Fouille de données
Notes de cours

Ph. PREUX
Université de Lille 3
philippe.preux@univ-lille3.fr

31 août 2009

http://www.grappa.univ-lille3.fr/~ppreux/fouille

Table des matières

Notations

On résume ici les notations utilisées dans l'ensemble de ce document :

- $\mathcal{X}$: ensemble de données ou d'exemples (*cf.* chap. 1, sec. 1.2.1) ;
- $x \in \mathcal{X}$: un élément de $\mathcal{X}$, *i.e.*, une donnée ;
- $\mathcal{A}$: ensemble des attributs décrivant chacune des données (*cf.* chap. 1, sec. 1.2.1) ;
- $a \in \mathcal{A}$: un élément de $\mathcal{A}$, *i.e.*, un attribut ;
- $\mathcal{V}_a$: ensemble des valeurs possibles pour l'attribut $a \in \mathcal{A}$ (*cf.* chap. 1, sec. 1.2.1) ;
- $\mathcal{D}$: espace des données : espace contenant toutes les données possibles pour le problème considéré ; on a la relation $\mathcal{X} \subset \mathcal{D}$ (*cf.* chap. 1, sec. 1.2.1) ;
- $N = |\mathcal{X}|$: nombre de données disponibles (*cf.* chap. 1, sec. 1.2.1) ;
- $P = |\mathcal{A}|$: nombre d'attributs décrivant chaque donnée (*cf.* chap. 1, sec. 1.2.1) ;
- K : nombre de segments (en segmentation) ;
- E : erreur (*cf.* chap. 3, sec. 3.7) ;
- $\mathcal{Y}$: ensemble des étiquettes possibles dans un problème de classification donné, ou un problème de régression (*cf.* chap. 2 et chap. 13) ;
- $y \in \mathcal{Y}$: un élément de $\mathcal{Y}$, *i.e.*, une classe dans un problème de classification ou une valeur associée à une donnée dans un problème de régression ;
- α : taux d'apprentissage ;
- $a \propto b$ signifie que la valeur de a (typiquement, une probabilité) est estimée par la valeur de b ;
- $a \approx b$ signifie que la valeur numérique de a est à peu près celle du nombre décimal b (à la précision donnée pour b, généralement le centième ou le millième).

Exceptionnellement, K et α peuvent signifier autre chose : K dans le chap. 5, α dans le chap. 8.

Résumé

J'ai rassemblé ici mes notes de cours et un certain nombre d'informations concernant la fouille de données.

On adopte une approche pragmatique et pratique, tout en essayant de donner le matériel nécessaire pour comprendre ce que l'on fait : le but n'est pas d'appliquer aveuglément des algorithmes, mais de connaître des algorithmes et de savoir quand et comment les appliquer, d'être capable de les utiliser et de juger les résultats qu'ils fournissent. En fouille de données, on ne peut pas se contenter d'appliquer aveuglément une méthode et de se contenter tout aussi aveuglément du résultat obtenu, comme s'il s'agissait de LA réponse au problème. Les algorithmes d'extraction d'information constituent une boîte à outils ; ayant cette boîte à disposition, il nous faut apprendre à les utiliser, comme l'artisan apprend à manier ces outils. Dit autrement, la fouille de données est un art : outre les connaissances plus ou moins techniques à acquérir, il faut ensuite accumuler beaucoup de pratique.

Au niveau pratique, on s'appuie exclusivement sur des logiciels libres : ils sont aisément accessibles sur la Toile. Certains sont remarquables. Malheureusement, il n'y a pas à l'heure actuelle de véritable atelier de fouille de données qui soit libre. Ceux-ci intègrent de très nombreux outils d'analyse et de fouille de données, de visualisation de données et des résultats de fouille, de présentation des résultats (création de tableaux de bord) et de liaison avec des bases et entrepôts de données : ces logiciels sont assez onéreux.

On ne s'attaque pas au problème de la gestion de gros volumes de données ; ce que l'on raconte ici s'applique à des volumes de données raisonnables (ordre de grandeur : méga-octets stockés dans de simples fichiers – fichiers Unix : suite de caractères non structurée – ou des bases de données traditionnelles – type sql). Au-delà, des architectures spécialisées (entrepôts de données) sont nécessaires pour cette gestion. Ici et là, on indique comment passer à l'échelle en ce qui concerne les algorithmes de fouille.

Ces notes ne sont pas particulièrement destinées aux étudiants en informatique, au contraire. Il est certain qu'un étudiant en informatique peut les lire ; il est tout aussi certain que la rédaction se veut d'une accessibilité beaucoup plus générale. Les notions d'informatique pure qui sont nécessaires pour une bonne compréhension sont introduites dans le texte ou en annexe.

Pré-requis : une connaissance minimale en mathématiques (algébre, analyse, probabilités, statistiques) est nécessaire ainsi qu'une connaissance minimale en algorithmique. Ensuite, j'essaie d'introduire les notions nécessaires. En fait, le plus important est : avoir envie de comprendre, se poser des questions, essayer de comprendre et expérimenter sur ordinateur. Ce dernier point est essentiel : pour

fouiller les données, l'ordinateur est un outil indispensable. En parallèle, pour comprendre les méthodes, les mathématiques constituent l'outil indispensable.

Remarque

Des remarques ponctuent le texte ; elles sont écrites dans des caractères plus petits que le texte normal. Leur lecture n'est pas indispensable lors d'un premier contact. Elles ont pour but de donner des détails qui permettent de mieux comprendre le texte en justifiant certains points ou en soulevant des questions pour aiguiser le regard du lecteur. Il est clair que pour vraiment comprendre les choses, la lecture de ces remarques est nécessaire.

Chapitre 1

Introduction

Contenu

1.1 Qu'est ce que la fouille de données ?

La fouille de données consiste à rechercher et extraire de l'information (utile et inconnue) de gros volumes de données stockées dans des bases ou des entrepôts de données. Le développement récent de la fouille de données (depuis le début des années 1990) est lié à plusieurs facteurs : une puissance de calcul importante est disponible sur les ordinateurs de bureau ou même à domicile ; le volume des bases de données augmente énormément ; l'accès aux réseaux de taille mondiale, ces réseaux ayant un débit sans cesse croissant, qui rendent le calcul distribué et la distribution d'information sur un réseau d'échelle mondiale viable ; la prise de conscience de l'intérêt commercial pour l'optimisation des processus de fabrication, vente, gestion, logistique, ...

La fouille de données a aujourd'hui une grande importance économique du fait qu'elle permet d'optimiser la gestion des ressources (humaines et matérielles). Elle est utilisée par exemple :

– organisme de crédit : pour décider d'accorder ou non un crédit en fonction du profil du demandeur de crédit, de sa demande, et des expériences

passées de prêts ;
- optimisation du nombre de places dans les avions, hôtels, ... ⇒ surréservation
- organisation des rayonnages dans les supermarchés en regroupant les produits qui sont généralement achetés ensemble (pour que les clients n'oublient pas bêtement d'acheter un produit parce qu'il est situé à l'autre bout du magasin). Par exemple, on extraira une règle du genre : « les clients qui achètent le produit X en fin de semaine, pendant l'été, achètenet généralement également le produit Y » ;
- organisation de campagne de publicité, promotions, ... (ciblage des offres)
- diagnostic médical : « les patients ayant tels et tels symptomes et demeurant dans des agglomérations de plus de 10^4 habitants développent couramment telle pathologie » ;
- analyse du génome
- classification d'objets (astronomie, ...)
- commerce électronique
- analyser les pratiques et stratégies commerciales et leurs impacts sur les ventes
- moteur de recherche sur internet : fouille du *web*
- extraction d'information depuis des textes : fouille de textes
- évolution dans le temps de donnés : fouille de séquences.

Le processus complet de fouille de données comprend plusieurs étapes :

1. collecte des informations et organisation de ces infos dans une base de données ;
2. nettoyage de la base de données : attributs sans valeur, ayant une valeur invalide (bruit), ... ; normalisation ;
3. sélection des attributs utiles ;
4. extraction d'information d'une base de données (*Knowledge Discovery in Databases*, ou KDD) ;
5. visualisation des données : histogramme, camembert, arbre, visualisation 3D et plus généralement, exploration interactive de données ;
6. évaluation des résultats de l'extraction de connaissance.

Dans ce cours, on s'intéressera essentiellement à la phase 4 et un peu aux phases 2, 3 et 5. Les aspects concernant les bases de données seront vues dans le cours du même nom. La phase 4 fait appel à des statistiques et des algorithmes d'intelligence artificielle (apprentissage automatique). L'étude de quelques exemples typiques de ces algorithmes constitue le corps de ce cours, suivie de l'étude de quelques applications réelles. Avant tout, nous discutons de la notion de données.

1.2 Qu'est ce qu'une donnée ?

Cette section a pour objet de fixer un vocabulaire et de rappeler quelques faits importants concernant les attributs des données et ce que représente la valeur d'un attribut. Mais tout d'abord quelques notations que nous retrouverons dans l'ensemble du cours, notations résumées page ix.

1.2.1 Notations

On notera $\mathcal{X}$ un ensemble de données. Chaque donnée est décrite par un ensemble $\mathcal{A}$ d'attributs. Chaque attribut $a \in \mathcal{A}$ prend sa valeur dans un certain ensemble de valeurs $\mathcal{V}_a$. Ainsi, on peut considérer l'ensemble des données x dont les coordonnées balayent toutes les valeurs possibles des attributs : c'est l'espace des données que nous noterons $\mathcal{D}$. Si l'on note a_1, ... a_P les P attributs, $\mathcal{D} = \mathcal{V}_{a_1} \times \mathcal{V}_{a_2} \times ... \times \mathcal{V}_{a_P}$. Toute donnée appartient à cet ensemble et on a $\mathcal{X} \subset \mathcal{D}$.

Il est souvent utile d'avoir une représentation géométrique de l'espace des données ; chaque attribut correspondant à un axe de coordonnées. S'il y a P attributs, l'espace des données est un espace euclidien à P dimensions.

1.2.2 Les différentes natures d'attributs

Une donnée est un enregistrement au sens des bases de données, que l'on nomme aussi « individu » (terminologie issue des statistiques) ou « instance » (terminologie orientée objet en informatique) ou même « tuple » (terminologie base de données) et « point » ou « vecteur » parce que finalement, d'un point de vue abstrait, une donnée est un point dans un espace euclidien ou un vecteur dans un espace vectoriel. Une données est caractérisée par un ensemble de « champs », de « caractères », ou encore d'« attributs » (en suivant les 3 terminologies précédemment évoquées : bases de données, statistiques et conception orientée objet).

attribut qualitatif

Un attribut peut être de nature qualitative ou quantitative en fonction de l'ensemble des valeurs qu'il peut prendre. Un attribut est qualitatif si on ne peut pas en faire une moyenne ; sa valeur est d'un type défini en extension (une couleur, une marque de voiture, ...).

attribut quantitatif

Sinon, l'attribut est de nature quantitative : un entier, un réel, ... ; il peut représenter un salaire, une surface, un nombre d'habitants, ... On peut donc appliquer les opérateurs arithmétiques habituels sur les attributs quantitatifs, ce qui n'est pas le cas des attributs qualitatifs.

Un attribut peut également être un enregistrement (une date par exemple), donc composé lui-même de sous-attributs (jour, mois, année dans le cas d'une date), sur lequel on peut définir les opérateurs arithmétiques habituels : donc quantitatif ne signifie pas forcément « numérique » et, réciproquement,

numérique ne signifie pas forcément quantitatif : un code postal est numérique, mais pas quantitatif.

1.2.3 Les différentes natures de valeur d'attribut

Il n'est pas inutile ici de consacrer quelques lignes à ce qu'est la valeur d'un attribut. Naturellement, cette valeur est censée représenter une certaine mesure d'une quantité dans le monde. Ainsi, quand on dit qu'une couleur est « bleue », cela signifie que nous en avons une certaine perception visuelle qui est associée à ce que, en français, on désigne par le mot « bleu » ; elle aurait pu être verte et on l'aurait appelée verte. Il est *a priori* impossible de comparer bleu et vert ; ce sont deux couleurs, un point c'est tout : la couleur est un attribut nominal. Si on dit qu'aujourd'hui, la température est de 20 ° C et qu'hier, il faisait 18 ° C, on peut dire que la température est plus élevée aujourd'hui qu'hier : cette fois-ci, on peut comparer les deux valeurs d'attributs, cela a un sens. Mais, si on se rappelle ses cours de physique, on sait bien que ce 20 et ce 18 sont aussi arbitraires que les mots « bleu » et « vert » : ces valeurs dépendent, notamment, des unités de mesure : la température est un attribut ordinal. Maintenant, si on dit que le nombre d'enfants de Paul est 2 et que Jacques a 3 enfants, d'une part on peut bien affirmer que Jacques a plus d'enfants que Paul et, de plus, ces deux nombres 2 et 3 ne sont pas arbitraires : le nombre d'enfants est un attribut absolu.

attribut nominal : valeurs non ordonnées

attribut ordinal : valeurs ordonnées

attribut à valeur absolue

Au-delà de la distinction qualitatif/quantitatif, on voit donc apparaître des distinctions plus subtiles entre des attributs dont les valeurs sont arbitraires et incomparables (attribut nominal), des attributs dont la valeur est arbitraire mais que l'on peut comparer (attribut ordinal) et des attributs dont la valeur n'est pas arbitraire (attribut absolu).

opérations sur des attributs de différentes natures

Ces différentes natures entraînent le fait que les opérations que l'on peut faire sur ces attributs ne sont pas les mêmes : on ne peut pas soustraire deux couleurs, on peut soustraire deux températures mais on ne peut pas en faire le rapport[1], alors que l'on peut soustraire et faire le rapport du nombre d'enfants de deux personnes.

Quand on fait des statistiques ou de la fouille de données, on effectue de nombreuses opérations arithmétiques ; hors, selon la nature de l'attribut, ces opérations sont licites ou non... Il importe donc de ne pas faire n'importe quel calcul, d'appliquer n'importe quel algorithme sans prendre garde aux attributs sur lesquels on les effectue.

Par ailleurs, il faut observer un principe d'indépendance du résultat des calculs par rapport aux unités de mesure dans lesquelles sont exprimées les

[1] réfléchissez-y par exemple sur cet exemple : si hier il faisait 20 °C et qu'aujourd'hui il en fait 10, fait-il deux fois plus froid aujourd'hui qu'hier ? et si aujourd'hui il fait 0, il fait combien de fois plus froid qu'hier ? et s'il fait -5 ?

valeurs des attributs : il n'y a aucune raison que l'information extraite d'une base de données change selon qu'une longueur est exprimée en millimètres, mètres ou années-lumières... De cette observation, on pose la règle suivante : on doit toujours s'arranger pour que le résultat d'une analyse ne dépende pas de l'unité de mesure. (On verra une petite illustration de ce principe au chapitre 11.1, section 11.1.1.)

principe d'indépendance par rapport aux unités de mesure

En attendant, si ces quelques lignes vous ont mis en appétit, lisez [Sarle, 1997b].

1.2.4 Le bruit

Il importe de ne pas faire comme si toutes les données ont une valeur connue, et encore moins une valeur valide ; il faut donc gérer des données dont certains attributs ont une valeur inconnue ou invalide ; on dit que les données sont « bruitées ». La simple élimination des données ayant un attribut dont la valeur est inconnue ou invalide pourrait vider complétement la base de données ! On touche le problème de la collecte de données fiables qui est un problème pratique très difficile à résoudre. En fouille de données, il faut faire avec les données dont on dispose sans faire comme si on disposait des valeurs de tous les attributs de tous les individus.

1.2.5 Différentes tâches d'extraction d'information

problème de classification

Dans certains problèmes (problème de classification), chaque donnée est affectée d'une caractéristique, par exemple une couleur. Supposons que l'ensemble des couleurs possibles soit fini et de faible cardinalité. Le problème de classification consiste alors à prédire la couleur d'un point quelconque étant donné un ensemble de points colorés. Géométriquement, cela revient à trouver un moyen de séparer les points les uns des autres, en fonction de leur couleur. S'il n'y a que deux couleurs, un simple (hyper)plan[2] peut suffire à les séparer ; ceux d'une certaine couleur sont d'un côté de l'hyperplan, les autres étant de l'autre côté. Dans ce cas, les points sont linéairement séparables (séparables par un objet géométrique qui ressemble à une droite, un hyperplan pour être plus précis au niveau du vocabulaire). Généralement, des points d'une couleur donnée se trouvent du mauvais côté de l'hyperplan. Cela peut résulter d'erreurs dans la valuation des attributs (on s'est trompé en mesurant certains attributs, ou en attribuant sa couleur à la donnée) ; dans ce cas, les données sont bruitées. Cela peut

[2]Un hyper-espace est un espace ayant plus de 3 dimensions ; dans notre cadre, l'espace des données est un hyper-espace à P dimensions : chaque axe de coordonnées est associé à un attribut. Un hyper-plan est un objet géométrique à $P-1$ dimensions. Dans le cas particulier où $P = 3$, un hyper-plan est donc un objet en 2 dimensions, soit ce que l'on dénomme habituellement un plan. La notion d'hyper-plan généralise celle de plan à un espace de dimension quelconque.

aussi être intrinsèque aux données qui ne peuvent pas être séparées linéairement. Il faut alors chercher à les séparer avec un objet non hyperplanaire. Le problème de classification sera défini précisément au chap. 2. On verra ensuite diverses approches à ce problème :

- construction d'un modèle arborescent permettant de prédire la classe d'une donnée (*cf.* chap. 3) ou d'un modèle exprimé sous forme de règles (*cf.* chap. 6). Dans ces deux cas, le modèle obtenu est interprétable par un humain ;
- estimation directe de la classe d'une donnée en fonction des exemples : une approche probabiliste au chap. 4 et une approche par cas au chap. 5. Dans ces deux cas, il n'y a pas d'apprentissage, pas de construction de modèle ;
- construction d'un modèle non interprétable par un humain : les réseaux de neurones au chap. 7 qui permettent de s'attaquer à des problèmes dans lesquels les données sont décrites par de très nombreux attributs (des images par exemple), ou les machines à vecteurs supports au chap. 8. Dans ces deux cas, le modèle construit n'est pas interprétable par un humain ; il est utilisé par l'algorithme pour estimer la classe de nouvelles données.

Enfin, le chap. 9 terminera ce survol des méthodes de résolution du problème de classification en ouvrant quelques pistes.

problème de régression

Pour revenir à notre exemple des points colorés, dans d'autres cas, l'ensemble des couleurs possibles est infini. On est alors dans un problème de régression. On en parlera brièvement au chap. 13.

problème de segmentation

Dans d'autres problèmes (problème de segmentation), on dispose d'un ensemble de points est la tâche consiste à repérer des groupes de points qui se ressemblent. On verra alors deux ensembles de techniques : les algorithmes qui proposent une segmentation des données au chap. 10 puis des algorithmes qui construisent une représentation géométrique interprétable par un humain (dans un plan) d'un ensemble de données au chap. 11. C'est alors un humain qui détecte les groupes et effectue la segmentation : l'algorithme de projection est une aide pour l'humain ; on effectue de la fouille de données assistée par ordinateur.

problème de recherche de règles d'association

Plus généralement, on peut disposer d'un ensemble de données pour lesquel on veut détecter des relations entre la valeur de leurs attributs. Il s'agît alors de chercher des règles d'association. Cela sera discuté au chap. 12.

problème d'estimation

On peut aussi disposer d'un ensemble de points et d'autres points dont tous les attributs ne sont pas valués. On doit alors déterminer la valeur de ces attributs manquants. C'est alors un problème d'estimation.

problème de prédiction

On peut aussi disposer d'un ensemble de données qui décrit l'évolution au cours du temps d'une certaine entité. On veut alors prédire le comportement futur à partir du comportement observé : c'est un problème de prédiction.

corrélation

Notons que dans tous ces problèmes, la notion de « corrélation » est omniprésente : l'extraction d'information repose sur la recherche de corrélations entre des données. Ces corrélations peuvent être linéaires : c'est le cas simple. En général, on doit chercher des corrélations non linéaires.

transformation de l'espace des données

Enfin, l'espace de données dans lequel celles-ci nous sont fournies initialement n'est pas forcément le plus adéquat. Il s'agît alors de préparer les données pour en faciliter l'extraction d'information. Cette préparation peut consister en la diminution du nombre d'attributs, à divers traitements sur les valeurs d'attributs (lissage, ...), au changement d'espace de représentation des données (projection dans un espace où les relations sont plus faciles à détecter, projection dans un sous-espace plus petit, ou représentation dans un espace obtenu par combinaison des attributs initiaux – création de *features*). La représentation des données dans un espace inadéquat entraîne lui-aussi du bruit, différent de celui rencontré plus haut. On discutera ces questions au chap. 14.

1.3 Références

Pour une introduction assez complète à la fouille de données illustrée par l'utilisation du logiciel libre `weka`, on lira [Witten and Franck, 2000]. Beaucoup de problématiques sont traitées si ce n'est le problème des gros volumes de données.

Pour une bonne introduction, qui commence à dater, sur les algorithmes d'apprentissage automatique lesquels sont largement utilisés en fouille de données, on lira [Mitchell, 1997].

Un polycopié pédagogique sur la classification de données : [Denis and Gilleron, 2000] ; un autre dans le même genre où l'on parle plus des aspects données : [Gilleron and Tommasi, 2000].

Un bon polycopié sur la fouille de données où la gestion de gros volumes de données est traitée explicitement : [Ullman, 2000]. Ce document est accompagné d'un certain nombre de papiers qui sont faciles à trouver sur le web. La lecture n'est pas toujours aisée ; ce n'est pas une lecture pour un débutant.

Un livre qui fait le tour de la fouille de données de manière assez précise : [Han and Kamber, 2001].

Chapitre 2

La classification

Contenu

2.1 Introduction

On commence par définir le problème de classification. Ce problème est parfois aussi dénommé « problème de classification supervisé » ou de « reconnaissance des formes »[1]. Attention cependant, en statistiques, un problème dit de classification est ce que nous dénommons un « problème de segmentation » (*cf.* chap. 10). En statistiques, ce que dénommons ici un problème de classification est souvent dénommé un « problème de décision ».

Définition 1 *On dispose d'un ensemble $\mathcal{X}$ de N données étiquetées. Chaque donnée x_i est caractérisée par P attributs et par sa classe $y_i \in \mathcal{Y}$. Dans un problème de classification, la classe prend sa valeur parmi un ensemble fini. Le problème consiste alors, en s'appuyant sur l'ensemble d'exemples $\mathcal{X} = \{(x_i, y_i)_{i\in\{1,\dots,N\}}\}$, à prédire la classe de toute nouvelle donnée $x \in \mathcal{D}$.*

On parle de classification binaire quand le nombre de classes $|\mathcal{Y}|$ est 2 ; il peut naturellement être quelconque. Dans tous les cas, il s'agît d'un attribut qualitatif pouvant prendre un nombre fini de valeurs.

Dans l'absolu, une donnée peut appartenir à plusieurs classes : c'est alors un problème multi-classes. Ici, on considère que chaque donnée appartient à une et une seule classe.

[1] *pattern recognition* en anglais

Concernant le vocabulaire, on utilise le mot « étiquette » comme synonyme de « classe ».

Définition 2 *Un exemple est une donnée pour laquelle on dispose de sa classe.*

On utilise donc un ensemble d'exemples classés pour prédire la classe de nouvelles données ; c'est une tâche d'« apprentissage à partir d'exemples », ou « apprentissage supervisé ».

hypothèse de représentativité des exemples

Une hypothèse est implicitement admise : les exemples dont on dispose sont représentatifs de l'ensemble des données. Notons qu'en général, on est incapable de déterminer si cette hypothèse est vérifiée pour un jeu d'exemples donné.

Notons que typiquement, dans un problème de classification, le nombre d'exemples (les données pour lesquelles on dispose de leur classe) est petit : l'étiquetage des données est en général effectué à la main, ce qui entraîne un coût élevé de cet étiquetage, donc de l'obtention d'un nombre important d'exemples. Nous sommes alors dans un problème de statistique où l'échantillon est petit.

Définition 3 *Un « classeur » est une procédure (un algorithme) qui, à partir d'un ensemble d'exemples, produit une prédiction de la classe de toute donnée.*

D'une manière générale, un classeur procède par « induction » : à partir d'exemples (donc de cas particuliers), on construit une connaissance plus générale[2]. La notion d'induction de connaissances implique la notion de « généralisation » de la connaissance : à partir de connaissances éparses, les exemples, on induit une connaissance plus générale. Naturellement, même si l'on suppose que la classe des étiquettes n'est pas erronée, il y a un risque d'erreur lors de la généralisation ; ce risque est quantifié par la notion de « taux d'échec », ou d'« erreur en généralisation », qui sera définie précisément plus loin (*cf.* chap. 3, sec. 3.7).

Quand on tente d'induire de la connaissance, il faut déterminer, au moins implicitement, la pertinence des attributs pour la prédiction de l'étiquette d'une donnée quelconque : c'est cela « généraliser ». D'une manière ou d'une part, explicitement ou pas, généraliser implique de construire un modèle des données. La taille de ce modèle est un paramètre important. À l'extrême, il est aussi gros que l'ensemble des exemples : dans ce cas, on n'a rien appris, rien généralisé et on est incapable d'effectuer une prédiction fiable pour une donnée qui ne se trouve pas dans l'ensemble des exemples : on a sur-appris. À un autre extrême, on peut n'avoir appris que les proportions des différentes étiquettes dans l'espace des données : par exemple, 1/3 des données sont bleues et les autres sont rouges, cela sans lien avec la description des données ; prédire la classe revient alors à tirer la classe au hasard avec ces proportions un tiers/deux tiers : on a pris trop

[2] dans le cas de la déduction, on part d'une connaissance générale pour obtenir une information sur un cas particulier.

de recul et on n'est plus capable d'effectuer une prédiction fiable pour une donnée particulière. Entre ces deux extrêmes, il y a un juste milieu où le modèle a pris du recul par rapport aux exemples, a su extraire les informations pertinentes du jeu d'exemples pour déterminer l'étiquette de n'importe quelle donnée avec une probabilité élevée de succès ; le modèle est alors de taille modérée et la probabilité d'erreur de ce modèle est la plus faible que l'on puisse obtenir : on a un modèle optimisant le rapport qualité/prix, *i.e.* probabilité d'effectuer une prédiction correcte/coût du modèle. La recherche d'un modèle optimisant ce rapport est l'objectif de l'apprentissage automatique, lequel est l'un des outils indispensables pour la réaliser de la fouille de données.

Rappelons[3] que parmi les modèles, on distingue ceux qui peuvent être interprétés par un humain (arbre de décision par exemple) et ceux qui ne le peuvent pas (réseau de neurones par exemple).

Notons aussi que l'on peut voir la construction d'un modèle comme une manière de compresser l'information contenue dans un jeu d'exemples.

On distingue deux grands types de classeurs :
- ceux qui utilisent directement les exemples pour prédire la classe d'une donnée ;
- ceux pour lesquels on a d'abord construit un modèle et qui, ensuite, utilisent ce modèle pour effectuer leur prédiction.

Il ne faut quand même pas tomber dans une vision trop simpliste : il existe naturellement une gradation entre ces deux extrêmes.

Plusieurs problèmes sont à résoudre :
- méthode d'induction du classeur ?
- comment utiliser le classeur obtenu ?
- comment évaluer la qualité du classeur obtenu : taux d'erreur (ou de succès) ?
- comment traiter les attributs manquants dans le jeu d'apprentissage ?
- comment traiter les attributs manquants dans une donnée à classer ?
- estimer la tolérance au bruit : le bruit concerne ici la valeur des attributs de l'exemple avec lequel on construit le classeur.

2.2 Une approche naïve

Idée : utiliser une table de *look-up* et y stocker tous les exemples.

Pour prédire la classe d'une instance : rechercher cette instance dans la table. Si elle s'y trouve, renvoyer sa classe ; sinon, tirer sa classe au hasard.

[3] on l'a déjà dit au chap. 1, sec. 1.2.5.

Évaluation du classeur : 100 % de réussite sur les données de l'ensemble d'exemples ; 50 % pour les autres données[4].

Discussion : l'induction de classeur est une tâche complexe : si le classeur obtenu n'a pas de performance supérieure à une procédure aléatoire, il n'est pas intéressant. Donc, ce classeur n'est pas intéressant.

Pour être intéressant, un classeur doit pouvoir « généraliser » à partir des exemples qui lui ont été fournis.

Remarque : le classeur proposé plus haut fait de l'apprentissage par cœur.

Dans la suite, on va s'efforcer de construire des classeurs réellement intéressant. Ainsi, on va présenter plusieurs algorithmes d'induction de classeurs. Ce sont les plus connus et les plus répandus ; d'autres existent : en fait, seule l'imagination nous limite !

2.3 Exercice

Exercice 1 *Avant de lire la suite, essayer d'imaginer des méthodes de classification.*

[4] en supposant que les deux valeurs de classe sont équi-réparties dans l'espace des données : dans ce cas, on tire la classe au hasard.

Chapitre 3

Classification par arbres de décision

Contenu

Qu'est ce qu'un arbre de décision ?

On se donne un ensemble $\mathcal{X}$ de N exemples notés x_i dont les P attributs sont quantitatifs ou qualitatifs. Chaque exemple x est étiqueté, c'est-à-dire qu'il lui est associée une « classe » ou un « attribut cible » que l'on note $y \in \mathcal{Y}$.

À partir de ces exemples, on construit un arbre dit « de décision » tel que :
- chaque nœud correspond à un test sur la valeur d'un ou plusieurs attributs ;
- chaque branche partant d'un nœud correspond à une ou plusieurs valeurs de ce test ;
- à chaque feuille est associée une valeur de l'attribut cible.

Utilisation d'un arbre de décision

L'arbre de décision peut être ensuite exploité de différentes manières :

1. en y classant de nouvelles données ;
2. en faisant de l'estimation d'attribut (*cf.* sec. 3.3) ;
3. en en extrayant un jeu de règles de classification concernant l'attribut cible (pour cela, *cf.* chap. 6) ;
4. en interprétant la pertinence des attributs (*cf.* sec. 3.2.2).

Un exemple très simple

Dans ce qui suit et dans les chapitres qui suivent, on illustrera notre propos avec un exemple de jeu de données : un ensemble de jours (un jour = un exemple), chacun caractérisé par un numéro et ses conditions météorologiques (température, humidité de l'air, force du vent, ciel), l'attribut cible étant « jouer au tennis ? », dont les valeurs possibles sont oui et non. Une fois l'arbre de décision construit, on pourra classer une nouvelle donnée pour savoir si on joue ou non ce jour-là (*cf.* table 3.1).

Notons que si la classe ne prend que deux valeurs (comme dans l'exemple « jouer au tennis ? »), on parle de classification binaire. Par convention, une donnée dont la classe est « oui » est qualifiée de positive, alors qu'une donnée dont la classe est « non » est qualifiée de négative. Plus généralement, les classes peuvent signifier n'importe quoi : rouge ou vert, grand ou petit, ... Dans chaque cas, on peut définir adéquatement ce que l'on entend par positif et négatif.

La figure 3.1 présente un arbre de décision pour l'exemple. (Cet arbre est un arbre de décision possible pour ce jeu de données ; nous ne prétendons pas qu'il résulte de l'application des algorithmes qui vont être décrits plus loin, ni même qu'il est d'un quelconque intérêt, sinon pédagogique.) La donnée 1 de la table 3.1 est prédite comme « oui » car son attribut « humidité » vaut « Élevée », donc on suit la branche droite partant de la racine et on atteint le nœud « Vent », et l'attribut « Vent » vaut « Faible », ce qui nous fait suivre la branche gauche de ce nœud et atteindre une feuille étiquetée « oui » (celle du milieu sur la figure).

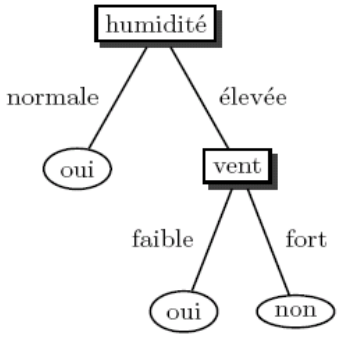


FIG. 3.1 – Un exemple d'arbre de décision sur le jeu de données « jouer au tennis ? ».

De même, la donnée 2 sera prédite comme de classe « non » et la donnée 5 sera prédite de classe « oui » en s'aiguillant directement depuis la racine vers la feuille « oui » à la gauche de la figure.

couverture d'une donnée par un nœud ou une feuille d'un arbre de décision

Concernant le vocabulaire, on dira qu'une donnée est couverte par un nœud (resp., une feuille) si sa description amène la donnée de la racine jusque ce nœud (resp., cette feuille). Ainsi, toutes les données sont couvertes par la racine (puisque l'on a encore testé aucun attribut) ; toutes les données dont l'attribut « Humidité » vaut « Élevée » sont couvertes par le nœud testant l'attribut « Vent »...

3.1 Construction d'un arbre de décision

Précisons tout de suite que la construction d'un arbre de décision optimal est un problème $\mathcal{NP}$-complet (*cf.* annexe C), optimal au sens où il minimise le nombre d'erreur de classification. Aussi, il ne faut pas avoir l'espoir de construire l'arbre de décision optimal pour un jeu d'exemples donné. On va se contenter d'en construire un qui soit correct. Plusieurs algorithmes ont été proposés, notamment CART dans les années 1980 par Breiman et al. [1984]. On décrit ici l'algorithme ID3 de R. Quinlan proposé en 1986 qui a été raffiné par la suite (C4.5 puis C5) (*cf.* Quinlan [1993]). On constate expérimentalement que ces algorithmes sont très performants : ils construisent rapidement des arbres de décision qui prédisent avec une assez grande fiabilité la classe de nouvelles données.

ID3 ne prend en compte que des attributs nominaux. Son successeur, C4.5, prend également en charge des attributs quantitatifs. Dans la suite, on suppose donc pour l'instant que tous les attributs sont nominaux. On suppose que la

Principe de construction d'un arbre de décision par l'algorithme ID3

classe est binaire.

Les tests placés dans un nœud par l'algorithme ID3 concernent exclusivement le test de la valeur d'un et seul attribut (comme l'exemple d'arbre de décision rencontré plus haut, *cf.* fig. 3.1). ID3 fonctionne récursivement : il détermine un attribut à placer en racine de l'arbre. Cette racine possède autant de branches que cet attribut prend de valeurs. À chaque branche est associé un ensemble d'exemples dont l'attribut prend la valeur qui étiquette cette branche ; on accroche alors au bout de cette branche l'arbre de décision construit sur ce sous-ensemble des exemples et en considérant tous les attributs excepté celui qui vient d'être mis à la racine. Par cette procédure, l'ensemble des exemples ainsi que l'ensemble des attributs diminuent petit à petit au long de la descente dans l'arbre.

Ayant l'idée de l'algorithme, il reste à résoudre une question centrale : quel attribut placer en racine ? Une fois cette question résolue, on itérera le raisonnement pour les sous-arbres.

Pour cela, nous avons besoin de définir quelques notions. Commençons par l'entropie introduite initialement par Shannon [1948], notion héritée de la thermodynamique où l'entropie d'un système est d'autant plus grande qu'il est désordonné, hétérogène.

Entropie

Définition 4 *Soit un ensemble $\mathcal{X}$ d'exemples dont une proportion p_+ sont positifs et une proportion p_- sont négatifs. (Bien entendu, $p_+ + p_- = 1$.) L'entropie de $\mathcal{X}$ est :*

$$H(\mathcal{X}) = -p_+ \log_2 p_+ - p_- \log_2 p_- \tag{3.1}$$

Remarques :

1. $0 \leq H(\mathcal{X}) \leq 1$;
2. si $p_+ = 0$ ou $p_- = 0$, alors $H(\mathcal{X}) = 0$: ainsi, si tous exemples sont soit tous positifs, soit tous négatifs, l'entropie de la population est nulle ;
3. si $p_+ = p_- = 0.5$, alors $H(\mathcal{X}) = 1$: ainsi, s'il y a autant de positifs que de négatifs, l'entropie est maximale.

La figure 3.2 représente la fonction entropie en fonction de p_+.

En résumé, l'entropie mesure l'hétérogénéité d'une population du point de vue de la classe de ses membres. Elle se mesure en bits. C'est en fait une mesure de la quantité d'information qu'il y a dans $\mathcal{X}$ du point de vue de la classe de ses éléments. Cette quantité est liée à la probabilité d'existence de cette combinaison de répartition de la classe parmi ces éléments si la classe résulte d'un tirage aléatoire : il est très improbable que tous les exemples soient de la même classe ; il est au contraire très probable que les exemples soient à peu près équitablement répartis entre les différentes classes. Une autre interprétation de

la notion d'entropie concerne l'incertitude du résultat du tirage au sort dans une population : quand les deux classes sont équitablement réparties dans la population, c'est là que l'issue du tirage d'un élément au hasard est la plus incertaine, l'entropie est alors maximale.

