersphériques et moins bons pour les autres formes de classes. Le fait de prendre plusieurs représentants par classe permet de mieux traiter le cas de classes de formes complexes.

Avec plusieurs représentants par classe, le réseau se compose de 2 couches : la première est constituée de neurones codant chacun un représentant d'une classe, la deuxième de cellules réalisant des fonctions OU sur les sorties de neurones de la première. Les neurones de la dernière couche codent chacun une classe.

Différentes versions

LVQ1

La modification des poids du neurone vainqueur est définie par :

$$w_k(t+1) = w_k(t) + s(t).\alpha\left(x(t) - w_k(t)\right)$$

où *c* est l'indice de classe, α un coefficient d'apprentissage (fixe), et

s(t)=+1 si la classification est correcte (le neurone vainqueur représente la bonne classe)

-1 si la classification est incorrecte (le neurone vainqueur ne représente pas la bonne classe)

LVQ2

Dans cette version on ne modifie pas uniquement le prototype le plus proche du vecteur à classer mais les deux prototypes les plus proches, quand les conditions suivantes sont réunies :

- le vecteur à classer est proche de la frontière (hyperplan) qui sépare les 2 prototypes, c'est à dire qu'il est à peu près à égale distance du prototype correct et du prototype incorrect ;
- le prototype le plus proche correspond à une classe incorrecte et le second prototype le plus proche à la classe correcte.

La première condition peut être implémentée de la manière suivante : soient d_i et d_j les distances euclidienne entre une entrée *x* et respectivement w_i, le représentant de la classe

incorrecte, et w_j, celui de la classe correcte. *x* est défini comme tombant dans une fenêtre de largeur L si

$$\min\left(\frac{d_i}{d_j},\frac{d_j}{d_i}\right) > s \text{ avec } s=\frac{1-L}{1+L}$$

L'algorithme éloigne alors w_i (le prototype incorrect) du vecteur à classer x et en rapproche w_j (le prototype correct), comme dans la version de base.

Les valeurs recommandées pour L sont 0,2 à 0,3.

L'intérêt de cette variante est de déplacer les vecteurs prototypes uniquement dans les cas où il y a mauvaise classification, et où le vecteur à classer se trouve proche de la frontière entre 2 classes, zone d'ambiguïté.

LVQ2.1

Dans cette version on modifie les deux prototypes les plus proches comme dans LVQ2, mais l'un des deux peut indifféremment représenter la classe correcte et l'autre une mauvaise classe. La modification n'est appliquée que si le vecteur à classer est proche de la frontière qui sépare les deux prototypes.

LVQ3

Dans le cas où les 2 prototypes les plus proches du vecteur à classer représentent une classe différente, ils sont modifiés de la même manière que dans LVQ2.1.

Mais dans le cas où ces 2 prototypes appartiennent à la même classe que le vecteur à classer, on ajoute un facteur constant dans la règle d'apprentissage :

$$w_k(t+1) = w_k(t) + \varepsilon.\alpha\left(x(t) - w_k(t)\right),\ k \in \{i,j\}$$

où i et j sont les indices de ces 2 prototypes.

L'effet de cette règle supplémentaire est d'éviter les dérives des prototypes à long terme (c'est à dire dans le cas où les mêmes entrées sont présentées indéfiniment, cycliquement), possibles avec LVQ2.1.

Des valeurs empiriques pour ε vont de à 0,1 à 0,5.

OLVQ1

Il s'agit d'une variante de l'algorithme de base LVQ1, dans laquelle les valeurs optimales des taux d'apprentissage sont déterminées. Chaque classe possède son propre taux d'apprentissage.

Le taux d'apprentissage est défini par :

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1+s(t).\alpha_c(t-1)}$$

où *c* est l'indice de classe,
et s(t)=+1 si la classification est correcte ;
-1 si la classification est incorrecte.

Il diminue quand le nombre de bonnes classifications augmente (s(t)=1), et augmente quand le nombre de mauvaises classification augmente (s(t)=-1). Ainsi, quand la classification s'améliore, les prototypes ont tendance à se stabiliser.

Empiriquement, on observe que le taux de reconnaissance maximal est atteint au bout d'un nombre de présentations de toute la base d'apprentissage égal à 30 à 50 fois le nombre de vecteurs prototypes.

On ajoute simplement l'indice de la classe dans la règle d'apprentissage :

$$w_k(t+1) = w_k(t) + s(t).\alpha_c(t)\left(x(t) - w_k(t)\right)$$

V) Fonctions radiales de base

a) RCE (Reduced Coulomb Energy)

Le réseau RCE est un réseau à construction dynamique. Il s'agit d'une méthode supervisée, c'est à dire dans laquelle l'appartenance aux classes est connue à l'avance.

Principe

L'idée de cette méthode est de représenter les classes par une réunion d'hypersphères (des cercles en dimension 2). Chaque hypersphère est codée par une cellule (qu'on appellera cellule de codage).

Une classe est codée par une cellule recevant les sorties d'une ou plusieurs cellules de codage.

Initialement le réseau ne comporte aucun neurone de classe. Un nouveau neurone est créé dès que le besoin s'en fait sentir, c'est à dire quand les neurones déjà créés ne peuvent coder correctement une nouvelle entrée. Un neurone possède une zone d'influence dans l'espace des entrées, de forme circulaire et définie par son rayon, initialement centrée sur l'entrée ayant provoqué la création de ce neurone. Cette zone circulaire est définie par la règle d'activation des neurones :

pour le neurone j : $y_j=1$ si $dist(x, w_j) \leq \lambda_j$
0 sinon

où :

- λ_j est le seuil d'entrée du neurone j (également le rayon de son cercle d'influence),
- x le vecteur d'entrée actuel,
- w_j le vecteur poids du neurone.

Propriétés

Inconvénients

- le nombre de prototypes (et donc de cellules) dépend de l'ordre de présentation des entrées
- l'algorithme est très consommateur de mémoire
- tendance à l'apprentissage par coeur (même du bruit) et donc mauvaise généralisation
- la méthode est non optimale dans le cas des classes qui se chevauchent, contrairement aux méthodes qui définissent des frontières entre les classes

Avantages

- apprentissage rapide, permettant donc des applications temps-réel
- simplicité : un seul paramètre initial λ_1 + un échantillon d'apprentissage
- faible connectivité : linéaire en fonction du nombre de neurones
- possibilité d'apprentissage de nouvelles données sans avoir à ré-apprendre toute la base

Algorithme

Initialisation : rayon initial des hypersphères λ_0
Pour chaque vecteur à classer
- Si ce vecteur n'appartient pas à la région d'influence d'une cellule de codage, **création** d'une nouvelle cellule de codage
  - S'il existe déjà un neurone de classe pour la classe de ce vecteur, **connexion** de cette nouvelle cellule de codage à cette cellule de classe
  - Sinon, **création** d'une nouvelle cellule de classe pour la classe de ce vecteur
- Si ce vecteur appartient à la région d'influence d'une ou plusieurs cellule(s) de codage

- Pour chacune de ces cellules de codage
  - Si cette cellule de codage représente la dasse du vecteur présenté, la classification est correcte et rien ne se passe
  - Sinon, cela signifie que la région d'influence de cette cellule de codage est trop grande :
  - **Diminution** de cette région : λ_k=dist(x, w_k)-e (x vecteur à classer, w_k vecteur poids de cette cellule)
  - Si aucune de ces cellules de codage ne représente la classe correcte
  - **Création** d'une nouvelle cellule de codage, dont le rayon de la région d'influence doit être légèrement inférieur à la distance entre le vecteur à classer et tous les autres vecteurs représentants (*). Dans le cas général de n cellules de codage, cette région d'influence doit donc satisfaire la condition :

$$\lambda_n < \min_k[\text{dist}(x_n, w_k)],\ k=1,\ldots,n-1$$

  - s'il existe déjà un neurone de classe pour la classe de ce vecteur, **connexion** de cette nouvelle cellule de codage à cette cellule de classe
  - (suite de) Si aucune de ces cellules de codage ne représente la classe correcte
- sinon, **création** d'une nouvelle cellule de classe pour la classe de ce vecteur

(*) pour que cette cellule ne s'active pas lors de la présentation de ces vecteurs représentants

La base d'apprentissage est présentée plusieurs fois, jusqu'à ce qu'il n'y ait plus de modifications dans le réseau.

Comparaison avec LVQ

Comme dans version de la méthode LVQ avec plusieurs représentants par classe, on a 2 couches neurones : une couche intermédiaire de neurones, dans laquelle un neurone code une partie de l'espace d'entrée, et une couche de sortie dont les neurones reçoivent les sorties d'un ou plusieurs neurones de la couche intermédiaire, et codent les différentes classes.

Les différences avec LVQ sont :

- dans LVQ les neurones calculent les distances entre leur vecteur poids et le vecteur d'entrée, et seul le neurone possédant la réponse minimale voit sa sortie passer à 1 (compétition) ; dans RCE la même distance est calculée, mais elle est comparée à un seuil.
- dans RCE l'apprentissage ne consiste pas à modifier les poids des vecteurs représentant mais à modifier le seuil de comparaison de chaque neurone de codage ;
- dans LVQ la structure est définie une fois pour toutes ; dans RCE elle évolue au cours de l'apprentissage.

b) Réseaux à neurones gaussiens

Par comparaison avec la méthode RCE, la sortie des neurones de la première couche (la couche de codage) n'est pas du type tout-ou-rien. Comme dans RCE, une distance est calculée entre un vecteur d'entrée à classer et un vecteur représentant, mais plutôt que d'appliquer cette différence à une fonction seuil, elle est appliquée à une fonction pouvant avoir une autre forme : gaussienne, triangulaire, carrée... En général c'est la fonction gaussienne qui est utilisée. Contrairement à RCE la sortie des neurones de la couche de sortie n'est pas le résultat d'une fonction OU, mais une somme pondérée des sorties des neurones de la couche de codage. RCE peut donc être vu comme un cas particulier de FRB.

La sortie *y* des neurones de sortie, dans le cas gaussien, est définie par :

$$y_j = \sum_{k=1}^{P} w_{kj} G_k(x)$$

où *P* est le nombre de neurones dans la couche intermédiaire, w_{kj} est le poids reliant le k^e neurone de la couche intermédiaire au j^e neurone de sortie, *x* est le vecteur d'entrée actuel du réseau, $G_k(x)$ est la fonction de transfert, gaussienne dans notre cas, avec ses paramètres spécifiques. L'indice de l'exemple est omis pour simplifier l'écriture.

La gaussienne est définie par :

$$G_k(x) = \exp\left[-\frac{1}{2}(x - x_k^{(c)})^T C_k^{-1}(x - x_k^{(c)})\right]$$

où C_k^{-1} est la matrice de covariance des vecteurs d'entrée, $x_k^{(c)}$ le centre de la k^e gaussienne.

Pour les fonctions radiales de base utilisées on utilise le terme "noyaux gaussiens".

Normalement, l'expression d'une gaussienne comporte un coefficient constant multiplicatif de l'exponentielle, mais dans les FRB ce coefficient n'est pas nécessaire dans la mesure où les différentes gaussiennes sont pondérées par les poids des neurones de sortie, qui sont ajustables.