La définition précédente de l'entropie se généralise aisément à un attribut pouvant prendre plus de deux valeurs distinctes :

Définition 5 *Pour une classe prenant n valeurs distinctes (numérotées de 1 à n), notons $p_{i\in[1,n]}$ la proportion d'exemples dont la valeur de cet attribut est i dans l'ensemble d'exemples considéré $\mathcal{X}$. L'entropie de l'ensemble d'exemples $\mathcal{X}$ est :*

$$H(\mathcal{X}) = -\sum_{i=1}^{i=n} p_i \log_2 p_i \tag{3.2}$$

Intuitivement, avoir des sous-sensembles dont l'entropie est minimale est intéressant : cela signifie que l'attribut placer à la racine discrimine les exemples en fonction de leur classe. Il est naturel de sommer ces entropies en les pondérant en fonction de la proportion d'exemples dans chacun des sous-ensembles.

Intuitivement toujours, voulant mettre un attribut discriminant au mieux, on peut utiliser la différence entre l'entropie de l'ensemble d'exemples initial (utilisé pour déterminer la racine) et cette somme pondérée pour trouver l'attribut le plus intéressant à placer dans la racine. L'attribut qui maximise cette différence est l'attribut qui discrimine le mieux les exemples en fonction de leur classe.

Gain d'information

Définition 6 *Soit une population d'exemples $\mathcal{X}$. Le gain d'information de $\mathcal{X}$ par rapport à un attribut a_j donné est la réduction d'entropie causée par la partition de $\mathcal{X}$ selon a_j :*

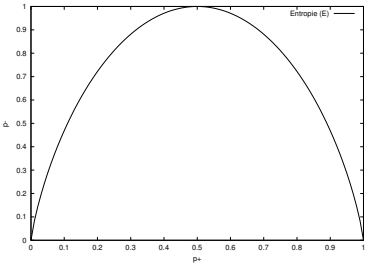


FIG. 3.2 – Graphe de l'entropie d'une population : en abscisse, la proportion d'exemples positifs dans la population p_+. En ordonnée, l'entropie de la population.

$$Gain\ (\mathcal{X}, a_j) = H(\mathcal{X}) - \sum_{v \in\ valeurs(a_j)} \frac{|\mathcal{X}_{a_j=v}|}{|\mathcal{X}|}\ H(\mathcal{X}_{a_j=v}) \tag{3.3}$$

où $\mathcal{X}_{a_j=v} \subset \mathcal{X}$ est l'ensemble des exemples dont l'attribut considéré a_j prend la valeur v, et la notation $|\mathcal{X}|$ indique le cardinal de l'ensemble $\mathcal{X}$.

À première vue, cette formule peut sembler compliquée. En fait, chacun des termes de la somme correspond à l'une des branches du nœud si l'attribut a_j y est testé, pondéré par la proportion d'exemples qui sont couverts par cette branche (couvert, *i.e.* les exemples dont l'attribut testé dans le nœud a la valeur correspondant à cette branche). La somme constitue donc la somme des entropies pour l'ensemble des branches du nœud associé à l'attribut a_j. Le gain est donc la différence entre l'entropie de l'ensemble des exemples et la somme des entropies de chacune des branches suite au test de cet attribut dans le nœud. Plus ce gain est grand, plus le partitionnement de la population selon cet attribut entraîne des sous-ensembles homogènes.

Exemple : supposons que $\mathcal{X}$ soit constitué de 14 exemples, 9 positifs et 5 négatifs. Parmi ces exemples, 6 positifs et 2 négatifs prennent la valeur « oui » pour l'attribut a, tandis que les autres exemples prennent la valeur « non » pour cet attribut.

$$\text{Gain}\ (\mathcal{X}, a) = H(\mathcal{X}) - \sum_{v \in \{\text{oui, non}\}} \frac{|\mathcal{X}_{a=v}|}{|\mathcal{X}|}\ H(\mathcal{X}_{a=v})$$

$$\text{Gain}\ (\mathcal{X}, a) = H(\mathcal{X}) - \frac{8}{14}\ H(\mathcal{X}_{a=\text{oui}}) - \frac{6}{14}\ H(\mathcal{X}_{a=\text{non}})$$

On a :

$$H(\mathcal{X}_{\text{oui}}) = -(\frac{6}{8}\ln_2\frac{6}{8} + \frac{2}{8}\ln_2\frac{2}{8}) \approx 0.811$$

et

$$H(\mathcal{X}_{\text{non}}) = -(\frac{3}{6}\ln_2\frac{3}{6} + \frac{3}{6}\ln_2\frac{3}{6}) = 1.0$$

D'où :

$$\text{Gain}\ (\mathcal{X}, a) \approx 0.940 - \frac{8}{14}\ 0.811 - \frac{6}{14}\ 1.00$$

$$\text{Gain}\ (\mathcal{X}, a) \approx 0.048$$

Le gain d'information en classant le jeu de d'exemples $\mathcal{X}$ par rapport à a est donc : 0.048.

Le principe de l'algorithme ID3 pour déterminer l'attribut à placer à la racine de l'arbre de décision peut maintenant être exprimé : rechercher l'attribut qui possède le gain d'information maximum, le placer en racine, et itérer pour chaque fils, c'est-à-dire pour chaque valeur de l'attribut. Cela étant dit, on peut donner l'algorithme ID3 : *cf.* l'algorithme 1.

Algorithme 1 ID3

Nécessite: 2 paramètres : l'ensemble d'exemples $\mathcal{X}$, l'ensemble d'attributs $\mathcal{A} = \{a_{j\in\{1,\dots p\}}\}$ où p est le nombre d'attributs restants à considérer

Créer un nœud racine

si tous les élements de $\mathcal{X}$ sont positifs **alors**

racine. étiquette $\leftarrow \oplus$

return racine

fin si

si tous les éléments de $\mathcal{X}$ sont négatifs **alors**

racine. étiquette $\leftarrow \ominus$

return racine

fin si

si $\mathcal{A} = \emptyset$ **alors**

racine. étiquette $\leftarrow$ valeur la plus présente de la classe parmi les $\mathcal{X}$

return racine

fin si

$a^* \leftarrow \arg\max_{a\in\mathcal{A}} \text{gain}(\mathcal{X}, a)$

racine. étiquette $\leftarrow a^*$

pour toutes les valeurs v_i de a^* **faire**

ajouter une branche à racine correspondant à la valeur v_i

former $\mathcal{X}_{a^*=v\,i} \subset \mathcal{X}$ dont l'attribut a^* vaut v_i

si $\mathcal{X}_{a^*=v\,i} = \emptyset$ **alors**

à l'extrémité de cette branche, mettre une feuille étiquetée avec la valeur la plus présente de la classe parmi les $\mathcal{X}$

sinon

à l'extrémité de cette branche, mettre ID3 ($\mathcal{X}_{a^*=v\,i}$, $\mathcal{A} - \{a^*\}$)

fin si

fin pour

return racine

TAB. 3.1 – Jeu de données « jouer au tennis ? »

Jour	Ciel	Température	Humidité	Vent	Jouer au tennis ?
1	Ensoleillé	Chaude	Élevée	Faible	Non
2	Ensoleillé	Chaude	Élevée	Fort	Non
3	Couvert	Chaude	Élevée	Faible	Oui
4	Pluie	Tiède	Élevée	Faible	Oui
5	Pluie	Fraîche	Normale	Faible	Oui
6	Pluie	Fraîche	Normale	Fort	Non
7	Couvert	Fraîche	Normale	Fort	Oui
8	Ensoleillé	Tiède	Élevée	Faible	Non
9	Ensoleillé	Fraîche	Normale	Faible	Oui
10	Pluie	Tiède	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui
12	Couvert	Tiède	Élevée	Fort	Oui
13	Couvert	Chaud	Normale	Faible	Oui
14	Pluie	Tiède	Élevée	Fort	Non

3.2 Exemple de construction d'un arbre de décision par ID3

3.2.1 Construction

Sur un exemple, on montre la construction d'un arbre de décision par ID3. On considère l'ensemble exemplees de la table 3.1. L'attribut cible est donc « Jouer au tennis ? ». Déroulons ID3 :

1. création d'une racine
2. les exemples n'étant ni tous positifs, ni tous négatifs, l'ensemble des attributs n'étant pas vide, on calcule les gains d'information pour chaque attribut :

Attribut	Gain
Ciel	0.246
Humidité	0.151
Vent	0.048
Température	0.029

   Donc, la racine de l'arbre de décision testera l'attribut « Ciel » ;
3. l'attribut « Ciel » peut prendre trois valeurs. Pour « Ensoleillé », ID3 est appelé récursivement avec 5 exemples : $\{x_1, x_2, x_8, x_9, x_{11}\}$). Les gains d'information des 3 attributs restants sont alors :

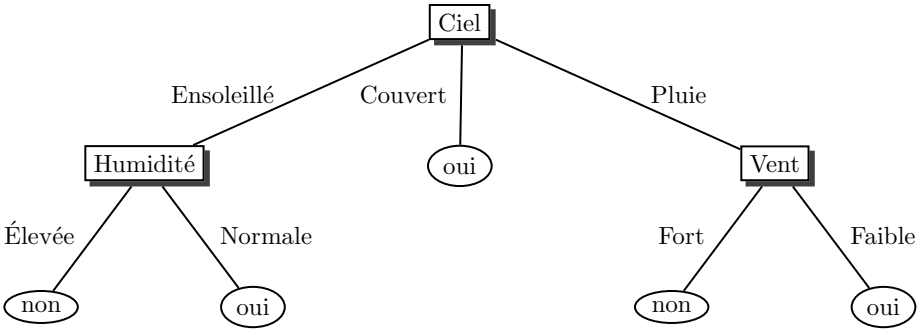


FIG. 3.3 – Arbre de décision obtenu pour l'exemple du texte « jouer au tennis ? ».

Attribut	Gain
Humidité	0.970
Vent	0.570
Température	0.019

(Rappel : le gain d'info est ici calculé uniquement sur les exemples dont l'attribut « Ciel » vaut « Ensoleillé », soit $\mathcal{X}_{\text{Ciel=Ensoleillé}}$ avec notre notation.) L'attribut « Humidité » sera donc choisi ; on continue la construction de l'arbre de décision récursivement ;

4. pour la branche « Pluie » partant de la racine, ID3 est appelé récursivement avec 5 exemples : $\{x_4, x_5, x_6, x_{10}, x_{14}\}$; on continue la construction de l'arbre de décision récursivement ;

5. pour la branche « Couvert » partant de la racine, ID3 est appelé récursivement avec 4 exemples : $\{x_3, x_7, x_{12}, x_{13}\}$; dans ce dernier cas, tous les exemples sont positifs : on affecte donc tout de suite la classe « oui » à cette feuille.

3.2.2 Interprétation de l'arbre

Analyse de l'arbre de décision ; pretinence des attributs

Remarquons que l'arbre de décision qui vient d'être construit nous donne des informations sur la pertinence des attributs vis-à-vis de la classe. Ainsi, l'attribut « Température » n'étant pas utilisé dans l'arbre ; ceci indique que cet attribut n'est pas pertinent pour déterminer la classe. En outre, si l'attribut « Ciel » vaut « Ensoleillé », l'attribut « Vent » n'est pas pertinent ; si l'attribut « Ciel » vaut « Pluie », c'est l'attribut « Humidité » qui ne l'est pas.

3.3 Utilisation de l'arbre de décision pour classer une donnée

Algorithme de détermination de la classe d'une donnée dont tous les attributs sont valués : *cf.* algorithme 2.

Algorithme 2 Classification d'une donnée dans un arbre de décision

Nécessite: 2 paramètres : arbre de décision AD, exemple x
// on utilise une variable nc (nœud courant) pour parcourir l'arbre
nc ← racine (AD)
tant-que nc ≠ feuille **faire**
en fonction de l'attribut testé dans nc et de sa valeur dans x, suivre l'une des branches de nc. Le nœud atteint devient nc
fin tant-que
retourner étiquette(nc)

Par exemple, on peut prédire la classe pour les données suivantes :
- (Ensoleillé, Fraîche, Élevée, Fort) est classée comme « non » ;
- (Ensoleillé, Fraîche, Normale, Fort) est classée comme « oui » ;
- (Pluie, Chaude, Normale, Faible) est classée comme « oui » ;
- (Pluie, Fraîche, Élevée, Fort) est classée comme « oui ».

3.4 Les attributs numériques

C4.5 et attributs numériques

Successeur d'ID3, C4.5 prend en compte les attributs numériques, c'est-à-dire, des attributs dont l'arité est élevée (voire infinie). Hormis cela et quelques détails décrits plus loin, la construction d'un arbre de décision par C4.5 est identique dans son principe à la construction par ID3.

Dans le cas de C4.5, un nœud de l'arbre de décision peut contenir un test du fait que la valeur d'un attribut numérique est inférieure à un certain seuil : cela correspond donc à un nouveau pseudo-attribut binaire. C4.5 ne dispose pas d'autres possibilités de prendre en compte ce type d'attributs. Nous ne décrivons donc que cette possibilité.

Dans ce qui suit, nous illustrons notre propos sur le jeu de données « jouer au tennis ? » dans lequel les attributs « Température » et « Humidité » ont été numérisés (*cf.* table 3.2).

3.4.1 Test d'un attribut numérique

On explique le principe sur un exemple simple basé sur le jeu de données « jouer au tennis ? ». Considérons les exemples dont l'attribut « Ciel » vaut

TAB. 3.2 – Jeu de données « jouer au tennis ? » avec des attributs quantitatifs et nominaux.

Jour	Ciel	Température	Humidité	Vent	Jouer au tennis ?
1	Ensoleillé	27.5	85	Faible	Non
2	Ensoleillé	25	90	Fort	Non
3	Couvert	26.5	86	Faible	Oui
4	Pluie	20	96	Faible	Oui
5	Pluie	19	80	Faible	Oui
6	Pluie	17.5	70	Fort	Non
7	Couvert	17	65	Fort	Oui
8	Ensoleillé	21	95	Faible	Non
9	Ensoleillé	19.5	70	Faible	Oui
10	Pluie	22.5	80	Faible	Oui
11	Ensoleillé	22.5	70	Fort	Oui
12	Couvert	21	90	Fort	Oui
13	Couvert	25.5	75	Faible	Oui
14	Pluie	20.5	91	Fort	Non

« Ensoleillé », soit l'ensemble $\mathcal{X}_{\text{Ciel=Ensoleillé}}$ d'exemples ayant un seul attribut numérique comme suit :

Jour	Température	« jouer au tennis »
1	27.5	non
2	25	non
8	21	non
9	19.5	oui
11	22.5	oui

On commence par trier les exemples sur la valeur de leur attribut numérique. À chaque attribut, on associe le numéro de son exemple associé ainsi que la valeur de l'attribut cible :

Température	19.5	21	22.5	25	27.5
Jour	9	8	11	2	1
« jouer au tennis ? »	oui	non	oui	non	non

On détermine le seuil s pour partitionner cet ensemble d'exemples. C4.5 utilise les règles suivantes :

1. ne pas séparer deux exemples successifs ayant la même classe ; donc, on ne peut couper qu'entre les exemples x_9 et x_8, x_8 et x_{11}, x_{11} et x_2 ;

2. si on coupe entre deux valeurs v et w ($v < w$) de l'attribut, le seuil s est fixé à v (on aurait pu aussi utiliser $\frac{v+w}{2}$) ;
3. choisir s de telle manière que le gain d'information soit maximal.

Remarque : une fois le seuil s fixé et le nœud créé, chaque sous-arbre pourra à nouveau tester la valeur de cet attribut ; en effet, contrairement au cas des attributs qualitatifs qui produisent des nœuds ayant autant de branches que l'attribut prend de valeurs différentes, l'ensemble des valeurs prises par un attribut numérique est coupé en deux : chaque partie peut donc encore être raffinée jusqu'à ne contenir que des exemples ayant même valeur cible.

Application : l'entropie de l'ensemble d'exemples est :

$$H(\mathcal{X}) = -(\frac{2}{5}\ln_2\frac{2}{5} + \frac{3}{5}\ln_2\frac{3}{5}) \approx 0.971$$

Pour $s = 21$, le gain d'information est :

$$\text{Gain}(\mathcal{X}, \text{Température}, s = 21) = H(\mathcal{X}) - (\frac{1}{5}H(\mathcal{X}_{\text{Température}<21}) + \frac{4}{5}H(\mathcal{X}_{\text{Température}\geq 21}))$$

avec :

$$H(\mathcal{X}_{\text{Température}<21}) = -(1\ln_2 1 + 0\ln_2 0) = 0$$

et

$$H(\mathcal{X}_{\text{Température}\geq 21}) = -(\frac{1}{4}\ln_2\frac{1}{4} + \frac{3}{4}\ln_2\frac{3}{4}) \approx 0.608$$

soit

$$\text{Gain}(\mathcal{X}, s = 21) \approx 0.971 - (\frac{1}{5}\times 0 + \frac{4}{5}\times 0.608) \approx 0.485$$

De la même manière, en fonction du seuil, le gain d'information est alors :

seuil	Gain($\mathcal{X}$, Température, s)
$s = 21$	0.485
$s = 22.5$	0.02
$s = 25$	0.42

Nous avons montré sur un exemple comment choisir le seuil pour un attribut donné. Naturellement, dans son action, C4.5 effectue ce traitement pour chaque attribut quantitatif et détermine donc pour chacun un seuil produisant un gain d'information maximal. Le gain d'information associé à chacun des attributs quantitatifs est celui pour lequel le seuil entraîne un maximum. Finalement, l'attribut choisi (parmi les quantitatifs et les nominaux pour lesquels le principe est identique ID3) est celui qui produit un gain d'information maximal.

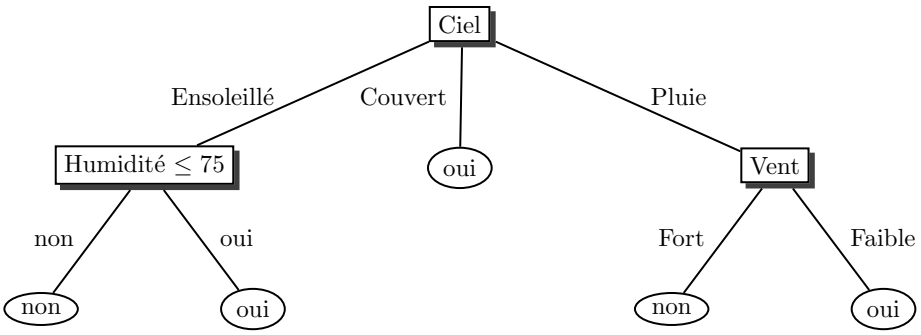


FIG. 3.4 – Arbre de décision obtenu pour des attributs numériques sur le jeu de données « jouer au tennis ? ».

3.4.2 Rapport de gain

En présence d'attribut numérique ou d'attribut d'arité élevée, ceux-ci sont automatiquement favorisés pour être sélectionnés comme test dans les nœuds. Pour contrecarrer cet effet, C4.5 utilise le rapport de gain au lieu du gain d'information pour déterminer l'attribut à utiliser dans un nœud.

Le rapport de gain est défini par :

$$\text{Rapport de gain}(\mathcal{X}, a) = \frac{\text{Gain}(\mathcal{X}, a)}{\text{SplitInfo}(\mathcal{X}, a)}$$

où :

$$\text{SplitInfo}(\mathcal{X}, a) = \sum_{v \in \text{valeurs}(a)} \frac{|\mathcal{X}_{a=v}|}{|\mathcal{X}|} \ln_2 \frac{|\mathcal{X}_{a=v}|}{|\mathcal{X}|}$$

Contrairement à ID3 qui choisit l'attribut ayant le gain maximal, C4.5 choisit l'attribut ayant le rapport de gain maximal pour le placer à la racine de l'arbre de décision.

3.4.3 Application : construction d'un arbre de décision en présence d'attributs numériques

On reprend le même exemple (« jouer au tennis ? ») avec cette fois-ci des valeurs numériques pour les attributs « Température » (en °C) et « Humidité » (en pourcentage) (*cf.* table 3.2).

On obtient l'arbre de décision de la figure 3.4.

3.5 Valeurs d'attributs manquantes

Nous abordons le problème des exemples dont la valeur de certains attributs est manquante, problème concret extrêmement fréquent. ID3 ne dispose

d'aucune possibilité pour gérer ce manque d'information ; C4.5 dispose d'un mécanisme qui est décrit ci-dessous.

On distingue deux cas : certains attributs des exemples de l'ensemble d'apprentissage sont non valués ; certains attributs de la donnée à classer sont non valués. On traite successivement ces deux cas dans la suite de cette section.

3.5.1 Attributs non valués dans l'ensemble d'apprentissage

différents remèdes pour les attributs manquants

Plusieurs solutions peuvent être envisagées, les plus générales étant :

- on laisse de côté les exemples ayant des valeurs manquantes ; ennuyeux car le nombre d'exemples diminue ;
- le fait que la valeur de l'attribut soit manquante est une information en soit : on ajoute alors une valeur possible à l'ensemble des valeurs de cet attribut qui indique que la valeur est inconnue ;
- la valeur la plus courante pour l'attribut en question parmi les exemples classées dans ce nœud est affectée à la place de la valeur manquante ;
- les différentes valeurs observées de cet attribut parmi les exemples couverts par le même nœud sont affectées avec des proids différents en fonction de la proportion d'exemples de l'ensemble d'apprentissage couverts par ce nœud pour les différentes valeurs de cet attribut.

stratégie C4.5 pour les attributs manquants à la construction de l'arbre

C'est cette dernière possibilité qui est utilisée par C4.5. Dès lors, des fractions d'exemples sont utilisées pour continuer la construction de l'arbre. Il faut alors adopter le calcul du gain d'information.

Pour calculer le gain d'information, on ne tient compte que des exemples dont l'attribut est valué. Soit $\mathcal{X}$ l'ensemble d'exemples couverts par le nœud courant (dont on est en train de déterminer l'attribut à tester) et $\mathcal{X}_{\text{sans ?}} \subset \mathcal{X}$ les exemples dont l'attribut est valué. On redéfinit :

$$H(\mathcal{X}) = H(\mathcal{X}_{\text{sans ?}}) \tag{3.4}$$

et

$$\begin{aligned} \text{Gain } (\mathcal{X}, a) \quad = \quad & (H(\mathcal{X}) - \sum_{v \in \text{valeurs}(a)} \frac{|\mathcal{X}_{\text{sans ?},a=v}|}{|\mathcal{X}_{\text{sans ?}}|} H(\mathcal{X}_{\text{sans ?},a=v})) \\ & \frac{|\mathcal{X}_{\text{sans ?}}|}{|\mathcal{X}|} \end{aligned} \tag{3.5}$$

Exemple : supposons que l'exemple x_{12} ait ? à la place de Couvert comme valeur de son attribut « Ciel ». Déterminons le test à placer en racine de l'arbre de décision.

L'entropie de l'ensemble d'apprentissage $\mathcal{X}$ est maintenant :

$$H(\mathcal{X}) = -\frac{8}{13}\ln_2\frac{8}{13} - \frac{5}{13}\ln_2\frac{5}{13} \approx 0.961 \text{ bits}$$

$$\begin{aligned}\text{Gain } (\mathcal{X}, \text{Ciel}) \approx \quad & \tfrac{13}{14}(0.961- \\ & (\tfrac{5}{13}(-\tfrac{2}{5}\ln_2\tfrac{2}{5} - \tfrac{3}{5}\ln_2\tfrac{3}{5})+ \\ & \tfrac{3}{13}(-\tfrac{3}{5}\ln_2\tfrac{3}{5} - \tfrac{0}{3}\ln_2\tfrac{0}{3})+ \\ & \tfrac{5}{13}(-\tfrac{3}{5}\ln_2\tfrac{3}{5} - \tfrac{2}{5}\ln_2\tfrac{2}{5}))) \\ & \approx 0.199 \text{ bits}\end{aligned}$$

Demeurant l'attribut fournissant un gain maximal, « Ciel » est placé à la racine de l'arbre. L'exemple 12 est affecté avec les poids $\frac{5}{13}$, $\frac{3}{13}$ et $\frac{5}{13}$ à chacune des branches, respectivement « Ensoleillé », « Couvert » et « Pluie » ; les autres exemples sont affectés à leur branche respective avec un poids 1 pour chacun.

3.5.2 Classification d'une donnée ayant des attributs non valués

On se place ic non plus dans la phase de construction de l'arbre de décision, mais lors de son utilisation pour rpédire la classe d'une donnée.

Lors de sa descente dans l'arbre, si un nœud teste un attribut dont la valeur est inconnue, C4.5 estime la probabilité pour la donnée de suivre chacune des branches en fonction de la répartition des exemples du jeu d'apprentissage couverts par ce nœud. Cela détermine une fraction de donnée qui poursuit sa descente selon chacune des branches.

Arrivé aux feuilles, C4.5 détermine la classe la plus probable à partir de ces probabilités estimées. Pour chaque classe, il fait la somme des poids ; la classe prédite est celle dont le poids est maximal.

Exemple : dans l'arbre de décision obtenu sur « jouer au tennis ? » avec des attributs nominaux (*cf.* fig. 3.3), classons la donnée (Ciel = ?, Température = Tiède, Humidité = ?, Vent = Faible).

Le nœud racine testant l'attribut « Ciel », sa valeur étant inconnue dans cette donnée à classer, on calcule la proportion d'exemples correspondant à chaque valeur :

- 5 « Ensoleillé » ;
- 4 « Couvert » ;
- 5 « Pluie ».

Donc, on poursuit la classification en transmettant les poids 5/14 vers le nœud testant l'attribut « Humidité », 4/14 le long de la branche « Couvert » vers l'étiquette « oui » et 5/14 vers le nœud testant l'attribut « Vent ».

La valeur de l'attribut « Humidité » est inconnue également. Parmi les exemples « Ensoleillé », il y en a 3 dont l'attribut « Humidité » vaut « Élevée »,

2 dont cet attribut vaut « Normale », soit 3/5 et 2/5 respectivement. Puisque 5/14 exemple a suivi cette branche depuis la racine, on obtient $5/14 \times 3/5 = 3/14$ exemple atteignant l'étiquette « non » et $5/14 \times 2/5 = 1/7$ exemple atteignant l'étiquette « oui ».

L'attribut « Vent » a la valeur « Faible » ; le 5/14 d'exemple qui ont suivi cette branche depuis la racine est donc classé comme « oui ».

En résumé, il y a 3/14 exemple qui atteint une étiquette « non » et $1/7 + 4/14 + 5/14 = 11/14$ exemple qui atteint une étiquette « oui ». On en conclut que la classe la plus probable de cette donnée est « oui ».

3.6 ID3 *vs.* C4.5

Dans cette section, on résume les différences entre ID3 et C4.5 et on donne l'algorithme de C4.5. Rappelons que si notre exposé se concentre ici sur ces deux algorithmes, ils en existent d'autres. Simplement, ces deux algorithmes sont proches l'un de l'autre (C4.5 est une évolution d'ID3) et, d'un point de vue pratique, C4.5 et C5, la version commerciale de C4.5, sont connus comme très performants.

ID3 construit un arbre de décision :

- les attributs doivent tous être qualitatifs et ils sont considérés comme nominaux ;
- ne peut pas prendre en charge des exemples ou des données dans lesquels il manque la valeur d'attributs ;
- utilise les notions d'entropie et de gain pour déterminer l'attribut à tester dans chaque nœud.

En ce qui concerne C4.5 :

- les attributs peuvent être qualitatifs ou quantitatifs ;
- peut utiliser des exemples dans lesquels la valeur de certains attributs est inconnue lors de la construction de l'arbre de décision ;
- peut prédire la classe de donnée dont la valeur de certains attributs est inconnue ;
- utilise le rapport de gain pour déterminer l'attribut à tester dans chaque nœud.

3.7 Validation d'un arbre de décision

Une fois un arbre de décision construit, il est essentiel de le valider en essayant d'estimer les erreurs de classification qu'il fait, autrement dit, la probabilité que la classe prédite pour une donnée quelconque soit correcte[1]. Dépendant

[1] *error rate*

de l'ensemble de données qui est utilisé pour la mesurer, cette quantité est donc une variable aléatoire dont il faut estimer la valeur.

Notons que cette section ne concerne pas uniquement les arbres de décision : elle peut s'appliquer à tous les algorithmes de classification qui seront vus par la suite.

Définitions :

erreur de classification = erreur en généralisation = taux d'échec

taux de succès

Définition 7 *L'erreur de classification E d'un classeur est la probabilité que ce classeur ne prédise pas correctement la classe d'une donnée de l'espace de données.*

Le taux de succès[2] *est égal à* $1 - E$.

Une première approche, naïve, consiste à faire cette estimation en comptant le nombre d'exemples utilisés pour construire l'arbre qui sont mal classées.

erreur apparente = erreur d'entraînement = erreur d'apprentissage

Définition 8 *L'erreur apparente* [3] E_{app} *est mesurée avec les exemples utilisés pour la construction du classeur : c'est la proportion d'exemples dont la classe est mal prédite par le classeur.*

E_{app} n'est pas un bon estimateur de l'erreur qui serait commise face à de nouvelles données ; en effet, l'enjeu est bien là : à partir des exemples avec lesquels l'arbre de décision a été construit, l'apprentissage doit pouvoir être généralisé à de nouvelles données. C'est là l'objectif des algorithmes d'apprentissage : un algorithme ne peut être qualifié d'algorithme d'apprentissage que s'il est capable de généraliser ce qu'il a appris.

L'estimation de la qualité de l'arbre de décision construit en tant que classeur de nouvelles données est donc un point capital. On distingue :

- le jeu d'exemples d'apprentissage (noté $\mathcal{X}_{\text{app}}$) ou d'entraînement avec lesquelles l'arbre de décision est construit ; — jeu d'apprentissage
- le jeu d'exemples de test (noté $\mathcal{X}_{\text{test}}$) qui permet d'estimer les erreurs de classification : pour ces exemples, on connaît leur classe. On les classe avec l'arbre de décision construit avec $\mathcal{X}_{\text{app}}$ puis on regarde s'ils sont classés correctement. Bien entendu, idéalement, l'intersection entre jeu d'apprentissage et jeu de test doit être vide. On obtient alors une mesure de l'erreur de test E_{test}. — jeu de test

holdout

Si l'on ne dispose pas de jeu de test, on utilise le jeu d'apprentissage de la manière suivante : on le découpe en deux parties ; l'une constituera le jeu d'apprentissage effectif pour construire l'arbre de décision ; l'autre serviva à l'évaluer (technique de *holdout*, ou retenue).

[2] *success rate*
[3] *resubstitution error.*

3.7.1 Mesures de qualité d'un classeur

On définit ici quelques notions simples largement utilisées.

On suppose que l'on a construit un classeur (un arbre de décision ou un autre) à partir d'un jeu d'exemples ; ces mesures nécessitent un jeu d'exemples qui peut être le jeu d'apprentissage ou un jeu de test.

On définit dans le cas de classification binaire, sachant que les définitions sont aisément extensibles aux autres cas :

Définition 9
- *VP : le nombre de vrais positifs : les exemples de classe positive et dont la classe est prédite comme positive ;*
- *VN : le nombre de vrais négatifs : les exemples de classe négative et dont la classe est prédite comme négative ;*
- *FP : le nombre de faux positifs : les exemples de classe négative et dont la classe est prédite comme positive ;*
- *FN : le nombre de faux négatifs : les exemples de classe positive et dont la classe est prédite comme négative.*

matrice de confusion

On peut alors visualiser ces informations dans une « matrice de confusion » :

	+	− ← classe prédite
+	VP	FN
−	FP	VN
↑ classe		

S'il n'y a des nombres non nuls que sur la diagonale principale, c'est qu'aucun exemple n'est mal classé.

précision

rappel

On peut définir aussi deux statistiques, la précision et le rappel :
- précision pour les positifs = $\frac{VP}{VP+FP}$; précision pour les négatifs = $\frac{VN}{VN+FN}$;
- rappel pour les positifs = $\frac{VP}{VP+FN}$; rappel pour les positifs = $\frac{VN}{VN+FP}$.

Intuitivement parlant, la précision mesure la proportion d'exemples vraiment positifs (resp. négatifs) parmi ceux qui sont classés comme positifs (resp. négatifs). Le rappel mesure la proportion d'exemples vraiment positifs (resp. négatifs) parmi tous les exemples classés comme positifs (resp. négatifs).

mesure F

Il est toujours plus pratique de manipuler un seul nombre qui synthétise les autres. Ainsi la mesure F est définie par :

$$F = \frac{2 \text{ rappel } \times \text{ précision}}{\text{rappel } + \text{ précision}} = \frac{2\ VP}{2\ VP + FP + FN}$$

D'autres mesures existent.

ation croisée

3.7.2 Validation croisée

Une méthode plus sophistiquée est la « validation croisée ». Pour cela, on découpe l'ensemble des exemples en n sous-ensembles mutuellement disjoints. Il faut prendre garde à ce que chaque classe apparaisse avec la même fréquence dans les n sous-ensembles (stratification des échantillons). Si $n = 3$, cela produit donc 3 ensembles A, B et C. On construit l'arbre de décision $\mathrm{AD}_{A\cup B}$ avec $A \cup B$ et on mesure son taux d'erreur sur C, c'est-à-dire, le nombre d'exemples de C dont la classe est mal prédite par $\mathrm{AD}_{A\cup B}$: E_C.

Ensuite, on construit l'arbre de décision $\mathrm{AD}_{B\cup C}$ avec $B \cup C$ et on mesure l'erreur sur A : E_A. Enfin, on construit l'arbre de décision $\mathrm{AD}_{A\cup C}$ avec $A \cup C$ en mesurant l'erreur sur B : E_B.

Le taux d'erreur E est alors estimé par la moyenne de ces trois erreurs mesurées $E = \frac{E_A+E_B+E_C}{3}$.

Habituellement, on prend $n = 10$. Cette méthode est dénommée « validation croisée en n-plis » (*n-fold cross-validation*).

3.7.3 Technique du *leave-one-out*

Cette technique consiste à effectuer une validation croisée à $n = N$ plis en laissant à chaque fois un seul exemple de côté (N est le nombre d'exemples dont on dispose).

L'erreur est estimée par la moyenne des N erreurs mesurées.

3.7.4 Technique de *bootstrap* (= *bagging*)

Le jeu d'apprentissage est constitué en effectuant N tirages avec remise parmi l'ensemble des exemples. Cela entraîne que certains exemples du jeu d'apprentissage seront vraisemblablement sélectionnés plusieurs fois, et donc que d'autres ne le seront jamais. En fait, la probabilité qu'un certaine exemple ne soit jamais tiré est simplement :

$$(1 - \frac{1}{N})^N$$

La limite quand $N \to +\infty$ de cette probabilité est $e^{-1} \approx 0.368$.

Les exemples qui n'ont pas été sélectionnés constituent le jeu de test.

Le jeu d'apprentissage contient 63,2 % des exemples du jeu d'exemples initial (en moyenne).

L'erreur est calculée en combinant l'erreur d'apprentissage E_{app} et l'erreur de test E_{test} par la formule suivante :

$$E = 0.632 \times E_{\mathrm{test}} + 0.368 \times E_{\mathrm{app}}$$

Ensuite, cette procédure est itérée plusieurs fois et une erreur moyenne est calculée.

3.7.5 Confiance dans l'estimation de l'erreur

Les recettes que nous venons de décrire permettent de donner une idée de la probabilité de mal classer une nouvelle donnée. On présente ici comment C4.5 détermine un intervalle de confiance pour cette estimation.

Cas où on utilise un jeu de test différent du jeu d'apprentissage

On suppose pour l'instant que nous disposons d'un jeu d'exemples de test disjoint du jeu d'exemples d'apprentissage.

Supposons que 25% des exemples du jeu de test soient mal classés ; cela signifie-t-il que le taux d'erreur de l'algorithme est de 25 % ? ... Non.

On veut estimer l'erreur E, c'est-à-dire, la probabilité qu'une donnée soit mal classée par un classeur donné. Cette erreur est une variable aléatoire. On va s'intéresser à en déterminer la valeur moyenne : c'est-à-dire, disposant d'un classeur construit avec N exemples, quelle est la valeur probable de E ? peut-on en donner un intervalle ?

Donc, on suppose disposer d'un classeur. On lui présente une nouvelle donnée. Le classeur prédit une classe pour cette donnée ; soit cette classe est prédite correctement, soit elle l'est incorrectement. On est donc face à un événement aléatoire ayant deux possibilités : « bien classé » ou « mal classé ». E est donc la probabilité de l'événement « mal classé ».

L'erreur de test $\mathcal{X}_{\text{test}}$ est donc :

$$\mathcal{X}_{\text{test}} = \frac{\text{nombre d'exemples de } \mathcal{X}_{\text{test}} \text{ mal classés}}{|\mathcal{X}_{\text{test}}|}$$

On veut donc estimer E avec E_{test} : E_{test} est un estimateur de E.

Nous sommes en présence d'un processus de Bernouilli : une suite d'événements dont chacun prend une valeur parmi deux (bien ou mal classé). Nous sommes face à un problème exactement analogue à la détermination de la probabilité qu'une pièce tombe sur pile : on lance plusieurs fois une pièce en l'air et on compte le nombre de fois où on lance la pièce (N) ainsi que le nombre de fois où elle tombe sur pile (N_{pile}). Quand N tend vers l'infini, on sait que le rapport $\frac{N_{\text{pile}}}{N}$ tend vers p, la probabilité que la pièce tombe sur pile. La question que l'on se pose ici est la suivante : n'ayant effectué que N lancers (et non pas une infinité) et mesuré N_{pile}, que puis-je dire sur p ?

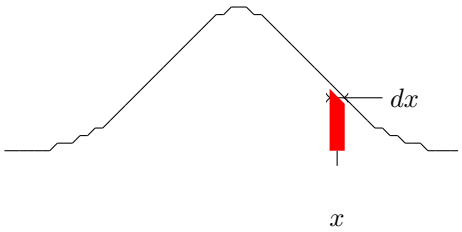


FIG. 3.5 – La probabilité qu'une variable aléatoire distribuée normalement prenne une valeur donnée x est la surface située sous la courbe en cloche sur une largeur infiniment petite (mais pas nulle !) centrée en x, ce qui est très grossièrement illustré par le trapèze rouge.

C'est cette question que nous étudions ci-dessous. Ce qui nous intéresse ici n'est pas la probabilité pour une pièce de tomber sur pile, mais la probabilité pour un classeur de mal prédire la classe d'une donnée, soit E.

distribution de E

On présente un exemple au classeur. On compte 0 si l'exemple est bien classé, 1 sinon. Si on a N exemples, on obtient ainsi une estimation de la valeur de E_{test}. Nous sommes en présence d'une loi binomiale. Aussi, on sait[4] que cette estimation de E_{test} tend vers E. On sait aussi que sa variance tend vers $\frac{E(1-E)}{N}$. Donc, $\frac{E_{\text{test}}-E}{\sqrt{\frac{E(1-E)}{N}}}$ est une variable aléatoire centrée réduite.

Le théorème central limite nous dit aussi que quand N tend vers l'infini, la distribution de E_{test} tend vers une distribution normale. Donc, la variable aléatoire $\frac{E_{\text{test}}-E}{\sqrt{E(1-E)/N}}$ est centrée, réduite, distribuée normalement quand N tend vers l'infini.

intervalle de confiance

On veut savoir si E_{test} est une bonne estimation de E. Pour cela, donnons-nous un seuil de confiance c et déterminons la probabilité avec laquelle E est dans un certain intervalle de valeurs centré sur E_{test}.

Rappelons que la probabilité qu'une variable aléatoire v distribuée normalement, centrée et réduite prenne une valeur dans un intervalle $x \pm dx$ est donnée par :

$$Pr[v = x] = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

[4] *cf.* bouquin de stats

TAB. 3.3 – Extrait de la table donnant $Pr[v \geq z] = \frac{1}{\sqrt{2\pi}} \int_z^{+\infty} e^{-\frac{x^2}{2}}$ pour une distribution normale et une variable aléatoire v centrée réduite. Cette probabilité $Pr[v \geq z] = \frac{1-c}{2}$ est la surface sous la cloche « à droite » de l'abscisse z. Pour les valeurs intermédiaires, on peut effectuer une interpolation linéaire ; par exemple, pour une confiance $c = 0.75$, on a $\frac{1-c}{2} = 0.125$, d'où $z = 1.28 - \frac{1.28-0.84}{4} = 1.17$. Une table statistique donne $z = 1.15$ pour $c = 0.75$.

$Pr[v \geq z]$	0.001	0.005	0.01	0.05	0.1	0.2	0.4
z	3.09	2.58	2.33	1.65	1.28	0.84	0.25

La probabilité que la valeur de v soit comprise dans un intervalle $[-z, +z]$ est donnée par :

$$Pr[-z \leq v \leq z] = \frac{1}{\sqrt{2\pi}} \int_{-z}^{z} e^{-\frac{x^2}{2}} dx$$

Quelle est maintenant l'intervalle de valeurs $[-z, +z]$ autour de E_{test} dans lequel nous avons la probabilité c que E s'y trouve ?

C'est z tel que :

$$Pr[-z \leq \frac{E_{\text{test}} - E}{\sqrt{E(1-E)/N}} \leq z] = c$$

dans laquelle E_{test} est connu, c est fixé (donc z s'en déduit) et E est cherché. Donc, on cherche bien l'intervalle $E \in [E_{inf}, E_{sup}]$ tel que la probabilité que E appartienne à cet intervalle soit c.

On trouve z dans une table de distribution normale, centrée, réduite. Ce type de table donne en fait $Pr[v \geq z]$ (*cf.* table 3.3). Pour une distribution normale, on sait que $Pr[v \leq -z] = Pr[v \geq z]$. La confiance c dans l'intervalle est $c = Pr[-z \leq v \leq z] = 1 - 2 \times Pr[v \geq z]$. Donc, c étant fixé, on détermine z tel que $Pr[v \geq z] = \frac{1-c}{2}$

Quand on a ainsi déterminé z, il nous reste à trouver E_{inf} et E_{sup}. En ré-écrivant comme il faut $Pr[-z \leq \frac{E_{\text{test}}-E}{\sqrt{E(1-E)/N}} \leq z] = c$, on obtient :

$$\begin{cases} E_{inf} = \dfrac{E_{\text{test}} + \frac{z^2}{2N} - z\sqrt{\frac{E_{\text{test}}}{N} - \frac{E_{\text{test}}^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \\ E_{sup} = \dfrac{E_{\text{test}} + \frac{z^2}{2N} + z\sqrt{\frac{E_{\text{test}}}{N} - \frac{E_{\text{test}}^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \end{cases}$$

Par exemple, si $E_{\text{test}} = 0.25$, $N = 1000$ et $c = 0.8$, on obtient un intervalle de confiance :

$$E \in \frac{0.25 + \frac{1.28^2}{2000} \pm 1.28\sqrt{\frac{0.25}{1000} - \frac{0.25^2}{1000} + \frac{1.28^2}{4\times 1000^2}}}{1 + \frac{1.28^2}{1000}} = [0.233, 0.268]$$

De même, si $N = 100$, on a $E \in [0.20, 0.31]$. On constate, on s'y attendait, que plus N est grand, plus l'intervalle est resserré.

Cas où le jeu de test est le même que le jeu d'apprentissage

On suppose maintenant que l'on mesure l'erreur en utilisant le jeu d'apprentissage, c'est-à-dire avec les données qui ont servi à construire l'arbre de décision. Cela introduit un biais que les statistiques n'aiment pas beaucoup. Cependant, C4.5 maximise la probabilité d'erreur en utilisant la borne supérieure de l'intervalle de confiance calculée ci-dessus :

$$E_{sup} = \frac{E_{\text{test}} + \frac{z^2}{2N} + z\sqrt{\frac{E_{\text{test}}}{N} - \frac{E^2_{\text{test}}}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

Des tests expérimentaux ont montré que cette estimation n'est pas déraisonnable.

3.8 Sur-apprentissage

Il est intéressant de dessiner l'erreur d'apprentissage et l'estimation de l'erreur réelle E en fonction de la proportion d'exemples utilisés pour l'apprentissage. Ainsi, sur le jeu « jouer au tennis ? », on a pris successivement 1, 2, 3, ... 14 exemples pour construire l'arbre de décision ; on a mesuré l'erreur sur le jeu d'exemples utilisés pour construire l'arbre et on a estimé l'erreur réelle ; on obtient un graphe comme celui de la figure 3.6. On voit que l'erreur d'apprentissage diminue constamment quand on utilise plus d'exemples pour construire l'arbre alors que E diminue d'abord puis stagne. L'erreur mesurée sur le jeu d'apprentissage atteint 0 et reste à 0 parce que l'on peut construire un arbre parfait (qui classe correctement tous les exemples pour ce jeu de données) ; si l'on ne peut pas construire d'arbre parfait (à cause d'une incohérence dans les exemples : deux exemples ayant la même description sont de classe différente), l'erreur d'apprentissage reste non nulle.

Il est très important de comprendre ce que ce schéma illustre. Plus le nombre d'exemples utilisés pour construire le modèle (ici, un arbre de décision) augmente, plus l'erreur sur ce jeu diminue et tend vers 0. Cependant, ce qui compte vraiment, c'est l'erreur en généralisation. Pour sa part, quand le nombre d'exemples utilisés augmente, l'erreur en généralisation commence par diminuer puis elle augmente : c'est précisèment là où elle est minimale que l'on a construit

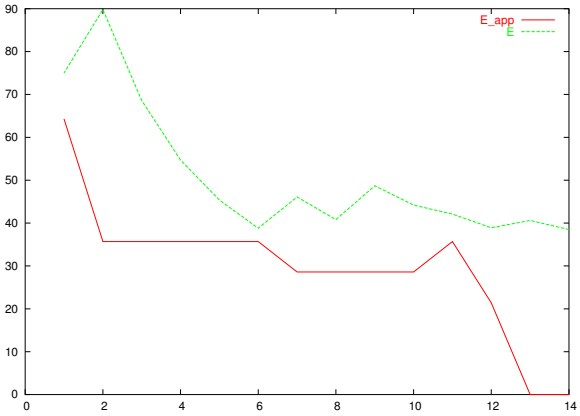


FIG. 3.6 – Variation de l'erreur d'apprentissage E_{app} et de l'erreur E (estimée) en fonction du nombre d'exemples utilisés pour construire l'arbre de décision. Les deux erreurs diminuent. Au bout d'un certain temps, alors que E_{app} continue de diminuer, E stagne. E est estimée ici avec une confiance $c = 0.75$. On utilise le jeu de données « jouer au tennis ? ».

le meilleur modèle, celui qui fait une erreur minimale. C'est ce modèle qui est capable de produire la meilleure généralisation de l'apprentissage, c'est-à-dire, qui a la plus petite probabilité de mal classer une donnée quelconque. Au-delà de ce modèle optimal, quand l'apprentissage se poursuit, le modèle se complique, la probabilité d'erreur augmente, et le modèle produit du sur-apprentissage. Le modèle alors construit colle de plus en plus près aux exemples et sa capacité à prédire correctement la classe d'autres données diminue ; le modèle « manque de recul » par rapport aux exemples. Le problème se sur-apprentissage (ou sur-spécialisation) est récurrent dans les algorithmes d'apprentissage automatique et, plus généralement, dans les modèles. On le rencontrera régulièrement dans les chapitres suivants.

3.9 Élagage

L'élagage consiste à simplifier un arbre de décision en coupant des branches. Il possède deux objectifs :

- simplifier l'arbre de décision ;
- diminuer le sur-apprentissage (= augmenter la capacité de généralisation) et, par là-même, diminuer le taux d'erreur.

Deux possibilités : élagage lors de la construction et élagage après la construction. Cette seconde approche est celle utilisée dans C4.5.

Deux types d'élagage sont effectués dans C4.5 :

– remplacement d'un sous-arbre : consiste à remplacer un sous-arbre par une feuille ;
– promotion d'un sous-arbre : consiste à rassembler deux nœuds dans un seul nœud.

Dans le premier cas (remplacement d'un sous-arbre), les nœuds sont considérés depuis les feuilles en remontant vers la racine et en effectuant un test au niveau de chaque nœud. La décision d'appliquer une transformation est prise en fonction du taux d'erreur du nœud et de ses fils comme suit. C4.5 estime l'erreur de chaque fils (*cf.* section 3.7.5) ainsi que l'erreur du nœud. Ces estimations sont combinées en les pondérant chacune par la proportion d'exemples couvert par chaque fils. C4.5 estime ensuite le taux d'erreur du nœud. Si celui-ci est plus petit que la combinaison des taux d'erreur des fils, alors C4.5 remplace le nœud par une feuille.

Exemple : on suppose qu'un nœud possède trois fils ; le premier couvre 6 exemples, dont 2 sont mal classés ; le deuxième couvre 2 exemples, 1 étant mal classé ; le dernier couvre 6 exemples dont, à nouveau, 2 sont mal classés. Les taux d'erreur estimés de ces trois fils sont respectivement 0.47, 0.72 et 0.47 (avec un taux de confiance de 75 %). La combinaison des trois donne : $\frac{6}{14} \times 0.47 + \frac{2}{14} \times 0.72 + \frac{6}{14} \times 0.47 = 0.504$. Le taux d'erreur du nœud lui-même est donc calculé avec les données : 14 exemples, dont 5 mal classés, ce qui fournit un taux d'erreur estimé de 0.46. Ce taux d'erreur étant plus petit, C4.5 remplace le nœud par une feuille étiquetée avec la valeur la plus présente parmi ces 14 exemples.

3.10 Illustration sur les iris

On illustre la construction d'un arbre de décision sur le jeu de données dénommé « iris ». Il contient 150 exemples ; chacun est une fleur (un iris) d'une des trois variétés suivantes : *setosa*, *versicolor* et *virginica*. La variété représente la classe de la donnée.

Chaque donnée est décrite par 4 attributs numériques : longueur et largeur des sépales ; longueur et largeur des pétales.

Avec ces 150 exemples, C4.5 construit l'arbre représenté à la figure 3.7 ; 3 exemples sont mal classés, 2 *versicolor* et 1 *virginica*.

Cet arbre incite à penser que la longueur et la largeur des sépales ne sont pas des attributs pertinents pour la détermination de la classe des données. C'est effectivement une caractéristique bien connue de ce jeu de données.

3.11 Critique

Les arbres de décision fonctionnent bien si :

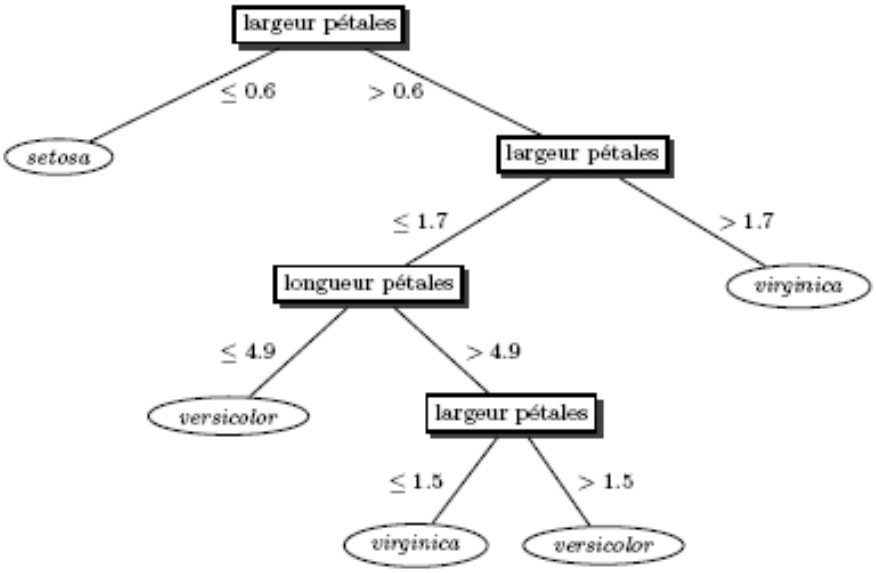


FIG. 3.7 – Arbre de décision construit par C4.5 sur le jeu de données « iris ».

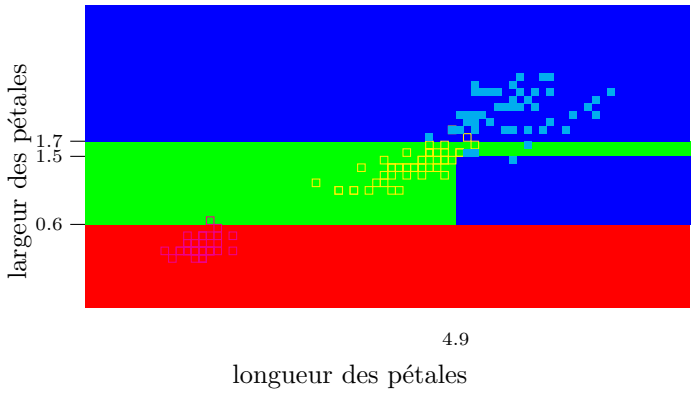


FIG. 3.8 – Découpage de l'espace des données par l'arbre de décision de la fig. 3.7 : en abcisse est portée la longueur des pétales, en ordonnées est portée la largeur des pétales. La zone rouge est la partie où la classe prédite est *setosa*, la zone en vert correspond à la classe *versicolor* et la zone bleue à la classe *virginica*.

– le nombre de valeurs possibles pour chaque attribut est faible ;
– la classe est à valeur qualitative ;
– même si l'ensemble des exemples contient des erreurs ;
– même si l'ensemble des exemples en contient dont certains attributs sont non valués.

De plus :

+ on dispose d'un algorithme de construction d'arbres de décision très utilisé et très performant (c4.5) ;
+ C4.5 est très rapide pour construire un arbre de décision de bonne qualité ;
+ on est capable d'en estimer la qualité ;
+ on obtient un modèle compréhensible, mais :
- sur un gros jeu de données, l'arbre devient gros et illisible ;
– on peut transformer un arbre de décision en un ensemble de règles de classification ce qui augmente la lisibilité du modèle (*cf.* chap. 6).

3.12 Logiciels libres

– C4.5 sur le site de quinlan : `http://www.cse.unsw.edu/~quinlan` ;
– C5 version démo : `http://www.rulequest.com`
– `dtree` de Ch. Borgelt : `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html#dtree`
– `weka` contient une implantation de l'algorithme C4.5 : `http://www.cs.waikato.ac.nz/~ml`

3.13 Exercices

Exercice 2 Vous avez été recruté comme expert en aide à la décision dans un laboratoire d'exo-biologie. Vous assistez un exo-biologiste qui analyse des échantillons de choses qui ressemblent à des plantes, recueillies sur une planète lointaine. Ces exo-plantes sont décrites par quatre informations (couleur, forme, taille et motif à leur surface) et une propriété importante : leur comestibilité. Votre mission est de déterminer une manière de décider si une exo-plante donnée est comestible ou non pour l'espèce humaine qui envisage de coloniser cette planète.

Au nombre de 8, les échantillons sont décrits dans le tableau ci-dessous :

identificateur	couleur	forme	taille	motif	Classe
SN-54	cyan	octogonal	petit	uniforme	comestible
AC-951	orange	hexagonal	petit	uniforme	comestible
ML-45	orange	octogonal	petit	rayé	non comestible
RT-3	magenta	hexagonal	gigantesque	rayé	comestible
ID-546	cyan	hexagonal	gigantesque	rayé	comestible
KG-7	orange	octogonal	grand	uniforme	comestible
XP-354	magenta	pentagonal	gigantesque	rayé	non comestible
EX-76	magenta	octogonal	grand	rayé	non comestible

Question 1 : quelle est l'entropie de la population ?

Question 2 : quel est l'attribut dont le gain d'information est maximal dans la population ?

Question 3 : construire l'arbre de décision avec l'algorithme ID3.

Question 4 : quelle est l'erreur de cet arbre mesurée sur le jeu d'apprentissage ?

Question 5 : estimer la probabilité de bien classer une nouvelle donnée avec une confiance de 75 %.

La mission vient de recueillir de nouveaux échantillons :

identificateur	couleur	forme	taille	motif	Classe
GH-45	magenta	octogonal	petit	rayé	comestible
GH-34	cyan	pentagonal	gigantesque	uniforme	non comestible
ML-57	magenta	hexagonal	grand	rayé	non comestible
NS-31	orange	hexagonal	grand	uniforme	comestible

Question 6 : quelle est la classe prédite par l'arbre que vous avez construit pour ces données ?

Question 7 : quelle est l'erreur mesurée avec ces 4 nouvelles données ?

Question 8 : à partir de cette erreur mesurée, estimer la valeur de l'erreur réelle avec une confiance de 75 %.

Question 9 : quelle est la classe prédite pour un petit échantillon pentagonal ?

Question 10 : quelle est la classe prédite pour un échantillon rayé et cyan ?

Exercice 3 Une banque dispose des informations suivantes sur un ensemble de ses clients :

client	M	A	R	E	I
1	moyen	moyen	village	oui	oui
2	élevée	moyen	bourg	non	non
3	faible	âgé	bourg	non	non
4	faible	moyen	bourg	oui	oui
5	moyen	jeune	ville	oui	oui
6	élevée	âgé	ville	oui	non
7	moyen	âgé	ville	oui	non
8	faible	moyen	village	non	non

L'attribut client indique le numéro du client ; l'attribut M indique la moyenne des crédits sur le compte du client ; l'attribut A donne la tranche d'âge ; l'attribut R décrit la localité du client ; l'attribut E possède la valeur oui si le client possède un niveau d'études supérieur au bac ; l'attribut I (la classe) indique si le client effectue ses opérations de gestion de compte via Internet.

Question 1 : quelle est l'entropie de la population ?

Question 2 : pour la construction de l'arbre de décision, utilisez-vous l'attribut numéro de client ? Pourquoi ?

Question 3 : lors de la construction de l'arbre de décision, quel est l'attribut à tester à la racine de l'arbre ?

Question 4 : construire l'arbre de décision complet.

Question 5 : quel est le taux d'erreur de cet arbre estimé sur l'ensemble de clients 1 à 8 ?

Question 6 : donner un intervalle de valeurs pour l'erreur réelle en utilisant une confiance de 90 %.

On se donne les 4 clients suivants :

client	M	A	R	E
9	moyen	âgé	village	oui
10	élevée	jeune	ville	non
11	faible	âgé	village	non
12	moyen	moyen	bourg	oui

Question 7 : comment chacun de ces clients est-il classé avec l'arbre de décision que vous avez proposé dans la question 4 ?

Pour ces 4 clients, on apprend par ailleurs que les clients 9 et 10 gèrent leur compte par Internet, et que les clients 11 et 12 ne le font pas.

Question 8 : quel est le taux d'erreur estimé sur les clients 9, 10, 11 et 12 ? Combien y a-t-il de faux positifs et de faux négatifs ?

Exercice 4 On considère des données décrites par quatre attributs :

- numéro : un entier ;
- forme : rond et carré ;
- taille : petit et grand ;
- couleur : bleu, blanc et rouge.

L'attribut cible est binaire de valeurs oui et non.

Les données disponibles sont les suivantes (le ? correspond à une valeur manquante) :

	forme	taille	couleur	classe
1	rond	petit	bleu	oui
2	carré	grand	rouge	non
3	rond	?	blanc	oui
4	carré	petit	bleu	oui
5	rond	grand	bleu	oui
6	carré	grand	blanc	non
7	carré	?	blanc	oui
8	carré	grand	bleu	non
9	carré	petit	rouge	oui
10	rond	grand	blanc	oui

Le problème est de savoir comment prendre en compte les données pour lesquelles on ne dispose de la valeur de tous les attributs.

Élimination des exemples incomplets

La solution la plus expéditive est de ne pas utiliser les exemples dans lesquels certains attributs ne sont pas valués.

Question 1 : peut-on trouver un arbre de décision parfait ? (un arbre de décision est parfait s'il prédit correctement la classe de tous les exemples). Répondre à cette question en réfléchissant, pas en essayant de le construire.

Question 2 : appliquer l'algorithme de construction d'arbre de décision vu en cours.

Valeur majoritaire de l'attribut

On remplace les valeurs manquantes par la valeur majoritaire prise par cet attribut sur l'échantillon complet.

Question 3 : quelle valeur associe-t-on aux attributs manquant sur les 10 données ?

Question 4 : peut-on trouver un arbre de décision parfait ? Répondre à cette question en réfléchissant, pas en essayant de le construire.

Question 5 : appliquer l'algorithme de construction d'arbre de décision vu en cours.

Valeur majoritaire de l'attribut par classe

Étant donné un exemple avec une valeur manquante, on remplace la valeur manquante par la valeur majoritaire prise par l'attribut correspondant pour les exemples de l'échantillon appartenant à la même classe.

On pose les 3 mêmes questions que précédemment :

Question 6 : quelle valeur associe-t-on aux attributs manquant sur les 10 données ?

Question 7 : peut-on trouver un arbre de décision parfait ?

Question 8 : appliquer l'algorithme de construction d'arbre de décision vu en cours.

Exercice 5 À la section 3.2.1, on a construit un arbre de décision avec le jeu de données « jouer au tennis ? » en laissant de côté l'attribut « Jour » sans autre forme de procès.

Question 1 : Calculer la gain de l'ensemble d'exemples pour cet attribut.

Question 2 : qu'en pensez-vous ? Construire l'arbre de décision complet en prenant en compte cet attribut.

À la section 3.4.2, on a indiqué que C4.5 utilise un autre critère que le gain d'information pour déterminer l'attribut à placer à la racine de l'arbre de décision.

Question 3 : calculer le rapport de gain pour les 5 attributs sur le jeu d'exemples « jouer au tennis ? »

Question 4 : quel attribut C4.5 place-t-il à la racine de l'arbre de décision ?

Question 5 : continuer la construction qu'effectue C4.5

Exercice 6 On dispose d'un jeu de N données décrites chacune par P attributs (on ne parle pas de classe ici). La valeur de certains attributs de certaines données est manquante.

Question 1 : Imaginez un moyen pour prédire la valeur des attributs manquants ?

Exercice 7 Supposons que l'on ajoute la possibilité d'avoir des tests du genre $a \in [b, B]$ où a est un attribut quantitatif. Comment calculeriez-vous le gain d'information apporté par cet attribut ?

Exercice 8 Exercice pour informaticien

Récupérer le source de C4.5, y implanter différentes stratégies de construction d'arbre de décision et effectuer une étude expérimentale.

On implantera :

- la *randomization* décrite dans Dietterich [1999]
- l'utilisation de l'indice de Gini à la place de l'entropie. Cet indice se définit par : $1 - \sum_{y \in \mathcal{Y}}(\text{proportion d'exemples de classe } y)^2$
- on utilisera les jeux de données fournis avec C4.5

Exercice 9 Exercice pour informaticien : construction incrémentale d'un arbre de décision

On suppose que les exemples sont disponibles non plus tous à la fois mais obtenus un par un (par exemple, un toutes les heures). On veut construire un arbre de décision en utilisant les exemples disponibles à un moment donné qui sera complété lorsqu'un nouvel exemple sera obtenu. Imaginer un algorithme pour cela.

Naturellement, l'algorithme commencera par construire un arbre dès l'obtention du premier exemple. Cet arbre initial sera ensuite modifié s'il le faut à la réception du deuxième exemple, ...

Chapitre 4

Classeur bayésien

Contenu

Dans ce chapitre, on présente une méthode de classification reposant sur une approche probabiliste basée sur la règle de Bayes, donc l'approche bayesienne des probabilités. Un intérêt de cette approche est qu'elle permet naturellement d'intégrer des connaissances *a priori*, ce que ne permettent pas les arbres de décision ou la plupart des méthodes que nous verrons par la suite.

Sans nous attarder longuement sur ce point, il est important de noter que l'approche bayésienne des probabilités diffère fondamentalement de l'approche classique, dite fréquentiste. Dans l'approche classique, on mesure $Pr[x]$, la probabilité d'occurence de l'événement x ; dans l'approche bayésienne, on mesure

$Pr[x|w]$, c'est-à-dire, la probabilité d'occurence de l'événement x si l'événement w est vérifié : w joue le rôle d'une hypothèse préliminaire que l'on suppose remplie pour estimer la probabilité d'occurcence de l'événement qui nous intéresse, noté x ici. w peut être un « événement » du genre « je possède telle connaissance ». Dans le premier cas, on fait comme si l'occurence de l'événement x était un fait absolu et sa mesure une mesure absolue ; dans le second, on a une attitude pragmatique qui peut s'interpréter comme : dans le monde où je vis, sachant que je suis dans telle situation et/ou que je connais telles informations (ou en supposant que j'ai ces informations) noté w, je mesure la probabilité d'occurence de l'événement x.

Disons-le encore autrement : il est important de ne pas confondre les notions de probabilité conditionnelle $Pr[x|w]$ et de probabilité jointe $Pr[x \wedge w]$: la probabilité conditionnelle estime la probabilité d'occurence d'un événement en supposant qu'un autre événement est vérifié ; la probabilité jointe estime la probabilité d'occurence de deux événements en même temps. Dans la probabilité conditionnelle, on ne s'intéresse pas au fait de savoir si w a réellement lieu ; il suppose qu'il a lieu et on estime la probabilité de x dans cette situation.

4.1 La règle de Bayes

4.1.1 Le théorème de Bayes

Soient A, B et C trois événements. Le théorème (ou règle) de Bayes démontre que :

$$Pr[A|B] = \frac{Pr[B|A]Pr[A]}{Pr[B]}C \tag{4.1}$$

où :

- $Pr[B|A, C]$ est la vraisemblance de l'événement B si A et C sont vérifiés ;
- $Pr[A|C]$ est la probabilité *a priori* de l'événement A sachant C ;
- $Pr[B|C]$ est la probabilité marginale de l'événement B sachant C ;
- $Pr[A|B, C]$ est la probabilité *a posteriori* de A si B et C.

Dans cette formulation de la règle de Bayes, C joue le rôle de la connaissance que l'on a.

4.1.2 Application à la classification

Supposons que :

- y soit l'événement : « j'observe la classe y » ;
- x soit l'événement : « j'observe la donnée x ».

$Pr[y|x, \mathcal{X}]$ est donc la probabilité d'observer la classe y si on observe la donnée x sachant que je dispose de l'ensemble d'exemples $\mathcal{X}$. Autrement dit, c'est l'estimation de la probabilité que la donnée x soit de classe y étant donné que je dispose des exemples $\mathcal{X}$. En appliquant la règle de Bayes, on obtient :

$$Pr[y|x, \mathcal{X}] = \frac{Pr[x|y, \mathcal{X}]Pr[y|\mathcal{X}]}{Pr[x|\mathcal{X}]}$$

Les probabilités de l'expression de droite doivent être estimées pour calculer la quantité qui nous intéresse, $Pr[y|x, \mathcal{X}]$:

- $Pr[y|\mathcal{X}]$ est la probabilité d'observer la classe y étant donné l'ensemble d'exemples $\mathcal{X}$. Bien entendu, si pour une raison ou une autre, on dispose de cette information, on peut utiliser la proportion dans l'ensemble des données de la classe y ; — estimation de la probabilité *a priori*
- $Pr[x|y, \mathcal{X}]$, la vraisemblance de l'événement « observer la donnée x » si elle est de classe y, disposant des exemples $\mathcal{X}$. Ce terme est plus difficile à estimer que le précédent. En absence d'autre information, on utilise « l'hypothèse de Bayes naïve » (HBN) : la donnée x est une conjonction de valeur d'attributs ; l'HBN consiste à supposer que les attributs sont des variables aléatoires indépendantes, c'est-à-dire que les valeurs de ses attributs ne sont pas corrélées entre-elles. Clairement, cette hypothèse n'est à peu près jamais vérifiée ; cependant, elle permet de faire des calculs simplement et, finalement, les résultats obtenus ne sont pas sans intérêt d'un point de vue pratique. Notons que si l'on a des informations concernant ces corrélations entre valeurs d'attributs, on pourra les utiliser. — estimation de la vraisemblance — hypothèse naïve de Bayes

  Si on applique l'HBN, en supposant que la donnée x est décrite par P attributs notés a_j dont les valeurs sont notées v_j, on écrit :

$$\begin{aligned} & Pr[x|y, \mathcal{X}] \\ = \quad & Pr[a_1 = v_1|y, \mathcal{X}] \times ... \times Pr[a_P = v_P|y, \mathcal{X}] \\ = \quad & \Pi_{i=1}^{i=P} Pr[a_i = v_i|y, \mathcal{X}] \end{aligned} \tag{4.2}$$

  Chaque terme $Pr[a_j = v_j|y]$ est estimé à l'aide de l'ensemble d'exemples ; l'estimation de ce terme dépende de la nature, qualitative ou quantitative, de l'attribut. Les exemples donnés plus bas illustrent ce point ;
- $Pr[x|\mathcal{X}]$ est la probabilité d'observer la donnée x, ayant l'ensemble d'exemples $\mathcal{X}$: *a priori*, on ne voit pas comment calculer cette quantité. Aussi, on utilise une astuce très simple : si la classe est binaire (on généralise sans difficulté aux autres cas), on constate que la somme de la probabilité d'observer une donnée x si elle est positive et de la probabilité d'observer cette même donnée x si elle est négative vaut 1. On peut donc écrire : — estimation de la probabilité marginale

$$\sum_{y_i \in \mathcal{Y}} Pr[y_i|x, \mathcal{X}] = 1$$

On peut toujours diviser un nombre par 1 sans changer sa valeur. Donc,

$$\begin{aligned} Pr[y|x,\mathcal{X}] &= \frac{Pr[y|x,\mathcal{X}]}{1} \\ &= \frac{Pr[y|x,\mathcal{X}]}{\sum_{y_i\in\mathcal{Y}} Pr[y_i|x,\mathcal{X}]} \\ &= \frac{\frac{Pr[x|y,\mathcal{X}]Pr[y|\mathcal{X}]}{Pr[x,\mathcal{X}]}}{\sum_{y_i\in\mathcal{Y}} \frac{Pr[x|y_i,\mathcal{X}]Pr[y_i|\mathcal{X}]}{Pr[x|\mathcal{X}]}} \end{aligned}$$

soit finalement :

$$Pr[y|x,\mathcal{X}] = \frac{Pr[x|y,\mathcal{X}]Pr[y|\mathcal{X}]}{\sum_{y_i\in\mathcal{Y}} Pr[x|y_i,\mathcal{X}]Pr[y_i|\mathcal{X}]} \tag{4.3}$$

Dans cette équation, en appliquant ce que l'on a dit précédemment pour estimer la vraisemblance et la probabilité *a priori*, on peut calculer le terme de droite.

classe MAP

Une fois que l'on a calculé la probabilité d'appartenance à chacune des classes de la donnée x ($Pr[y|x,\mathcal{X}], \forall y \in \mathcal{Y}$), on peut prédire sa classe comme étant celle qui maximise la probabilité *a posteriori* : c'est la classe MAP (*Maximal A Posteriori*), soit :

$$y_{\text{MAP}} = \arg\max_{y\in\mathcal{Y}} Pr[y|x,\mathcal{X}] \tag{4.4}$$

qui peut s'écrire aussi en appliquant l'éq. (4.3) :

$$y_{\text{MAP}} = \arg\max_{y\in\mathcal{Y}} Pr[x|y,\mathcal{X}]Pr[y|\mathcal{X}] \tag{4.5}$$

classe ML

Si l'on ne tient pas compte de $Pr[y|\mathcal{X}]$ et que l'on ne tient compte que de la vraisemblance $Pr[x|y]$, on obtient la classe ML (*Maximal Likelihood*), soit :

$$y_{\text{ML}} = \arg\max_{y\in\mathcal{Y}} Pr[x|y,\mathcal{X}] \tag{4.6}$$

Clairement, si les exemples sont uniformément répartis entre toutes les classes, soit $Pr[y|\mathcal{X}] = \frac{1}{|\mathcal{Y}|}$, l'hypothèse ML et l'hypothèse MAP sont équivalentes.

4.2 Exemple

On applique ce que l'on vient d'exposer sur le jeu de données « jouer au tennis ? ». L'ensemble des exemples $\mathcal{X}$ est dans la table 3.1 au chapitre 3.

On souhaite prédire la classe de la donnée x = (Ciel = Ensoleillé, Température = Fraiche, Humidité = Élevée, Vent = Fort).

Pour cela, on utilise la règle de Bayes :

$$Pr[\text{jouer} = \text{oui}|x,\mathcal{X}] = \frac{Pr[x|\text{jouer} = \text{oui},\mathcal{X}]Pr[\text{jouer} = \text{oui}|\mathcal{X}]}{Pr[x|\mathcal{X}]}$$

TAB. 4.1 – Sur le jeu de données « jouer au tennis ? », pour chaque valeur de la classe, nombre d'exemples pour chacune des valeurs possibles des attributs.

attribut	valeur	jouer = oui	jouer = non
Ciel	Ensoleillé	2	3
	Couvert	4	0
	Pluie	3	2
Température	Chaude	2	2
	Tiède	4	2
	Fraiche	3	1
Humidité	Élevée	3	4
	Normale	6	1
Vent	Fort	3	3
	Faible	6	2
Jouer		9	5

Comme on l'a indiqué plus haut, sans autre information, on peut estimer le numérateur mais pas le dénominateur. Aussi, on utilise la formulation suivante :

$$Pr[\text{jouer} = \text{oui}|x, \mathcal{X}] = \frac{Pr[x|\text{jouer} = \text{oui}, \mathcal{X}]Pr[\text{jouer} = \text{oui}|\mathcal{X}]}{\sum_{y\in\{\text{oui, non}\}} Pr[x|\text{jouer} = \text{y}, \mathcal{X}]Pr[\text{jouer} = \text{y}|\mathcal{X}]}$$

Reste à évaluer les quatre termes : $Pr[x|\text{jouer} = \text{oui}, \mathcal{X}]$, $Pr[\text{jouer} = \text{oui}|\mathcal{X}]$, $Pr[x|\text{jouer} = \text{oui}, \mathcal{X}]$ et $Pr[\text{jouer} = \text{non}, \mathcal{X}]$ à partir du jeu d'exemples.