Il est rappelé que dans le cas à 1 dimension, l'expression d'une gaussienne est :

$$G_k(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x - x_k^{(c)})}{2\sigma^2}\right]$$

Exemple d'implémentation

Dans cet exemple, les 3 ensembles de paramètres sont déterminés de la façon suivante :

- Les **centres des gaussiennes** $x_k^{(c)}$: on choisit d'utiliser la méthode des kppv pour sélectionner des vecteurs représentatifs des données, pour ces centres ;
- La **matrice de covariance des gaussiennes C_k** : on choisit d'utiliser une matrice diagonale avec la même variance sur tous les éléments de la diagonale (ce qui correspond à des gaussiennes isotropes, c'est à dire avec le même étalement dans toutes les dimensions)
- Les **poids de sortie** : on choisit de les adapter par apprentissage supervisé à l'aide de la règle suivante :

$$\Delta w_{ij} = \alpha . y_i . (y_j^d - y_j)$$

où :

w_{ij} est le poids de la connexion entre les neurones i et j ;

y_i est la sortie du neurone gaussien i ;

y_j est la sortie du neurone de sortie j ;

y_j^d est la sortie désirée du neurone j.

α est un coefficient d'apprentissage, en général inférieur à 1 : il règle la quantité d'adaptation des poids à chaque présentation d'un exemple ;

Algorithme d'apprentissage des poids de sortie d'un réseau à FRB

- Chargement des données d'apprentissage
- *Chargement des données de test*
- Recherche d'un nombre fixé 'nb_noyaux' de vecteurs représentatifs (ou par exemple chargement d'un fichier .cod généré par le programme 'eveninit' du LVQ_PAQ)
- Initialisation d'un tableau des sorties désirées du réseau (de taille 'nb_noyaux'×'nb_classes'), pour le calcul de l'erreur de sortie
- Initialisation d'un tableau (de taille 'nb_noyaux'×'nb_classes') des poids de sortie à des valeurs aléatoires (par exemple avec la fonction 'rand')
- Classification, par présentation des entrées et calcul des sorties du réseau : **pour chaque** épisode d'apprentissage (présentation de toute la base d'apprentissage)
    - **Pour chaque** vecteur d'apprentissage
        - Calcul des sorties de la couche des neurones gaussiens
        - **Pour chaque** neurone de sortie
            - Calcul de la sortie
            - Calcul de l'erreur de cette sortie
            - Modification des poids de ce neurone en vue de la correction des erreurs
    - Nouveau calcul de l'erreur avec les données de test : **pour chaque** vecteur de test
        - Calcul des sorties de la couche des gaussiennes
        - **Pour chaque** neurone de sortie
            - Calcul de la sortie
            - Calcul de l'erreur de cette sortie
        - Compétition en sortie (pour mise en évidence de la réponse maximale) : remplacement par 1 de la réponse maximale ; les autres par 0
    - Calcul du pourcentage d'erreur pour chaque classe

c) Fenêtres de Parzen

Historiquement, les fenêtres de Parzen sont plus anciennes que les fonctions radiales de base. Elles datent des années 60, et peuvent être formulées dans le cadre des réseaux de neurones.

Le réseau est composé de 2 couches : la première est constituée des neurones-noyaux, un pour chaque exemple d'apprentissage. La seconde est constituée de neurones de classification, connectés à tous les neurones-noyaux, et réalisant une simple somme de leurs entrées.

La sortie du neurone j de la couche de classification est définie par une somme de fonctions noyaux, pondérée par un coefficient fixe :

$$y_j = \frac{1}{N}\sum_{i=1}^{N} K_i(x)$$

N est le nombre d'exemples dans la base d'apprentissage ; ce qui signifie que l'on utilise un noyau pour chaque exemple d'apprentissage. Le coefficient 1/N a un rôle de normalisation

La fonction noyau k(x) peut avoir une forme quelconque : carrée, triangulaire, gaussienne. La seule condition à respecter est que la surface qu'elle définit avec l'axe des abscisses x soit égale à 1.

Si les noyaux sont de forme gaussienne, dans le cas de la dimension 1 (et centré sur 0) ils sont définis par :

$$G(x) = \frac{1}{\sqrt{2\pi}.\sigma.h} \exp\left(-\frac{x^2}{2\sigma^2.h^2} \right)$$

Le rôle du paramètre h est d'assurer que l'intégrale de la fonction soit toujours égale à 1. C'est l'unique paramètre du modèle.
Plus généralement, en dimension *d* :

$$K(x) = G(x) = \frac{1}{(2\pi)^{d/2} h^d \sqrt{|C|}} \exp\left[-\frac{(x - x_i)^T C^{-1} (x - x_i)}{2h^2} \right]$$

Le paramètre h est déterminé empiriquement.

Comparaison avec Fonctions Radiales de Base

Comme dans le cas des FRB, on a une première couche de neurones à fonction d'activation gaussienne, et une 2e couche de neurones linéaires.

Mais plutôt que d'avoir des gaussiennes de différentes tailles comme dans les FRB, on a un seul paramètre d'étendue des gaussiennes, commun à tous les neurones d'entrée. Ce paramètre est réglé une fois pour toutes.

De plus, au lieu d'avoir des poids sur les liens d'entrée des neurones de sortie, ces poids n'existent pas (ou on peut considérer qu'ils soient tous égaux à 1).

VI) Réseaux de neurones multicouches avec rétropropagation du gradient

La rétropropagation du gradient est un algorithme d'apprentissage applicable aux réseaux de neurones multicouches non-linéaires. Ce type de réseau est historiquement appelé perceptron.

Cette méthode est basée sur les propriétés de classification des neurones non-linéaires, sans les limitations des réseaux mono-couche.

a) Propriétés d'un réseau monocouche de neurones à seuil

Un neurone à seuil à 2 entrées peut séparer un plan en 2 parties par une droite, et donc réaliser une classification en 2 classes. En effet, la sortie de ce neurone est définie par :

$$y = s(w_1 x_1 + w_2 x_2)$$

où *s* est la fonction seuil (voir annexe sur les réseaux de neurones).

Les vecteurs d'entrée de ce neurone sont de dimension 2 et donc représentent des points dans un plan.

On peut distinguer 2 cas :

$$w_1x_1+w_2x_2>0 \ (1) \rightarrow y=1$$
$$w_1x_1+w_2x_2<0 \ (2) \rightarrow y=0$$

$$(1) \Leftrightarrow x_2 > -\frac{w_1}{w_2} x_1 \ (2)$$

$x_2 = -\frac{w_1}{w_2} x_1$ est l'équation d'une droite passant par l'origine et de pente $-w_1/w_2$.

(2) $\Leftrightarrow$ le point est au dessus de la droite

En rajoutant un 3e lien d'entrée à ce neurone (entrée x_0 et poids w_0), relié à une entrée constante, on peut avoir une droite quelconque :

$$y = s(w_0 x_0 + w_1 x_1 + w_2 x_2)$$

L'équation de la droite devient alors :

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} x_0$$

En prenant par exemple x_0=1 (choix arbitraire), l'ordonnée à l'origine de la droite est $-w_0/w_2$. La pente reste la même.

Un perceptron mono-couche permet donc de réaliser une classification en 2 classes dans le cas d'un problème linéairement séparable, c'est à dire pour lequel tous les exemples d'une classe peuvent être séparés de tous les exemples de l'autre classe par une droite.

b) Apprentissage du perceptron

Le but de l'apprentissage est d'obtenir automatiquement les droites de séparation, à partir d'exemples d'entrées/sorties. Les coefficients des droites de séparation sont codés dans les poids du réseau.

L'apprentissage est de type supervisé car il utilise l'information de classe correcte des exemples.

La règle dite d'apprentissage supervisé est définie par :

$$\Delta w_{ij} = \text{taux d'apprentissage} \times \text{erreur de sortie} \times \text{sortie}$$

soit

$$\Delta w_{ij} = \alpha.(y_j^{(d)} - y_j).x_i$$

où

$y_j^{(d)}$ est la valeur désirée (c'est à dire la valeur exacte) de la sortie du neurone j.

y_j est la valeur fournie par le neurone.

c) Apprentissage (supervisé) par descente de gradient

Il s'agit d'une autre méthode d'apprentissage, basée sur la diminution du gradient d'une erreur par rapport aux poids du réseau.

La règle d'apprentissage est définie par :

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

- α est le taux d'apprentissage, positif et compris entre 0 et 1 (la plupart du temps très inférieur à 1).
- E est l'erreur de sortie : dans le cas d'un apprentissage instantané (voir paragraphe suivant), elle est égale à la somme des erreurs individuelles des neurones de sortie :

$$E = \frac{1}{2} \sum_{i=1}^{m} (y_i^{(d)} - y_i)^2$$

- $y_j^{(d)}$ est la sortie désirée du neurone j et y_j sa sortie effective. L'apprentissage est du type supervisé.

- *m* est le nombre de neurones de la couche de sortie.

Le coefficient 1/2 n'est pas obligatoire.

Le calcul de la dérivée de $\partial E/\partial w_{ij}$ permet de déterminer l'expression de la règle d'apprentissage :

$$\Delta w_{ij} = \alpha.(y_j^{(d)} - y_j).x_i$$

Il est intéressant de constater qu'elle est identique à la règle d'apprentissage du perceptron énoncée dans le paragraphe précédent. Ce qui signifie que réduire l'erreur locale à chaque neurone est équivalent à réduire la somme des erreurs des neurones de sortie.

Pour un apprentissage en temps différé, la règle d'apprentissage n'est appliquée qu'une fois que tous les vecteurs d'apprentissage ont été présentés au réseau (voir paragraphe suivant). L'erreur est calculée sur tous les exemples de la base d'apprentissage :

$$E = \frac{1}{2}\sum_{f=1}^{F}\sum_{i=1}^{m}(y_i^{(d,f)} - y_i^f)$$

où F est le nombre de formes présentes dans cette base. On peut démontrer par calcul que dans ce cas la règle d'apprentissage devient :

$$\Delta w_{ij} = 2\alpha\sum_{f=1}^{F}\left(y_j^{(f,d)} - y_j^{(f)}\right)x_i^{(f)} = 2\alpha\sum_{f=1}^{F}E_f\,x_i^{(f)}$$

Les réseaux monocouches ne peuvent traiter que les problèmes linéairement séparables, ce qui n'est pas le cas de la plupart des problèmes réels. Par exemple, le problème pourtant simple du OU-Exclusif n'est pas linéairement séparable. L'extension de l'apprentissage par descente de gradient aux réseaux multi-couches a donc été développée pour traiter les problèmes dans lesquels les classes peuvent avoir des formes quelconques.

d) 2 modes d'apprentissage

Selon que l'on modifie les poids des neurones après chaque présentation d'une entrée ou après la présentation de tous les exemples de la base d'apprentissage (en fonction de l'erreur cumulée), on est en mode d'apprentissage instantané ou différé.

L'algorithme est un peu différent dans les deux cas.

Mode instantané (temps-réel, en ligne, incrémental)

```
Initialisation des poids
  Pour chaque cycle d'apprentissage (jusqu'à convergence)
  Début
    Pour chaque exemple de la base d'apprentissage
    Début
        Présentation de l'exemple au réseau
        Activation du réseau (=propagation d'activité directe=calcul de sa sortie)
        Calcul de l'erreur de sortie
        Modification des poids en fonction de l'erreur
    Fin
  Fin
```

Mode différé (hors-ligne, batch mode)

```
Initialisation des poids
  Pour chaque cycle d'apprentissage (jusqu'à convergence)
  Début
    Pour chaque exemple de la base d'apprentissage
    Début
        Présentation de l'exemple au réseau
        Activation du réseau
        Calcul de l'erreur de sortie et cumul
    Fin
    Modification des poids en fonction de l'erreur cumulée
Fin
```

e) Algorithme de rétropropagation du gradient

Enoncé de l'algorithme

Le nom de cette méthode désigne la façon dont le gradient de l'erreur de sortie est utilisé pour adapter les poids synaptiques du réseau, de la couche de sortie vers la ou les couche(s) en amont, dans le but de réduire ses erreurs.

Comme dans le cas mono-couche (décrit plus haut), la règle d'apprentissage par descente de gradient est définie par :

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

Dans le cas des réseaux multicouches, elle peut donner naissance à la règle d'apprentissage dite "delta généralisée", définie par :

$$\Delta w_{ij} = \alpha.\delta_j^{(c)}.y_i^{(c-1)}$$

avec

$$\delta_j^{(c)} = e_j.f'\left(a_j^{(c)}\right) \text{ pour le neurone j de la couche de sortie}$$

$$\delta_j^{(c)} = f^{(c)'}(a_j^{(c)}).\sum_k \delta_k^{(c+1)}.w_{jk}^{(c+1)} \text{ pour le neurone j de la couche cachée}$$

et

- $a_j = \sum_i w_{ij} y_i$.
- $\delta_j^{(c)}$ est appelé gradient local au neurone j (caché ou de sortie).
- C est l'indice de la couche
- f(x) est une fonction linéaire pour au moins une des couches, mais ne peut pas être la fonction seuil

Caractéristiques

Cet algorithme permet de résoudre des problèmes de classification dans lesquels les classes ne sont pas linéairement séparables, donc les cas de données réelles.

Il faut au moins une couche non-linéaire dans le réseau car un réseau multicouche composé de couches linéaires est équivalent à un réseau monocouche linéaire, donc ne peut résoudre des problèmes non-linéairement séparables.

L'algorithme de rétropropagation est applicable à tout réseau multi-couches dont les fonctions d'activation sont dérivables (la plus utilisée est la fonction sigmoïde), car la dérivée de cette fonction intervient dans la règle de modification des poids.

Si le nombre de neurones dans les couches cachées est trop grand, le réseau a tendance à apprendre par coeur les données d'apprentissage (et donc à mal généraliser).

Le problème est le même quand on effectue un nombre trop important d'itérations d'apprentissage.

VII) Classification bayesienne

Dans la classification bayesienne, les classes et les données à classer sont considérées comme des variables aléatoires (VA).

Les classes sont en nombre fini. Ce sont donc des VA discrètes et elles sont décrites par des probabilités (voir annexe).

Les données sont des VA continues, donc elles sont décrites par des fonctions de densité de probabilité (FDP).

Théorème de Bayes

Le théorème de Bayes donne la probabilité qu'une entrée *x* (un vecteur de données) appartienne à une classe donnée y_i. Cette probabilité est définie par :

$$P(y_i \mid x) = \frac{p(x \mid y_i)P(y_i)}{p(x)}$$

avec $p(x) = \sum_{i=1}^{N} p(x \mid y_i)P(y_i)$, où *N* est le nombre de classes.
P() désigne une probabilité et p() une FDP.

- $P(y_i|x)$ est appelée probabilité a posteriori de la classe y_i. Il s'agit d'une probabilité conditionnelle, car on connaît la donnée à classer, et ce qu'on cherche à déterminer est sa classe.
- $P(y_i)$ est appelée probabilité a priori de la classe y_i. Elle représente la probabilité qu'une donnée (indépendamment de sa valeur) lui appartienne.
- $p(x|y_i)$ est la FDP de la classe y_i. Elle représente la façon dont sont réparties les données dans l'espace (de dimension égale à celle des données).
- p(x) est la somme des FDP pour toutes les classes, pondérées par les probabilités a priori de ces dernières.

Estimation des paramètres

En général, ce dont on dispose pour mettre au point un classifieur, c'est un certain nombre de vecteurs d'exemples pour chacune des classes. Dans le cas d'une classification bayesienne, on cherche à modéliser chaque classe par une fonction, en générale une gaussienne (de dimension égale à celle des vecteurs de donnée).

$$G(x) = \frac{1}{(2\pi)^{n/2}\sqrt{|C|}} \exp\left[-\frac{(x-x_c)^T C^{-1}(x-x_c)}{2}\right]$$

Modéliser une classe par une fonction consiste à déterminer la FDP de chacune des classes $p(x|y_i)$:

$$p(x|y_i) = G(x)$$

Les paramètres de chaque gaussienne sont estimés à partir des données disponibles.

- Le *centre* de chaque gaussienne x_c est estimé à partir des données disponibles, comme étant la moyenne de ces données :

$$x_c = \frac{1}{m}\sum_{i=1}^{m} x_i$$

où m est le nombre d'exemples de la classe,
x_i est le i^e exemple de x.

- La matrice de covariance est estimée à partir des mêmes données (voir l'annexe sur les statistiques pour plus de détails sur la définition de la covariance). On peut l'obtenir en calculant :

$$C = \frac{1}{n}.A.A^T$$

où *A* est une matrice composée de tous les vecteurs de données, auxquels on a soustrait leur moyenne, disposés en colonnes. n est le nombre de ces vecteurs, c'est à dire le nombre de colonnes de A.

Décision de classification pour une ressemblance maximale

Le type de décision le plus simple n'utilise que l'information de densité de probabilité des exemples, et donc ne nécessite pas l'application du théorème de Bayes. Cette méthode consiste à décider que la classe d'un exemple *x* donné est celle donnant la densité de probabilité maximale. Cette règle est appelé "pour une ressemblance maximale" parce que la densité de probabilité des exemples représente une ressemblance entre les exemples d'une classe et la classe elle-même. Soit $d(x)$ la décision de classification pour le vecteur *x*, on a :

$$d(x) = y_i \text{ si } p(x \mid y_i) > p(x \mid y_j)$$

avec i,j=1, 2, ..., N et j≠i, où N est le nombre de classes.

Décision de classification pour une erreur de classification minimale

La règle de décision pour une erreur de classification minimale utilise en plus de l'information de densité de probabilité, l'information de probabilité a priori de chaque classe. En fait elle découle du théorème de Bayes : la classe pour laquelle la fonction d'appartenance de *x* (fonction résultant du théorème de Bayes) est maximale, est choisie comme étant la classe de *x* :

$$d(x) = y_i \text{ si } P(y_i \mid x) > P(y_j \mid x)$$

$$\Leftrightarrow \qquad d(x) = y_i \text{ si } \frac{p(x \mid y_i)P(y_i)}{\sum_{i=1}^{N} p(x \mid y_i)P(y_i)} > \frac{p(x \mid y_j)P(y_j)}{\sum_{i=1}^{N} p(x \mid y_i)P(y_i)}$$

$$\Leftrightarrow \qquad d(x) = y_i \text{ si } p(x \mid y_i)P(y_i) > p(x \mid y_j)P(y_j)$$

avec i, j=1, 2, ..., N et j≠i

On peut remarquer que quand on dispose d'autant d'exemples pour chaque classe, on a :

$$P(y_i) = P(y_j) \quad \text{pour } i,j=1,...,n \text{ et } i \neq j$$

et donc que les deux règles de décision sont équivalentes.

VIII) Reconnaissance de parole

a) Introduction

La reconnaissance de parole est un problème spécifique, pour des raisons qui ont trait à sa nature temporelle. Dans le cas de la reconnaissance de mots isolés (cas le plus simple), un même mot peut être prononcé plus ou moins rapidement : s'il est traité par tranches temporelles (trames), il sera codé par un nombre plus ou moins important de vecteurs de caractéristiques.

Si les mots de références (les classes) étaient codés par le même nombre de vecteurs de caractéristiques que les mots à classer, la classification pourrait se faire par une simple mesure de distance entre les 2 ensembles de vecteurs, la distance totale étant la somme des distances entre les vecteurs des 2 ensembles pris deux à deux. Mais ça n'est pas le cas et une distance ne peut pas être calculée directement. La méthode de *comparaison dynamique* (ou DTW pour Dynamic Time Warping) permet justement de comparer deux ensembles comprenant un nombre différent de vecteurs. DTW est l'application à la parole d'une méthode plus générale, appelée *programmation dynamique*. La distance calculée par cette méthode peut alors être utilisée pour la classification (comme dans les kPPV) avec k=1.

Les principale méthodes de reconnaissance de parole existantes sont :

- DTW
- Modèles de Markov cachés (MMC)
- Modèles de Markov cachés associés à un réseau de neurones (MMC-RN)

DTW était la méthode la plus répandue dans les années 70-80, dans les applications commerciales de la reconnaissance de parole. Aujourd'hui ce sont les MMC. Les MMC-RN donnent de meilleurs résultats que les MMC seuls, mais ne sont pas encore très répandus.

Seule la méthode DTW est étudiée ici.

b) Caractéristiques extraites du signal de parole

Tel quel, un signal de parole n'est pas facilement exploitable. Des informations pertinentes doivent en être extraites, qu'on appelle également éléments caractéristiques.

De plus il s'agit d'un signal non stationnaire (c'est à dire qui change en permanence), ce qui pose un problème pour en extraire des informations fréquentielles. On le découpe donc en morceaux, ou tranches, régulièrement espacées dans le temps.

Dans le cas le plus simple de la reconnaissance de mots isolés, un mot est codé par un ensemble de vecteurs de caractéristiques, chaque vecteur codant une des tranches temporelles.

Une méthode classique d'extraction de caractéristiques consiste à appliquer la Transformée de Fourier (TF) à chaque trame de signal. On parle alors de TF glissante. Comme on est dans le domaine échantillonné, il s'agit de la Transformé de Fourier Discrète (TFD). Les entrées du bloc de calcul de la TFD sont n échantillons du signal, ses sorties sont n points, correspondant à des composantes fréquentielles différentes. La fréquence du i^{e} point est $i \times f_e / n$. Seuls l'information des $n/2$ premiers points est utile (les autres constituent une symétrie des premiers par rapport à $f_e/2$.

Les différentes étapes de cette méthode sont répétées en boucle jusqu'à ce que le mot complet ait été traité. Ces étapes sont les suivantes :

1. une "tranche temporelle" (ou trame) du signal est découpée
2. cette trame est multipliée par une fonction fenêtre
3. la TFD est appliquée à la trame "fenêtrée"
4. le résultat de la TFD est mis au carré

L'intérêt de l'étape de fenêtrage (2) est d'atténuer de manière continue et douce les extrémités de la trame. En effet un arrêt brutal de ces extrémités revient à ajouter des composantes fréquentielles parasites au signal.

L'étape 4 permet d'obtenir une information homogène à un aspect physique du signal : son énergie à une fréquence donnée.

Les trames successives doivent se chevaucher (par exemple sur 1/3 ou 1/2 de leur longueur), pour éviter de perdre l'information d'une partie du signal.

En général, on utilise l'algorithme rapide de calcul de la TFD, appelé FFT (pour Fast Fourier Transform). Cet algorithme est basé sur les propriété de l'exponentielle, présente dans la formule de la TFD, et peut s'appliquer si le nombre d'échantillons est une puissance de 2 (128, 256, 512, 1024, etc).