Pour cela, nous avons calculé les proportions d'exemples correspondant à chacune des valeurs de chacun des attributs (*cf.* la table 4.1). De là, on tire :

- $Pr[\text{jouer} = \text{oui}|\mathcal{X}]$ est estimée par $\frac{9}{14}$;
- $Pr[\text{jouer} = \text{non}|\mathcal{X}]$ est estimée par $\frac{5}{14}$;
- $Pr[x|\text{jouer} = \text{oui}|\mathcal{X}]$ est estimée en appliquant l'hypothèse de Bayes naïve, soit :

$$\begin{aligned}
& Pr[x|\text{jouer} = \text{oui}, \mathcal{X}] \\
= \quad & Pr[(\text{Ciel=Ensoleillé, Température=Fraiche, Humidité=Élevée,} \\
& \text{Vent=Fort})|\text{jouer} = \text{oui}, \mathcal{X}] \\
\overset{HBN}{=} \quad & Pr[\text{Ciel=Ensoleillé}|\text{jouer} = \text{oui}, \mathcal{X}]\times \\
& Pr[\text{Température=Fraiche}|\text{jouer} = \text{oui}, \mathcal{X}]\times \\
& Pr[\text{Humidité=Élevée}|\text{jouer} = \text{oui}, \mathcal{X}]\times \\
& Pr[\text{Vent=Fort}|\text{jouer} = \text{oui}, \mathcal{X}] \\
\propto \quad & \tfrac{2}{9} \times \tfrac{3}{9} \times \tfrac{3}{9} \times \tfrac{3}{9} \\
\propto \quad & \tfrac{2}{243}
\end{aligned}$$

où le symbole $\overset{HBN}{=}$ signifie que le membre droit de l'équation est égal au

membre gauche sous l'hypothèse de Bayes naïve et le symbole $\propto$ indique que le membre gauche est estimé par celui de droite ;

– de même, on estime également $Pr[x|\text{jouer} = \text{non}, \mathcal{X}]$:

$$
\begin{array}{lll}
Pr[x|\text{jouer} = \text{non}, \mathcal{X}] & & \\
= & Pr[(\text{Ciel=Ensoleillé, Température=Fraiche, Humidité=Élevée,} & \\
& \text{Vent=Fort})|\text{jouer=non}, \mathcal{X}] & \\
\overset{HBN}{=} & Pr[\text{Ciel=Ensoleillé}|\text{jouer=non}, \mathcal{X}]\times & \\
& Pr[\text{Température=Fraiche}|\text{jouer=non}, \mathcal{X}]\times & \\
& Pr[\text{Humidité=Élevée}|\text{jouer=non}, \mathcal{X}]\times & \\
& Pr[\text{Vent=Fort}|\text{jouer=non}, \mathcal{X}] & \\
\propto & \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} & \\
\propto & \frac{36}{625} &
\end{array}
$$

Donc :

$$
\begin{array}{lll}
Pr[\text{jouer=oui}|x, \mathcal{X}] & \propto & \frac{\frac{2}{243} \times \frac{9}{14}}{\frac{2}{243} \times \frac{9}{14} + \frac{36}{625} \times \frac{5}{14}} \approx 0.205 \\
Pr[\text{jouer=non}|x, \mathcal{X}] & \propto & \frac{\frac{36}{625} \times \frac{5}{14}}{\frac{2}{243} \times \frac{9}{14} + \frac{36}{625} \times \frac{5}{14}} \approx 0.795
\end{array}
$$

Le symbole $\approx$ indique que la valeur de la fraction (elle-même une estimation de la probabilité) située à sa gauche prend pour valeur décimale approchée celle qui est indiquée à sa droite.

4.3 Attributs numériques

Si un attribut est numérique, la détermination de la probabilité d'observation d'une valeur particulière ne peut pas se faire comme on vient de le présenter. Dans ce cas, en absence d'autre information, on suppose que la distribution de la valeur de l'attribut est normale. Pour cela, on calcule la moyenne et l'écart-type de chaque attribut numérique et pour chaque valeur de l'attribut cible (*cf.* table 4.2).

Rappelons que la fonction de densité de probabilité d'une distribution normale est :

$$Pr[X = x] = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\overline{x})^2}{2\sigma^2}} dx$$

Remarque

Dans cette équation, on peut en première approche ne pas comprendre ce dx... Le lecteur curieux pourra essayer de calculer, pour une variable normale centrée d'écart-type 0.1, la probabilité que celle-ci prenne la valeur 0. Qu'en pense alors le lecteur curieux ? S'il n'en pense rien, qu'il se rappelle qu'une probabilité est forcément positive et inférieure à 1.

TAB. 4.2 – Statistiques sur les attributs Température et Humidité

	Température		Humidité	
	oui	non	oui	non
	26.5	27.5	86	85
	20	25	96	90
	19	17.5	80	70
	17	21	65	95
	19.5	20.5	70	91
	22.5		80	
	22.5		70	
	21		90	
	25.5		75	
moyenne	21.5	22.3	79.1	86.2
écart-type	2.91	3.53	10.2	9.7

On peut alors calculer par exemple :

$$Pr[\text{Température} = 18|\text{jouer=oui}, \mathcal{X}] \propto \frac{1}{\sqrt{2\pi\ 2.91^2}}\ e^{-\frac{(18-21.5)^2}{2\ 2.91^2}} \approx 0.0665$$

$$Pr[\text{Température} = 18|\text{jouer=non}, \mathcal{X}] \propto \frac{1}{\sqrt{2\pi\ 3.53^2}}\ e^{-\frac{(18-22.3)^2}{2\ 3.53^2}} \approx 0.0538$$

et

$$Pr[\text{Humidité} = 90|\text{jouer=oui}, \mathcal{X}] \propto \frac{1}{\sqrt{2\pi\ 10.2^2}}\ e^{-\frac{(90-79.1)^2}{2\ 10.2^2}} \approx 0.0221$$

$$Pr[\text{Humidité} = 90|\text{jouer=non}, \mathcal{X}] \propto \frac{1}{\sqrt{2\pi\ 9.7^2}}\ e^{-\frac{(90-86.2)^2}{2\ 9.7^2}} \approx 0.0381$$

Dès lors, pour la journée J =(Ciel=Ensoleillé, Température=18, Humidité=90, Vent=Fort), on obtient :

$$Pr[\text{jouer=oui}|J, \mathcal{X}] \propto \frac{\frac{2}{3} \times 0.0665 \times 0.0221 \times \frac{3}{9} \times \frac{9}{14}}{Pr[J|\mathcal{X}]} \approx \frac{2.1\ 10^{-4}}{Pr[J|\mathcal{X}]}$$

$$Pr[\text{jouer=non}|J, \mathcal{X}] \propto \frac{\frac{3}{5} \times 0.0538 \times 0.0381 \times \frac{3}{5} \times \frac{5}{14}}{Pr[J|\mathcal{X}]} \approx \frac{2.635\ 10^{-4}}{Pr[J|\mathcal{X}]}$$

d'où :

$$Pr[\text{jouer=oui}|J, \mathcal{X}] \propto \frac{2.1\ 10^{-4}}{2.1\ 10^{-4} + 2.635\ 10^{-4}} \approx 0.44$$

et

$$Pr[\text{jouer=oui}|J, \mathcal{X}] \propto \frac{2.635\ 10^{-4}}{2.1\ 10^{-4} + 2.635\ 10^{-4}} \approx 0.56$$

4.4 Valeur d'attribut manquante

4.4.1 Absence de la valeur d'un attribut dans une donnée dont on veut prédire la classe

Dans ce cas, c'est très simple, on ne prend pas cet attribut en compte dans l'estimation de la probabilité.

Par exemple, si on veut prédire la classe de la donnée x = (Ciel = Ensoleillé, Humidité = Élevée), on écrira $Pr[x|y, \mathcal{X}] \propto Pr[\text{Ciel=Ensoleillé}|y, \mathcal{X}] \times Pr[\text{Humidité=Élevée}|y, \mathcal{X}]$.

4.4.2 Absence de la valeur d'un attribut dans le jeu d'apprentissage

probabilité nulle ...

La méthode de Bayes n'est pas applicable telle qu'elle vient d'être montrée si une certaine valeur d'attribut n'apparaît pas dans le jeu d'entraînement en conjonction avec chaque valeur de l'attribut cible. En effet, dans ce cas apparaissent des probabilités estimées à 0 et comme les probabilités sont multipliées les unes par les autres, on obtient 0 à la fin du compte.

Ainsi, il n'existe pas d'exemple négatif pour lequel « Ciel » = « Couvert » ; l'estimation de $Pr[\text{Ciel} = \text{Couvert}|\text{jouer=non}] \propto 0$. D'un point de vue conceptuel, cela n'a pas de sens d'estimer que cette probabilité soit nulle. D'un point de vue pratique, ce 0 pose problème.

et estimateur de Laplace

La solution consiste à s'arranger pour que cette estimation ne soit pas nulle. Pour cela, on utilise la technique dite de l' « estimateur de Laplace ». Pour cela, on ajoute une valeur (par exemple 1) à chaque décompte de la table 4.1. Par exemple, pour l'attribut Ciel, au lieu d'avoir les probabilités estimées 2/9, 4/9 et 3/9, on aurait 3/12, 5/12 et 4/12 en ajoutant 1 à chaque numérateur et en ajoutant 3 (car l'attribut Ciel prend 3 valeurs distinctes) au dénominateur. Plus généralement, on peut ajouter une valeur μ à chaque dénominateur pour un attribut donné et ajouter μ/arité de l'attribut considéré au numérateur.

Cette quantité, μ/arité de l'attribut considéré, peut être vue comme une probabilité *a priori* de l'observation de chacune des valeurs de l'attribut ; on n'est donc pas obligé d'avoir une même probabilité *a priori* pour chacune des valeurs de l'attribut, mais des valeurs $p_1, p_2, \ldots p_n$ pour les n valeurs possibles de l'attribut considéré, du moment que les $p_{i\in[1,n]}$ sont positifs et que leur somme vaut 1.

De cette manière, on voit comment intégrer une connaissance *a priori* dans la méthode de classification, ici la connaissance de la distribution des classes sur l'ensemble des données.

On peut maintenant estimer $Pr[\text{jouer} = \text{oui}|\text{Ciel} = \text{Couvert}]$:

$$
\begin{aligned}
Pr \quad & [\text{jouer} = \text{oui} | \text{Ciel} = \text{Couvert}] \\
& = \frac{Pr[\text{Ciel} = \text{Couvert} | \text{jouer} = \text{oui}] \times Pr[\text{jouer} = \text{oui}]}{Pr[\text{Ciel} = \text{Couvert}]} \\
& \propto \frac{4/9 \times 9/14}{Pr[\text{Ciel} = \text{Couvert}]} \\
Pr \quad & [\text{jouer} = \text{non} | \text{Ciel} = \text{Couvert}] \\
& = \frac{Pr[\text{Ciel} = \text{Couvert} | \text{jouer} = \text{non}] \times Pr[\text{jouer} = \text{non}]}{Pr[\text{Ciel} = \text{Couvert}]} \\
& \propto \frac{1/8 \times 5/14}{Pr[\text{Ciel} = \text{Couvert}]} \\
Pr \quad & [\text{jouer} = \text{oui} | \text{Ciel} = \text{Couvert}] \\
& \propto \frac{4/9 \times 9/14}{4/9 \times 9/14 + 1/8 \times 5/14}
\end{aligned}
$$

en prenant par exemple ici $\mu = 3$.

4.4.3 Application au jeu de données « iris »

On applique l'approche bayésienne qui vient d'être décrite au jeu de données « iris » que l'on a présenté plus haut (*cf.* chap. 3, sec. 3.10).

On n'utilise que les deux attributs longueur et largeur des pétales.

On suppose que la distribution des données est normale pour chacune des 3 classes. Les paramètres de ces 3 distributions sont les suivants :

classe	longueur des pétales		largeur des pétales	
	moyenne	écart-type	moyenne	écart-type
setosa	1.46	0.17	0.24	0.11
versicolor	4.26	0.47	1.33	0.20
virginica	5.55	0.55	2.03	0.27

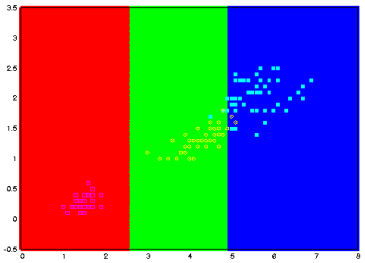
La vraisemblance d'appartenance d'une donnée à une classe est estimée de quatre manières différentes :

1. en caractérisant une donnée par la longueur L de ses pétales uniquement, donc en utilisant la distribution normale de la longueur des pétales que l'on suppose $\mathcal{N}(\overline{L}, \sigma_L)$:

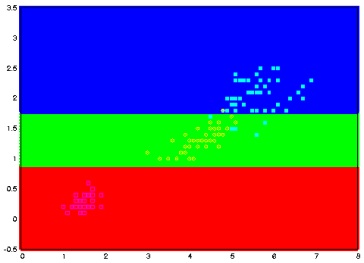
$$Pr[L | \mathcal{N}(\overline{L}, \sigma_L), \mathcal{X}] = \frac{1}{\sqrt{2\pi\sigma_L^2}} e^{-\frac{(L-\overline{L})^2}{2\sigma_L^2}}$$

2. en caractérisant une donnée par la largeur l de ses pétales uniquement, donc en utilisant la distribution normale de la largeur des pétales que l'on suppose $\mathcal{N}(\bar{l}, \sigma_l)$:

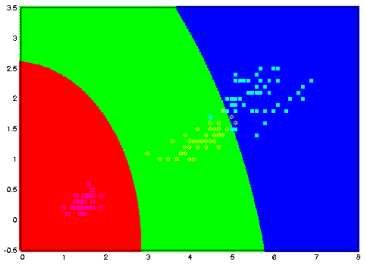
$$Pr[l | \mathcal{N}(\bar{l}, \sigma_l), \mathcal{X}] = \frac{1}{\sqrt{2\pi\sigma_l^2}} e^{-\frac{(l-\bar{l})^2}{2\sigma_l^2}}$$



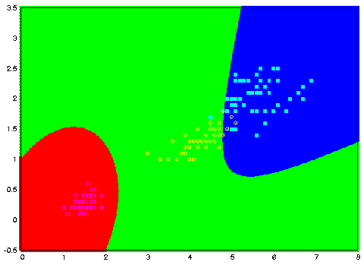
(a) On n'utilise que l'attribut longueur des pétales.



(b) On n'utilise que l'attribut largeur des pétales.



(c) On utilise les deux attributs longueur et largeur des pétales combinés de manière naïve.



(d) On utilise les deux attributs longueur et largeur des pétales et une distribution de probabilité bivariée.

FIG. 4.1 – Partitionnement de l'espace des données par la règle de Bayes en considérant différents attributs ou combinaison d'attributs. Sur chacun des 4 graphiques, la longueur des pétales est en abcisses, leur largeur en ordonnées ; les couleurs indiquent la classe MAP pour chaque donnée : rouge pour *setosa*, vert pour *versicolor* et bleu pour *virginica* ; enfin, on a représenté les 150 exemples dans des couleurs différentes selon leur classe : rose pour les *setosa*, jaune pour les *versicolor* et bleu clair pour les *virginica*.

3. en caractérisant une donnée par la longueur et la largeur de ses pétales et en utilisant l'hypothèse naïve de Bayes :

$$Pr[(L,l)|\mathcal{N}((\overline{L},\overline{l}),(\sigma_L,\sigma_l)),\mathcal{X}] = \frac{1}{2\pi\sigma_L\sigma_l} e^{-\frac{(L-\overline{L})^2}{2\sigma_L^2} - \frac{(l-\overline{l})^2}{2\sigma_l^2}}$$

4. en caractérisant une donnée par la longueur et la largeur de ses pétales et en utilisant une distribution de probabilité bivariée :

$$Pr[(x,y)|\mathcal{N}((0,0),(\sigma_x,\sigma_y)),\mathcal{X}] = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} e^{-\frac{x^2+y^2-2\rho xy}{2\sigma_x\sigma_y(1-\rho^2)}}$$

dans laquelle ρ est le coefficient de corrélation linéaire entre L et l et, pour simplifier les notations, on a pris $x = L - \overline{L}$ et $y = l - \overline{l}$.

Pour les 3 classes, ce coefficient de corrélation linéaire est :

classe	ρ
setosa	0.31
versicolor	0.79
virginica	0.32

La fig. 4.1 montre le partitionnement de l'espace des données résultant de chacune de ces 4 manières d'estimer la vraisemblance.

4.5 Exemple : classification de textes

En guise d'application de cette méthode de classification, intéressons-nous à la classification de textes. Disposant d'un ensemble de textes et d'un ensemble de classes, le but est de déterminer la classe la plus probable pour un texte. Les textes peuvent être des emails et la classe peut-être spam ou non-spam. Notons d'ailleurs que des filtres d'emails très performants fonctionnent exactement comme cela (*cf.* spam assassin ou bogofilter).

D'autres applications possibles, et couramment rencontrées, sont :

- des messages de *newsgroups* à classer dans leurs groupes respectifs ;
- des textes de dépêches d'agence de presse, chaque classe étant alors un sujet ;
- des pages *web* en fonction du fait qu'elles traitent de tel ou tel sujet ;
- des pages *web* en fonction de leur intérêt, ce qui, couplé à un moteur de recherche, peut permettre de trier les pages trouvées selon leur intérêt pour un utilisateur donné.

4.5.1 Représentation d'un texte

A priori, un texte est stocké dans un fichier ; c'est donc une suite de caractères, lettres, chiffres, ponctuation, espaces, ... Ce texte doit être mis sous

une forme telle qu'il puisse être traité par un classificateur de Bayes. Pour cela, chaque donnée (un texte) doit être décrite par un certain nombre d'attributs.

Plusieurs possibilités sont envisageables :

1. on définit un attribut par position dans le texte : 1$^{\text{er}}$ mot du texte, 2$^{\text{e}}$ mot du texte, ... Chaque attribut prend donc sa valeur dans un ensemble de mots possibles. Le problème ici est que si les textes ne sont pas tous strictement de la même longueur, le nombre d'attributs va être différent pour chaque texte ;
2. on définit un attribut par mot du vocabulaire et on lui affecte le nombre d'occurences de ce mot dans le texte (sac de mots). Encore plus simplement, les attributs ont une valeur booléenne et indiquent simplement la présence d'un mot particulier dans le texte. Dans ces deux cas, on ne se pré-occupe pas de l'ordre des mots dans le texte, ni même de l'organisation des mots dans le texte. Dans le deuxième cas, on ne se pré-occupe même pas de savoir si un mot apparaît souvent ou pas dans le texte ;
3. en suivant l'idée précédente, on peut ne pas inclure dans le vocabulaire les mots tels que les articles, pronoms, ... Cela réduit la taille de l'ensemble des mots considérés, ces mots n'ajoutent rien au sens du texte ;
4. d'autres possibilités seront vues plus loin (*cf.* chapitre 5, sec. 5.4).

Remarquons que dans ces différentes manières de représenter un texte, nous faisons totalement abstraction de l'ordre des mots, de la syntaxe et de la sémantique des textes.

4.5.2 Application de la règle de Bayes

Cette section reprend directement le travail de [Joachims, 1996].

On choisit de représenter un texte par, pour chaque mot d'un certain vocabulaire, le nombre d'occurence dans ce texte. Soit :

- $\mathcal{V}$ l'ensemble de mots considéré (un vocabulaire) ;
- $\mathcal{Y}$ l'ensemble des classes de textes ;
- un texte t est représenté par $|\mathcal{V}|$ attributs à valeur naturelle ;
- $n(m, y)$ est le nombre d'occurences du mot $m \in \mathcal{V}$ parmi tous les textes de classe $y \in \mathcal{Y}$;
- $n(y)$ est le nombre d'occurences de mots apparaissant dans l'ensemble des textes de classe $y \in \mathcal{Y}$, soit

$$n(y) = \sum_{m \in \mathcal{V}} n(m, y)$$

L'utilisation de la règle de Bayes va nous amener à calculer la probabilité d'observer une certaine classe de textes si on observe un certain texte : $Pr[y|t]$; on a immédiatement :

$$Pr[y|t,\mathcal{X}] = \frac{Pr[t|y,\mathcal{X}]Pr[y|\mathcal{X}]}{Pr[t|\mathcal{X}]}$$

dont il reste à estimer les différents termes, soit les $Pr[t|y,\mathcal{X}]$ et $Pr[y|\mathcal{X}]$.

$Pr[y|\mathcal{X}]$ est estimée par la proportion de textes de classe y dans l'ensemble des textes d'apprentissage $\mathcal{X}$.

Pour $Pr[t|y,\mathcal{X}]$, on va utiliser l'hypothèse de Bayes naïve : le texte considéré étant composé d'un certain ensemble de mots $m \in \mathcal{V}$, on simplifie cette probabilité avec $\Pi_{m\in\mathcal{V}} Pr[m|y,\mathcal{X}]$.

Reste à estimer $Pr[m|y,\mathcal{X}]$ avec le jeu d'apprentissage. On pourrait utiliser simplement l'équation :

$$Pr[m|y,\mathcal{X}] \propto \frac{n(m,y)}{n(y)}$$

c'est-à-dire, le nombre d'occurences du mot m dans les textes de classe y divisé par le nombre d'occurences de tous les mots apparaissant dans les textes de classe y.

Cependant, $n(m,y)$ peut être nul (tous les mots n'apparaissent pas forcément dans les textes de toutes les classes). Aussi, on utilise un estimateur de Laplace :

$$Pr[m|y,\mathcal{X}] \propto \frac{n(m,y)+1}{n(y)+|\mathcal{V}|}$$

Cela nous permet de calculer le numérateur de $Pr[y|t,\mathcal{X}]$ pour toutes les valeurs de classe possibles $y \in \mathcal{Y}$. Comme d'habitude, on n'a pas besoin d'évaluer le dénominateur qui est constant pour toutes les classes possibles. On affecte t à la classe :

$$y(t) = \arg\max_{y\in\mathcal{Y}} Pr[y|t,\mathcal{X}]$$

Appliquant cette méthode de classification, [Joachims, 1996] considère 20000 textes appartenant à 20 classes différentes. Il utilise les 2/3 des textes pour estimer les probabilités nécessaires à l'application de la règle de Bayes et teste sa méthode avec le tiers de textes restant. De l'ensemble des mots, il a retiré les 100 mots les plus fréquents et il reste ensuite 38500 mots différents ; un texte est donc représenté par 38500 attributs et sa classe. Sur le jeu de test, le taux d'erreur est de 11 %.

4.6 Critique

L'utilisation de la règle de Bayes fonctionne bien, comme elle vient d'être décrite, si :

– quand on utilise l'hypothèse naïve de Bayes, l'hypothèse d'indépendance des attributs est vérifiée. Celle-ci peut être contrôlée en calculant la matrice de corrélation entre les attributs (*cf.* ACP) et ses valeurs propres. Ainsi, on vérifie qu'il n'existe pas de corrélation linéaire entre les attributs ;
– l'hypothèse de distribution normale des valeurs des attributs numériques est vérifiée. Cela est plus difficile à vérifier car le fait que les valeurs observées parmi les exemples d'apprentissage soit ou non normalement distribuées n'implique pas que la valeur de l'attribut le soit vraiment.

De plus,

+ approche incrémentale : à chaque nouvel exemple ajouté, on peut raffiner les probabilités ;
+ possibilité d'insérer des connaissances *a priori*, ce qui est impossible dans les arbres de décision (du moins, ce n'est pas prévu de manière standard dans la méthode) ;
- nécessité de connaître de nombreuses probabilités ; sinon, elles peuvent être estimées avec le jeu d'exemples.

4.7 Logiciels libres

– `weka` contient une implantation d'un classificateur de Bayes (naïf ou non) `http://www.cs.waikato.ac.nz/~ml`
– `bayes` de Ch. Borgelt : `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html#bayes`

4.8 Exercices

Exercice 10 Sur le jeu de données « jouer au tennis ? » :

Question 1 : quelle est la classe prédite en utilisant la règle de Bayes d'une journée ensoleillé avec vent faible ?

Question 2 : quelle est la classe d'une journée ensoleillé, température de 23 ° C, humidité de 70 % et vent faible ?

Question 3 : quelle est la probabilité de jouer un jour où la température est de 23 ° C ?

Question 4 : quelle est la probabilité de jouer un jour où l'humidité est comprise entre 60 et 75 % ?

Exercice 11 La population P est un ensemble de champignons. Il y a deux classes : vénéneux et comestible. On s'intéresse à l'attribut binaire volve. On dispose des informations suivantes :

classe k	1 : vénéneux	2 :comestible
P(k)	0.05	0.95
P(volve/k)	0.9	0.2

Je ramasse les champignons si la règle de Bayes détermine leur classe MAP comme étant comestible.

Question 1 : Est-ce que je ramasse les champignons ayant une volve ? Appliqueriez-vous cette règle si vous alliez ramasser des champignons ?

On définit un coût pour tout couple de classes (k,i) noté cout(k,i). On définit alors le coût moyen de l'affectation à la classe k d'une description d de D par :

$$\text{coutMoyen}(k|d) = \sum_{i \in \{1,\ldots,c\}} cout(k,i) \times Pr[i|d].$$

La règle de décision du coût minimum est : choisir C_{coutmin} qui à toute description d associe la classe k qui minimise coutMoyen$(k|d)$.

On définit sur notre exemple les coûts suivants : $\text{cout}(1,1) = \text{cout}(2,2) = 0$, $\text{cout}(1,2) = 2$, $\text{cout}(2,1) = \infty$.

J'utilise la règle du coût minimum.

Question 2 : Est-ce que je ramasse les champignons ayant une volve ?

Exercice 12 On dispose d'une population P constituée d'un ensemble de pièces qui sont équitables, biaisées avec une probabilité 1/3 pour Face, ou encore biaisées avec une probabilité de 1/4 pour Face. Une expérience consiste à jeter une pièce 20 fois de suite. Au vu du résultat d'une telle expérience, on souhaite classifier la pièce. On considère donc les trois classes 1,2,3 qui correspondent à une probabilité de Face égale à respectivement 1/2, 1/3 et 1/4. On fera l'hypothèse a priori que les classes sont équiprobables. Une description est un mot de l'ensemble $\{P,F\}^{20}$, où P correspond à Pile et F à Face. Une procédure de classification doit associer à un mot de cet ensemble une classe. Soit la description :

d = PPFPPFFPFPPPFPPPPPPFP.

Question 1 : Trouver les formules de calcul des trois quantités $Pr[1|d]$, $Pr[2|d]$ et $Pr[3|d]$.

Question 2 : Comment d serait-elle classée si on utilise la règle de décision de Bayes ?

On décide de prolonger cette expérience, on relance cette même pièce 20 fois.

Question 3 : Indiquer un choix à faire sur les probabilités a priori qui serait plus intéressant que l'hypothèse initiale d'équiprobabilité.

Exercice 13 Les anglais et les américains orthographient le mot rigueur respectivement *rigour* et *rigor*. Un homme ayant pris une chambre dans un hôtel a écrit ce mot sur un bout de papier. Une lettre est prise au hasard dans ce mot ; c'est une voyelle.

Question 1 : sans autre information, quelle est, à votre avis, la probabilité que cet homme soit anglais ?

Question 2 : on sait que l'hôtel héberge 60 % d'américains, les autres clients étant anglais. Que devient la probabilité que cet homme soit anglais ?

Exercice 14 La nuit, vous êtes témoin d'un accident impliquant un taxi qui a pris la fuite. Dans la pénombre, vous pensez que le taxi est bleu. Tous les taxis de la ville sont bleux ou verts. Des savants ont calculé que distinguer, de nuit, la couleur bleue de la couleur verte est fiable à 75 %.

Question 1 : peut-on trouver la couleur la plus probable du taxi ?

Question 2 : même question si on sait que neuf taxis sur 10 sont verts ?

Exercice 15 Alice et Bob jouent au jeu suivant :

1. d'un jeu de carte, ils extraient les as et les deux de pique et de cœur qu'ils mettent en paquet ;
2. Alice prend deux cartes dans ce paquet ;
3. si elle a pris (au moins) un as, elle le dit à Bob.

Question 1 : quelle est, d'après Bob, la probabilité qu'Alice ait pris deux as ?

Question 2 : si, au lieu de dire « j'ai un as », Alice dit « j'ai l'as de pique » (en supposant qu'elle l'a vraiment), quelle est, selon Bob, la probabilité qu'elle ait les deux as ?

Question 3 : en quoi connaître la couleur de l'as renseigne-t-il plus que juste savoir qu'on possède un as ?

voir le poly [Denis and Gilleron, 2000].

Exercice 16 Exercice pour informaticien

Écrire un programme qui prédit la classe de données en utilisant la règle de Bayes qui ait la même interface que C4.5 (même format de fichier de données, même type d'interrogation, ...).

Chapitre 5

Classification à base d'exemples représentatifs

Contenu

Les k plus proches voisins

Pour commencer, on donne le principe intuitif de l'algorithmes des plus proches voisins (*k nearest neighbors*) :

1. on stocke les exemples tels quels dans une table ;
2. pour prédire la classe d'une donnée, on détermine les exemples qui en sont le plus proche ;
3. de ces exemples, on déduit la classe ou on estime l'attribut manquant de la donnée considérée.

On note à l'étape 2 l'utilisation d'une notion de proximité entre exemples. Il nous faut éclaircir cette notion.

5.1 Mesure de la dissimilarité entre deux données

La notion de proximité induit celle de distance. Cette notion entend formaliser celle de dissimilarité entre données. Mathématiquement parlant, une

distance

distance d est une application définie par :

$$\begin{aligned} d: \quad & \mathcal{D} \times \mathcal{D} \to \mathbb{R}^+ \\ & (x, y) \mapsto d(x, y) \end{aligned}$$

qui respecte les 3 propriétés suivantes $\forall (x, y, z) \in \mathcal{D}^3$:

- $d(x, y) = 0 \iff x = y$
- $d(x, y) = d(y, x)$ (symétrie)
- $d(x, z) \leq d(x, y) + d(y, z)$ (inégalité triangulaire[1])

dissimilarité

Pour sa part, une dissimilarité est également une application qui associe un réel positif à un couple de données. Cependant, on n'exige que les deux premières propriétés ci-dessus (l'inégalité triangulaire n'est pas exigée).

Il nous faut être capable de mesurer la dissimilarité entre 2 données. Notons x_i et x_j deux données. Pour des données dont les attributs sont tous quantitatifs, des mesures de dissimilarité souvent utilisées sont les trois suivantes :

- $L_1(x_i, x_j) = \sum_{k=1}^{k=P} |x_{i,k} - x_{j,k}|$
- $L_2(x_i, x_j) = \sum_{k=1}^{k=P} (x_{i,k} - x_{j,k})^2$
- $L_\infty(x_i, x_j) = \max_{k \in \{1,\ldots,P\}} |x_{i,k} - x_{j,k}|$

On distingue et discute ci-dessous le cas d'attributs numériques d'attributs nominaux.

5.1.1 Attribut numérique

Pour un attribut numérique, la dissimilarité introduite plus haut s'applique immédiatement.

ordre de grandeur des attributs

Cependant, un problème se pose si l'ordre de grandeur des attributs à combiner n'est pas le même pour tous ; naturellement, les attributs d'ordre de grandeur les plus grands vont dominer les autres.

mise à l'échelle

Pour éviter ce problème, il faut mettre à l'échelle les attributs en normalisant leur valeur. Si l'attribut a_i prend sa valeur dans l'intervalle $[\min(a_i), \max(a_i)]$, on utilise l'attribut normalisé :

$$\hat{a_i} = \frac{a_i - \min(a_i)}{\max(a_i) - \min(a_i)}$$

dont la valeur est comprise dans l'intervalle $[0, 1]$.

importance des attributs

[1] intuitivement parlant, cela signifie que le chemin le plus court entre deux points est le chemin direct entre les deux points.

D'autre part, certains attributs peuvent être plus importants que d'autres. On peut donc pondérer les attributs en fonction de leur importance.

5.1.2 Attribut nominal et attribut ordinal

La dissimilarité entre deux valeurs nominales doit être définie. On prend généralement :

- une dissimilarité nulle si la valeur de l'attribut est la même dans les deux données ;
- dissimilarité = 1 sinon.

Ainsi, on n'a pas à normaliser la valeur des attributs nominaux.

attribut !ordinal

Cependant, cette définition peut être un peu trop simple et n'est pas adaptée pour les attributs ordinaux. Ainsi, considérons un attribut « couleur » dont les valeurs possibles sont { rouge, jaune, rose, vert, bleu, turquoise }. On peut se dire que la dissimilarité entre bleu et rouge est plus grande qu'entre rouge et jaune. On peut alors essayer de se définir une mesure de dissimilarité qui en tienne compte.

5.1.3 Valeur d'attribut manquante

Si une valeur d'attribut est manquante dans une donnée, on considère généralement que la dissimilarité pour cet attribut est maximale.

5.2 L'algorithme des plus proches voisins

Tout cela ayant été précisé, on peut détailler le principe de l'algorithme donné plus haut.

5.2.1 Les k plus proches voisins

Détermination des k plus proches voisins

L'algorithme de détermination des k plus proches voisins est donné en 3. Dans cet algorithme, la fonction de mesure de dissimilarité δ est à définir en suivant les recommendations fournies dans la section précédente.

Prédire la classe d'une donnée

Ayant déterminé les k plus proches voisins, il reste à prédire la classe de la donnée x. Plusieurs stratégies sont envisageables. On suppose ci-dessous que l'on est dans un problème de classification binaire et on définit classe $(x) = +1$ pour une donnée x positive, classe $(x) = -1$ si elle est négative. On a alors par exemple :

Algorithme 3 Prédiction de la classe d'une donnée par la méthode des k plus proches voisins

Nécessite: 3 paramètres : un ensemble d'exemples $\mathcal{X}$ et une donnée $x \in \mathcal{D}$; $k \in \{1, ..., N\}$
pour chaque exemple $x_i \in \mathcal{X}$ **faire**
calculer la dissimilarité entre x_i et x : $\delta(x_i, x)$
fin pour
pour $\kappa \in \{1, ..., k\}$ **faire**
kppv $[\kappa] \leftarrow$ arg $\min_{i \in \{1,...,N\}} \delta(x_i, x)$
$\delta(x_i, x) \leftarrow +\infty$
fin pour
déterminer la classe de x à partir de la classe des exemples dont le numéro est stocké dans le tableau kppv

– la classe majoritaire parmi les k plus proches voisins, soit :
classe $(x) \leftarrow$ sgn $\sum_{\kappa=1}^{\kappa=k}$ classe (kppv[κ])

le terme apparaissant dans la somme est soit $+1$, soit -1 ; donc la somme elle-même est de signe positif s'il y a plus de positifs parmi les k plus proches voisins que de négatifs, et réciproquement. La fonction signe notée sgn est ici définie comme suit :

$$\text{sgn } (x) = \begin{cases} +1 \text{ si } x \geq 0 \\ -1 \text{ si } x < 0 \end{cases}$$

– la classe majoritaire parmi les k plus proches en pondérant la contribution de chacun par l'inverse de sa dissimilarité à x, soit :
classe $(x) \leftarrow$ sgn $\sum_{\kappa=1}^{\kappa=k} f(\delta(x_{\text{kppv}[\kappa]}, x))$ classe (kppv[κ]).
f peut être choisie de différentes manières :
  – normale : $f(\delta) = e^{-\delta^2}$;
  – inverse : $f(\delta) = 1/\delta$;
  – ...

Comme toujours en fouille de données, toute autre solution est envisageable, à partir du moment où elle est cohérente avec les données et la définition de leurs attributs.

Choisir k

Comment choisir k ? Il n'y a pas de méthode particulière sinon qu'il faut choisir le k qui va bien pour les données que l'on a !

D'une manière générale, on peut aussi prendre $k = N$ dans le cas où l'influence de chaque exemple est pondérée par sa dissimilarité avec la donnée à classer.

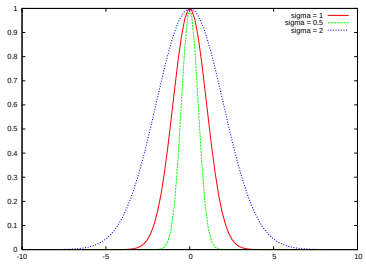


FIG. 5.1 – Le graphe de la fonction $e^{-\frac{d(x,y)}{2\sigma^2}}$, avec $\sigma \in \{0.5, 1, 2\}$. Plus σ est grand, plus large est le rayon d'influence. À l'extrême, si σ est nul, le rayon d'influence est nul : la fonction est nulle en tout point, sauf en 0 où elle est infinie.

Mesurer la similarité

Plutôt que mesurer la dissimilarité entre deux données, on peut naturellement mesurer leur similarité et inverser le raisonnement par rapport à ce que l'on vient de dire (en particulier, dans l'algorithme 3, il suffit de remplacer l'arg min par un arg max).