Les n sorties du bloc calculant la TFD peuvent être utilisées comme vecteurs caractéristiques d'une trame de signal. Cependant en général on réduit la dimension de ce vecteur en calculant des moyennes. De plus on effectue également des moyennes temporelles (sur quelques trames adjacentes), pour réduire l'aspect aléatoire du signal de parole (c'est ce qu'on appelle le périodogramme de Welch).

c) Description de la méthode DTW

Principe

Soit deux ensembles de vecteurs A et B (par exemple des spectrogrammes), caractéristiques de 2 signaux de parole, de longueur I et J (par exemple deux mots) :

$$A=\{a_1, a_2, ..., a_I\}, B=\{b_1, b_2, \ldots, b_J\}$$

Les a_i et les b_j sont des vecteurs de dimension n, extraits par exemple de la façon décrite dans le paragraphe précédent.

Un des deux ensembles constitue la forme de référence et l'autre la forme à reconnaître.

La méthode nécessite de calculer d'abord toutes les distances entre les vecteurs des ensembles A et B, deux à deux. Ces distances sont rangées dans un tableau :

	a_1	a_2		a_I
b_1	$d(a_1,b_1)$	$d(a_2,b_1)$		$d(a_I,b_1)$
b_2	$d(a_1,b_2)$	$d(a_2,b_2)$		...
b_J	$d(a_1,b_J)$	...		$d(a_I,b_J)$

où $d(a_i, b_j)$,par exemple, est la distance euclidienne, entre les vecteurs de caractéristiques a_i et b_j.

La méthode DTW consiste à rechercher dans ce tableau le chemin partant de la case en haut à gauche. En faisant cela on impose que le début du mot à reconnaître correspond au début du mot de référence.

A chaque passage d'une case à l'autre, on autorise par exemple 3 directions possibles (ce que l'on appelle des contraintes), et on choisit la direction qui mène à la case dans laquelle la distance est minimale.

On cumule ces distances minimales au fur et à mesure de la progression dans le tableau, ce qui donne une distance cumulée. Cette distance cumulée permet au bout du compte de déterminer le chemin avec la distance cumulée la plus petite. Cette méthode est une méthode de programmation dynamique, et il a été démontré en 1967 que le chemin de coût minimal était la somme de tous les sous-chemins minimaux.

De manière plus formelle, on calcule la distance cumulée D(i, j), avec $i \in [1, I]$ et $j \in [1, J]$, en tenant compte des seules transitions autorisées. L'algorithme consiste à choisir à chaque itération, la distance cumulée D(i, j) (avec d(i, j) la distance entre les deux 'tranches' A(i) et B(j)) :

$$D(i, j) = \min[(D(i-1, j-1), D(i-1, j), D(i, j-1)] + d(i, j)$$

Cette opération de comparaison est répétée avec toutes les formes de référence, et la forme pour laquelle la distance est minimale est retenue comme la classe de la forme à comparer.

Le résultat doit être normalisé, pour le rendre indépendant de la taille des 2 ensembles (en nombre de vecteurs), en divisant par la somme des tailles des 2 ensembles. La distance entre les 2 formes A et B est donc finalement :

$$\text{dist}(A,B) = \frac{D(i,j)}{I+J}$$

Algorithme

```
très_grand=10000
g[0][0]=0
pour j variant de 1 à J
        g[0][j]=très_grand
        pour i variant de 1 à I
                d=dist(a_i, b_j)
                g[i][j]= min(g[i-1][j]+d, g[i-1][j-1]+d, g[i][j-1]+d)
        fin pour
fin pour
D=g[I][J]/(I+J)
```

Annexes

A1. Affichage de données multi-dimensionnelles

Intérêt

Il n'est pas possible de visualiser mentalement, ni même de représenter graphiquement, des données de dimension supérieure à 3. Dans certains cas il est pourtant nécessaire de les visualiser, par exemple pour repérer des regroupements et initialiser certains algorithmes.

Il est possible de réaliser des projections en 2 ou 3 dimensions qui conservent les relations existant entre les points dans l'espace multidimensionnel dans lequel ils sont définis. Un algorithme bien connu est celui de Sammon.

Principe

Le principe de cette méthode est de définir un critère d'erreur qui mesure la somme des différences au carré entre les distances entre tous les points considérés deux à deux dans l'espace de départ, et les distances entre les mêmes points dans l'espace d'arrivée, et de minimiser cette erreur.

Soit n le nombre de points, D la dimension de l'espace de départ, d la dimension de l'espace d'arrivée (en général, 2), $D_{i,j}$ $\forall$ i, j=1,....,n, $i \neq j$, la distance (en général, euclidienne) entre 2 points dans l'espace de départ, $d_{i,j}$ $\forall$ i, j=1,....,n, $i \neq j$ la distance entre 2 points dans l'espace d'arrivée. Le critère à minimiser est :

$$E = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} D_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{(D_{ij} - d_{ij})^2}{D_{ij}}$$

Ce critère varie entre 0 et 1.

$$c = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} D_{ij}} \text{ est une constante}$$

Les coordonnées des points dans l'espace d'arrivée constituent les variables à déterminer par l'algorithme. N'étant initialement pas connues, elles sont initialisées à des valeurs aléatoires. Les points sont déplacés à chaque itération de l'algorithme par petites quantités, jusqu'à ce que le critère E atteigne un minimum. Ce minimum est local, c'est à dire que l'algorithme peut converger vers cette valeur alors qu'il existe un ou plusieurs minima plus petit(s). La modification s'effectue selon la descente la plus pentue (steepest descent) du gradient du critère *E* (c'est également la méthode qui est à la base de l'apprentissage par rétropropagation du gradient).

A chaque itération de l'algorithme, chaque coordonnée dans l'espace d'arrivée est modifiée d'une quantité proportionnelle au négatif du gradient du critère :

$$x_{pq}(t+1) = x_{pq}(t) - \alpha . \frac{\partial E(t)}{\partial x_{pq}(t)}$$

où x_{ik} est la k^e coordonnée du q^e point dans l'espace d'arrivée. Une valeur usuelle du taux de modification α est 0,3 à 0,4.

x et $\frac{\partial E}{\partial x}$ sont des vecteurs multidimensionnels. Pour le p^e point :

$$x_i = (x_{i1}, x_{i2}, ..., x_{in})^t$$

et

$$\frac{\partial E(t)}{\partial x_p} = \left(\frac{\partial E(t)}{\partial x_{p1}}, \frac{\partial E(t)}{\partial x_{p2}}, ..., \frac{\partial E(t)}{\partial x_{pn}} \right)^t$$

où n est la dimension de l'espace d'arrivée.

On peut démontrer que les dérivées par rapport à chacune des variables sont définies par :

$$\frac{\partial E}{\partial x_{pq}} = -2c. \sum_{\substack{j=1 \\ j \neq p}}^{n} \frac{D_{pj} - d_{pj}}{D_{pj}.d_{pj}} (x_{pq} - x_{jq})$$

C'est la relation que l'on peut alors programmer.

A2. Introduction aux réseaux de neurones

Définition

Le domaine des réseaux de neurones est également appelé *connexionnisme.*

Un neurone est un processeur élémentaire (ultra-simple) qui modélise grossièrement un neurone naturel. L'opération qu'il réalise est une somme pondérée d'un certain nombre de signaux d'entrée (ce nombre peut être variable), et applique cette somme à une fonction de transfert qui peut avoir différentes formes (voir figure ci-dessous). Les coefficients de pondération sont appelés coefficients synaptiques, ou poids synaptiques.

Les signaux d'entrée sont constitués par les signaux de sortie d'autres neurones connectés au neurone considéré, ou des signaux d'entrée. Les coefficients synaptiques modélisent les "forces" de connexion entre les neurones. Ces forces de connexion sont adaptées par *apprentissage*.

Relation entrées-sortie

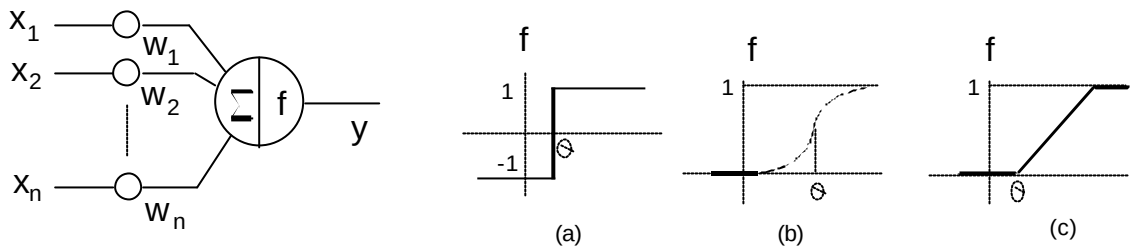
L'opération réalisée par un modèle de neurone est :

$$y = f\left(\sum_{i=1}^{n} w_i x_i \right)$$

ou

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta \right)$$

où θ est appelé biais. Il correspond à un décalage de la fonction f sur l'axe des abscisses. L'argument de f est appelé "état interne" du neurone ou encore potentiel post-synaptique. F est appelée fonction d'activation ou encore fonction de transfert.



La fonction d'activation peut avoir diverses formes :
(a) *seuil logique*, (neurone de McCulloch et Pitts) ; les états de sortie peuvent alors être 0 et 1 ou –1 et +1
(b) *logistique* ; elle a une forme similaire à la fonction seuil, mais est continue : elle peut varier entre –1 et +1, ou 0 et +1
(c) *linéaire* avec ou sans saturation

La fonction de la figure (b) ci-dessus s'appelle sigmoïde. Elle s'exprime par :

$$f(x) = \frac{1}{1+e^{-(x-\theta)/a}}$$

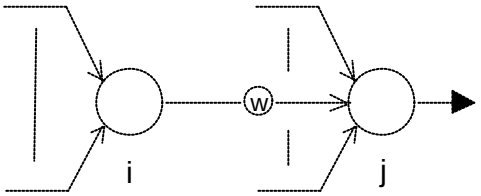
Le paramètre *a* est la pente de la sigmoïde au niveau de son point d'inflexion.

Le décalage de la fonction de transfert sur l'axe des abscisses θ peut être vu comme un lien d'entrée supplémentaire du neurone, dont l'entrée est une valeur fixe. Cette entrée constante peut être notée par exemple x_0, et le poids du lien connecté à cette entrée w_0. Dans ce cas on peut écrire la sortie du neurone de la façon suivante :

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

où f est centrée sur 0, et n est le nombre d'entrées du neurone.

Il y a 2 cas possibles pour une entrée d'un neurone : être constituée par :
- la sortie d'un autre neurone
- une entrée, c'est à dire une composante d'un vecteur de données



Les poids synaptiques sont les paramètres libres du réseau. Ils sont déterminés automatiquement en fonction du problème par apprentissage, une fois la structure du réseau établie.

Connexion entre neurones

Plusieurs neurones peuvent être reliés entre eux selon divers schémas de connexion, pour définir des réseaux.

Le schéma de connexion peut définir des couches, et dans ce cas la connexion d'une couche à l'autre peut être totale (les neurones d'une couche sont connectés à tous les neurones

de la couche en amont) ou locale (les neurones d'une couche sont connectés à une partie des neurones de la couche en amont).

Si tous les neurones sont connectés entre eux, on parle de réseaux récurrents car les sorties d'un neurone sont reliées à ses entrées, de manière directe ou indirecte.

Règles d'apprentissage

Les règles d'apprentissage indiquent la quantité de modification à appliquer à chaque poids, en fonction des exemples d'entrée (et des sorties désirées associées, dans le cas de l'apprentissage supervisé) :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

où t désigne un instant avant la modification des poids, et t+1 un instant après.

Il existe un grand nombre de règles d'apprentissage différentes. Les 3 plus répandues sont les suivantes :

- $\Delta w_{ij} = \alpha . y_i . y_j$ règle de Hebb
- $\Delta w_{ij} = \alpha . (y_i - w_{ij}) . y_j$ règle d'apprentissage compétitif
- $\Delta w_{ij} = \alpha . y_i . (y_j^d - y_j)$ règle d'apprentissage supervisé

où :

- w_{ij} est le poids de la connexion entre les neurones i et j ; cette connexion est un lien d'entrée du neurone j ;
- y_i et y_j sont respectivement les sorties des neurones source i et cible j ;
- α est un coefficient d'apprentissage, en général inférieur à 1 : il règle la quantité d'adaptation des poids à chaque présentation d'un exemple ;
- y_j^d est la sortie désirée du neurone j.

Remarque : dans le cas d'un neurone d'une couche d'entrée y_i représente la i^e coordonnée du vecteur d'entrée.

La 1ère règle est une règle d'apprentissage *non-supervisé* (ou *sans professeur*), c'est à dire dans laquelle n'apparaît pas de terme de sortie désirée des neurones. Ceci est équivalent aux méthodes de classification sans professeur, dans lesquelles les classes correctes ne sont pas connues. Hebb était un neurobiologiste qui a cherché à modéliser les phénomènes d'apprentissage dans les neurones naturels.

La 2^e peut être utilisée pour un apprentissage supervisé ou non-supervisé (par exemple VQ ou LVQ).

La 3^e est une règle d'apprentissage *supervisé* (ou *avec professeur*). La classe correcte est connue au moins pour une partie des exemples disponibles. Dans le cadre des réseaux de neurones ces informations constituent des sorties désirées des neurones. Les 2 termes de la règle correspondent à :

$$\Delta w_{ij} = \text{taux d'apprentissage} \times \text{entrée} \times \text{erreur de sortie}$$

Parfois la combinaison entre le vecteur d'entrée et le vecteur de poids d'un neurone n'est pas constitué d'un produit de corrélation mais d'une distance (en général euclidienne) entre ces deux vecteurs (voir plus loin : réseaux VQ et LVQ, réseau RCE). En fait, quand le produit de corrélation entre deux vecteurs est maximal, la distance entre eux est minimale dans le cas où ces vecteurs sont normalisés (c'est à dire que leur norme est égale à 1). Donc dans certaines conditions, les méthodes qui utilisent les produits de corrélation et celles qui utilisent les distances sont équivalentes.

Utilisation de Réseaux de neurones en Reconnaissance de Formes

Comme dans le cas des méthodes "classiques" de reconnaissance de formes, les 2 grandes étapes d'utilisation d'un classifieur sont :

- sa construction, qui comprend ici la définition de la structure du réseau et l'adaptation des poids par apprentissage ;
- son utilisation pour la classification (en généralisation), qui consiste à activer le réseau avec des nouvelles données, sans adapter les poids.

A3. Quelques éléments de calcul vectoriel et matriciel

Vecteurs

La notation usuelle pour un vecteur est sous forme d'une colonne.

Norme

Soit $x = (x_1, x_2, ..., x_n)^t$ un vecteur de dimension n.

Sa norme euclidienne est définie par :

$$\|x\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$$
$$= \sqrt{\sum_{i=1}^{n} x_i^2}$$

Un vecteur est dit normal si sa norme vaut 1.

Produit scalaire

Le produit scalaire de 2 vecteurs $x = (x_1, x_2, ..., x_n)^t$ et $y = (y_1, y_2, ..., y_n)^t$ est défini par :

$$x.y = \sum_{i=1}^{n} x_i.y_i$$

La norme d'un vecteur x est également la racine du produit scalaire de son transposé avec lui-même :

$$\|x\|^2 = x^t.x$$

La normalisation d'un vecteur consiste à rendre sa norme égale à 1. Ce résultat est obtenu en divisant toutes les composantes du vecteur par la norme de ce dernier :

$$x' = \frac{x}{\|x\|}$$

est normalisé. On dit également qu'il est normal.

Orthogonalité entre vecteurs

Le cosinus de 2 vecteurs est défini par :

$$\cos(x, y) = \frac{x^t.y}{\|x\|.\|y\|}$$

Il correspond au produit scalaire des versions normalisées des deux vecteurs.
Les deux vecteurs sont orthogonaux si le cosinus entre eux est nul.

Une matrice est dite orthogonale si toutes ses colonnes sont orthogonales entre elles deux à deux.
Une matrice orthogonale multipliée par sa transposée donne une matrice diagonale.

Projection

La projection d'un vecteur x sur un autre y est définie par :

$$\frac{x^t.y}{\|y\|}$$

Distance

La distance entre deux vecteurs est une grandeur souvent utilisée pour quantifier la similarité entre eux. Elle ne peut être calculée que si les deux vecteurs ont la même dimension.
La distance euclidienne entre deux vecteurs x et y est définie par :

$$d_E(x,y) = \sqrt{\|x-y\|} = \sqrt{(x-y)^t(x-y)} = \sqrt{\sum_i (x_i - y_i)^2}$$

Un autre exemple de distance est la distance de Hamming, définie par :

$$d_H(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

Lien entre distance euclidienne et produit scalaire

Soit d(x,y) la distance euclidienne entre les vecteurs x et y. On a :

$$d^2(x,y) = (x-y)^t(x-y)$$

? $$d^2(x,y) = x^t x + y^t y - 2x^t y = \|x\|^2 + \|y\|^2 - 2x^t y$$

Si les vecteurs x et y sont normalisés, on a

$$d^2(x,y) = 2(1 - x^t y)$$

Si le produit scalaire est minimal, le produit scalaire est maximal.

On a également

$$d^2(x,y) = \|x\|^2 + \|y\|^2 - 2\|x\|\|y\|\cos(x,y)$$

Indépendance

Soient $\{x_1, x_2, \ldots, x_m\}$ un ensemble de vecteurs. Un vecteur y est une combinaison linéaire de ces vecteurs s'il peut être mis sous la forme

$$y = c_1x_1 + c_2x_2 + \ldots + c_mx_m$$

où c_i sont des nombres réels dont au moins l'un d'entre eux est non nul. Par exemple, $y = x_1$ est une combinaison linéaire.
L'ensemble des vecteurs x_i est linéairement indépendant si on ne peut pas trouver un ensemble de valeurs $\{c_1, c_2, \ldots, c_m\}$ tel que

$$c_1x_1+c_2x_2+\ldots+c_mx_m=\mathbf{0} \quad \text{(vecteur nul)}$$

dont au moins un des coefficients est non nul ; ce qui revient à dire qu'aucun des vecteurs n'est une combinaison linéaire des autres.

Ces notions s'appliquent également aux lignes et aux colonnes des matrices.

Par exemple, dans la matrice A suivante, les colonnes sont linéairement dépendantes :

$$A=\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 1\\ 3 & 6 & 0\end{pmatrix}$$

Dans la matrice B suivante également, bien que cela soit moins évident :

$$B=\begin{pmatrix}3 & 4 & 2\\ 1 & 0 & 2\\ 2 & 1 & 3\end{pmatrix}$$

En effet :

$$\begin{pmatrix}4\\ 0\\ 1\end{pmatrix}=2\times\begin{pmatrix}3\\ 1\\ 2\end{pmatrix}-\begin{pmatrix}2\\ 2\\ 3\end{pmatrix}$$

Rang d'une matrice

Le rang d'une matrice n'est pas forcément égal à sa dimension. Il est égal au plus grand nombre de lignes (ou de colonnes) constituant un ensemble linéairement indépendant.

Par exemple, la dimension de la matrice A est 3×3, et son rang est 3×2.

Inverse d'une matrice

Cette opération n'est applicable qu'aux matrices carrées, et uniquement de plein rang, c'est à dire des matrices dont le rang est égal à la dimension.

Propriétés :

- $A\times A^{-1}=I$
- $A\times A^{-1}= A^{-1}\times A$

Dans le cas d'une matrice diagonale $A=\text{diag}\{a_{i,i}\}$, l'inverse est égale à :

$$A^{-1}=\text{diag}\{1/a_{i,i}\}$$

Dans le cas de matrices non carrées, on définit la notion d'inverse généralisée (encore appelée pseudo-inverse ou inverse de Moore-Penrose).

Vecteurs propres et valeurs propres

Un vecteur u qui vérifie l'équation

$$A.u=\lambda.u \quad (1)$$

est un vecteur propre de la matrice A et λ est la valeur propre qui lui est associée. On a également :

$$(1) \leftrightarrow A.u-\lambda.u=0$$
$$\leftrightarrow (A-\lambda.I).u=0$$

où I est la matrice identité (des 1 sur la diagonale, des 0 partout ailleurs). Cette équation est appelée équation caractéristique de la matrice. Si u est un vecteur propre, tout vecteur k.u avec k constante est un autre vecteur propre.

Une matrice de dimension n×n possède n vecteurs propres (associés chacun à une valeur propre).

En général on rassemble les vecteurs propres dans une matrice, sous la forme de ses colonnes, et les valeurs propres dans une autre matrice, disposées en colonne, dans l'ordre correspondant à celui des vecteurs propres. Soient U et Λ ces deux matrices, respectivement. L'équation matricielle correspondant à (1) est :

$$A.U=\Lambda.U$$

Analyse en composantes principales (ACP)

L'ACP est l'application de la décomposition en vecteurs et valeurs propres, à la matrice de covariance d'un ensemble de vecteurs.

Cette décomposition est également appelée décomposition (ou transformation) de Karhunen-Loeve, du nom des chercheurs ayant proposé cette méthode dans le cadre du traitement du signal.

L'ACP d'un ensemble de n vecteurs va donner comme résultat un ensemble de n autres vecteurs de même dimension, associés chacun à un coefficient scalaire appelé valeur propre. De plus, les vecteurs propres sont rangés par ordre de décroissance de leur valeur propre associée.

Elle consiste à utiliser les vecteurs propres et les valeurs propres de la matrice de corrélation d'images de référence, et ne garder que les vecteurs propres associés aux plus grandes valeurs propres. Elle permet une réduction de la dimension des données, intéressante par exemple dans la compression de signaux ou d'images.

A4. Quelques éléments de statistiques

Moyenne d'une variable

Par définition, une variable peut prendre différentes valeurs. Soient x_i n exemples (réalisations) de la variables x. La moyenne d'une variable x (par exemple la variable "longueur de pétale" pour les données Iris) est définie par :

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

On dit qu'une variable est centrée si sa moyenne est nulle.

Variance d'une variable

La variance d'une variable est égale à la somme des carrés des différences entre les exemples de cette variable et leur moyenne :

$$\mathrm{var}(x) = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n}$$

La variance caractérise donc la somme des écarts d'une variable par rapport à sa moyenne.