Notons $\bar{\delta}$ une fonction de similarité en général, celle-ci doit respecter :
- $\bar{\delta}(x, x)$ est maximal, voire infini ;
- $\bar{\delta}(x, y) = \bar{\delta}(y, x), \forall (x, y)$.

Une fonction populaire respectant ces deux propriétés est la fonction normale :

$$\bar{\delta}(x, y) = e^{-\frac{\delta(x,y)}{2\sigma^2}}$$

où $\delta(x, y)$ est une certaine distance ou dissimilarité entre $x \in \mathcal{D}$ et $y \in \mathcal{D}$. On en rappelle le graphe à la figure 5.1. Le paramètre σ permet de régler le rayon d'influence de chaque donnée : plus σ est grand, plus ce rayon est grand. Cette fonction porte aussi le nom de « fonction à base radiale »[2].

5.2.2 Application à « jouer au tennis ? »

On reprend l'application « jouer au tennis ? » avec le jeu de données comprenant des attributs numériques et d'autres nominaux (*cf.* table 3.2) sur lequel on applique l'algorithme des k plus proches voisins.

Dissimilarité entre deux données

On peut définir la dissimilarité entre deux données par :

[2] *radial basis function* en anglais, soit RBF en abrégé.

$$\delta^2(x_i, x_j) = \delta^2_{\text{Ciel}}(x_i, x_j) + \delta^2_{\text{Température}}(x_i, x_j) + \delta^2_{\text{Humidité}}(x_i, x_j) + \delta^2_{\text{Vent}}(x_i, x_j)$$

$\delta^2_{\text{Température}}(x_i, x_j)$ et $\delta^2_{\text{Humidité}}(x_i, x_j)$ sont aisément définies puisque ce sont des attributs numériques. On utilisera généralement la distance euclidienne, soit :

$$\delta^2_{\text{Température}}(x_i, x_j) = (x_{i,\text{Température}} - x_{j,\text{Température}})^2$$

pour la température.

Pour $\delta^2_{\text{Ciel}}(x_i, x_j)$, on peut la définir comme suit (par exemple) :

$$\delta^2_{\text{Ciel}}(x_i, x_j) = \begin{cases} 1 \text{ si la valeur de l'attribut Ciel diffère pour } x_i \text{ et } x_j \\ 0 \text{ sinon} \end{cases}$$

De même, on peut définir la dissimilarité concernant l'attribut « Vent ».

Il reste à régler le problème des ordres de grandeur. Tels que nous venons de définir les dissimilarités, elle vaut au plus 1 pour Ciel et Vent, peut valoir plus de 20 pour la température et l'humidité. Il peut être judicieux de multiplier les premiers par 20, ou diviser les seconds par 20.

5.2.3 Critique

- relativement lourd en terme de temps de calcul et d'utilisation d'espace mémoire si le nombre d'exemples est important ; nécessité d'utiliser des structures de données plus sophistiquées : Kd-tree ou autres ;
- difficulté de la définition de la bonne distance/dissimilarité ;
- problème s'il y a du bruit dans les données ;
- problème si la pertinence des attributs n'est pas uniforme pour tous.

Remarque : l'algorithme des k plus proches voisins est un algorithme dit « fainéant » parce qu'il ne réalise un traitement qu'au moment où on lui fournit une donnée à classer : il travaille à la demande.

5.3 Plus proches voisins sur le jeu de données « iris »

On applique l'algorithme des plus proches voisins sur le jeu de données « iris » que l'on a déjà rencontré plus haut (*cf.* chap. 3, sec. 3.10 et chap. 4, sec. 4.4.3).

Pour prédire la classe d'une donnée, on utilise k exemples, k variant entre 1 et tous les exemples. On détermine la similarité entre une donnée x et chaque exemple x_i par la fonction noyau gaussien :

$$e^{-\frac{||x-x_i||^2}{2\sigma^2}}$$

Pour chaque classe, on fait la moyenne de cette mesure de similarité ; la classe prédite est celle pour laquelle cette moyenne est la plus élevée, soit :

$$y(x) \leftarrow \arg\max_{y \in \{setosa, versicolor, virginiva\}} \frac{\sum_{\{x_i, y(x_i)=y\}} e^{-\frac{||x-x_i||^2}{2\sigma^2}}}{|\mathcal{X}_{\text{classe}=y}|}$$

La figure 5.2 montre le partitionnement de l'espace des données quand on fait varier le nombre de voisins considérés.

La figure 5.3 montre le partitionnement de l'espace des données quand le paramètre σ varie.

On pourra comparer ces résultats au partitionnement effectué par un classeur bayésien (*cf.* fig. 4.1 page 56) et à celui effectué par C4.5 (*cf.* fig. 3.8 page 40).

5.4 Plus proches voisins et classification de textes

L'algorithme des plus proches voisins a été utilisé pour faire de la classification de textes. Étant donnée sa simplicité, un fait expérimental des moins attendus sont ses excellentes performances sur certains jeux de données, ce qui en fait un algorithme générique plus performant que tous les autres, mêmes très sophistiqués, sur certains jeux de données (*cf.* voir Yang [1999]).

On reprend ce que nous avons vu au chapitre 4 à propos de la classification de textes, ainsi que les notations qui y ont été introduites.

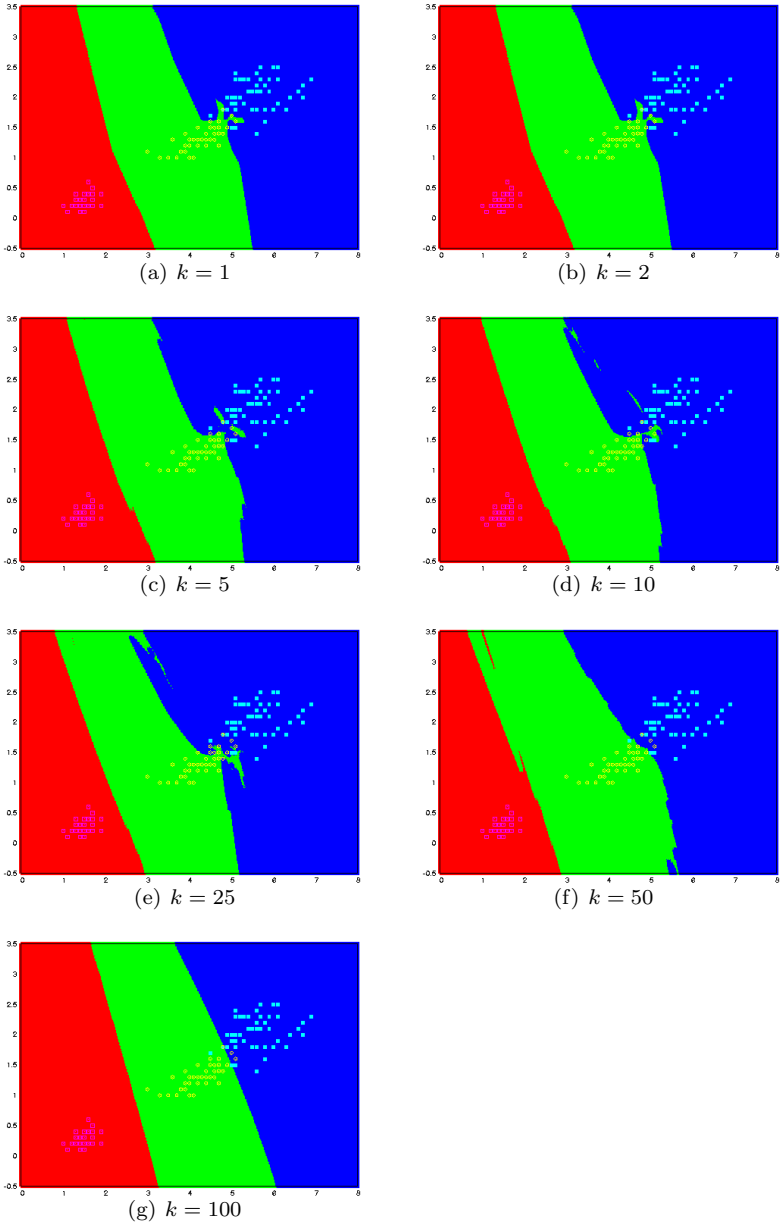
Un texte est représenté par un ensemble d'attributs, chaque attribut étant associé à un mot du vocabulaire $\mathcal{V}$ et ayant pour valeur le nombre d'occurences de ce mot dans le texte. On note $n(m,t)$ le nombre d'occurences du mot m dans le texte t.

Notons qu'un mot qui apparaît 2 fois dans un texte n'est pas forcément 2 fois plus important qu'un mot qui n'apparaît qu'une seule fois. Aussi, pour mieux tenir compte de cet effet non linéaire de l'importance des mots dans un texte en fonction de leur nombre d'occurences, on peut utiliser la quantité :

$$\begin{cases} 1 + \log(n(m,t)) \text{ si } n(m,t) \neq 0 \\ 0 \text{ sinon} \end{cases} \tag{5.1}$$

qui se nomme la fréquence du terme.

Une représentation fréquemment utilisée est dite « TF.IDF » (*Term frequency.Inverse Document Frequency*) ; dans cette représentation, l'attribut associé au mot m prend la valeur suivante :



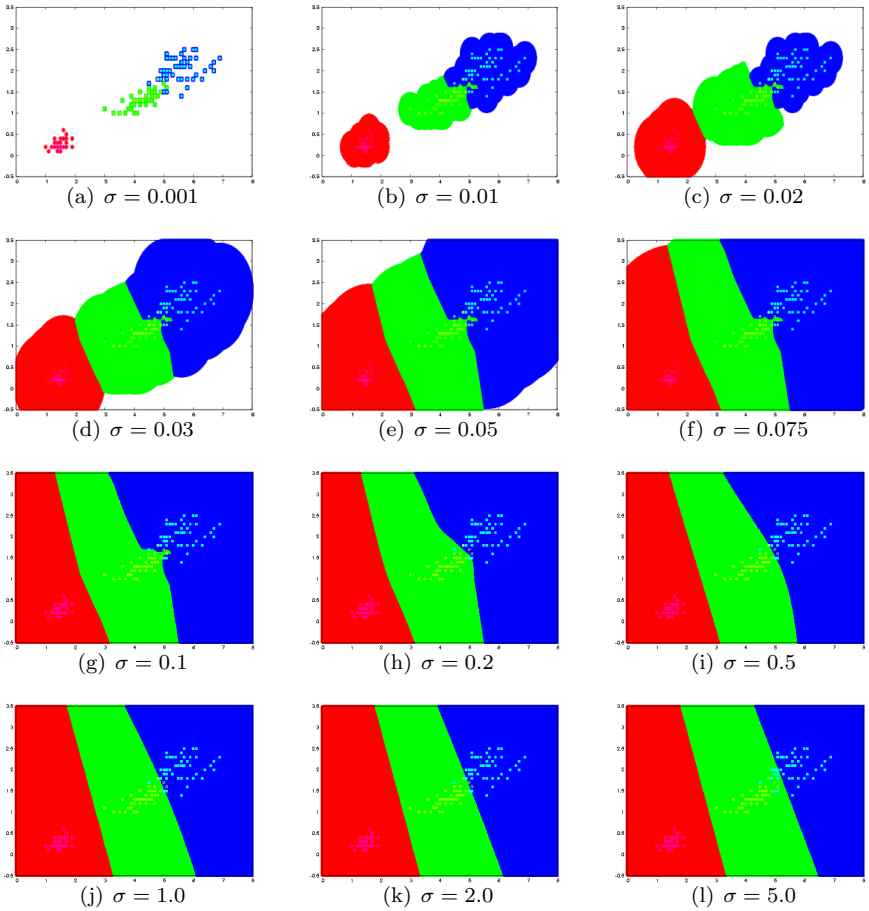
(a) $k = 1$ (b) $k = 2$

(c) $k = 5$ (d) $k = 10$

(e) $k = 25$ (f) $k = 50$

(g) $k = 100$

FIG. 5.2 – Partitionnement de l'espace des données par les plus proches voisins en considérant différentes valeurs du paramètre k, *i.e.*, le nombre de voisins. Sur chacun des graphiques, la longueur des pétales est en abcisses, leur largeur en ordonnées ; les couleurs indiquent la classe prédite pour chaque donnée : rouge pour *setosa*, vert pour *versicolor* et bleu pour *virginica* ; enfin, on a représenté les 150 exemples dans des couleurs différentes selon leur classe : rose pour les *setosa*, jaune pour les *versicolor* et bleu clair pour les *virginica*.



(a) $\sigma = 0.001$ (b) $\sigma = 0.01$ (c) $\sigma = 0.02$

(d) $\sigma = 0.03$ (e) $\sigma = 0.05$ (f) $\sigma = 0.075$

(g) $\sigma = 0.1$ (h) $\sigma = 0.2$ (i) $\sigma = 0.5$

(j) $\sigma = 1.0$ (k) $\sigma = 2.0$ (l) $\sigma = 5.0$

FIG. 5.3 – Partitionnement de l'espace des données par les plus proches voisins en considérant différentes valeurs du paramètre σ de la fonction noyau gaussien, en utilisant tous les exemples. Sur chacun des graphiques, la longueur des pétales est en abcisses, leur largeur en ordonnées ; les couleurs indiquent la classe prédite pour chaque donnée : rouge pour *setosa*, vert pour *versicolor* et bleu pour *virginica* ; enfin, on a représenté les 150 exemples dans des couleurs différentes selon leur classe : rose pour les *setosa*, jaune pour les *versicolor* et bleu clair pour les *virginica*. Pour les plus petites valeurs de σ, les zones blanches sont des zones où la précision des calculs est trop limitée pour déterminer la classe des points s'y trouvant (précision : 10^{-300}). Pour bien faire, il faudrait adapter notre programme pour qu'il puisse traiter des quantités plus petites que cela.

$$\begin{cases} 0, \text{ si } n(m,t) = 0 \\ (1 + \log(n(m,t))) \log \frac{N}{n(m)} \text{ sinon} \end{cases} \tag{5.2}$$

où :

– $n(m)$ est le nombre de textes dans lesquels le mot m apparaît ;

– N est le nombre de données, soit ici, le nombre de textes.

Le terme $\log \frac{N}{n(m)}$ est dénommée l'IDF : *Inverse Document Frequency*. $IDF(m)$ est faible si m apparaît dans beaucoup de textes, élevé s'il n'apparaît que dans un seul. Dans le premier cas, la présence de m ne donne aucune information puisqu'il est présent dans tous les textes ; dans le second, il indique des textes particuliers, donc, éventuellement, une classe particulière.

Un texte complet peut alors être représenté par un vecteur dont chaque composante correspond à un mot présent au moins une fois dans l'ensemble des textes d'apprentissage, cette composante prenant pour valeur le nombre d'occurences de ce mot dans ce texte particulier ou son IDF. On peut ensuite normaliser ce vecteur pour qu'il soit de longueur unité. Donc, la i^e composante du vecteur document, qui correspond au i^e terme du vocabulaire, est : $\frac{IDF(i)}{\sqrt{\sum_j IDF(j)^2}}$

Par exemple, le texte suivant (extrait du `man ls` de Linux) :

> « La commande ls affiche tout d'abord l'ensemble de ses arguments fichiers autres que des répertoires. Puis ls affiche l'ensemble des fichiers contenus dans chaque répertoire indiqué. dir et vdir sont des versions de ls affichant par défaut leurs résultats avec d'autres formats. »

fournit le corpus composé des racines[3] de mots : commande, affich, ensemble, argument, dir, fichier, ls, vdir, répertoire, contenu, indiqué, version, résultat, format. Le vecteur de ce document est indiqué à la table 5.1.

interprétation géométrique

En utilisant la représentation TF.IDF, un texte est représenté par un ensemble d'attributs (P) et de sa classe. Ces P attributs ont chacun une valeur positive : c'est un vecteur dans un espace à P dimensions. Ce vecteur peut aisément être normé. Dès lors, un texte est représenté par un vecteur situé dans le premier quadrant et tous les textes ont la même norme (ainsi, ils ont tous la même importance).

Dans cette représentation, les composantes correspondent aux proportions de mots dans les textes, une composante par mot. Des vecteurs ayant à peu près les mêmes coordonnées sont donc des textes à peu près composés des mêmes mots, dans les mêmes proportions (*cf.* fig. 5.4). Dans ce cas, on peut penser qu'ils appartiennent à la même classe. Donc, il est important de pouvoir déterminer

[3]les racines de mots permettent de considérer équivalentes toutes les formes accordées ou conjuguées d'un même mot comme. Pour chaque langue, il existe une méthode pour obtenir ces racines pour n'importe quel mot.

TAB. 5.1 – Vecteur représentatif du texte donné en exemple. Il y a autant de lignes que de racines de mot. La colonne i donne le numéro de la racine du mot, la deuxième donne sa racine, la troisième le nombre d'occurences du mot dans l'ensemble des textes, la quatrième donne son IDF (voir la définition dans le texte) et la dernière est la composante correspondante dans le vecteur-document du texte considéré.

i	mot	DF	IDF	$\overrightarrow{d}(i)$
1	affich	3	-1.09	0.56
2	argument	1	0.0	0.0
3	commande	1	0.0	0.0
4	contenu	1	0.0	0.0
5	dir	1	0.0	0.0
6	ensemble	2	-0.69	0.35
7	fichier	2	-0.69	0.35
8	format	1	0.0	0.0
9	indiqué	1	0.0	0.0
10	ls	3	-1.09	0.56
11	répertoire	2	-0.69	0.35
12	résultat	1	0.0	0.0
13	vdir	1	0.0	0.0
14	version	1	0.0	0.0

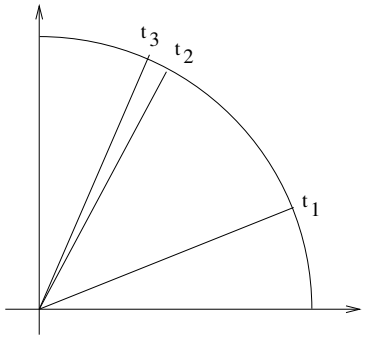


FIG. 5.4 – Représentation très schématique de l'espace des textes. Chaque axe de coordonnées correspond à un mot du vocabulaire. Dans un cas réel, ce schéma est de dimension $P \gg 2$. Un texte est représenté par un vecteur de norme 1 situé dans le 1^{er} quadrant. Deux textes proches (composés des mêmes mots en proportions similaires) forment un angle aigu (textes t_2 et t_3) alors que deux textes composés de termes différents forment un angle obtu (texte t_1 par rapport à t_2 et t_3).

cette proximité entre deux vecteurs, ce qui peut se faire par la détermination de l'angle entre les deux vecteurs. On peut en fait facilement obtenir le cosinus de cet angle : plus cette valeur est proche de 1, plus les textes associés ont une composition similaire. Rappelons que le cosinus de deux vecteurs est lié à leur produit scalaire par :

$$< \overrightarrow{V}, \overrightarrow{W} >= \| \overrightarrow{V} \| \times \| \overrightarrow{W} \| \times \cos(\widehat{\overrightarrow{V}, \overrightarrow{W}})$$

Si les vecteurs sont de norme unité, cette équation se simplifie en :

$$< \overrightarrow{V}, \overrightarrow{W} >= \cos(\widehat{\overrightarrow{V}, \overrightarrow{W}}) \tag{5.3}$$

et la bonne nouvelle est que le produit scalaire de deux vecteurs est une opération des plus simples à effectuer :

$$< \overrightarrow{V}, \overrightarrow{W} >= \sum_{i=1}^{i=P} V_i W_i$$

Donc, pour déterminer la dissimilarité entre deux textes x_i et x_j, il suffit de calculer leur produit scalaire $\sum_{k=1}^{k=P} a_{i,k} a_{j,k}$. Ensuite, en appliquant l'algorithme des plus proches voisins, on peut estimer la classe d'un texte à partir d'un ensemble d'exemples.

Par cette méthode très simple, Yang [1999] a obtenu d'excellents résultats pour la classification de textes.

5.5 Logiciel libre

- `weka` possède plusieurs implantations d'algorithmes de type « plus proches voisins » `http://www.cs.waikato.ac.nz/~ml`
- `rainbow` est un logiciel permettant de représenter des textes et de faire de la classification de textes avec différents algorithmes : `http://www.cs.cmu.edu/~mccallum/bow`. Outre les k plus proches voisins, `Rainbow` inclut un classeur de Bayes naïf, les MVS (*cf.* chap. 8) et d'autres algorithmes.

5.6 Exercices

Chapitre 6

Classeur à base de règles

Contenu

Dans ce chapitre, on s'intéresse à l'obtention d'un classeur composé de règles de classification : pour prédire la classe d'une donnée, on applique ces règles. Les règles sont construites à partir du jeu d'exemples $\mathcal{X}$.

Une règle de classification possède la forme générale :

$$\textbf{si } \text{condition}(x) \textbf{ alors } \text{classe}(x) = ...$$

où :
- x est une donnée ;
- condition(x) est une condition exprimée sur les attributs de la donnée x de la forme « attribut = valeur » ;
- ... est une valeur possible pour la classe : « vrai » ou « faux » si on est dans un problème de classification binaire par exemple.

L'utilisation d'une règle de classification est tout à fait intuitive : si la condition est vérifiée sur une certaine donnée, alors on en prédit sa classe.

Exemple : soit un ensemble de données, chaque donnée étant caractérisée par trois attributs $a_1 \in \{\alpha_1, \alpha_2, \alpha_3\}$, $a_2 \in \{\beta_1, \beta_2\}$ et $a_3 \in \{\gamma_1, \gamma_2\}$ et de classe « vrai » ou « faux ». Une règle de classification pourrait être :

$$\text{si } a_1 = \alpha_1 \text{ et } a_3 = \gamma_2 \text{ alors classe } \textit{gets} \text{ faux}$$

Pour déterminer la classe d'une donnée, on utilise un ensemble de règles de classification. Ces règles sont ordonnées (numérotées) ; pour classer une donnée,

on regarde si la donnée vérifie la condition de la première règle ; si elle est vérifiée, on applique la conclusion de la règle ; sinon, on regarde si la donnée vérifie la condition de la deuxième règle ; si elle est vérifiée, on applique sa conclusion ; *etc.*

Le problème est donc le suivant : disposant d'un jeu de données, induire un ensemble de règles de classification.

Différentes approches ont été proposées :

- on construit un arbre de décision que l'on transforme ensuite en règles ; c'est ce que fait `C4.5rules` par exemple, algorithme accompagnant C4.5 (*cf.* chap. 3) ;
- on génère directement des règles : exemples : Prism Cendrowska [1987], Ripper [Cohen, 1995], slipper [Cohen and Singer, 1999], ...
- on passe par des règles d'association.

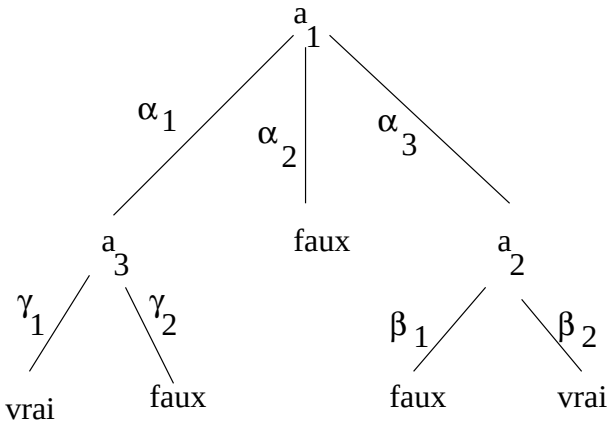
Nous examinons successivement 3 approches dans la suite : `C4.5rules`, Prism et l'obtention d'un jeu de règles en passant au préalable par des règles d'association.

Ce chapitre est présenté sous forme d'exercice.

6.1 Méthode « c4.5rules »

La première méthode que nous examinons est celle utilisée par c4.5rules. Elle consiste à construire un arbre de décision correspondant au jeu de données et d'en déduire un ensemble de règles de classification.

Par exemple, de l'arbre de décision représenté ci-dessous :



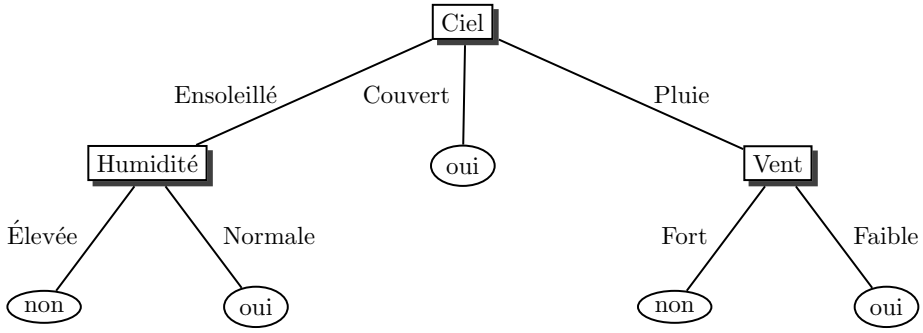
on peut déduire le jeu de règles de classification suivant :

1. **si** $a_1 = \alpha_1$ et $a_3 = \gamma_1$ **alors** classe = vrai
2. **si** $a_1 = \alpha_1$ et $a_3 = \gamma_2$ **alors** classe = faux
3. **si** $a_1 = \alpha_2$ **alors** classe = faux
4. **si** $a_1 = \alpha_3$ et $a_2 = \beta_1$ **alors** classe = faux

5. **si** $a_1 = \alpha_3$ et $a_2 = \beta_2$ **alors** classe = vrai

Pour obtenir ces règles, on a simplement pris chaque chemin entre la racine et une feuille ; on combine les tests présents dans les nœuds rencontrés sur le chemin par des « et » : cela fournit la condition de la règle ; la conclusion est l'étiquette de la feuille. On a produit les règles en effectuant ce traitement en parcourant l'arbre de décision de gauche vers la droite.

On illustre la suite sur le jeu de données « jouer au tennis ? » (*cf.* chap. 3, table 3.1). On rappelle que l'on a déduit l'arbre de décision suivant de ce jeu de données :



Question 1 : Donner l'ensemble de règles que l'on obtient à partir de cet arbre de décision en suivant la méthode indiquée ci-dessus.

Une fois ce jeu de règles obtenu, `c4.5rules` le simplifie. Pour cela, il élimine certaines règles ou en simplifie la condition. Nous expliquons maintenant comment se déroule cette simplification.

Soit une condition de la forme :

$$c_1 \text{ et } c_2 \text{ et } ... \ c_n$$

où les c_i sont des conditions sur les attributs (comme dans l'exemple ci-dessus).

Le principe est le suivant :

1. on estime la probabilité d'erreur $e_{c_1,c_2,...c_n}$ de la règle avec toutes ses conditions (c_1 et c_2 et ... c_n) ;
2. on estime la probabilité d'erreur $e_{c_1,c_2,...c_{n-1}}$ de la règle dont la condition est privée de sa dernière partie (ici c_n), soit la règle de condition c_1 et c_2 et ... c_{n-1} ;
3. si $e_{c_1,c_2,...c_{n-1}} \leq e_{c_1,c_2,...c_n}$, on simplifie la condition puisque la règle simplifiée commet moins d'erreur que la règle non simplifiée. Donc, la condition devient c_1 et c_2 et ... c_{n-1}. Et on recommence : on voit si l'on peut encore simplifier en retirant c_{n-1} en estimant, à nouveau, la probabilité d'erreur de cette nouvelle règle, ... On continue tant que cette estimation diminue ;

4. au contraire, si $e_{c_1,c_2,...c_{n-1}} > e_{c_1,c_2,...c_n}$, on laisse la règle inchangée. C'est terminé.

Comment estime-t-on la probabilité d'erreur d'une règle ? Comme on l'a vu au chap. 3, sec. 3.7.5 pour estimer l'erreur effectuée par un arbre de décision. On compte le nombre d'exemples n vérifiant la condition de la règle et on compte, parmi ceux-ci, le nombre d'exemples s dont la classe est bien prédite par la règle. Donc, le nombre d'exemples vérifiant la condition de la règle et mal classés est $f = n - s$. À partir du rapport f/n, en se donnant un taux de confiance c ($c = 75\%$ par exemple), on détermine un intervalle de confiance pour la probabilité de mal classer une donnée et on considère sa borne supérieure : cette borne supérieure est prise comme estimation de la probabilité recherchée.

À l'issue de la simplification de chacune des règles, on peut avoir plusieurs règles identiques (même condition, même conclusion) ; on n'en garde bien entendu qu'une seule.

Question 2 : Appliquer cette méthode de simplification au jeu de règles obtenu à la question précédente. On prendra $c = 75\%$.

Question 3 : Quelle est l'erreur d'apprentissage de ce jeu de règles ?

Question 4 : Quelle est l'erreur estimée du jeu de règles obtenu (on prendra encore un taux de confiance de 75 %) ?

6.2 Approche par couverture : l'algorithme Prism

Une deuxième méthode pour générer un jeu de règles de classification est la suivante. On cherche une règle de la forme :

$$\text{si ? alors classe} = \text{vrai}$$

où ? est une condition sur la valeur d'un attribut. Sur l'exemple donné plus haut, ? pourrait être :

- $a_1 = \alpha_1$
- $a_1 = \alpha_2$
- $a_1 = \alpha_3$
- $a_2 = \beta_1$
- $a_2 = \beta_2$
- $a_3 = \gamma_1$
- $a_3 = \gamma_2$

Pour chacune de ces possibilités, on détermine le rapport :

$$\frac{\text{nombre d'exemples couverts correctement classés}}{\text{nombre d'exemples couverts par la règle}}$$

où :

- le nombre d'exemples couverts par la règle est le nombre d'exemples qui vérifient la condition de la règle ;
- le nombre d'exemples couverts correctement classés est le nombre d'exemples qui vérifient la condition de la règle et dont la classe est celle présente en conclusion de la règle (vrai dans cet exemple).

Enfin, on choisit la condition pour laquelle ce rapport est maximal. Si plusieurs conditions ont même valeur pour ce rapport, on prend celle dont le numérateur est maximal. Si, à nouveau, plusieurs conditions sont à égalité, on en prend une au hasard.

Cette première étape fournit une règle de la forme :

$$\text{si } c_1 \text{ alors classe} = \text{vrai}$$

Ensuite, on essaie de trouver une deuxième condition pour obtenir une règle de la forme :

$$\text{si } c_1 \text{ ET } c_2 \text{ alors classe} = \text{vrai}$$

c_2 est l'une des conditions restantes (une fois que l'on a choisi c_1). On fait le choix de c_2 comme pour c_1.

Et on continue comme cela jusqu'à ce que le rapport :

$$\frac{\text{nombre d'exemples couverts correctement classés}}{\text{nombre d'exemples couverts par la règle}}$$

soit égal à 1.

À ce moment-là, on arrête : on a obtenu notre première règle R_1.

On retire du jeu d'apprentissage tous les exemples couverts par cette règle (ils sont nécessairement bien classés). Et on recommence tout ; cela va fournir une nouvelle règle R_2, puis R_3 ... jusqu'à ce qu'il n'y ait plus d'exemples positifs dans le jeu de données.

Ensuite, on refait exactement la même chose pour les exemples négatifs ce qui fournit un nouvel ensemble de règles.

Le jeu de règles de classification recherché est l'union de ces deux ensembles de règles (celles qui concluent sur classe = vrai et celles qui concluent sur classe = faux).

Cet algorithme fournit un ensemble de règles de classification.

Question 5 : Appliquer cet algorithme sur le jeu de données « jouer au tennis ? »

Question 6 : Quelle est l'erreur d'apprentissage de ce jeu de règles ?

Question 7 : Quelle est l'erreur estimée du jeu de règles obtenu (on prendra encore un taux de confiance de 75 %) ?

Question 8 : Que se passe-t-il s'il existe des données ayant la même description (même valeur pour tous leurs attributs) et de classe différente ? Proposer une solution pour y remédier.

6.3 Approche par règles d'association

Remarque : pour pouvoir lire et comprendre cette section, la lecture préalable du chap. 12 est indispensable.

Cette méthode consiste à construire des règles d'association. Pour cela, l'étape préalable consiste à constuire des ensembles d'items fréquents de plus en plus grands. Pour cela, on commence par des singletons, puis des couples, puis des ensembles de 3 items, ... Un item est de la forme attribut = valeur. Ici, on considère que la classe est un attribut comme un autre ; un item peut donc être classe = vrai. À chaque item, on associe son support, c'est-à-dire le nombre de fois où il apparaît dans le jeu d'exemples.

Question 9 : Donner la liste des ensembles d'1 item apparaissant dans le jeu de données « joue-je au tennis aujourd'hui ? » et, pour chacun, son support.

Dans la suite, on ne s'intéressera qu'aux ensemble d'items de support ≥ 3.

Quand on dispose des ensembles de n items dont le support est $\geq S$, un ensemble de $n+1$ items doit nécessairement avoir chacun de ses sous-ensembles de n items qui sont des ensembles de n items de support $\geq S$. Autrement dit, notons H, I, J, K et L cinq items ; supposons que $\{H, I, J\}$, $\{H, I, K\}$, $\{H, J, K\}$, $\{H, J, L\}$ et $\{I, J, K\}$ soient de support $\geq S$; alors, $\{H, I, J, K\}$ est peut-être un ensemble de 4 items de support $\geq S$ parce que ses sous-ensembles de 3 items ($\{H, I, J\}$, $\{H, J, K\}$ et $\{I, J, K\}$) sont également de support $\geq S$.

Cela étant dit, il faut ensuite vérifier que le support de l'ensemble d'items est bien $\geq S$ pour qu'il soit fréquent.

Question 10 : Donner la liste des ensembles de 2 items fréquents de support ≥ 3.

Question 11 : Idem pour les ensembles de 3, 4, 5 items fréquents de support ≥ 3.

Il reste à passer des ensembles d'items fréquents à des règles de classification. De l'ensemble d'items fréquents $\{H, I, J\}$ on peut tirer un ensemble de règles d'association :

- si H et I alors J
- si H et J alors I
- si I et J alors H
- si H alors I et J
- si I alors H et J
- si J alors H et I
- si $\emptyset$ alors H, I et J

À chaque règle, on peut associer sa précision :

$$\frac{\text{nombre d'exemples couverts pour lesquels la conclusion de la règle est correcte}}{\text{nombre d'exemples couverts par la règle}}$$

Pour des règles de classification, on ne s'intéresse qu'aux règles d'association dont la conclusion est de la forme classe = vrai ou classe = faux.

Question 12 : À partir des ensembles d'items fréquents calculés précédemment, donner la liste des règles de classification dont la précision est 1.

Question 13 : Quelle est l'erreur d'apprentissage de ce jeu de règles ?

Question 14 : Quelle est l'erreur estimée du jeu de règles obtenu (on prendra encore un taux de confiance de 75 %) ?

6.4 Synthèse

Question 15 : Parmi les 3 méthodes que nous venons d'étudier, laquelle semble la plus performante, c'est-à-dire, laquelle possède la plus faible probabilité d'erreur de classification telle qu'elle a été estimée dans les différentes parties du problème ?

6.5 Logiciels libres

- `C4.5rules` accompagne C4.5 ; il est disponible sur le site de quinlan : `http://www.cse.unsw.edu/~quinlan` ;
- différents algorithmes sont disponibles dans `weka`, dont Prism et Ripper ; `http://www.cs.waikato.ac.nz/~ml`
- slipper peut être obtenu auprès de son auteur pour des usages non commerciaux.

Chapitre 7

Classification par réseaux de neurones

Contenu

Il existe plusieurs types de réseaux de neurones. Nous décrivons ici l'utilisation de l'un d'entre-eux, le perceptron multi-couches. Un autre type de réseau de neurones, le réseau de Kohonen, est présenté au chap. 11 dans le cadre de l'apprentissage non supervisé.

Malgré leur nom, les réseaux de neurones n'ont qu'un très lointain rapport avec celui que nous avons dans notre tête, nous humains, ou même les vers de terre. Aussi, il ne faut pas s'attendre à des propriétés extraordinaires et miraculeuses que la difficulté d'appréhension du fonctionnement d'un réseau de neurones pourrait entraîner dans des esprits non prévenus.

Nous commençons par décrire l'unité de base d'un réseau de neurones, le neurone formel.

7.1 Le neurone formel

En terme de vocabulaire, on parlera indifféremment de neurone formel, d'unité et de perceptron. Dans ce qui suit, on considère que ces trois termes on la même signification.

7.1.1 Description d'un neurone formel

Nous considérons ici un type particulier de neurone formel, le perceptron. Un perceptron est une unité de traitement qui a les caractéristiques suivantes :

- il possède $P+1$ entrées que l'on notera $e_{i\in\{0,...P\}}$. L'entrée e_0 est particulière : on la nomme le « biais » ou « seuil » et elle vaut toujours 1[1] ;
- il possède une sortie notée s ;
- chacune des entrées est pondérée par un poids noté $w_{i\in\{0,...P\}} \in \mathbb{R}$;
- une fonction d'activation, notée $\varphi(.)$, détermine la valeur de s en fonction des valeurs présentes en entrée pondérées par leur poids : $\sum_{i=0}^{i=P} w_i e_i$, soit :

$$s = \varphi(\sum_{i=0}^{i=P} w_i e_i) \tag{7.1}$$

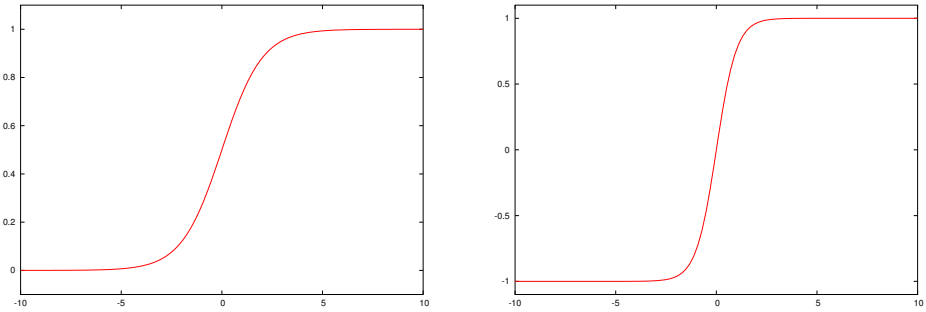
- on nommera « potentiel » la valeur $v = \sum_{i=0}^{i=P} w_i e_i$). Donc, la sortie s est obtenue par l'application de la fonction d'activation à ce potentiel.

Le fonctionnement d'un perceptron est très simple : il fonctionne de manière itérative et, à chaque itération, calcule sa sortie en fonction de ses entrées. Dans une tâche de classification, cette sortie (numérique forcément) indique la classe prédite pour la donnée qui avait été placée en entrée sur les $e_{i\in\{1,...P\}}$.

Notons que la sortie du perceptron ne dépend que des poids $w_{i\in\{0,...P\}}$ (une fois l'entrée fixée bien entendu). Aussi, l'apprentissage consiste ici à trouver la valeur des poids qui fait en sorte que lorsqu'une donnée est placée sur l'entrée du perceptron, la sortie prédit correctement sa classe. Donc, pour un perceptron, apprendre signifie fixer la valeur de ces $P+1$ paramètres réels.

Pour une tâche de classification, il est d'usage d'utiliser un perceptron dont la sortie $s \in \{0, 1\}$ ou un perceptron dont la sortie $s \in \{-1, 1\}$. D'un point de vue théorique, utiliser l'un ou l'autre ne change rien ; d'un point de vue pratique, c'est moins simple.

[1] si on veut entrer dans les subtilités de vocabulaire qui n'ont aucune conséquence sur le reste, on précisera que si c'est un biais, e_0 vaut 1 et que si c'est un seuil, e_0 vaut -1. Cela n'a aucune importance.



(a) fonction logistique (b) fonction tanh

FIG. 7.1 – Aspect de la fonction logistique $\frac{1}{1+e^{-v}}$ et de la fonction $\tanh(v)$. Bien noter l'axe des ordonnées : à gauche, la fonction varie entre 0 et 1 ; à droite, elle varie de -1 à 1. Notons que dans les deux cas, la pente à l'origine est ajustable et rendre le 'S' plus ou moins aplati ; à l'extrême, la pente tend vers une verticale et donc, la fonction de Heaviside n'est qu'un cas limite de fonction sigmoïde.

La fonction d'activation peut être la fonction de Heaviside :

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ 0 \text{ si } v < 0 \end{cases} \tag{7.2}$$

si $s \in \{0, 1\}$, ou

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ -1 \text{ si } v < 0 \end{cases} \tag{7.3}$$

si $s \in \{-1, 1\}$.

Ce peut être aussi une fonction sigmoïde[2] dite « logistique » (*cf.* fig. 7.1(a)) :

$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

avec $a \in \mathbb{R}$ si $s \in [0, 1]$, ou tangente hyperbolique (*cf.* fig. 7.1(b)) :

$$\varphi(v) = \tanh(v)$$

si $s \in [-1, 1]$.

Ces deux dernières fonctions d'activation ont l'immense avantage d'être continues et dérivables autant de fois que l'on veut sur $\mathbb{R}$ (*cf.* fig. 7.1). De plus, leurs dérivées s'expriment facilement en fonction d'elles-mêmes ce qui nous sera utile un peu plus loin.

[2]sigmoïde = en forme de 'S' aplati

Enfin, ce peut être simplement la fonction linéaire $\varphi(v) = v$. Dans ce cas, la sortie du neurone est la somme pondérée des entrées. On parle alors de neurone « linéaire ». La sortie prend alors une valeur dans $\mathbb{R}$.

prédiction

Pour prédire la classe d'une donnée placée en entrée, on regarde la sortie du perceptron (classification binaire). Si on a utilisé une fonction d'activation de type seuil, chacune des deux valeurs possibles en sortie correspond à une classe. Si on a utilisé une fonction d'activation logistique (resp. tanh), on peut dire que la classe est donnée par la valeur extrême la plus proche, 0 (resp. -1) ou 1 (resp. 1). Avec un neurone linéaire, on peut classer l'entrée en fonction du signe de la sortie.

interprétation géométrique

Notons tout de suite que l'équation $s = \sum_{i=0}^{i=P} w_i e_i$ définit un hyperplan dans un espace à $P+1$ dimensions, chacune correspondant à un attribut. Comme on l'a dit plus haut, notre objectif ici est de déterminer les w_i pour prédire correctement la classe des données placées en entrée du perceptron. Une fois correctement appris, ces poids déterminent donc un hyperplan $\sum_{i=0}^{i=P} w_i e_i$ qui sépare les exemples positifs des exemples négatifs : idéalement, ils sont de part et d'autre de l'hyperplan. Cependant, il arrive, et c'est le cas général, que cet hyperplan n'existe pas : dans ce cas, on dit que les exemples ne sont pas « linéairement séparables ». Dans ce cas, on doit utiliser plusieurs perceptrons, autrement dit, un « réseau de perceptrons ». On commence par présenter l'apprentissage des poids par un perceptron avant de passer aux réseaux de neurones.

7.1.2 Apprentissage des poids d'un perceptron

Cas séparable

On peut trouver les w_i par inversion de matrice, mais chacun sait que dans la pratique, l'inversion d'une matrice est quelque chose à éviter à tout prix. Un autre algorithme simple pour trouver ces w_i est la règle d'apprentissage du perceptron (*cf.* algorithme 4).

Le passage de l'ensemble des exemples (la boucle pour de la règle d'apprentissage du perceptron) se nomme un « épisode ».

Notons bien que les exemples sont « mélangés » avant la boucle **pour**, ce qui signifie que cette boucle ne les passe pas toujours dans le même ordre. Ce point est très important pour que l'algorithme fonctionne.

Notons aussi que ce faisant, nous introduisons du hasard dans cet algorithme : dans ce cours, c'est la première fois que nous étudions un algorithme dont le fonctionnement repose sur du hasard. Une autre source de hasard ici est l'initialisation des poids. Ce type d'algorithme dont le fonctionnement repose au moins en partie sur le hasard est qualifié de « non déterministe », ou encore « stochastique ». Tous les algorithmes concernant les réseaux de neurones sont non déterministes. Une conséquence de cela est que deux exécutions successives

Algorithme 4 Règle d'apprentissage du perceptron

Nécessite: les N exemples d'apprentissage $\mathcal{X}$. Chaque exemple x_i possède P attributs dont les valeurs sont notées $x_{i,j}$. Pour chaque donnée, $x_{i,0}$ est un attribut virtuel, toujours égal à 1. La classe de l'exemple x_i est y_i.
Nécessite: taux d'apprentissage $\alpha \in]0, 1]$
Nécessite: un seuil ϵ
  initialiser les w_i aléatoirement
  répéter
    // E mesure l'erreur courante
    $E \leftarrow 0$
    mélanger les exemples
    pour tous les exemples du jeu d'apprentissage $\mathcal{X}$ **faire**
      $E_i \leftarrow y_i - \varphi(\sum_{j=0}^{j=P} w_j x_{i,j})$
      $E \leftarrow E \ + |E_i|$
      pour tous les poids $w_k, k \in \{0, 1, ...P\}$ **faire**
        $w_k \leftarrow w_k + \alpha \ E_i \ x_{i,k}$
      fin pour
    fin pour
  jusque $E < \epsilon$

de l'algorithme sur le même jeu de données donne généralement des résultats différents.

Un résultat rassurant :

Propriété 1 *dite « Théorème de convergence du perceptron » : la règle d'apprentissage du perceptron converge pour les problèmes linéairement séparables en un nombre fini d'itérations.*

Par « converge », on veut dire qu'au bout d'un certain nombre d'itérations, les corrections apportées aux poids sont nulles : les poids ne varient plus ; la suite de valeurs prises par les poids a convergé.

Deux remarques :

- un apprentissage peut ne pas être robuste et être remis en cause par un nouvel exemple ⇒ difficulté de trouver un ensemble d'exemples qui permette l'apprentissage au mieux ;
- des exemples peuvent être mal classés à cause de bruit, d'erreurs. Dans ce cas, l'algorithme d'apprentissage ne converge pas.

Ainsi, même dans le cas où un jeu de données pourrait être linéairement séparable, celui-ci peut être non séparable linéairement à cause du bruit dans les données. Aussi, on passe maintenant au cas réellement intéressant, celui du cas non linéairement séparable.

Remarque

Il existe un algorithme numérique, l'algorithme de Ho et Kashyap, qui permet de déterminer si un jeu d'exemples est linéairement spéarable. Dans ce cas, cet algorithme fournit un hyperplan. Pourquoi n'utilise-t-on pas cet algorithme pour trouver les poids du perceptron ? Parce qu'il est numériquement lourd.

Exemples non linéairement séparables

Dans le cas où les exemples ne sont pas linéairement séparables, on cherche à minimiser les erreurs de classification. Dans ce cas, on utilise un neurone dont la sortie est continue, par exemple $s \in [-1, 1]$, indiquant ainsi que l'exemple est plus ou moins dans l'une des deux classes. On considère ici un neurone linéaire dont la sortie prend la valeur $s = \sum_{j=0}^{j=P} w_j x_{i,j}$ pour la donnée x_i.

Il faut maintenant définir cette notion d'erreur, par exemple en essayant de minimiser le nombre d'exemples mal classés.

On peut noter $\overrightarrow{w}$ l'ensemble des $P+1$ poids $w_0, ..., w_P$ du neurone mis sous forme vectorielle :

$$\overrightarrow{w} = \begin{pmatrix} w_0 \\ w_1 \\ ... \\ w_P \end{pmatrix}$$

mesure de l'erreur

On peut alors se donner :

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{i=1}^{i=N} (y_i - \varphi(x_i))^2 \tag{7.4}$$

qui représente l'erreur d'apprentissage due aux poids $\overrightarrow{w}$. Si la fonction d'activation est linéaire, cette erreur E s'écrit :

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{i=1}^{i=N} (y_i - \sum_{j=0}^{j=P} w_j x_{i,j})^2 \tag{7.5}$$

correction des poids

On voit donc que l'erreur E est un paraboloïde dans l'espace des poids. Donc, E possède un seul minimum et il s'agît de trouver la valeur des poids correspondant à ce minimum. Pour cela, la technique classique consiste à utiliser un algorithme de descente de gradient (*cf.* fig. 7.2). Pour cela, on part d'un point (dans l'espace des poids, espace de dimension $P+1$) $\overrightarrow{w}_0$; ensuite, itérativement, on corrige ce point pour se rapprocher du minimum de E. Pour cela, on corrige chaque poids d'une quantité proportionnelle au gradient de E en ce point, cela dans chaque direction, donc par rapport à chacun des poids. Ce gradient est donc la dérivée (partielle) de E par rapport à chacun des poids. Reste à déterminer le coefficient à appliquer à cette correction. En principe, une bonne valeur dépend

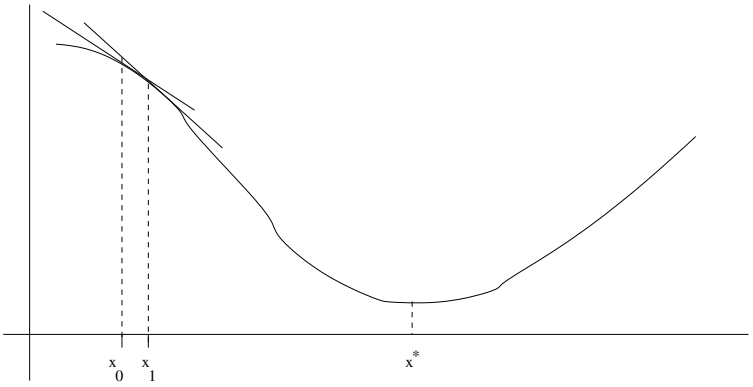


FIG. 7.2 – Principe de la méthode de descente de gradient pour trouver le minimum d'une fonction. Partant d'un $x_{i=0}$ initial, on passe itérativement de x_i à x_{i+1} en corrigeant x_i d'une quantité proportionnelle à la pente de la fonction en x_i (ce qui est indiqué par le segment de droite), autrement dit, la dérivée de la fonction en x_i. Ainsi, la méthode converge vers le minimum noté x^* sur la figure. Naturellement, l'algorithme converge vers un minimum local et l'on n'a aucune garantie qu'il converge vers un optimum global. Cependant, l'intérêt de cette méthode est qu'elle permet d'atteindre un minimum de n'importe quelle fonction, pas seulement pour les fonctions dont on a une expression analytique, du moment que l'on peut calculer sa dérivée.

de la dérivée seconde (dans ce cas, on a une méthode de Newton qui est donc un cas plus élaborée de méthode de gradient). Pour faire plus simple, on peut utiliser un coefficient arbitraire $\alpha \in [0, 1]$.

Nous devons donc calculer le gradient de E par rapport à $\overrightarrow{w}$ (noté $\nabla E(\overrightarrow{w})$), c'est-à-dire, la dérivée partielle de E par rapport à chacun des $P+1$ poids : $\frac{\partial E}{\partial w_j}$ pour $j \in \{0, 1, ...P\}$:

$$\nabla E(\overrightarrow{w}) = (...\frac{\partial E}{\partial w_j}...)^T$$

Quand on disposera de ce terme, on effectuera la correction de $\overrightarrow{w}$ par :

$$\overrightarrow{w} \leftarrow \overrightarrow{w} - \underbrace{\alpha \nabla E(\overrightarrow{w})}_{\Delta(\overrightarrow{w})} \tag{7.6}$$

Il nous reste à calculer ce terme général $\frac{\partial E}{\partial w_j}$:

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}[\frac{1}{2}\sum_{i=1}^{i=N}(y_i - \varphi(x_i))^2]$$

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\frac{\partial}{\partial w_j}[\sum_{i=1}^{i=N}(y_i - \varphi(x_i))^2]$$

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\sum_{i=1}^{i=N}\frac{\partial}{\partial w_j}[(y_i - \varphi(x_i))^2]$$

$$\frac{\partial E}{\partial w_j} = \frac{1}{2}\sum_{i=1}^{i=N} 2(y_i - \varphi(x_i))\frac{\partial}{\partial w_j}[(y_i - \varphi(x_i))]$$

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{i=N}(y_i - \varphi(x_i))\frac{\partial}{\partial w_j}[(y_i - \sum_{k=0}^{k=P} w_k x_{i,k}]$$

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{i=N}(y_i - \varphi(x_i))(-x_{i,j})$$

d'où :

$$\Delta w_j = \alpha \sum_{i=1}^{i=N}(y_i - \varphi(x_i))x_{i,j} \tag{7.7}$$

Maintenant que l'on sait quelle correction apporter aux poids, on en tire l'algorithme d'apprentissage des poids d'un perceptron dans le cas d'exemples non séparables. On en présente en fait deux :

- la régle de gradient standard (*cf.* algo. 5) qui réalise un traitement par lot (*batch*) : dans ce cas, on présente tous les exemples, on cumule l'erreur sur l'ensemble des exemples et on corrige les poids à partir de cette erreur cumulée ;
- la régle de gradient stochastique, ou règle Δ, ou règle Adaline, ou règle de Widrow-Hoff, ou règle des moindres carrés (*Least-Mean-Square* : LMS) (*cf.* algo. 6). Dans ce cas, on corrige les poids après présentation de chacun des exemples en entrée.

Notons bien que le calcul réalisé précédemment pour la correction des poids s'appuie sur une évaluation de l'erreur sur l'ensemble des exemples, donc la version standard de l'algorithme. Cependant, on constate expérimentalement que la règle Δ donne de bons résultats. L'intérêt de la règle Δ est qu'elle ne nécessite pas de disposer de l'ensemble des exemples une fois pour toute ; elle permet un apprentissage incrémental, lorsque les exemples sont disponibles ; de plus, des poids ayant été appris avec un jeu d'exemples, ils peuvent être améliorés par la suite si de nouveaux exemples sont acquis. Dans le cas de la règle de gradient standard, tout l'apprentissage doit être recommencé lorsque le jeu d'exemples change. Enfin, dans le cas où l'on dispose de l'ensemble du jeu d'exemples, on dispose de méthodes bien plus performantes pour apprendre les poids que la règle standard. En résumé, la règle standard n'a un intérêt que du point de vue théorique car c'est le cas que l'on est capable d'analyser formellement ; dans la pratique, on ne l'utilise pas. Par contre, la règle Δ, elle,

est très utilisée.

Remarque

Les méthodes alternatives à la règle de gradient standard évoquées ci-dessus sont des méthodes venant de l'algèbre linéaire. En effet, quand on cherche un jeu de poids $\overrightarrow{w}$ permettant d'approximer au mieux un jeu d'exemples, on peut disposer les exemples dans une matrice X dans lequel chaque ligne est occupée par un exemple et chaque colonne par un attribut (ce sont les $x_{i\in\{1,\ldots,N\},j\in\{1,\ldots P\}}$) ; on peut disposer les valeurs attendues en sortie pour chaque exemple dans un vecteur S dans lequel $s_i = y_i$ et l'on doit trouver $\overrightarrow{w}$ qui minimise la différence entre $X\overrightarrow{w}$ et S, soit $||S - X\overrightarrow{w}||$. C'est un problème classique d'algèbre linéaire qui est très bien résolu par l'algorithme du gradient conjugué (*cf.* Shewchuk [1994]).

Le problème avec l'approche présentée plus haut est que la correction apportée à chaque poids dépend uniquement du gradient de l'erreur. Aussi, la correction apportée à un poids ne dépend pas des corrections apportées aux autres poids. Hors, pour que la descente de gradient fonctionne bien, il faut que la correction apportée à un poids soit liée à la correction apportée aux autres poids. En effet, il faut que globalement, les poids aient des corrections qui soient à peu près du même ordre de grandeur. Outre le gradient conjugué mentionné plus haut, d'autres algorithmes ont été conçus dans cet esprit, en particulier le QuickProp et le RProp qui ont de bonnes performances expérimentales. On pourra consulter Gibb [1996].

Notons que ces deux algorithmes (gradient standard et règle Δ) sont très proches de la règle d'apprentissage du perceptron. La seule différence tient dans le fait que la sortie du perceptron est ici continue et non discrète.

taux d'apprentissage

Il faut que le taux d'apprentissage diminue au cours de l'apprentissage. Par exemple, $\alpha = 1/t$ où t est le numéro de l'itération courante du répéter/jusque. On peut prendre aussi $\alpha = 1/\sqrt{t}$.

Propriété 2 *La règle Δ converge asymptotiquement, que les données soient séparables ou non.*

Notons que la convergence est réalisée vers un optimum local.

Notons aussi que la règle Δ peut ne pas converger vers un séparateur séparant correctement les positifs des négatifs (même si les exemples sont linéairement séparables).

L'initialisation des poids étant aléatoire et celle-ci influençant le point de convergence, il est bon de recommencer plusieurs fois l'exécution de l'algorithme avec des initialisations différentes à chaque fois.

7.1.3 Illustration sur les iris

On illustre sur un exemple l'apprentissage réalisé par la règle Δ. On utilise pour cela un jeu de données dénommé « iris » que l'on a déjà rencontré précédemment.

Rappelons que ce jeu de données contient 150 exemples ; chaque exemple est une fleur (un iris) d'une des trois variétés suivantes : *setosa*, *versicolor* et

Algorithme 5 Règle de gradient standard

Nécessite: les N instances d'apprentissage $\mathcal{X}$
Nécessite: taux d'apprentissage $\alpha \in]0, 1]$
Nécessite: un seuil ϵ
  initialiser les $w_{j,j\in\{0,\ldots,P\}}$ aléatoirement
  répéter
    // E mesure l'erreur courante
    $E \leftarrow 0$
    pour tous les poids $w_{j,j\in\{0,\ldots,P\}}$ **faire**
      $\Delta(w_j) \leftarrow 0$
    fin pour
    mélanger les exemples
    pour tous les exemples du jeu d'apprentissage $x_{i,i\in\{1,N\}}$ **faire**
      $E \leftarrow E + (y_i - \varphi(x_i))^2$
      pour tous les poids $w_{j,j\in\{0,P\}}$ **faire**
        $\Delta(w_j) \leftarrow \Delta(w_j) + \alpha(y_i - \varphi(x_i))x_{i,j}$
      fin pour
    fin pour
    pour tous les poids $w_{j,j\in\{0,P\}}$ **faire**
      $w_j \leftarrow w_j + \Delta(w_j)$
    fin pour
  jusque $E < \epsilon$

Algorithme 6 Règle Δ

Nécessite: les N instances d'apprentissage $\mathcal{X}$
Nécessite: taux d'apprentissage $\alpha \in]0, 1]$
Nécessite: un seuil ϵ
  initialiser les $w_{j,j\in\{0,\ldots P\}}$ aléatoirement
  répéter
    // E mesure l'erreur courante
    $E \leftarrow 0$
    mélanger les exemples
    pour tous les exemples du jeu d'apprentissage $x_{i,i\in\{1,\ldots,N\}}$ **faire**
      $E \leftarrow E + (y_i - \varphi(x_i))^2$
      pour tous les poids $w_{j,j\in\{0,\ldots,P\}}$ **faire**
        $w_j \leftarrow w_j + \alpha(y_i - \varphi(x_i))x_{i,j}$
      fin pour
    fin pour
  jusque $E < \epsilon$

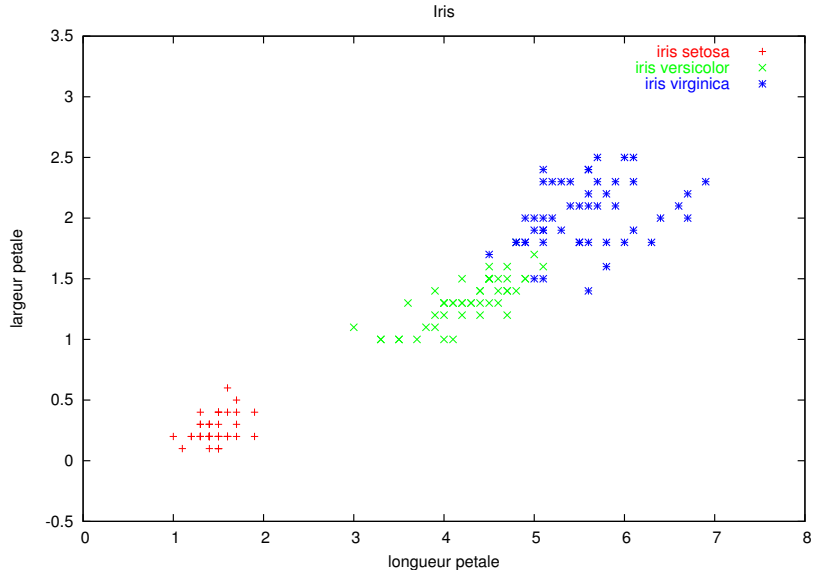


FIG. 7.3 – Chaque exemple (= une fleur) est représenté ici en fonction de la longueur (abscisses) et de la largeur (ordonnées) de ses pétales. On voit que les deux premières variétés (*setosa* et *versicolor*) sont bien distinguables en fonction de ces deux attributs ; on voit aussi que les deux dernières (*versicolor* et *virginica*) ne le sont pas tout à fait.

virginica. La variété représente la classe de la donnée. Chaque fleur est décrite par 4 attributs (longueur et largeur des sépales et des pétales).

En fait, pour ces 3 variétés, la longueur et la largeur des pétales suffisent pour déterminer la variété dans la grande majorité des cas. Aussi, on a simplifié le jeu de données en ne conservant que ces 2 attributs.

On utilise un perceptron possédant trois entrées : une par valeur numérique (longueur et largeur des pétales) et le biais. On doit donc apprendre 3 poids. Le perceptron a sa sortie $s = \tanh \sum_{i=0}^{i=2} w_i x_i$ ($\in [-1, 1]$).

On peut représenter chaque donnée dans le plan de ses deux attributs (voir la figure 7.3). Ayant les trois poids du perceptron, ceux-ci fournissent l'équation de la droite séparant les instances positives des instances négatives par : $y = -\frac{w_0 + w_1 \times x}{w_2}$.

À la figure 7.4, on ne prend en compte que deux variétés, les *setosa* et les *versicolor*. On représente alors la droite séparatrice des deux classes au fur et à mesure des itérations de la règle Δ. L'itération 0 correspond à la valeur initiale des 3 poids ($w_0 = 0.3$, $w_1 = 1$, $w_2 = -0.5$, soit la droite d'équation $y = \frac{x+0.3}{0.5}$). Dès l'itération suivante, la droite sépare les deux classes. L'erreur diminue ensuite au cours des itérations successives. On a arrêté l'algorithme

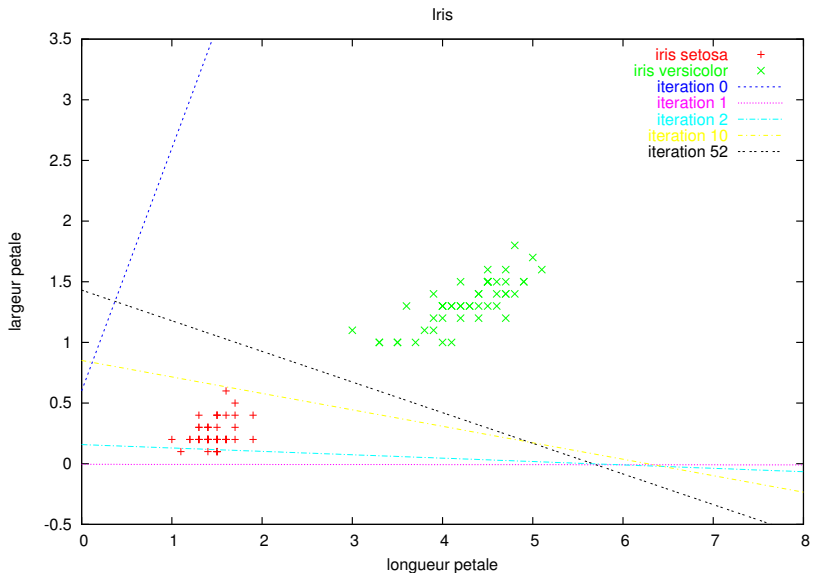


FIG. 7.4 – Évolution de la séparatrice au cours des itérations de la règle Δ en ne prenant en compte que les deux variétés *setosa* et *versicolor*.

lorsque l'erreur est inférieure à 0.5.

À la figure 7.5, on ne prend que les deux variétés *setosa* et *versicolor* et on représente la droite séparatrice des deux classes obtenues avec différentes valeurs initiales de poids. L'exécution 1 correspond à la figure précédente : $w_0 = 0.3$, $w_1 = 1$, $w_2 = -0.5$. L'exécution 2 correspond aux poids initiaux : $w_0 = -1$, $w_1 = 2$, $w_2 = 3.14$, l'exécution 3 : $w_0 = 10$, $w_1 = 12$, $w_2 = -56.14$.

7.2 Perceptron multi-couches

Comme on vient de le voir, un perceptron est capable de prédire correctement la classe de données linéairement séparables (*cf.* fig. 7.6(a)) et uniquement dans ce cas-là. De nombreux autres cas sont envisageables. Par exemple, les données peuvent être séparées de manière non linéaire (*cf.* fig. 7.6(b)) ou par paquets (*cf.* fig. 7.6(c)).

On commence par donner l'intuition de ce que l'utilisation de plusieurs perceptrons judicieusement interconnectés peut apporter.

Intéressons-nous au cas où les données d'une classe sont regroupées par paquets.

Suppsons que l'on dispose d'un perceptron dont la sortie est +1 si son entrée est à une distance inférieure à un certain seuil d'un certain point, 0 sinon. Dans le cas de la fig. 7.6(c), on pourra utiliser 3 perceptrons de ce type, chaque per-

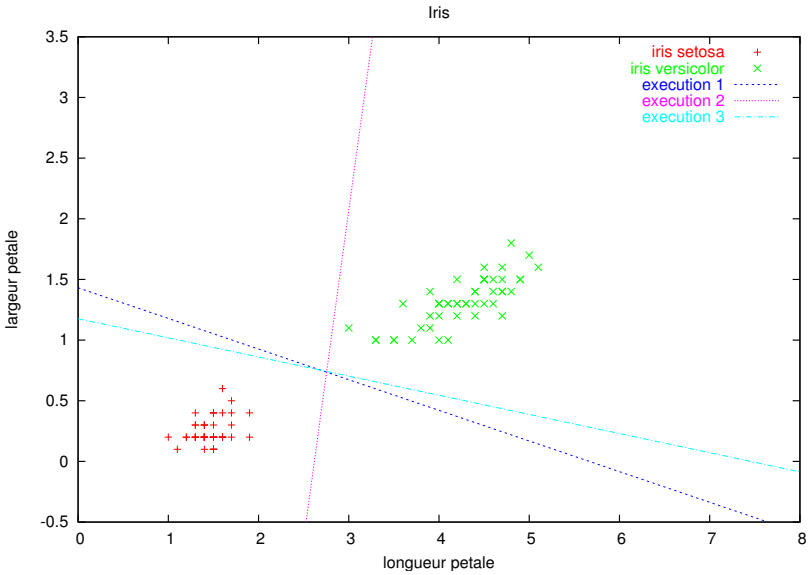
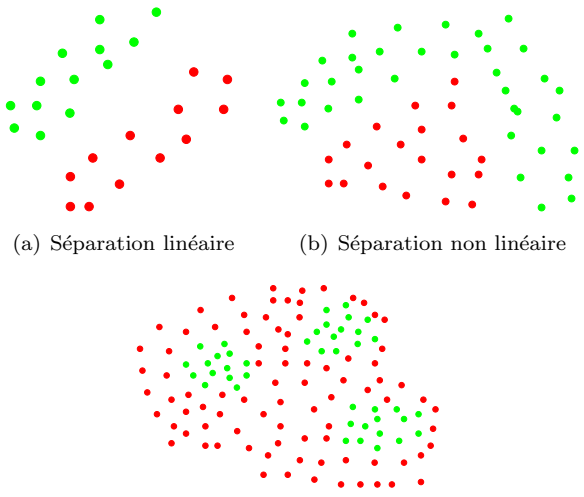


FIG. 7.5 – Différentes séparatrices obtenues à l'issue de plusieurs exécutions de la règle Δ avec des initialisations différentes des poids pour chacune. On ne prend en compte que les deux variétés *setosa* et *versicolor*.



(a) Séparation linéaire (b) Séparation non linéaire

(c) Séparation non linéaire par paquets

FIG. 7.6 – Différents cas de séparation des données de deux classes (les rouges et les verts).

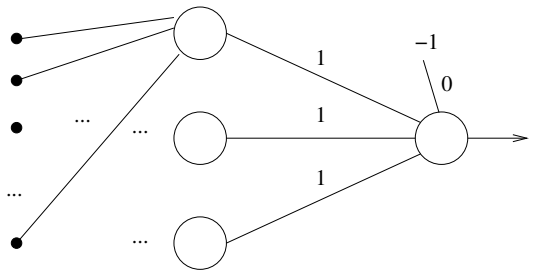


FIG. 7.7 – Voir le texte.

ceptron étant associé à l'une des trois zones contenant des points verts. La sortie de ces 3 perceptrons peut être connectée à l'entrée d'un quatrième perceptron dont la sortie sera +1 si l'une de ses entrées vaut +1, vaudra 0 sinon (*cf.* fig. 7.7). Cette sortie indiquera finalement par +1 que la donnée se situe dans une zone verte, par 0 qu'elle se situe dans la zone rouge ; donc elle indiquera bien la classe de la donnée.

Un perceptron de ce type est obtenu en utilisant une fonction d'activation particulière, dite RBF (*radial basis function*). Elle s'exprime par :

$$\varphi(x) = e^{-\frac{||x-c||^2}{2\sigma^2}}$$

où x est une donnée placée en entrée du perceptron, c le centre de la zone dans laquelle la sortie du perceptron doit être significative, σ l'étalement (écart-type) de cette zone. Cette fonction a la forme en cloche d'une distribution normale.

Avec cette fonction d'activation, on n'a pas un perceptron qui sort +1 ou 0, mais on peut facilement transformer ce perceptron en un perceptron de ce type en mettant un perceptron à seuil sur sa sortie ; par ailleurs, le fait d'avoir une sortie continue comprise dans l'intervalle $]0, 1]$ est en fait généralement considéré comme plus intéressant. De plus, la fonction d'activation est alors continue et dérivable ce qui en permet des manipulations mathématiques aisées.

Dans le cas où la séparatrice est une fonction non linéaire non fermée, c'est-à-dire de type polynomiale, l'intuition est plus difficile à faire passer. Cependant, la combinaison de plusieurs perceptrons permet également de résoudre ce cas. Ce type de réseau se dénomme le « perceptron multi-couches » : il est constitué de plusieurs couches de neurones. Nous le décrivons maintenant.

7.2.1 Topologie d'un perceptron multi-couches

Un perceptron multi-couches (PMC) est constitué des éléments suivants (*cf.* fig. 7.8) :

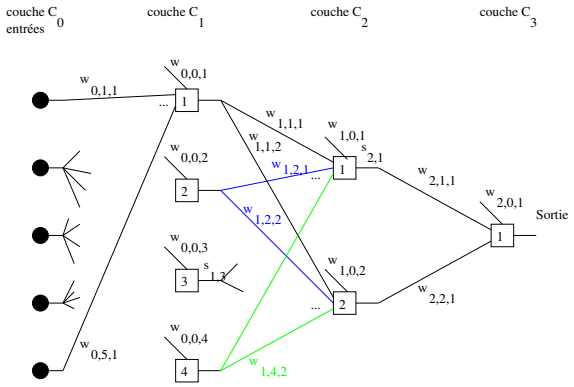


FIG. 7.8 – Schéma de principe de la topologie d'un perceptron multi-couches. On n'a pas représenté toutes les connexions pour lui garder sa lisibilité. On a indiqué la notation utilisée pour les poids w et les sorties s de certains neurones.

- une couche d'entrées : P entrées si les données sont décrites par P attributs (il faut que les attributs soient numériques). Par convention, nous noterons cette couche C_0 ;
- une couche de sortie qui peut en principe contenir plusieurs neurones. Ici, pour une tâche de classification, on suppose que la couche de sortie ne contient qu'un seul neurone dont la sortie fournira la prédiction de la classe de la donnée qui aura été mise en entrée. Par convention, nous noterons cette couche C_q ;
- une ou plusieurs couches intermédiaires situées entre la couche d'entrée et la couche de sortie. Chaque couche est nommée C_i où i varie entre 1 et $q-1$; chacune est composée d'un certain nombre de perceptrons (le nombre de perceptrons de la couche C_i sera noté $|C_i|$) ; chaque perceptron de la couche $C_{i,i\in\{1,\ldots,q\}}$ possède $|C_{i-1}|+1$ entrées : chacune de ces entrées correspondant à la sortie d'une unité de la couche précédente C_{i-1} et une entrée auxiliaire (biais) vaut toujours 1. Ces couches intermédiaires sont dites « cachées » ;
- chaque connexion entre deux unités est caractérisée par un poids réel.

Notons que dans un PMC, les connexions sont toutes orientées de la couche d'entrée vers la couche de sortie. Pour une unité u de la couche l, il n'y a aucune connexion vers :

- une autre unité de la même couche l,
- une unité d'une autre couche que la couche $l+1$.

7.2.2 Apprentissage des poids d'un PMC

Ayant imaginé un perceptron multi-couches, il faut maintenant un algorithme pour apprendre les poids de ce PMC. On peut utiliser pour cela l'algorithme de rétro-propagation du gradient de l'erreur qui est une généralisation de la règle Δ proposée par Le Cun en 1986 ; comme dans le cas du perceptron, on calcule le gradient de l'erreur en fonction de chaque poids pour les corriger ; ainsi, on fait à nouveau une descente de gradient. Les neurones ont leurs entrées et leur sortie réelles et la fonction d'activation est sigmoïde.