Covariance entre des variables différentes

Dans le cas de données multi-dimensionnelles, il y a plusieurs variables (par exemple dans les données Iris il y a 4 variables). Ces variables peuvent être corrélées ou non. Si elles sont corrélées, leur covariance et leur produit de corrélation, définis ci-dessous, ne sont pas nuls.

Soient x_i et x_j deux variables et leurs moyennes respectivement $\overline{x_i}$ et $\overline{x_j}$. La covariance entre elles est définie par :

$$\mathrm{cov}(x_i, x_j) = \frac{1}{n}\sum_{k=1}^{n}\left(x_{i,k} - \overline{x_i}\right)\left(x_{j,k} - \overline{x_j}\right)$$

où n est le nombre de réalisations des variables (=nombre de vecteurs d'exemple), $x_{i,k}$ est le k^e exemple de la variable i (et $x_{j,k}$ est la k^e réalisation de la variable j).

Par exemple, dans les données *Iris*, la covariance entre la longueur et la largeur de pétale n'est pas nulle.

La covariance entre une variable et elle-même est égale à sa variance.

Corrélation entre des variables différentes

Quand la covariance entre des variables différentes n'est pas nulle, ces variables varient ensemble et elles sont également corrélées.

Le produit de corrélation est la version normalisée (il est compris entre 0 et 1) de la covariance :

$$\mathrm{cor}(x_i, x_j) = \frac{\mathrm{cov}(x_i, x_j)}{\sigma_i \sigma_j}$$

où

$$s_i = \sqrt{\mathrm{var}(x)}\ ,\ i=1,2$$

est appelé écart-type de la variable *x*.

Dans le cas d'un nombre plus grand de variables, celles-ci peuvent être corrélées 2 à 2. On définit alors une matrice de variance-covariance C :

$$C = \left\{\mathrm{cov}(x_i, x_j), i = 1,...,n; j = 1,...,n\right\}$$

où i est l'indice de la colonne dans la matrice C et j l'indice de la ligne, ou inversement puisque cette matrice est symétrique.

Cette notation signifie qu'à l'intersection de la ligne j et de la colonne i de cette matrice se trouve la covariance entre les variables x_i et x_j. La dimension de C est donc $n \times n$.

La matrice de covariance peut également être obtenue de la façon suivante : à partir du produit d'une matrice A et de sa transposée, où A contient tous les vecteurs de données, auxquels on a soustrait leur moyenne (définie ci-dessus), disposés en colonnes. A^T contient donc les mêmes vecteurs mais disposés en lignes :

$$C = \frac{1}{n}.A.A^T$$

A5. Quelques éléments de probabilités

Les mêmes variables que celles qu'on a considérées jusqu'à maintenant peuvent être vues comme des variables aléatoires (VA). On utilise alors l'outil des *probabilités* pour les étudier.

Distribution de probabilité et fonction de densité de probabilité

Il existe des variables *discrètes* (par exemple le résultat d'un lancé de dé) et des variables *continues* (par exemple la longueur d'un pétale d'un *Iris*). Les variables discrètes peuvent être décrites par des probabilités, les variables continues par des *densités de probabilité.*

Par exemple, le résultat du tirage d'un dé comporte 6 cas possibles. La probabilité P de chaque cas ("événement") x_i est 1/6. On dit que la distribution de probabilité est uniforme. La représentation sur un repère donnera une constante $P(x_i)=1/6$ pour x prenant les valeurs entières de 1 à 6.

Supposons que le nombre des valeurs possibles d'une variable augmente ; alors la probabilité de chacune des possibilités diminue, et peut approcher 0.

Dans une distribution de probabilité, si le nombre d'évènements possibles augmente, les points sur la représentation graphique se rapprochent, et si ce nombre tend vers l'infini, la distribution se transforme en fonction de densité de probabilité.

Par exemple, dans le cas d'un générateur de nombres aléatoires réels compris entre 0 et 1, il s'agit d'une VA continue. Le nombre de valeurs possibles est quasi-infini, donc la probabilité de chaque évènement est quasi-nulle. On doit alors définir des intervalles. Par exemple, la probabilité que x soit compris entre 0 et 0,1, si la loi de probabilité est uniforme, est égale à 1/10. On parle alors de fonction de densité de probabilité (FDP), qui dans cet exemple est uniforme.

Une VA continue est définie par une fonction de densité de probabilité et non plus par des probabilités.

Les variables réelles suivent des FDP (par exemple la longueur d'un pétale d'iris, la taille des individus, leur poids, etc).

L'intégrale d'une FDP est égale à 1, puisque sa surface correspond à l'évènement certain : on est sûr que le résultat est l'une des valeurs possibles.

La fonction de répartition est l'intégrale de la FDP. La fonction de répartition représente la somme de ces probabilités, en partant de la gauche vers la droite. Par exemple dans le cas d'un dé, $P(x<3)=P(x=1)+P(x=2)=2/6$. La limite de cette fonction quand x tend vers $-\infty$ est 0, et 1 quand *x* tend vers l'infini.

Variables aléatoires discrètes

On peut définir la moyenne et la variance d'une VA discrète. Elles prennent en compte la probabilité de chaque exemple de valeur de cette variable. Les différentes valeurs que peut prendre la variable sont appelées *réalisations* de la variable.

La *moyenne* de la VA discrète *x* est définie par :

$$E(x) = \sum_i p_i . x_i$$

où p_i est la probabilité de la réalisation x_i. La moyenne est également appelée *espérance*.

Dans le cas où l'on a n réalisations de la variable et que toutes ces réalisations sont équiprobables, cette définition est la même que pour les variables non aléatoires, car $p_i=1/n$ pour les n réalisations différentes.

La *variance* d'une VA discrète x est définie par :

$$\sigma^2 = \sum_i p_i (x_i - \eta)^2$$

avec ?=E(x).

L'ensemble des probabilités des différents évènements possibles constitue ce qu'on appelle la *distribution de probabilité* de la VA x. On la représente dans un repère par les différents évènements en abscisse et des probabilités correspondantes en ordonnée.

Variables aléatoires continues

La *moyenne* d'une VA continue *x* est définie par :

$$E(x) = \int_{-\infty}^{+\infty} x.f(x)dx$$

Exemple : espérance d'une variable aléatoire suivant une densité de probabilité d(x) uniforme entre 0 et 1 :

$$E(x) = \int_0^1 x dx$$

$$E(x) = \left[\frac{x^2}{2} \right]_0^1$$

$$E(x) = \frac{1}{2}$$

Par exemple, variable générée par programmation : un réel compris entre 0 et 1 (comme la fonction du fichier rdf.c).

La variance de cette même variable est définie par :

$$\sigma^2 = \int_{-\infty}^{+\infty} (x - \eta)^2 . f(x)dx$$

(*Remarque* : σ^2 est la moyenne de la VA $(x-\eta)^2$)

Avec une VA continue, les probabilités n'ont plus de sens, puisque le nombre de réalisations de la variable tend vers l'infini, et il faut raisonner sur des intervalles. On définit donc une VA continue par une *fonction de densité de probabilité* (FDP). La FDP d'une VA continue x est souvent notée p(x).

Exemple 1 : tirage d'un dé. On obtient pour la variance environ 4,9.

Exemple 2 : variable aléatoire uniforme comprise entre 0 et 1. On obtient pour la variance :

$$\sigma^2 = \int_0^1 (x - \frac{1}{2})^2 dx$$
$$\sigma^2 = \int_0^1 x^2 - x + \frac{1}{4} dx$$
$$\sigma^2 = \left[\frac{x^3}{3} - \frac{x^2}{2} + \frac{x}{4} \right]_0^1$$
$$\sigma^2 = \left[\frac{1}{3} - \frac{1}{2} + \frac{1}{4} \right]$$
$$\sigma^2 = \frac{1}{12}$$

Variables aléatoires (continues) gaussiennes

Une VA continue gaussienne de dimension 1 (équivalent à un vecteur de dimension 1, que l'on appelle un scalaire) possède une FDP définie par l'expression d'une fonction gaussienne.

Si cette FDP est centrée sur 0, on a :

$$p(x) = G(x)$$

avec

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2} \right)$$

σ^2 représente la variance et σ l'écart-type ; ce paramètre détermine l'étalement de la courbe sur l'axe des abscisses. L'intégrale de cette fonction (qui représente la surface située entre la courbe et l'axe des abscisses) pour *x* variant de -∞ à +∞ est égale à 1. x_c est appelée moyenne, à ne pas confondre avec une moyenne arithmétique. Il représente le centre de la courbe sur l'axe des abscisses.

Si cette variable est centrée sur x_c :

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - x_c)^2}{2\sigma^2} \right)$$

Dans le cas plus général de vecteurs de variables de dimension n (nombre de composantes des vecteurs), G(x) est définie par :

$$G(x) = \frac{1}{(2\pi)^{n/2}\sqrt{|C|}} \exp\left[-\frac{(x - x_c)^T C^{-1} (x - x_c)}{2} \right]$$

C est la matrice de covariance (de dimension n×n) des coordonnées de x, C^{-1} son inverse et |C| son déterminant. Cette matrice contient les covariances entre les différentes variables $x_i.x_j$, i,j=1,...,d où d est la dimension des données. x_c est le centre (ou moyenne) de la gaussienne.

Il s'agit du cas général, où les variables sont corrélées entre elles, ce qui n'est pas toujours le cas. Dans le cas ou elles ne sont pas corrélées, on a $\sqrt{|C|} = \sigma$ et C^{-1}=I, la matrice identité.

La distribution gaussienne est également appelée loi normale. Sa notation compacte peut alors s'écrire : $x \sim N(x_c, \sigma^2)$. On dit alors que la variable aléatoire x suit une loi normale de moyenne x_c et de variance σ^2. Cette notation n'indique pas la dimension des vecteurs x.

De nombreuses données réelles sont définies par une FDP gaussienne. Cette propriété résulte du *théorème central limite*, qui énonce que sous certaines conditions générales, la distribution de x approche une FDP gaussienne de moyenne h et de variance s^2 quand le nombre de réalisations de cette variable augmente. Selon ce théorème, la distribution d'observations résultant de plusieurs causes non corrélées, où aucune ne prédomine sur une autre, tend vers une distribution gaussienne (ce qui est le cas de très nombreuses données réelles).

On peut obtenir une FDP gaussienne en ajoutant n VA indépendantes x_i. Leur somme est une VA de moyenne égale à la somme des moyennes des x_i, et de variance égale à la somme de leurs variances :

$$\eta = \eta_1 + \eta_2 + ... + \eta_n$$

et

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 + ... + \sigma_n^2$$

alors la variable $\frac{x - \eta}{\sigma^2}$ est une VA gaussienne centrée réduite, c'est à dire de moyenne nulle (=centrée sur 0) et délimitant une surface égale à 1 avec l'axe des abscisses.

Avec une somme pondérée de plusieurs gaussiennes, on peut approcher n'importe quelle distribution décrivant un phénomène réel. On parle alors d'approximateur universel.

Probabilités conditionnelles

Les probabilités conditionnelles relient différents évènements entre eux. Un exemple d'évènement est "le résultat d'un lancé de dé est 6".

La probabilité d'un évènement A sachant qu'un évènement B s'est réalisé est définie par :

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \quad (1)$$

avec : $P(A \cap B) = P(B \cap A)$: probabilité que les évènements *A* et *B* se réalisent en même temps. Donc on a (d'après (1)) :

$$P(A \cap B) = P(A \mid B).P(B)$$

et

$$P(A \cap B) = P(B \mid A).P(A)$$

Remarque : les notations $P(A \cap B)$, P(A et B) et P(A, B) sont équivalentes.