Remarque

Le calcul de la dérivée de l'erreur par rapport à chacun des poids d'un PMC se fait itérativement, par rétro-propagation depuis la couche de sortie vers la couche d'entrée. Ainsi, on peut calculer directement cette dérivée pour un neurone de sortie. Nous allons faire ce calcul pour un PMC ayant une seule sortie ; il n'y a aucune difficulté à passer à plusieurs sorties. On rappelle les notations en insistant sur un point important :

- $v_b = \sum_a s_a w_{a\to b}$ le potentiel du neurone b qui est la somme pondérée de ses entrées, c'est-à-dire, les sorties des neurones (a) de la couche précédente, par les poids sur ces connexions que l'on note ici $w_{a\to b}$ pour qu'il soit bien clair qu'il s'agît de la connexion entre la sortie du neurone a et une entrée du neurone b ;
- $s_b = \varphi(v_b)$, la sortie du neurone b, φ étant la fonction d'activation du neurone b.

Faire ce calcul permet de bien comprendre comment adapter l'algorithme si l'on change la fonction d'activation des neurones.

On veut donc calculer $\frac{\partial E}{\partial w_{a\to b}}$ pour tout poids $w_{a\to b}$.

On suppose que l'erreur est :

$$E = \frac{1}{2}(\text{sortie attendue } - s_b)^2$$

Pour tout neurone b, on a :

$$\begin{aligned}\frac{\partial E}{\partial w_{a\to b}} &= \frac{\partial E}{\partial v_b}\frac{\partial v_b}{\partial w_{a\to b}} \\ &= \frac{\partial E}{\partial v_b} s_a\end{aligned}$$

Reste à calculer ce terme $\frac{\partial E}{\partial v_b}$. On distingue le cas où b est un neurone de sortie du cas où b est dans une couche cachée.

Pour le neurone de sortie :

$$\frac{\partial E}{\partial v_b} = \frac{\partial E}{\partial s_b}\frac{\partial s_b}{\partial v_b}$$

- $\frac{\partial E}{\partial s_b}$: peut se ré-écrire :

$$\begin{aligned}\frac{\partial E}{\partial s_b} &= \frac{\partial}{\partial s_b}\frac{1}{2}(\text{sortie attendue } - s_b)^2 \\ &= -(\text{sortie attendue } - s_b)\end{aligned}$$

- $\frac{\partial s_b}{\partial v_b}$: c'est simplement la dérivée de la fonction d'activation. (*cf.* exercice 17.)

Ainsi, on peut calculer directement ce terme pour un neurone de sortie.

Pour un neurone d'une couche cachée : pour calculer ce terme pour les neurones de la couche c, on suppose que l'on sait le calculer pour la couche $c+1$. Ainsi, on va commencer

par calculer ce terme pour la couche juste avant la couche de sortie, puis remonter ainsi vers la première couche cachée. Soit donc un neurone b d'une couche cachée :

$$\frac{\partial E}{\partial v_b} = \sum_{k \text{ : les neurones de la couche suivant celle du neurone } b \text{ considéré}} \frac{\partial E}{\partial v_k}\frac{\partial v_k}{\partial v_b} = \sum_k \frac{\partial E}{\partial v_k}\frac{\partial v_k}{\partial s_b}\frac{\partial s_b}{\partial v_b}$$

dans laquelle :

- $\frac{\partial s_b}{\partial v_b}$: on a déjà rencontré ce terme : c'est la dérivée de la fonction d'activation du neurone b ;
- $\frac{\partial v_k}{\partial s_b}$:

  $$v_k = \sum_{\text{neurones } b' \text{ de la couche précédent celle du neurone } k} w_{b' \to k} s_{b'}$$

  Il faut bien noter que le neurone b est l'un des neurones de la couche précédent celle où se trouve le neurone k. Donc :

  $$v_k = w_{b \to k} s_b + \sum_{\text{autres neurones } b' \text{ de la couche précédent celle du neurone } k} w_{b' \to k} s_{b'}$$

  Dans cette expression s_b (la variable par rapport à laquelle on dérive) apparaît une seule fois, dans le terme qui a été extrait de la somme. Donc,

  $$\frac{\partial v_k}{\partial s_b} = w_{b \to k}$$

- $\frac{\partial E}{\partial v_k}$: c'est la dérivée de E par rapport au potentiel du neurone k qui se situe dans la couche suivante, donc plus proche de la sortie. C'est donc le même terme que nous sommes en train de calculer pour un neurone plus proche de la sortie ; puisque nous faisons un calcul itératif couche par couche, en commençant par la couche de sortie et en remontant vers la couche d'entrée, ce terme a déjà été calculé précédemment.

En résumé, nous sommes capables de calculer la dérivée de l'erreur par rapport à n'importe quel poids d'un PMC. Quand on dispose de cette dérivée, on l'utilise pour corriger le poids, comme dans le cas du perceptron, en le pondérant par un taux d'apprentissage.

L'algorithme de rétro-prapagation du gradient de l'erreur est donné par l'algorithme 7.

7.2.3 Quelques compléments sur l'algorithme d'apprentissage des poids

On donne ici quelques « trucs » pour obtenir un apprentissage qui ne soit pas trop long. On ne dira jamais assez combien l'utilisation d'un réseau de neurones demande beaucoup de soin dans les plus fins détails de l'algorithme d'apprentissage.

Le critère d'arrêt

Le critère d'arrêt de l'algorithme de retro-propagation du gradient de l'erreur peut être :

Algorithme 7 Algorithme de rétro-propagation du gradient de l'erreur (version stochastique) pour un perceptron multi-couches ayant $P+1$ entrées (P attributs + le biais), $q+1$ couches numérotées de 0 (C_0 : couche d'entrée) à q (C_q : couche de sortie) et une seule sortie ; notation : $s(x_i)$ est la sortie du PMC pour la donnée x_i, $s_{l,k}$ est la sortie de la k^{e} unité (entrée ou neurone) de la couche l, $w_{l,k,m}$ est le poids de la connexion entre l'unité k de la couche l et le neurone m de la couche $l+1$ ($k=0$ correspond au biais), $|C_l|$ est le nombre d'unités composant la couche C_l. L'algorithme donné ici correspond à des neurones à fonction d'activation logistique $\frac{1}{1+e^{-x}}$.

Nécessite: les N instances d'apprentissage $\mathcal{X}$
Nécessite: taux d'apprentissage $\alpha \in]0,1]$
  initialiser les w_i
  tant-que critère d'arrêt non rempli **faire**
    mettre à jour α
    mélanger les exemples
    pour tout exemple x_i **faire**
      $s(x_i) \leftarrow$ sortie du réseau pour l'exemple x_i
      $\delta_{q,1} \leftarrow s(x_i)(1-s(x_i))(y_i - s(x_i))$
      pour toutes les couches cachées : l décroissant de $q-1$ à 1 **faire**
        pour tous les neurones k de la couche C_l **faire**
          $\delta_{l,k} \leftarrow s_{l,k}(1-s_{l,k}) \sum_{m \in \{0,\ldots,|C_{l+1}|\}} w_{l,k,m}\ \delta_{l+1,m}$
        fin pour
      fin pour
      // mise à jour des poids
      pour toutes les couches l croissant de 0 à $q-1$ **faire**
        pour toutes les unités k de la couche l, k variant de 1 à $|C_l|$ **faire**
          pour tous les neurones m connectés sur la sortie du neurone k de la couche l, m variant de 1 à $|C_{l+1}|$ **faire**
            $w_{l,k,m} \leftarrow w_{l,k,m} + \alpha\ \delta_{l+1,m}\ s_{l,k}$
          fin pour
        fin pour
      fin pour
    fin pour
  fin tant-que

- la somme des corrections (les $\delta_{...}$) est inférieure à un seuil fixé lors d'une itération de la boucle **pour** la plus externe (somme des corrections sur l'ensemble des exemples d'apprentissage) ;
- le nombre d'itérations effectuées qui a été fixé *a priori* ;
- l'erreur sur un jeu de validation est inférieure à un seuil fixé. Pour cela, dans l'algorithme 7, il faut ajouter, à la suite du passage de tous les exemples du jeu d'apprentissage, l'évaluation de l'erreur sur le jeu de validation ;
- stabilisation de l'erreur d'une itération du **tant-que** à la suivante.

L'initialisation des poids

L'initialisation des poids n'est pas un simple détail (en fait, dans l'utilisation des PMC, rien n'est un détail...). Le Cun et al. [1998] conseille d'initialiser chaque poids avec une valeur tirée aléatoirement dans une distribution gaussienne centrée (de moyenne nulle) et d'écart-type égal à la racine carrée du nombre d'entrées de l'unité à l'entrée de laquelle se trouve ce poids.

Par exemple, les poids des neurones de la couche C_1 de la figure 7.8 seront initialisés en tirant des nombres pseudo-aléatoires d'une distribution normale de moyenne nulle et de variance égale à 6.

L'objectif est que le potentiel des neurones soit proche du point d'inflexion de la sigmoïde (en 0 pour tanh).

Le taux d'apprentissage

Le taux d'apprentissage doit diminuer à chaque épisode, par exemple comme $\alpha = \frac{1}{\text{numéro de l'épisode}^c}$ où $c \in \mathbb{R}^+$.

Ajouter de l'inertie aux corrections de poids

Pour éviter que la variation de valeur des poids ne soit trop brutale, on peut utiliser un moment noté η : la mise à jour des poids devient alors :

$$\Delta_w(t+1) \leftarrow \alpha\ \delta_w\ y + \eta\ \Delta_w(t)$$

où $\Delta_w(t)$ est la correction du poids w à l'itération t.
On prend $\Delta_w(t=0) = 0$.

Fonction d'activation

A priori, la fonction d'activation doit être la fonction tangente hyperbolique : $\varphi(v) = a \tanh(bv)$ où a et b sont deux constantes. Sinon, il faut prendre une fonction qui soit symétrique par rapport à l'origine, donc fuir la trop célèbre fonction logistique.

Pré-traitement des sorties

Utilisant une fonction d'activation *tanh*, on pense naturellement à coder la classe des données par $+1$ et -1. Cependant, ± 1 sont des valeurs asymptotiques pour la fonction *tanh*. Aussi, elles ne sont jamais atteintes et donc l'apprentissage n'est pas possible... Au lieu d'utiliser ces valeurs, on utilise généralement des valeurs un peu plus petite que 1 en valeur absolue ; on pourra utiliser ± 0.6 par exemple.

Pré-traitement des entrées

Il est recommandé que les entrées du réseau :
- ne soient pas corrélées, c'est-à-dire que leur matrice de covariance soit la matrice identité. Pour dé-corréler les entrées, on peut faire une ACP (*cf.* chap. 11.1 ; on appelle cela du *sphering* en anglais) ;
- aient une moyenne nulle ;
- aient une variance unité.

Attributs nominaux

Si un attribut est nominal et peut prendre ν valeurs non ordinales, on utilisera ν entrées pour coder sa valeur. Pour une valeur donnée de l'attribut, l'une de ces ν entrées sera à 1, toutes les autres valant 0.

Si un attribut est binaire, une seule entrée est suffisante.

Si on a des informations selon lesquelles un meilleur codage peut être imaginé, il faut utiliser celui-ci.

Attributs manquant

Dans le jeu d'apprentissage, si la valeur de certains attributs est manquante (de manière non systématique), plusieurs approches peuvent être envisagées : remplacement par une valeur moyenne, remplacement par une valeur particulière, remplacement par la valeur la plus présente, ... On peut aussi dupliquer l'exemple en remplaçant la valeur manquante par une valeur différente dans chacun de ces nouveaux exemples.

Notons que si le nombre d'attributs manquants est faible dans le jeu d'apprentissage, cela ne doit pas perturber l'apprentissage puisque les perceptrons multi-couches sont naturellement peu sensibles au bruit.

Pour prédire la classe d'une donnée dont la valeur de certains attributs manque, on pourra à nouveau remplacer cette valeur manquante par une certaine valeur (moyenne, la plus présente, ...). On pourra aussi prédire la classe de donnée ayant différentes valeurs pour cet attribut et déterminer ainsi la classe la plus probable.

Il est important de noter que le traitement adéquat des attributs manquants et l'impact de ces manques sur les résultats sont des questions qui ne sont pas assez étudiées. Sur un cas concret, lorsque le nombre d'attributs manquants est significatif, il faut garder à l'esprit que les résultats de la fouille peuvent dépendre de la manière dont ils ont été traitées. On consultera avec intérêt Sarle [1998] et Information Technology Service [2004].

Plus de deux classes

Si le nombre de classes est supérieur à 2, on utilise $|\mathcal{Y}|$ sorties. Chaque sortie correspond à une classe. Une donnée étant placée en entrée, la sortie la plus active indique la classe prédite.

Si l'on veut, on utilise une unité dite *winner-takes-all* qui prend en entrée les $|\mathcal{Y}|$ sorties du réseau et fournit en sortie le numéro de son entrée la plus active : c'est le numéro de la classe prédite.

Quelques problèmes survenant lors de l'utilisation de l'algorithme de rétro-propagation du gradient de l'erreur

L'algorithme de rétro-propagation du gradient de l'erreur trouve un optimum local. Plusieurs exécutions en initialisant les poids différemment ou en présenter les exemples dans des ordres différents peuvent entraîner des convergences vers des poids différents (rappelons que les exemples sont mélangés à chaque épisode).

Si l'on n'y prend pas garde, on va rencontrer des problèmes de sur-apprentissage. Deux approches sont utilisées :

arrêt précoce : (*early stopping* en anglais) c'est une approche expérimentale : on utilise un jeu de validation composé d'exemples, disjoint du jeu d'apprentissage. Durant l'apprentissage, on mesure l'erreur sur le jeu de validation. Quand cette erreur stagne, on arrête l'apprentissage. Remarquons que si l'on mesure l'erreur sur le jeu d'apprentissage, au moment où l'erreur sur le jeu de validation stagne, l'erreur sur le jeu d'apprentissage continue à diminuer. C'est exactement cela le « sur-apprentissage » (*cf.* chap. 3, sec. 3.8) ;

régularisateur : c'est une approche qui s'appuie sur des résultats théoriques qui consiste à minimiser une fonction combinant la mesure de l'erreur sur le jeu d'apprentissage avec la valeur des poids : on cherche une erreur minimale et des poids les plus petits possibles, soit $E = E_{\text{app}} + \lambda \sum_i w_i^2$ où λ est une constante qui permet d'équilibrer l'importance des deux termes et les w_i représentent tous les poids du réseau. Typiquement, on teste plusieurs valeurs de λ. Cette technique se nomme *weight decay* en anglais. D'autres techniques de régularisation sont envisageables.

Le problème de l'apprentissage des poids On pourra également consulter Sarle [1997a], Peterson et al. [1993], Le Cun et al. [1998] pour avoir des points de vue pratiques sur la mise en œuvre des réseaux de neurones ; très souvent, les exposés des réseaux de neurones sont très académiques et ne permettent pas de les appliquer directement ; leur utilisation nécessite beaucoup de savoir-faire.

7.2.4 D'autres résultats rassurants

Propriété 3 *Toute fonction booléenne peut être apprise sans erreur par un perceptron multi-couches ayant 1 seule couche cachée.*

Cependant, ce résultat est essentiellement théorique. En effet, on démontre aussi qu'une fonction à P entrées peut nécessiter une couche cachée possédant $O(2^P)$ neurones. Cela signifie que le calcul de cette fonction peut être non polynomial, autrement dit, prendre beaucoup trop de temps (*cf.* annexe C).

Dans la pratique, pour éviter des temps d'apprentissage démeusurément longs, on utilise une deuxième couche cachée, voire plus.

Remarque

Pour commencer à comprendre la portée du résultat précédent, il n'est pas inutile de rappeler que toute fonction calculable (*i.e.*, dont on peut calculer la valeur à l'aide d'un ordinateur ou, pour être tout à fait précis, une machine de Turing, *cf.* annexe C) peut être mise sous forme d'une fonction booléenne (c'est exactement ce que fait un ordinateur d'ailleurs). Donc, ce résultat implique que toute fonction calculable par un ordinateur peut être calculée par un perceptron.

Par ailleurs, Hornik [1991] a montré qu'un perceptron multi-couches avec 1 seule couche cachée de perceptrons sigmoïdes et un neurone linéaire en sortie peut calculer n'importe quelle fonction réelle et bornée, autrement dit, n'importe quelle séparatrice.

Donc, cela signifie que tout problème de classification peut être résolu par un perceptron constitué d'une couche cachée d'unités dont la fonction d'activation est une sigmoïde et d'une unité linéaire en sortie.

De même, Hartman et al. [1990], Girosi and Poggio [1989] ont montré que toute fonction réelle peut être approximée avec autant de précision que l'on veut par un perceptron ayant une couche cachée d'unités RBF et une unité de sortie linéaire.

Donc, on a des théorèmes montrant que l'apprentissage de toute séparatrice avec différents types de perceptrons ayant une couche cachée d'unité dont la fonction d'activation est non linéaire et une sortie linéaire est possible. Cependant, ces théorèmes n'indiquent pas le nombre d'unités à mettre dans la couche cachée.

7.3 Application à « jouer au tennis ? »

7.3.1 Numérisation des attributs et de la classe

Les entrées

On applique ce que l'on a dit à la section 7.2.3.

L'attribut « Ciel » peut prendre 3 valeurs différentes. Aussi, on utilisera 3 entrées binaires, une par valeur possible, soit une entrée pour Ciel = Ensoleillé, une entrée pour Ciel = Couvert et une 3^{e} pour Ciel = Pluie.

C'est exactement la même chose pour l'attribut « Température ».

L'attribut « Humidité » est binaire. Aussi, une entrée prenant la valeur 0 ou 1 est-elle suffisante pour cette attribut.

C'est la même situation pour l'attribut « Vent ».

En résumé, on aura 8 entrées binaires.

La classe

La classe est binaire. Aussi, associer ± 0.6 aux deux classes est naturelle. On peut aussi avoir deux sorties, une par classe.

7.4 Critique

- \- difficile à utiliser dans la pratique : nécessite beaucoup de savoir-faire et d'expérience ;
- o quand ça marche, ça marche mais ce n'est pas parce que vous n'obtenez pas de bons résultats que les réseaux de neurones sont nuls !
- o à n'appliquer que sur des problèmes difficiles pour lesquels les autres méthodes ne fonctionnent pas ;
- \+ bonne capacité de généralisation ;
- \+ très utilisé pour des tâches difficiles : analyse signal (caractères écrits, analyse signal, ...) ;
- \+ fonctionne bien même en présence de données bruitées ;
- – calibrage du réseau pas forcément évident (nombre de couches, type de réseau, nombre de neurones par couche, ...) ;
- \- trouve un optimum local et non global ;
- \- très difficile d'extraire un modèle de l'apprentissage effectué par le réseau.

7.5 Les logiciels libres

Les logiciels libres proposant des implantations de réseaux de neurones sont nombreuses. Il y en a une dans `weka` que nous déconseillons : on atteint là les

limites de ce logiciel en terme de temps de calcul.

Au contraire, on pourra utiliser `JavaNNS` qui est un logiciel très puissant. Cependant, son utilisation n'est pas très simple. `http://www-ra.informatik.uni-tuebingen.de/SNNS`

7.6 Exercices

voir le poly [Denis and Gilleron, 2000].

Exercice 17 Calculer le terme $\frac{\partial s_b}{\partial v_b}$ qui apparaît dans la dérivation de la règle de rétro-propagation du gradient de l'erreur pour les différentes fonctions d'activation présentées ici : logistique, tanh et linéaire.

Exercice 18 Calculer $\frac{\partial f}{\partial w}$ quand la fonction minimisée combine l'erreur sur le jeu de données et l'amplitude des poids ; cela permet d'avoir des poids dont la valeur absolue demeure petite. Dans ce cas, la fonction f peut s'écrire :

$$f = \text{erreur} + \sum ||\overrightarrow{w}||$$

où $\overrightarrow{w}$ représente l'ensemble des poids du réseau.

Chapitre 8

Classification par machines à vecteurs supports

Contenu

La pré-occupation est ici la même que dans le cas de l'utilisation de réseau de neurones pour la classification (*cf.* chap. 7) : trouver un hyperplan qui sépare les exemples positifs des exemples négatifs. Cependant, alors qu'un réseau de neurones trouve un optimum local, une machine à vecteurs supports (MVS) trouve un optimum global. Notons aussi que les machines à vecteurs supports ne se cantonnent pas, elles non plus, aux hyperplans, mais peuvent construire des séparateurs non linéaires de diverses formes.

La simple lecture de ce qui précède pourrait laisser penser que les réseaux de neurones doivent être remisés aux oubliettes. Cependant, diverses raisons pratiques font que les deux techniques (réseau de neurones d'une part, machine à vecteurs supports d'autre part) ont chacune des qualités et des défauts et que les deux techniques doivent être connues et que les deux ont de bonnes raisons d'être utilisées.

Enfin, une petite note concernant ce nom barbare de « machine à vecteurs supports ». « Machine » signifie ici simplement « algorithme » (*cf.* le terme *machine learning* pour algorithme d'apprentissage). Pour ce qui est des « vecteurs supports », on va bientôt expliquer ce que cela signifie.

Comme dans les autres chapitres, on note $x_{i\in\{1,\ldots,N\}}$ le i^{e} individu et $x_{i,j}$ sa j^{e} composante. Les attributs sont numériques, quantitatifs (*cf.* ce qui a été dit concernant les attributs nominaux et les attributs manquants dans le cas du perceptron multi-couches, chap. 7, sections 7.2.3 et 7.2.3). On note y_i la classe de l'individu i. La classe de chacun des individus est ± 1.

8.1 Machine à vecteurs supports linéaire

On commence par présenter les notions essentielles des machines à vecteurs supports dans le cas le plus simple, celui où l'on cherche un hyperplan séparateur. On commence alors par le cas où les données sont linéairement séparables avant de regarder le cas non linéairement séparable.

8.1.1 Cas séparable

Principe de la méthode

Considérons chaque individu x_i comme un point dans un espace à P dimensions ; on peut le voir également comme un vecteur $\overrightarrow{x_i}$. Dans la suite, on alternera les deux points de vue en fonction du contexte : point ou vecteur.

Si le problème est linéairement séparable, les individus positifs sont séparables des individus négatifs par un hyperplan H. Notons H_+ l'hyperplan parallèle à H qui contient l'individu positif le plus proche de H, resp. H_- pour l'individu négatif (*cf.* fig. 8.1).

marge

Une MVS linéaire recherche l'hyperplan qui sépare les données de manière à ce que la distance entre H_+ et H_- soit la plus grande possible. Cet écart entre les deux hyperplans H_+ et H_- est dénommé la « marge ».

Un hyperplan a pour équation $y =< \overrightarrow{w}, \overrightarrow{x} > + b$, où $< \overrightarrow{w}, \overrightarrow{x} >$ dénote le produit scalaire entre les vecteurs $\overrightarrow{w}$ et $\overrightarrow{x}$. Pour une donnée $\overrightarrow{x}$ de classe y, on cherche $\overrightarrow{w}$ tel que :

$$\begin{cases} < \overrightarrow{w}, \overrightarrow{x} > + b \geq 1 \text{ si } y = +1 \\ < \overrightarrow{w}, \overrightarrow{x} > + b \leq -1 \text{ si } y = -1 \end{cases}$$

Donc, on a : $y(< \overrightarrow{w}, \overrightarrow{x} > + b) - 1 \geq 0$.

calcul de la marge

On veut maximiser la largeur de la marge. Calculons-la :

1. le vecteur $\overrightarrow{w}$ est perpendiculaire à l'hyperplan H (mathématiques élémentaires) ;

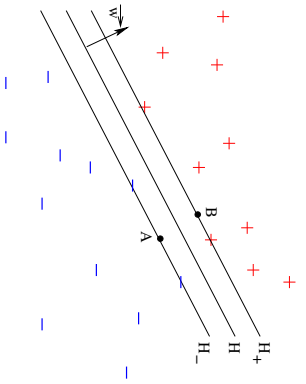


FIG. 8.1 – Schéma illustratif : les individus positifs sont représentés par un +, les négatifs par un −. On a représenté un hyperplan H qui sépare les positifs des négatifs (sur le schéma, ce n'est pas l'hyperplan qui sépare au mieux les deux ensembles). On a représenté H_+ et H_-, tous deux parallèles à H. B est un point de H_+ et A est le point le plus proche de B qui appartient à H_-.

2. soit B un point de H_+ et A le point le plus proche de B sur H_- (*cf.* fig. 8.1) ;
3. pour tout point O, on a : $\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB}$;
4. par définition des points A et B, $\overrightarrow{AB}$ est parallèle à $\overrightarrow{w}$. Donc, il existe $\lambda \in \mathbb{R}$ tel que $\overrightarrow{AB} = \lambda\overrightarrow{w}$, soit $\overrightarrow{OB} = \overrightarrow{OA} + \lambda\overrightarrow{w}$;
5. on veut que A, B, H_- et H_+ soient tels que :
$$\begin{cases} B \in H_+ \Rightarrow < \overrightarrow{w}, \overrightarrow{OB} > + b = 1 \\ A \in H_- \Rightarrow < \overrightarrow{w}, \overrightarrow{OA} > + b = -1 \end{cases}$$
6. donc, $< \overrightarrow{w}, (\overrightarrow{OA} + \lambda\overrightarrow{w}) > + b = 1$;
7. donc, $\underbrace{< \overrightarrow{w}, \overrightarrow{OA} > + b}_{-1} + < \overrightarrow{w}, \lambda\overrightarrow{w} >= 1$;
8. soit, $-1 + \underbrace{< \overrightarrow{w}, \lambda\overrightarrow{w} >}_{\lambda < \overrightarrow{w}, \overrightarrow{w} >} = 1$;
9. donc, $\lambda = \frac{2}{< \overrightarrow{w}, \overrightarrow{w} >} = \frac{2}{||\overrightarrow{w}||}$.
10. donc :
$$|\lambda| = \frac{2}{||\overrightarrow{w}||} \tag{8.1}$$

La largeur de ma marge est $||\lambda\overrightarrow{w}||$, $\overrightarrow{w}$ en donnant la direction et $|\lambda|$ son amplitude.

Conclusion : on veut maximiser la marge ; donc, on doit minimiser la norme de $\overrightarrow{w}$.

Minimiser $||\overrightarrow{w}||$ revient à peu près à la même chose que minimiser $||\overrightarrow{w}||^2$. Une différence néanmoins est que quand $||\overrightarrow{w}||^2$ est petit (proche de 0), son carré est encore plus petit ; donc, ce terme est négligeable pour une plus grande amplitude de valeur de $||\overrightarrow{w}||$. On la préfère donc ici.

De plus, on veut vérifier les contraintes $\gamma_i = y_i(< \overrightarrow{w}, \overrightarrow{x_i} > + b) - 1 \geq 0, \forall i \in \{1, ..., N\}$.

problème d'optimisation non linéaire

Cela est donc un problème d'optimisation non linéaire (minimiser $||\overrightarrow{w}||^2$) avec contraintes (les γ_i) que l'on résoud habituellement par la méthode de Lagrange [Fourer, 2004] (*cf.* annexe D). Cette méthode transforme un problème d'optimisation de fonction avec contraintes en un problème d'optimisation de fonction sans contrainte. Ces deux problèmes possèdent les mêmes solutions.

expression du lagrangien

Pour cela, on exprime le lagrangien L_P comme somme de la fonction à minimiser (fonction objectif) et de l'opposé de chaque contrainte γ_i multipliée par une constante $\alpha_i \in \mathbb{R}^+$. Les α_i constituent les « multiplicateurs de Lagrange ».

multiplicateur de Lagrange

On a donc :

$$L_P = \frac{1}{2}||\overrightarrow{w}||^2 - \sum_{i=1}^{i=N} \alpha_i y_i (< \overrightarrow{w}, \overrightarrow{x_i} > + b) + \sum_{i=1}^{i=N} \alpha_i \tag{8.2}$$

L_P doit être minimisé par rapport à $\overrightarrow{w}$ et b et il faut que les dérivées par rapport aux α_i soient nulles.

Le gradient de L_P devant être nul par rapport à $\overrightarrow{w}$ et b, on écrit :

$$\begin{cases} \frac{\partial L_P}{\partial \overrightarrow{w}} = 0 \\ \frac{\partial L_P}{\partial b} = 0 \end{cases}$$

d'où l'on tire aisément (en calculant ces dérivées et en les annulant) :

$$\begin{cases} \overrightarrow{w} = \sum_{i=1}^{i=N} \alpha_i y_i \overrightarrow{x_i} \\ \sum_{i=1}^{i=N} \alpha_i y_i = 0 \end{cases}$$

lagrangien dual

De la formulation de L_P et de ces deux équations, on tire la formulation duale du lagrangien en éliminant $\overrightarrow{w}$:

$$L_D = \sum_{i=1}^{i=N} \alpha_i - \frac{1}{2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} \alpha_i \alpha_j y_i y_j < \overrightarrow{x_i}, \overrightarrow{x_j} > \tag{8.3}$$

qui doit être maximisé. Le maximum de L_D et le minimum de L_P sont obtenus pour les mêmes valeurs de $\overrightarrow{w}$, b et α_i.

Pour que $\overrightarrow{w}$, b et les α_i existent, le problème doit vérifier les conditions de Karush-Kuhn-Tucker (KKT) :

$$\begin{cases} \frac{\partial L_P}{\partial w_\nu} = w_\nu - \sum_{i=1}^{i=N} \alpha_i y_i x_{i,\nu} = 0, \forall\ \nu = 1, 2, ...P \\ \frac{\partial L_P}{\partial b} = -\sum_{i=1}^{i=N} \alpha_i y_i = 0 \\ y_i(<\overrightarrow{w}, \overrightarrow{x_i}> + b) - 1 \geq 0, \forall\ i = 1, 2, ...N \\ \alpha_i \geq 0, \forall i = 1, ...N \\ \alpha_i(y_i(<\overrightarrow{w}, \overrightarrow{x_i}> + b) - 1) = 0, \forall\ i = 1, ...N \end{cases} \tag{8.4}$$

Ces cinq conditions résument ici ce qui a été dit précédemment. Les conditions KKT sont donc vérifiées et le problème que l'on cherche à résoudre possède bien une solution.

Les deux dernières lignes indiquent simplement que pour tout exemple $\overrightarrow{x_i}$, soit $\alpha_i = 0$, soit $y_i(<\overrightarrow{w}, \overrightarrow{x_i}> + b) - 1 = 0$.

vecteur support

On définit alors :

Définition 10 *un vecteur support est un vecteur (*i.e.*, une donnée) dont le multiplicateur de Lagrange associé est non nul.*

Quelques remarques :

1. les multiplicateurs de Lagrange étant positifs dans le problème posé ici, un vecteur support a donc un multiplicateur de Lagrange de valeur strictement positive ;
2. puisque l'on a $\alpha_i(y_i(<\overrightarrow{w}, \overrightarrow{x_i}> + b) - 1) = 0$ pour tous les points et que $\alpha_i \neq 0$ pour les vecteurs supports, cela entraîne que $y_i(<\overrightarrow{w}, \overrightarrow{x_i}> + b) - 1 = 0$ pour les vecteurs supports ; cela entraîne que les vecteurs supports sont situés exactement sur les hyperplans H_+ et H_- ;
3. en fait, seuls les exemples correspondant aux vecteurs supports sont réellement utiles dans l'apprentissage. Si on les connaissait *a priori*, on pourrait effectuer l'apprentissage sans tenir compte des autres exemples ;
4. les vecteurs supports synthétisent en quelque sorte les aspects importants du jeu d'exemples. On peut donc compresser l'ensemble des exemples en ne retenant que les vecteurs supports.

Classification d'une nouvelle donnée

On l'a vu plus haut, la classe d'une donnée $\overrightarrow{x}$ est ± 1 et elle est fournie par le signe de $<\overrightarrow{w}, \overrightarrow{x}> + b$. En effet, si cette quantité est ≥ 1, cela signifie que x est « au-dessus » de H_+. Sinon, $<\overrightarrow{w}, \overrightarrow{x}> \leq -1$, ce qui signifie que x est « en-dessous » de H_-. En utilisant la fonction signe sgn(.), on note sgn($<\overrightarrow{w}, \overrightarrow{x}> + b$) cette quantité.

Puisque $\overrightarrow{w} = \sum_{i=1}^{i=N} \alpha_i y_i \overrightarrow{x_i}$ (*cf.* plus haut) et que seuls les vecteurs supports ont un multiplicateur de Lagrange non nul, on a :

$$\text{sgn}(<\overrightarrow{w},\overrightarrow{x}>+b) = \text{sgn}\,(\sum_{i=1}^{i=N}\alpha_i y_i(<\overrightarrow{x_i},\overrightarrow{x}>+b)) = \text{sgn}\,(\sum_{j=1}^{j=N_s}\alpha_j y(s_j)(<\overrightarrow{s_j},\overrightarrow{x}>+b)) \tag{8.5}$$

où $\overrightarrow{x}$ est l'instance à classer, les $\overrightarrow{x_i}$ sont les exemples d'apprentissage, les $\overrightarrow{s_j}$ sont les vecteurs supports, N_s est le nombre de vecteurs supports.

8.1.2 Cas non séparable

On suppose maintenant que les exemples ne sont pas linéairement séparables. On cherche alors un hyperplan qui minimise les erreurs de classification sur le jeu d'exemples.

On introduit un jeu de variables mesurant l'erreur (notées ξ_i et appelées en anglais *slack variables*) telles que :

$$\begin{cases} <\overrightarrow{w},\overrightarrow{x_i}>+\,b \geq 1-\xi_i \text{ si } y_i=+1 \\ <\overrightarrow{w},\overrightarrow{x_i}>+\,b \leq -1+\xi_i \text{ si } y_i=-1 \\ \xi_i \geq 0, \forall\ i=1,...N \end{cases}$$

Un exemple $\overrightarrow{x_i}$ est bien classé si $\xi_i = 0$. Si $\xi_i \neq 0$, c'est-à-dire si x_i est mal classé, alors $\xi_i \geq 1$ (*cf.* fig. 8.2). Ainsi, ξ_i indique à quel point l'exemple x_i est du mauvais côté : si x_i est du mauvais côté de la séparatrice, plus x_i est loin de la séparatrice, plus ξ_i est grand. Donc, $\sum_{i=1}^{i=N}\xi_i$ est une borne supérieure du nombre d'erreurs de classification.

Le problème devient celui de la recherche de l'hyperplan impliquant la marge la plus grande et le nombre d'erreurs le plus petit. Donc, la fonction objectif devient (par exemple) : minimiser :

$$||\overrightarrow{w}||^2 + C\sum_{i=1}^{i=N}\xi_i \tag{8.6}$$

où C est une constante (dénommée « capacité » de la MVS) qui permet de donner plus ou moins d'importance aux erreurs de classification.

Donc, en résumé, on résoud maintenant le problème :

$$\begin{cases} \text{minimiser } ||\overrightarrow{w}||^2 + C\sum_{i=1}^{i=N}\xi_i \\ \text{en respectant les } \gamma_i\xi_i \geq 0, \forall i \end{cases} \tag{8.7}$$

On applique la méthode de Lagrange comme précédemment, ce qui fournit les multiplicateurs et les ξ_i.

On a $\alpha_i \in [0, C]$. Si $\alpha_i \in [0, C[$, alors $\xi_i = 0$. Si $\alpha_i = C$, alors $\xi_i \neq 0$: x_i est un vecteur support.

Pour classer une donnée, c'est exactement comme dans le cas séparable (*cf.* sec. 8.1.1).

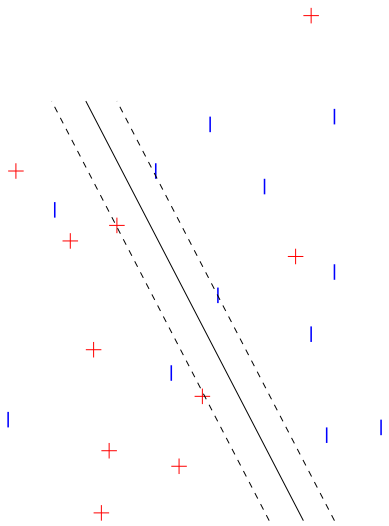


FIG. 8.2 – Schéma illustratif : les exemples ne sont plus linéairement séparables : deux positifs sont du côté des négatifs et trois négatifs du côté des positifs. La séparatrice est indiquée en trait plein et les deux droites en pointillés sont H_+ et H_-. Il y a 4 vecteurs supports, 2 positifs et 2 négatifs. Les ξ sont tous nuls sauf pour les 5 données du mauvais côté.

Dans le cas non séparable linéairement, nous avons trois sortes de données :
- les données mal classées ;
- les données correspondant aux vecteurs supports (bien classées) ;
- les données bien classées, qui ne sont pas des vecteurs supports.

Remarque : dans une MVS, l'hyperplan trouvé est un optimum global ; dans un perceptron multi-couches, c'est un optimum local. Les algorithmes mis en jeu dans chacune de ces approches entraînent leurs conséquences : formulation lagrangienne pour les MVS et descente de gradient de l'erreur de classification pour le perceptron ; étant plus sophistiquée, l'approche lagrangienne est également plus coûteuse en temps d'exécution et elle ne peut pas être appliquée à de très gros volumes de données.