Si A et B sont indépendants :

$$P(A|B) = P(A)$$

et donc :

$$P(A \cap B) = P(A) \times P(B)$$

Exemple : 2 lancés successifs d'un dé sont des évènements indépendants : le résultat du 2e lancé est indépendant du 1er.

Pour un plus grand nombre d'évènements A_1, A_2, …., A_k (cas général d'évènements non indépendants) :

$$P(A_1 \text{ et } A_2 \text{ et } ... \text{ et } A_k)=p(A_1|A_2 \text{ et } ... \text{ et } A_k) \times p(A_2|A_3 \text{ et } ... \text{ et } A_k) \times ... \\ ...\times p(A_{k-1}|A_k) \times P(A_k)$$

Remarque : on peut inverser l'ordre des évènements.

Si les évènements A_i sont mutuellement indépendants, on a :

$$P(A_1 \text{ et } A_2 \text{ et } ... \text{ et } A_k) = P(A_1) \times P(A_2) \times ... \times P(A_{k-1}) \times P(A_k)$$

Pour les probabilités conditionnelles :

$$P(A \text{ et } B \mid C) = P(A \mid B \text{ et } C) \times P(B \mid C)$$

Si A et B sont indépendants conditionnellement à C, alors

$$P(A \text{ et } B \mid C) = P(A \mid C) \times P(B \mid C)$$

Probabilités totales

Un évènement peut parfois se décomposer en plusieurs évènements exclusifs, c'est à dire distincts. Ces évènements peuvent également être exhaustifs, c'est à dire qu'un de ces évènements se réalise nécessairement. Par exemple, l'évènement "le lancé d'un dé donne un résultat supérieur à 4" peut se décomposer en deux évènements exclusifs et exhaustifs : "le résultat est 5" et "le résultat est 6".

La formule des probabilités totales permet de calculer la probabilité d'un évènement à partir d'une telle décomposition. Soient *A* et *B* deux évènements, *A* pouvant être décomposé en évènements mutuellement exclusifs et exhaustifs A_1, A_2,.... (cas général : évènements pas indépendants) Alors on a :

$$\begin{aligned} P(B) &= P(B \cap A_1) + P(B \cap A_2) + ... \\ &= \sum_i P(B \cap A_i) \\ &= \sum_i P(B \mid A_i) \times P(A_i) \end{aligned}$$

parce qu'un des évènements A_i se produit forcément.

Exemple 1 : évènements "pile" ou "face" lors du jeté d'une pièce : exclusifs et exhaustifs.

Exemple 2 : évènement "la carte est un as" (qui se décompose en 4 évènements) lorsqu'on tire une carte. Il est composé d'évènements exclusifs mais pas exhaustifs.

Formule de Bayes

La formule de Bayes est à la base de la reconnaissance de formes bayesienne. Elle peut être déduite des expressions ci-dessus :

$$P(A_i \mid B) = \frac{P(A_i \cap B)}{P(B)}$$

$$= \frac{P(B \cap A_i)}{P(B)}$$

$$= \frac{P(B|A_i) \times P(A_i)}{\sum_i P(B|A_i) \times P(A_i)}$$

les A_i sont exclusifs et exhaustifs.

Exemple des jumeaux : il existe des vrais et des faux jumeaux. Les faux jumeaux sont 2 fois plus nombreux que les vrais. Les vrais sont toujours de même sexe alors que les faux sont de même sexe avec une probabilité 1/2. Etant donné que 2 jumeaux sont de même sexe, quelle est la probabilité qu'ils soient de vrais jumeaux ?

$$P(vj|ms) = \frac{P(ms|vj) \times P(vj)}{P(ms|vj) \times P(vj) + P(ms|fj) \times P(fj)}$$
$$= \frac{1 \times 1/3}{1 \times 1/3 + 1/2 \times 2/3}$$
$$= \frac{1}{2}$$

Probabilités totales et conditionnelles

$$P(B|C) = P(B \cap A_1|C) + P(B \cap A_2|C) + ...$$

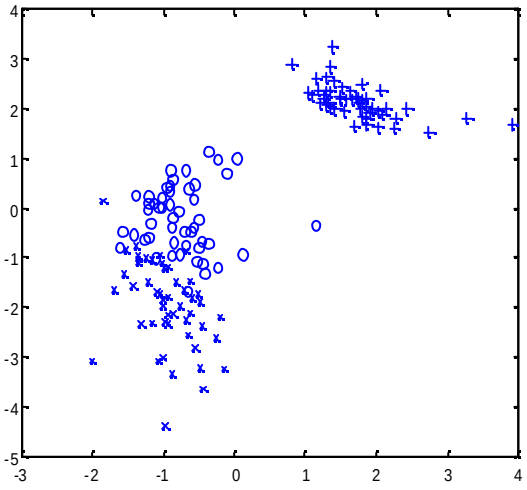
$$= \sum_i P(B \cap A_i|C)$$
$$= \sum_i P(B|A_i \cap C) \times P(A_i|C)$$

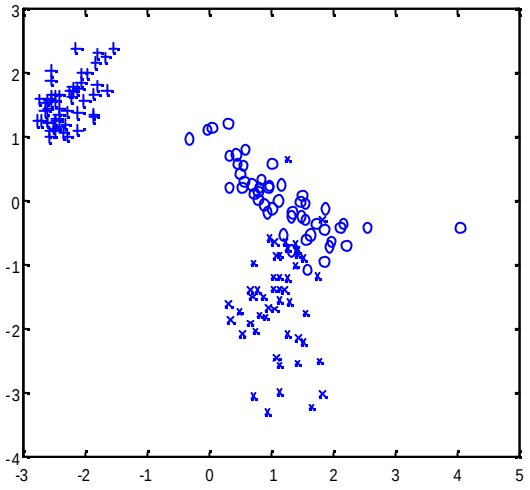
Exemples de données : Iris

Contenu du fichier original :

5.1,3.5,1.4,0.2,Iris-setosa	7.0,3.2,4.7,1.4,Iris-versicolor	6.3,3.3,6.0,2.5,Iris-virginica
4.9,3.0,1.4,0.2,Iris-setosa	6.4,3.2,4.5,1.5,Iris-versicolor	5.8,2.7,5.1,1.9,Iris-virginica
4.7,3.2,1.3,0.2,Iris-setosa	6.9,3.1,4.9,1.5,Iris-versicolor	7.1,3.0,5.9,2.1,Iris-virginica
4.6,3.1,1.5,0.2,Iris-setosa	5.5,2.3,4.0,1.3,Iris-versicolor	6.3,2.9,5.6,1.8,Iris-virginica
5.0,3.6,1.4,0.2,Iris-setosa	6.5,2.8,4.6,1.5,Iris-versicolor	6.5,3.0,5.8,2.2,Iris-virginica
5.4,3.9,1.7,0.4,Iris-setosa	5.7,2.8,4.5,1.3,Iris-versicolor	7.6,3.0,6.6,2.1,Iris-virginica
4.6,3.4,1.4,0.3,Iris-setosa	6.3,3.3,4.7,1.6,Iris-versicolor	4.9,2.5,4.5,1.7,Iris-virginica
5.0,3.4,1.5,0.2,Iris-setosa	4.9,2.4,3.3,1.0,Iris-versicolor	7.3,2.9,6.3,1.8,Iris-virginica
4.4,2.9,1.4,0.2,Iris-setosa	6.6,2.9,4.6,1.3,Iris-versicolor	6.7,2.5,5.8,1.8,Iris-virginica
4.9,3.1,1.5,0.1,Iris-setosa	5.2,2.7,3.9,1.4,Iris-versicolor	7.2,3.6,6.1,2.5,Iris-virginica
5.4,3.7,1.5,0.2,Iris-setosa	5.0,2.0,3.5,1.0,Iris-versicolor	6.5,3.2,5.1,2.0,Iris-virginica
4.8,3.4,1.6,0.2,Iris-setosa	5.9,3.0,4.2,1.5,Iris-versicolor	6.4,2.7,5.3,1.9,Iris-virginica
4.8,3.0,1.4,0.1,Iris-setosa	6.0,2.2,4.0,1.0,Iris-versicolor	6.8,3.0,5.5,2.1,Iris-virginica
4.3,3.0,1.1,0.1,Iris-setosa	6.1,2.9,4.7,1.4,Iris-versicolor	5.7,2.5,5.0,2.0,Iris-virginica
5.8,4.0,1.2,0.2,Iris-setosa	5.6,2.9,3.6,1.3,Iris-versicolor	5.8,2.8,5.1,2.4,Iris-virginica
5.7,4.4,1.5,0.4,Iris-setosa	6.7,3.1,4.4,1.4,Iris-versicolor	6.4,3.2,5.3,2.3,Iris-virginica
5.4,3.9,1.3,0.4,Iris-setosa	5.6,3.0,4.5,1.5,Iris-versicolor	6.5,3.0,5.5,1.8,Iris-virginica
5.1,3.5,1.4,0.3,Iris-setosa	5.8,2.7,4.1,1.0,Iris-versicolor	7.7,3.8,6.7,2.2,Iris-virginica
5.7,3.8,1.7,0.3,Iris-setosa	6.2,2.2,4.5,1.5,Iris-versicolor	7.7,2.6,6.9,2.3,Iris-virginica
5.1,3.8,1.5,0.3,Iris-setosa	5.6,2.5,3.9,1.1,Iris-versicolor	6.0,2.2,5.0,1.5,Iris-virginica
5.4,3.4,1.7,0.2,Iris-setosa	5.9,3.2,4.8,1.8,Iris-versicolor	6.9,3.2,5.7,2.3,Iris-virginica
5.1,3.7,1.5,0.4,Iris-setosa	6.1,2.8,4.0,1.3,Iris-versicolor	5.6,2.8,4.9,2.0,Iris-virginica
4.6,3.6,1.0,0.2,Iris-setosa	6.3,2.5,4.9,1.5,Iris-versicolor	7.7,2.8,6.7,2.0,Iris-virginica
5.1,3.3,1.7,0.5,Iris-setosa	6.1,2.8,4.7,1.2,Iris-versicolor	6.3,2.7,4.9,1.8,Iris-virginica
4.8,3.4,1.9,0.2,Iris-setosa	6.4,2.9,4.3,1.3,Iris-versicolor	6.7,3.3,5.7,2.1,Iris-virginica

Affichage de Sammon des données (2 exécutions différentes de l'algorithme)





Exemple de données : Wine

Il y a 3 classes, qui correspondent à 3 cultivateurs différents.

Dans le cadre ci-dessous sont données les 5 premières lignes de chaque classe telles qu'elles se présentent dans le fichier de données original :

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735
……
2,12.37,.94,1.36,10.6,88,1.98,.57,.28,.42,1.95,1.05,1.82,520
2,12.33,1.1,2.28,16,101,2.05,1.09,.63,.41,3.27,1.25,1.67,680
2,12.64,1.36,2.02,16.8,100,2.02,1.41,.53,.62,5.75,.98,1.59,450
2,13.67,1.25,1.92,18,94,2.1,1.79,.32,.73,3.8,1.23,2.46,630
2,12.37,1.13,2.16,19,87,3.5,3.1,.19,1.87,4.45,1.22,2.87,420
……
3,12.86,1.35,2.32,18,122,1.51,1.25,.21,.94,4.1,.76,1.29,630
3,12.88,2.99,2.4,20,104,1.3,1.22,.24,.83,5.4,.74,1.42,530
3,12.81,2.31,2.4,24,98,1.15,1.09,.27,.83,5.7,.66,1.36,560
3,12.7,3.55,2.36,21.5,106,1.7,1.2,.17,.84,5,.78,1.29,600
3,12.51,1.24,2.25,17.5,85,2,.58,.6,1.25,5.45,.75,1.51,650
……
```

Remarque : le premier chiffre sur chaque ligne correspond à un numéro de classe (arbitraire)

Informations disponibles dans le fichier associé :

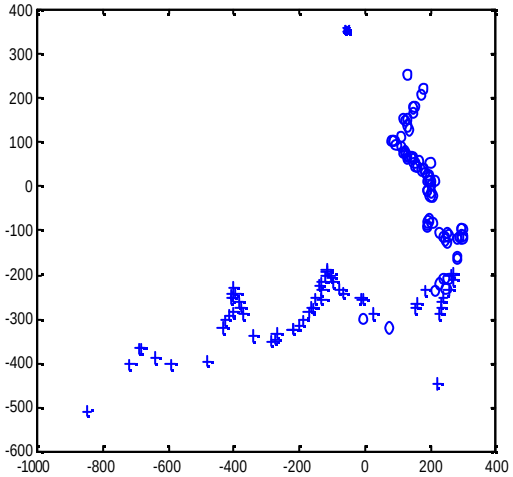
13 attributs (=caractéristiques) :

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

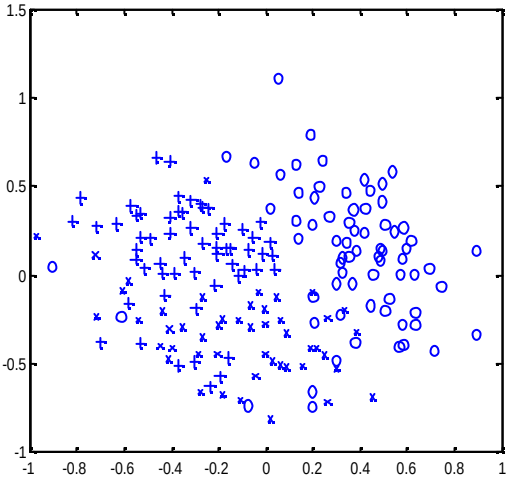
Répartition des exemples dans les 3 classes :

classe 1 : 59 classe 2 : 71 classe 3 : 48

Résultat de l'affichage de Sammon :



données non normalisées



données normalisées

Exemples de données : Caractères manuscrits

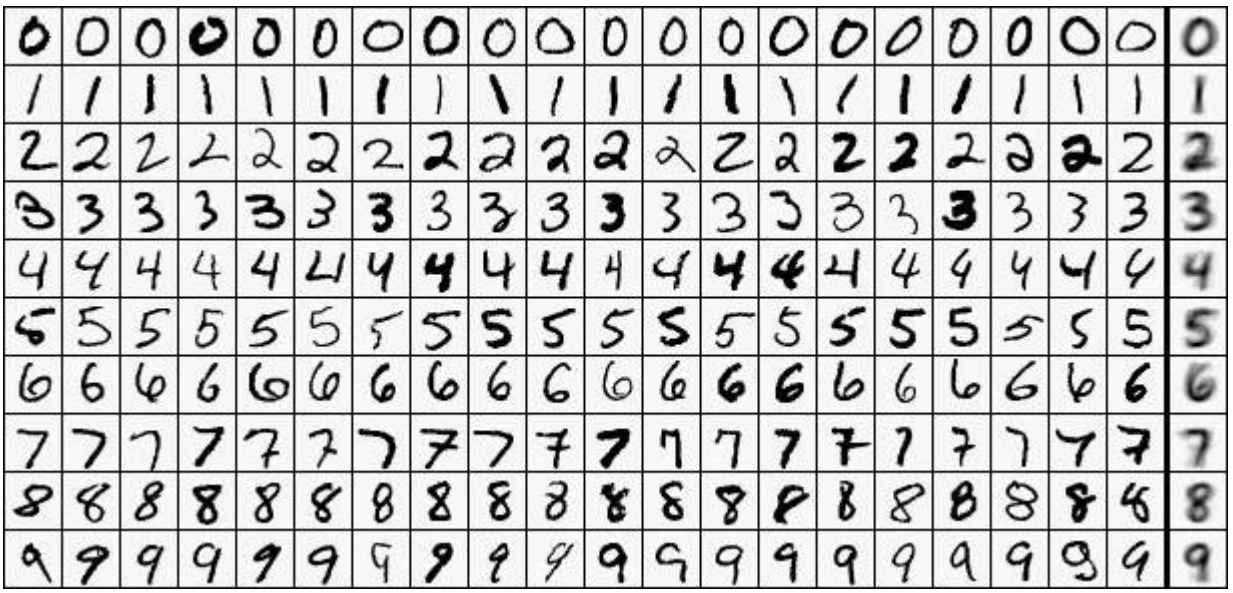
1er cas : Paramètres caractéristiques extraits d'images de caractères manuscrits (hollandais)

Extrait du fichier "caract_holland.don", qui contient 20000 exemples différents (1 exemple = 1 ligne) :

```
2 8 3 5 1 8 13 0 6 6 10 8 0 8 0 8 8 T
5 12 3 7 2 10 5 5 4 13 3 9 2 8 4 10 10 I
4 11 6 8 6 10 6 2 6 10 3 7 3 7 3 9 9 D
7 11 6 6 3 5 9 4 6 4 4 10 6 10 2 8 8 N
2 1 3 1 1 8 6 6 6 6 5 9 1 7 5 10 10 G
4 11 5 8 3 8 8 6 9 5 6 6 0 8 9 7 7 S
4 2 5 4 4 8 7 6 6 7 6 6 2 8 7 10 10 B
1 1 3 2 1 8 2 2 2 8 2 8 1 6 2 7 7 A
2 2 4 4 2 10 6 2 6 12 4 8 1 6 1 7 7 J
11 15 13 9 7 13 2 6 2 12 1 9 8 1 1 8 8 M
…
etc
…
```

2e cas : Images brutes (caractères normalisés en taille et en position)

Extrait du fichier "caract.don", qui contient 10000 exemples différents :



Exemple d'implémentation des kppv

Principe utilisé

Les vecteurs à classer sont rangés dans une 1ère liste. Au fur et à mesure de leur classification, ils sont rangés dans une 2e liste. Chaque vecteur à classer est comparé à tous les vecteurs de cette 2e liste (=les vecteurs déjà classés).

Algorithme des kppv pour un vecteur à classer

Paramètres d'entrée :
- *voisinage k*
- *vecteur à classer*
- *liste des vecteurs déjà classés*

Paramètre de sortie :
- *indice de la classe la plus proche (au sens des kppv)*

- calcul de la distance entre le vecteur à classer et tous les autres vecteurs déjà classés :
  - mettre les résultats dans un tableau de distances *T*
- recherche des *k* distances les plus petites :
  - répéter *k* fois
    - rechercher la distance minimale dans le tableau des distances *T*
    - retirer la valeur correspondante de *T*
    - copier le vecteur correspondant dans une liste temporaire *L*
- recherche de la classe la plus représentée parmi les *k* vecteurs les plus proches (vecteurs de la liste *L*):
  - initialiser *C* compteurs à 0 (où *C* nombre de classes)
  - pour chacun des *k* vecteurs de la liste temporaire *L*
    - incrémenter le compteur de la classe de ce vecteur
  - rechercher le compteur maximal, en déduire la classe la plus proche du vecteur à classer

Algorithme des kppv pour un ensemble de vecteurs

Paramètres d'entrée :
- *voisinage k*
- *liste des vecteurs à classer*
- *nombre de vecteurs pour initialiser chaque classe*

Paramètre de sortie :
- *taux de reconnaissance pour chaque classe*

- création d'une 2e liste (liste 2) pour y ranger les vecteurs au fur et à mesure de leur classification :
  liste 1 : vecteurs à classer (passée en paramètre)
  liste 2 : vecteurs déjà classés
- normalisation des vecteurs de la liste 1
- choix des exemples initiaux pour chaque classe :
  pour chaque classe i
  pour chaque exemple initial
  - choisir aléatoirement un exemple de la classe i dans la liste 1
  - copier cet exemple dans la liste 2
  - retirer cet exemple de la liste 1
- classification :
  tant que la liste 1 n'est pas vide
  - choisir aléatoirement un vecteur de cette liste
  - appliquer la méthode des kPPV à ce vecteur (algo ci-dessus)
  - comparer le résultat donné par les kPPV avec le bon résultat, et incrémenter un compteur de bonnes classifications
  - le copier dans la liste 2
  - le retirer de la liste 1 à partir des compteurs de bonne classification
- calcul des statistiques de classification

Bibliographie

Livres sur la Reconnaissance de Formes en général

- Belaïd A., Belaïd Y., « Reconnaissance des formes : méthodes et applications », InterEditions, 1992.
- Celeux G., Diday E., Govaert G., Lechevallier Y., Ralambondrainy H., « Classification automatique des données. Environnement statistique et informatique », Dunod Informatique, 1989
- Fabre P., « Exercice la reconnaissance des formes par ordinateur », Masson, 1989
- Horaud R., Monga O., « Vision par ordinateur, outils fondamentaux », Hermès, 1995
- Kunt M., Granlund G., Kocher M., « Traitement de l'information ; volume 2 : Traitement numérique des images », Presses Polytechniques et universitaires romandes, 1993
- Kunt M. (publié sous la direction de), «Reconnaissance de formes et analyse de scènes », Presses Polytechniques et Universitaires Romandes, 2000
- Miclet L., « Méthodes structurelles pour la reconnaissance de formes », Eyrolles, 1984
- Milgram Maurice, « Reconnaissance des formes ; méthodes connexionnistes et numériques », Armand Colin, 1993
- Simon J. C., « La reconnaissance de formes par algorithmes », Masson, 1985

Livres traitant de la reconnaissance de parole

- Boite R., Kunt M., « Traitement de la parole », Presses Polytechniques et Universitaires Romandes, 2000
- Calliope, « La parole et son traitement automatique », Masson, 1989
- Rabiner L. R., Juang B. H., « Fundamentals of speech recognition », Prentice Hall, 1993

Livres sur les réseaux de neurones

- Hérault J., Jutten Ch., « Réseaux neuronaux et traitement du signal », Hermès, 1994
- Jodouin J. F., « Les réseaux de neurones. Principes et définitions », Hermès, 1994
- Jodouin J. F., « Réseaux neuromimétiques », Hermès, 1994
- Nadal J.P., « Réseaux de neurones. De la physique à la psychologie », Armand Colin, 1993

Livres sur des sujets proches

- Cornuéjols A., Miclet L., « Apprentissage artificiel. Concept et Algorithmes », Eyrolles, 2002
- Coster, Chermant, « Précis d'analyse d'image », Press du CNRS, 1989

Internet

- Rauber T. W., « Cours de reconnaissance de formes », www.inf.ufes.br/~thomas/