8.2 Machine à vecteurs supports non linéaire

On suppose maintenant que les données ne sont pas linéairement séparables. L'idée est alors de trouver une transformation de l'espace des données dans un autre espace dans lequel les données sont à nouveau linéairement séparables (*cf.* fig. 8.3 et 8.4). Généralement, ce nouvel espace a une dimension plus grande que l'espace des données initial ; il peut même être de dimension infinie. Cela

FIG. 8.3 – Schéma illustratif : un ensemble d'exemples positifs et négatifs non linéairement séparable.

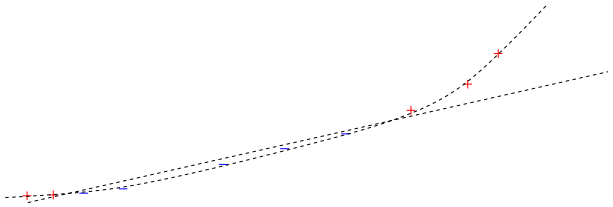


FIG. 8.4 – Schéma illustratif : les exemples sont les mêmes qu'à la fig. 8.3. En effectuant une transformation non linéaire des points ($x \rightarrow x^2$), on constate que les exemples deviennent linéairement séparables. C'est là tout le principe des machines à vecteurs supports non linéaires : transformer l'espace de données de manière à ce que les exemples soient linéairement séparables dans le nouvel espace.

s'appuie sur le théorème de Cover [1965] qui indique qu'un ensemble d'exemples transformé de manière non linéaire dans un espace de plus grande dimension a plus de chance d'être linéairement séparable que dans son espace d'origine.

8.2.1 Construction d'une MVS non linéaire

Remarque fondamentale pour ce qui suit : dans la résolution du problème quadratique (donc lors de l'apprentissage) et lors de la classification d'une nouvelle donnée, les données apparaissent toujours sous la forme de produits scalaires (*cf.* les sections précédentes sur les MVS linéaires).

On note :

$$\Phi \ : \mathbb{R}^P \rightarrow \mathcal{F}$$

une transformation de l'espace des données (ici, $\mathcal{D} = \mathbb{R}^P$) en un espace des caractéristiques $\mathcal{F}$ (*feature space*). Chacun des axes de coordonnées de $\mathcal{F}$ est une combinaison non linéaire des axes de coordonnées de $\mathcal{D}$.

fonction noyau

Supposons que l'on dispose d'une fonction K dite de « fonction noyau » :

$$K(x_i, x_j) =< \Phi(x_i), \Phi(x_j) >$$

kernel trick

On peut dès lors effectuer tous les calculs dans le cadre des MVS en utilisant K, sans devoir explicitement transformer les données par la fonction Φ, donc, sans nécessairement connaître cette fonction Φ :

– lors de l'optimisation du problème quadratique, on remplace les produits scalaires $< \overrightarrow{x_i}, \overrightarrow{x_j} >$ par $K(x_i, x_j)$, soit :

$$L_D = \sum_{i=1}^{i=N} \alpha - \frac{1}{2} \sum_{i=1}^{i=N} \sum_{j=1}^{j=N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

à comparer à l'éq. (8.3) ;
– lors de la classification d'une nouvelle donnée $\overrightarrow{x}$, on calcule sgn $(< \overrightarrow{w}, \overrightarrow{x} > + b) = \sum_{i=1}^{i=N_s} \alpha_i y(s_i) K(s_i, x) + b$.

Donc, il n'y a rien de bien différent dans le cas des MVS non linéaires par rapport aux MVS linéaires concernant le principe de la méthode et des calculs.

8.2.2 Fonctions noyaux

condition de Mercer

Peut-on utiliser n'importe quelle fonction comme fonction noyau ?

Les fonctions noyaux acceptables doivent respecter la condition de Mercer. Une fonction $K(x, y)$ respecte cette condition si pour toute fonction $g(x)$ telle que $\int g(x)^2 dx$ est finie, on a $\iint K(x, y) g(x) g(y) dx dy \geq 0$.

Le respect de la condition de Mercer garantit que le problème quadratique possède une solution.

On connaît un certain nombre de fonctions noyaux, parmi lesquelles :
– noyau polynomial : $K(x, y) = (< \overrightarrow{x}, \overrightarrow{y} > + c)^m$, $m \in \mathbb{N}$ et $c > 0$;
– noyau gaussien : $K(x, y) = e^{\frac{||\overrightarrow{x} - \overrightarrow{y}||^2}{2\sigma^2}}$;
– noyau neuronal/sigmoïde : $K(x, y) = \tanh(\kappa < \overrightarrow{x}, \overrightarrow{y} > - \delta)$ où $\kappa > 0$ et $\delta > 0$ (*stricto senso*, cette fonction ne respecte pas la condition de Mercer pour toutes les valeurs de κ et δ).

En outre, un théorème indique que la combinaison linéaire de fonctions noyaux acceptables est une fonction noyau acceptable.

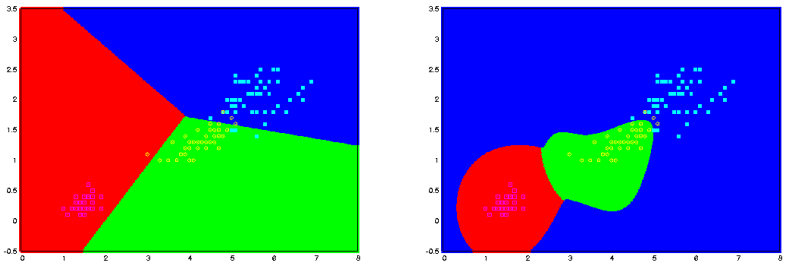
Question : face à un problème donné, quelle fonction noyau utiliser ?

Réponse : on ne sait pas.

8.3 Application

On applique ici les MVS au jeu de données « iris ». On ne considère que les deux attributs longueur et largeur des pétales. Vue la répartition par classes des données, on compare MVS linéaire et MVS à un noyau gaussien.

Pour la MVS linéaire, 6 exemples sont mal classés ; pour la MVS RBF, 4 sont mal classés. La figure 8.5 montre le découpage de l'espace des données en 3 classes pour les 2 MVS.



(a) MVS linéaire (b) MVS non linéaire à base de RBF

FIG. 8.5 – Sur le jeu de données « iris », découpage de l'espace des données par deux MVS, l'une linéaire, l'autre non linéaire. On a représenté également les exemples : les *setosa* en rose, les *versicolor* en jaune et les *virginica* en bleu clair.

8.4 Les logiciels libres pour MVS

Il y a un certain nombre de logiciels libres plus ou moins faciles à utiliser.

- `SVMTorch` : assez facile à utiliser : `http://www.idiap.ch/learning/SVMTorch.html`.
- `rainbow` : c'est un logiciel pour la classification de textes ; il comprend notamment un module MVS. Il est disponible à l'url `http://www.cs.cmu.edu/~mccallum/bow` ;
- `libsvm` : `www.csie.ntu.edu.tw/~cjlin/libsvm` : bibliothèque de fonctions ; nécessite de programmer ;
- `mySVM` : `http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html` bibliothèque de fonctions ; nécessite de programmer ;
- $\texttt{SVM}^{light}$: `http://svmlight.joachims.org` bibliothèque de fonctions ; nécessite de programmer.

8.5 Conclusion

- ne permet pas l'extraction d'un modèle compréhensible ;
- inadapté à la fouille de très grands volumes de données. Les meilleurs logiciels ne traitent qu'environ 10000 exemples : les calculs sont lourds ;
- choix du noyau : pas de solution à l'heure actuelle si ce n'est par essai/erreur ;
- fournit un optimum global, mais qu'est ce que cela signifie vraiment pour un jeu de données réel ?
- apprentissage incrémental possible ;
- pour faire de la fouille de données, il faut comprendre l'algorithme que l'on

utilise. Clairement, une bonne compréhension des MVS n'est pas aisée.

8.6 Exercices

Exercice 19 Soit le problème suivant :

$$\begin{cases} \text{minimiser } x_1^2 - 3x_2 \\ \text{avec la condition } 3x_1 - x_2 + 4 \geq 0 \end{cases}$$

Le résoudre par la méthode de Lagrange.

Exercice 20 Donner le lagrangien et son dual pour une machine à vecteurs supports linéaire dans le cas non séparable (*cf.* eq. (8.7)).

Exercice 21 Que pensez-vous de la fig. 8.5(b) ?

Exercice 22 Questions de réflexion :

1. est-ce-que les MVS constituent LA méthode à toujours utiliser puisque l'on a démontré qu'elle fournit un optimum global ?
2. dans la pratique, on constate que certains algorithmes ont de meilleures performances que les MVS. Comment expliquez-vous ce fait ?

Chapitre 9

Pour en finir avec la classification

Contenu

Pour terminer la discussion du problème de classification, on aborde ici deux problématiques :

- la combinaison de classeurs ;
- en plus des exemples (données étiquetées), si l'on dispose de données (non étiquetées) supplémentaires, ces données peuvent-elles être utilisées pour obtenir un classeur de meilleure qualité ?

Chacune de ces deux problématiques est discutée dans les deux sections qui suivent. Enfin, on termine ce chapitre par une synthèse sur les méthodes de classification.

9.1 Combinaison de classeurs

Plusieurs techniques de construction et combinaison de classeurs ont été proposées dont le *bagging* et le *boosting* sont les plus connues. On présente brièvement ces 2 approches.

Notons que d'autres concepts sont connexes à ces notions : d'une manière générale, on parle de méthodes d'ensemble, ou de *leveraging*.

9.1.1 *Bagging*

L'idée est de construire T classeurs à partir du jeu d'apprentissage, T étant fixé *a priori*. Quand ces classeurs sont construits, la prédiction de la classe d'une nouvelle donnée s'obtient par un vote majoritaire : *cf.* algorithmes 8 pour la construction des T classeurs et 9 pour la prédiction. On suppose ici que la classe y est codée par ± 1.

Algorithme 8 Algorithme de *bagging* : construction de T classeurs.

Nécessite: $\mathcal{X}_{\text{app}}$: ensemble d'exemples d'apprentissage
  pour t : $1 \rightarrow T$ **faire**
    échantillonner avec remise N exemples de $\mathcal{X}_{\text{app}}$
    $C_t \leftarrow$ classeur avec ces N exemples
  fin pour

Algorithme 9 Algorithme de *bagging* : prédiction à partir des T classeurs.

Nécessite: x : donnée dont il faut prédire la classe, T classeurs construits par l'algorithme 8
  pour t : $1 \rightarrow T$ **faire**
    $c_t \leftarrow$ classe prédite par la classeur C_t
  fin pour
  retourner la classe majoritaire parmi les $\{c_t\} =$ sgn $\sum_{t=1}^{t=T} C_t(x)$

L'algorithme des forêts aléatoires s'appuie sur le principe du *bagging* (*cf.* Breiman [2001]).

9.1.2 *Boosting*

Cette technique repose sur un résultat théorique fort que l'on peut énoncer intuitivement comme suit : à partir d'un ensemble de classeurs qui ont une probabilité supérieure au hasard de prédire correctement la classe d'une donnée, on peut construire par combinaison de ces classeurs de base un nouveau classeur dont la probabilité de succès peut être rendue arbitrairement proche de 1 (*cf.* Schapire [1989]).

Plusieurs algorithmes ont été proposés. Pratiquement, c'est AdaBoost qui est utilisé. Son principe est d'associer un poids à chaque exemple ; ce poids indique la difficulté de prédire la classe de cet exemple. Dès lors, AdaBoost va concentrer ses efforts sur les exemples dont il est difficile de prédire la classe : un premier classeur est construit sur l'ensemble des exemples ; ceux dont la classe est mal

prédite ont leur poids qui augmente, les autres ont leur poids qui diminue. Ce poids indique la probabilité que chaque exemple a d'être tiré lors d'un tirage au sort. Et on recommence avec la construction d'un second sur les exemples ainsi (re-)pondérés, et ainsi de suite jusqu'à obtenir un nombre T fixé de classeurs (*cf.* algorithme 10).

Algorithme 10 Algorithme AdaBoost : construction de T classeurs.

Nécessite: $\mathcal{X}_{\text{app}}$: ensemble d'exemples d'apprentissage
  pour chaque exemple $x_i \in \mathcal{X}_{\text{app}}$ **faire**
    $w_i \leftarrow 1/N$
  fin pour
  pour $t \in \{1, ..., T\}$ **faire**
    $C_t \leftarrow$ classeur construit avec ces N exemples pondérés par les w_i
    $e_t \leftarrow$ erreur de C_t mesurée sur ce jeu d'exemples avec cette pondération
    $\beta_t \leftarrow \frac{1}{2} \log\left(\frac{1-e_t}{e_t}\right)$
    pour chaque exemple $x_i \in \mathcal{X}_{\text{app}}$ **faire**
      $w_i \leftarrow w_i e^{-\beta_t y_i C_t(x_i)}$
    fin pour
    normaliser les poids (pour que leur somme fasse 1)
  fin pour

Pour prédire la classe d'une donnée, chaque classeur vote pour sa prédiction avec un poids β_t pour le classeur C_t. La classe prédite est la classe de poids maximal (*cf.* l'algorithme 11 dans le cas binaire), en supposant que la classe est codée par ± 1.

Algorithme 11 Algorithme AdaBoost : prédiction à partir des T classeurs.

Nécessite: x : donnée dont il faut prédire la classe, T classeurs construits par l'algorithme 10 et leurs coefficients β_t
  retourner $\text{sgn} \sum_{t=1}^{t=T} \beta_t C_t(x)$

Le *boosting* peut être appliqué à tout type de classeur. En particulier, il a été appliqué aux réseaux de neurones et aux arbres de décision. Dans ce dernier cas, pour des problèmes de classification binaire, plutôt que de construire des arbres de décision complets comme le fait C4.5, on peut ne construire que des arbres réduits à une simple racine (arbre n'ayant qu'un seul nœud : *decision stump*). Ainsi, on engendre autant de *stumps* qu'il y a d'attributs ; chacun teste un seul attribut et effectue ainsi une prédiction de la classe de la donnée qui lui est présentée.

On peut aussi construire des arbres de décision dans lesquels les nœuds contiennent des tests obtenus selon d'autres méthodes que dans le cas de C4.5 (*cf.* Dietterich [1999], Breiman [2001]). Ho [1998] propose de tester des combi-

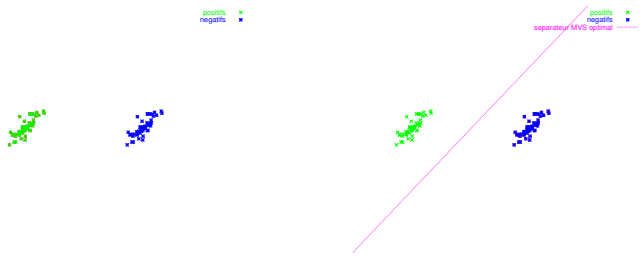


FIG. 9.1 – À gauche : un ensemble d'exemples linéairement séparables. À droite, le séparateur trouvé par une MVS linéaire.

naisons non linéaires d'attributs.

9.2 Apprendre avec des données non étiquetées

Problématique : on dispose rarement d'un grand nombre d'exemples car l'étiquetage, généralement manuel, est une opération coûteuse. Par contre, on dispose souvent en complément des exemples étiquetés d'un certain nombre, éventuellement important, de données non étiquetées. Peut-on utiliser ces données pour construire un classeur plus performant ?

La réponse peut-être « oui » : les figures 9.1 et 9.2 en illustrent l'idée.

On y reviendra au chapitre 10 lorsque certains outils auront été introduits (l'algorithme EM).

9.3 Synthèse des méthodes de classification

Au-delà de l'apparente diversité des approches que nous avons rencontrées, il est important de bien noter les réelles dissimilarités et les similarités entre chacune. L'explicitation du type de fonction de prédiction permet d'y voir plus clair que d'en rester aux niveaux descriptifs, tels que des arbres de décision, des réseaux de neurones, les k plus proches voisins, les machines à vecteurs supports ou les méthodes d'ensemble (*cf.* table 9.2). La table 9.1 résume également ce qui est appris, autrement dit, en quoi consiste le modèle construit par chacune des méthodes.

La table 9.2 permet de constater une grande similarité entre plusieurs des techniques, malgré leur apparente variétés. Ainsi, k plus proches voisins, perceptron multi-couches, machines à vecteurs supports, combinaisons de classeurs utilisent-ils des fonctions de prédiction très ressemblantes.

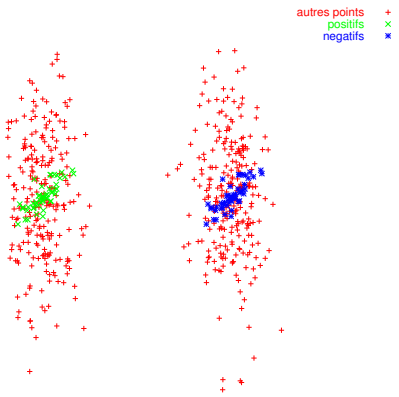


FIG. 9.2 – Schéma illustrant comment des données non étiquetées peuvent être utiles : on a le même jeu d'exemples qu'à la figure 9.1 auquel on a adjoint un ensemble de données non étiquetées (en rouge). On « voit » que le séparateur linéaire des positifs et des négatifs est probablement une droite à peu près verticale plutôt que le séparateur indiqué sur la figure 9.1.

TAB. 9.1 – Cette table résume en quoi consiste l'apprentissage dans les différentes techniques rencontrées.

méthode	ce qui est appris
ID3, C4.5, C5, CART	un arbre de décision
C4.5rules	un jeu de règles de classification
k plus proches voisins	rien
approche bayésienne	rien
perceptron multi-couches	les poids sur les connexions entre neurones
réseau RBF	les poids sur les connexions entre neurones
machines à vecteurs supports	les multiplicateurs de Lagrange (et éventuellement d'autres paramètres)
bagging	des classeurs
boosting	des classeurs et un poids pour chacun

TAB. 9.2 – Cette table résume comment est effectuée la prédiction de la classe d'une donnée quelconque pour les différentes techniques rencontrées. K est une fonction noyau (fonction à base radiale par exemple, notée G dans la table). S est une fonction sigmoïde (tanh par exemple).

méthode	fonction de prédiction de la classe d'une donnée x
ID3, C4.5, C5, CART	descente dans l'arbre de décision
C4.5rules	application des règles de classification
k plus proches voisins	$\sum_{v \in \{k\text{plus proches voisins}\}} \delta_v K(v,x)$
perceptron multi-couches	$\text{sgn}(\sum_i w_i S(\sum_j w_j x_j))$
réseau RBF	$\text{sgn}(\sum_i w_i G(\|\|x_j - c_j\|\|^2))$
machines à vecteurs supports	$\text{sgn}(\sum_{v \in \{\text{vecteurs supports}\}} \alpha_v K(v,x))$
bagging	$\text{sgn}(\sum_{t=1}^{t=T} C_t(x))$
boosting	$\text{sgn}(\sum_{t=1}^{t=T} \beta_t C_t(x))$

9.4 Logiciels libres

- le logiciel non libre C5 contient une option *boosting* qui n'est pas documentée. La version de démonstration permet d'expérimenter cette technique. Voir `http://www.rulequest.com`
- `weka` permet également d'expérimenter le *boosting* : `http://www.cs.waikato.ac.nz/~ml`
- `ADTree` possède une option de *boosting* de *stumps* : `http://www.grappa.univ-lille3.fr` ;
- voir aussi les *random forests* de L. Breiman : `http://www.stat.berkeley.edu/users/breiman/RandomForests`

9.5 Conclusion

On pourrait continuer encore à parler longuement du problème de classification. Par exemple, on n'a pas parlé du tout des problèmes multi-étiquettes où chaque donnée peut appartenir à plusieurs classes. On va cependant s'arrêter là.

On a passé beaucoup de temps sur la classification : c'est normal : c'est très utilisé, c'est ce que l'on maîtrise le mieux et on a présenté des choses qui sont utilisables en dehors du seul problème de classification.

De plus, au-delà des algorithmes et méthodes génériques que nous avons décrites, chacun peut imaginer de nouvelles méthodes de classification qui seront adaptées à son problème particulier. Quand on se lance dans la conception de

nouvelles méthodes, il est cependant indispensable de connaître ce qui existe déjà.

9.6 Exercices

Exercice 23 Vous voulez devenir riche ? Concevez un système qui prédise si une certaine chanson va être un tube.

Question 1 : Que pensez-vous de la faisabilité de l'entreprise ?

Question 2 : Comment vous y prenez-vous d'un point de vue technique ?

Chapitre 10

Segmentation

Contenu

Avec ce chapitre, on aborde la problématique de la segmentation de données. Cette tâche est très importante et plusieurs synonymes existent : partitionnement, catégorisation, classification non supervisée (anglicisme : *clustering*). En statistique, ce que nous dénommons ici un problème de segmentation est appelé un « problème de classification ».

10.1 Introduction

L'objectif de la segmentation est le suivant : on dispose de données non étiquetées. On souhaite les regrouper par données ressemblantes.

Cette manière de définir intuitivement l'objectif de la segmentation cache la difficulté de formaliser la notion de ressemblance entre deux données. Au-delà des algorithmes existant dont nous présentons quelques exemples dans la suite, une bonne partie de l'art à acquérir consiste ici à imaginer cette formalisation.

Ainsi, soit un ensemble $\mathcal{X}$ de N données décrites chacune par leurs P attributs. La segmentation consiste à créer une partition ou une décomposition de cet ensemble en groupes telle que :

critère 1. les données appartenant au même groupe se ressemblent ;

critère 2. les données appartenant à deux groupes différents soient peu ressemblantes.

Mesurer la ressemblance entre deux données

Clairement, la notion de « ressemblance » doit être formalisée. Cela est fait en définissant une distance entre tout couple de points du domaine $\mathcal{D}$. Toute la difficulté est là : définir correctement cette distance. De là dépend le résultat de la segmentation. On a déjà abordé ce point (*cf.* le chap. 5, section 5.1). Si l'utilisation d'une distance euclidienne est *a priori* une bonne idée, ce n'est pas la seule possibilité, loin s'en faut. En particulier, on a rarement d'informations concernant la pertinence des attributs : seuls les attributs pertinents doivent intervenir dans la définition de la distance. De plus, dans la segmentation que l'on aimerait obtenir, la pertinence des attributs peut dépendre du groupe auquel la donnée appartient.

Différents types de segmentation

Ne connaissant pas les critères de partitionnement *a priori*, on doit utiliser ici des algorithmes d'apprentissage non supervisés : ils organisent les données sans qu'ils ne disposent d'information sur ce qu'ils devraient faire.

Il existe deux grandes classes de méthodes :

- non hiérarchique : on décompose l'ensemble d'individus en k groupes ;
- hiérarchique : on décompose l'ensemble d'individus en une arborescence de groupes.

On peut souhaiter construire une décomposition :

- telle que chaque donnée appartienne à un et un seul groupe : on obtient une partition au sens mathématique du terme ;
- dans laquelle une donnée peut appartenir à plusieurs groupes ;
- dans laquelle chaque donnée est associée à chaque groupe avec une certaine probabilité.

Avant de poursuivre, notons que le problème de segmentation optimale en k groupes est $\mathcal{NP}$-complet (*cf.* annexe C). Il faut donc chercher des algorithmes calculant une bonne partition, sans espérer être sûr de trouver la meilleure pour les critères que l'on se sera donnés.

Idéalement, dans un processus de fouille de données, la segmentation doit être une tâche interactive : une segmentation est calculée et proposée à l'utilisateur qui doit pouvoir la critiquer et obtenir, en fonction de ces critiques, une nouvelle segmentation, soit en utilisant un autre algorithme, soit en faisant varier les paramètres de l'algorithme, ...

10.2 Segmentation non hiérarchique

Dans une approche hiérarchique, on souhaite obtenir une décomposition de l'ensemble de données $\mathcal{X}$ en K groupes non hiérarchisés que l'on notera G_1, G_2, ... G_K. On a : $\mathcal{X} = \bigcup_{i=1}^{i=K} G_i$.

Après un ensemble de définitions, on verra deux approches : dans la première, on obtient une partition au sens mathématique (chaque donnée est associée à un et un seul groupe) ; dans la seconde, chaque donnée est associée à chaque groupe avec une certaine probabilité.

centre de gravité

Définition 11 *Soit $\mathcal{X}$ un ensemble de donnée, chacune décrite par P attributs. On nomme « centre de gravité » g de $\mathcal{X}$ une donnée synthétique dont chaque attribut est égal à la moyenne de cet attribut dans $\mathcal{X}$. Soit, $g = (\overline{a_1}, \overline{a_2}, ...\overline{a_P})$.*

inertie

Définition 12 *L'inertie d'un ensemble $\mathcal{X}$ de N données :*

$$\mathcal{I} = \sum_{i=1}^{i=N} d^2(x_i, g) \tag{10.1}$$

où g est le centre de gravité de $\mathcal{X}$.

On peut aussi calculer l'inertie des individus appartenant à un groupe de la partition. On note $\mathcal{I}_{i\in\{1,...K\}}$ l'inertie du groupe G_i ; elle se calcule par l'équation (10.1). On note g_i le centre de gravité des points appartenant au groupe G_i.

inertie intraclasse

Définition 13 *On définit l'inertie intraclasse comme suit :*

$$\mathcal{I}_W = \sum_{i=1}^{i=K} w_i \mathcal{I}_i \tag{10.2}$$

où w_i est le poids du groupe G_i. Si toutes les données ont le même poids, le poids d'un groupe est $w_i = \frac{|G_i|}{N}$ où $|G_i|$ est le cardinal du groupe G_i.

inertie interclasse

Définition 14 *On définit l'inertie interclasse :*

$$\mathcal{I}_B = \sum_{i=1}^{i=K} w_i d^2(g_i, g) \tag{10.3}$$

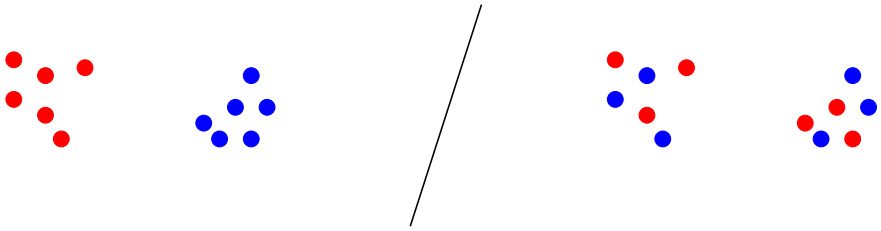


FIG. 10.1 – À gauche comme à droite, la distribution des points est la même, donc l'inertie $\mathcal{I}$ du nuage est la même. On a indiqué par des couleurs les points des deux groupes de la segmentation. À gauche, l'inertie intraclasse $\mathcal{I}_W$ est faible alors que l'inertie interclasse $\mathcal{I}_B$ est grande : les deux groupes sont nettement séparés spatialement. À droite, les deux groupes sont mélangés : l'inertie intraclasse $\mathcal{I}_W$ est grande alors que l'inertie interclasse $\mathcal{I}_B$ est faible.

Propriété 4 *Théorème de Huygens : pour un ensemble de données, on a*

$$\mathcal{I} = \mathcal{I}_W + \mathcal{I}_B \tag{10.4}$$

Remarque : $\mathcal{I}$ est une constante. Donc, $\mathcal{I}_W + \mathcal{I}_B$ est une constante, quelle que soit la segmentation en groupes du jeu de données (*cf.* fig. 10.1).

Pour créer une segmentation respectant les deux critères indiqués plus haut, il faut maximiser $\mathcal{I}_B$ (critère 1, *cf.* p. 130)) et minimiser en même temps $\mathcal{I}_W$ (critère 2, *cf.* p. 130)). Donc, d'après le théorème de Huygens, il suffit de respecter l'un des deux critères : le second sera respecté en même temps.

10.2.1 L'algorithme des centres mobiles

L'algorithme des centres mobiles est également dénommé k-moyennes[1], ou centroïdes. L'objectif est de segmenter les données en k groupes, k étant fixé *a priori*. L'idée de cet algorithme est très intuitive et, de fait, cet algorithme a été ré-inventé à plusieurs reprises. Il en existe de nombreuses variantes, en particulier l'algorithme bien connu des « nuées dynamiques ».

L'idée de l'algorithme des centres mobiles est la suivante : on part de K données synthétiques (c'est-à-dire des points de l'espace de données $\mathcal{D}$ ne faisant pas forcément parti du jeu de données) que l'on nomme des « centres ». Chaque centre caractérise un groupe. À chaque centre sont associées les données qui lui sont les plus proches ; cela crée un groupe autour de chaque centre. Ensuite, on calcule le centre de gravité de chacun de ces groupes ; ces k centres de gravité deviennent les nouveaux centres et on recommence tant que les groupes ne sont pas stabilisés, *i.e.* tant qu'il y a des données qui changent de groupe d'une itération à la suivante ou encore, tant que l'inertie varie substantiellement d'une

[1] en anglais : *k-means*

itération à la suivante (*cf.* l'algorithme 12). Cet algorithme converge en un nombre fini d'itérations.

Algorithme 12 Algorithmes des centres mobiles

Nécessite: 2 paramètres : le jeu de données $\mathcal{X}$, le nombre de groupes à constituer $K \in \mathbb{N}$

$I \leftarrow \infty$

prendre K centres arbitraires $c_k \in \mathcal{D}$

répéter

pour $k \in \{1, ...K\}$ **faire**

$G_k \leftarrow \emptyset$

fin pour

pour $i \in \{1, ...N\}$ **faire**

$k^* \leftarrow \arg\min_{k \in \{1,...,K\}} d(x_i, c_k)$

$G_{k^*} \leftarrow G_{k^*} \cup \{x_i\}$

fin pour

pour $k \in \{1, ...K\}$ **faire**

$c_k \leftarrow$ centre de gravité de G_k

fin pour

$I \leftarrow \mathcal{I}_W$

calculer $\mathcal{I}_W$

jusque $I - \mathcal{I}_W <$ seuil

10.2.2 Quelques remarques sur les centres mobiles

La segmentation obtenue dépend des centres initiaux

Lors de l'initialisation de l'algorithme, on prend K points dans l'espace de données au hasard. La segmentation calculée par les centres mobiles dépend de cette initialisation.

Pour contrer ce problème, on exécute plusieurs fois l'algorithme en prenant à chaque fois des centres initialisés différemment. On compare les segmentations obtenues à chaque itération et on retient celle dont l'inertie intraclasse est la plus faible. En général, un certain nombre de données se trouvent toujours regroupées ensemble, alors que d'autres ne le sont pas. On peut considérer que les premières indiquent nettement des regroupements, alors que les secondes correspondent à des données éventuellement atypiques, ou à des données bruitées. De toute manière, cette information est intéressante.

Le nombre de groupes

Le K choisi peut être mauvais. On peut tester plusieurs valeurs de K en exécutant plusieurs fois l'algorithme avec des K croissants. Pour chaque valeur

TAB. 10.1 – Caractéristiques du jeu de données représenté à la fig. 10.2. « moyenne théorique » : coordonnées des centres des 5 distributions. On n'indique que les moyennes des distributions normales ; les écarts-types sont tous égaux à 0,5. « moyenne trouvée par les centres mobiles » : les coordonnées des 5 centres trouvés par l'algorithme des centres mobiles. « moyenne trouvée par EM » : idem pour l'algorithme EM. Celui-ci estime également les écarts-types : la valeur estimée varie entre 0,44 et 0,54.

Groupe	G_1 μ_1	G_2 μ_2	G_3 μ_3	G_4 μ_4	G_5 μ_5
moyenne théorique	(0, 0)	(3, 5)	(-1, -3)	(7, 6)	(-1, 3)
moyenne trouvée par les centres mobiles	(0,03, 0,03)	(3.0, 5,0)	$(-1.0, -3{,}07)$	(7,02, 6,01)	$(--0{,}99, 2{,}99)$
moyenne trouvée par EM	(0, 0)	(3, 5)	$(-1, -3)$	(7, 6)	$(-1, 3)$

de K, on note l'inertie intraclasse ou le rayon moyen des groupes. Cette valeur décroît quand K augmente. En faisant un graphique représentant l'inertie intraclasse en fonction de K, on voit la bonne valeur de K : c'est celle à partir de laquelle $\mathcal{I}_W$ ne décroît plus substantiellement (*cf.* figure 10.3).

La convergence de l'algorithme des centres mobiles

Propriété 5 *L'algorithme des centres mobiles converge au bout d'un nombre fini d'itérations.*

On insiste sur le fait que l'algorithme trouve un optimum local.

10.2.3 Illustration des centres mobiles

On considère le jeu de données bi-dimensionnelles représenté à la figure 10.2. Ce jeu de données a été engendré en tirant 1000 points dans le plan, chacun étant issu de l'une des cinq distributions normales dont les paramètres sont indiqués dans la table 10.1. Il y a équi-probabilité de tirer un point dans chacune de ces cinq distributions.

On exécute plusieurs fois l'algorithme des centres mobiles pour des valeurs de K variant de 3 à 20. On obtient le graphique $\mathcal{I}_W$ en fonction de K de la figure 10.3.

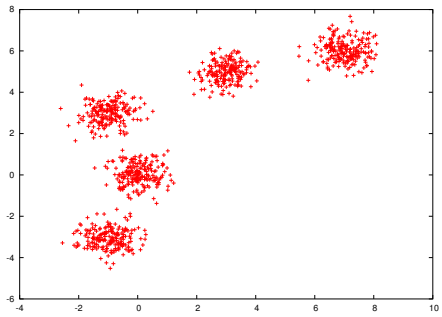


FIG. 10.2 – Jeu de données bi-dimensionnelles utilisé pour illustrer le fonctionnement de l'algorithme des centres mobiles.

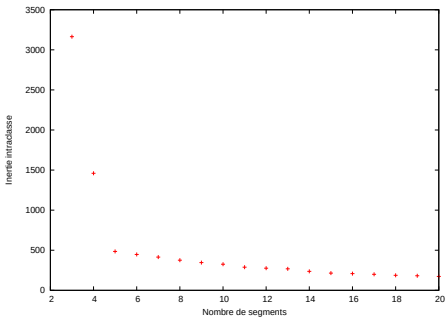


FIG. 10.3 – Exécution de l'algorithme des centres mobiles sur le jeu de données représenté à la fig. 10.2. On représente ici le rayon moyen des groupes en fonction de K. Celui-ci ne varie plus à partir de $K = 5$. Il y a donc 5 groupes.

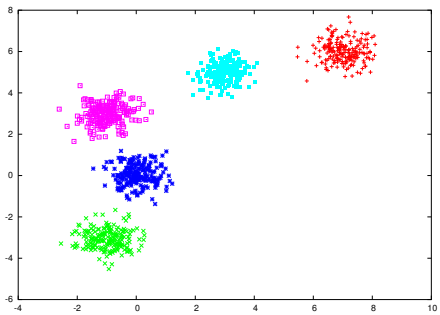


FIG. 10.4 – Affectation des données aux 5 groupes (un groupe = une couleur) par l'algorithme des centres mobiles sur le jeu de données représenté à la fig. 10.2.

Sur ce graphique, on voit une très nette inflexion pour $K = 5$. Pour cette valeur, la figure 10.4 représente l'affectation de chaque donnée aux 5 groupes. On obtient bien le résultat attendu.

Critique des centres mobiles

+ algorithme très simple à comprendre ;
+ algorithme très simple à programmer ;
o algorithme pas trop coûteux en temps d'exécution si le nombre de données n'est pas trop élevé. Quand N devient grand, les centres mobiles deviennent coûteux. D'autres algorithmes ont été proposés dans ce cas
- on peut obtenir des groupes vides, donc moins de K groupes non vides.

10.2.4 L'algorithme EM

On présente maintenant une toute autre approche de la segmentation non hiérarchique au travers de l'algorithme EM introduit par Duda and Hart [1973]. Une description détaillée est disponible dans MacLachlan and Krishnan [1997]. EM recherche un modèle statistique (une mixture) qui décrive au mieux le jeu de données à segmenter. Chaque donnée est associée à chaque segment (= une distribution de probabilités) avec une certaine probabilité.

Les mixtures

Une mixture est un ensemble de K distributions de probabilité. L'idée est que chaque distribution de probabilité décrit la distribution des valeurs d'attribut pour un groupe de la segmentation. Le nombre de segments est donc fixé ici à K.

Chaque distribution est décrite par un ensemble de paramètres θ. Une mixture de K distributions est la donnée de $\{(w_i, \theta_i), i \in \{1, ...K\}\}$ où w_i est le poids de la distribution i ; on a donc : $\sum_{i=1}^{i=K} w_i = 1$ et $w_i \geq 0, \forall i$. Les w_i sont les « coefficients de la mixture ». À partir du jeu de données, on se fixe K et le type de distributions de probabilités (donc le nombre et le sens des paramètres θ_i) et on cherche les w_i et θ_i qui modélisent les données au mieux.

Estimation des paramètres d'une distribution

Le cas le plus simple est celui où il n'y a qu'un seul attribut numérique qui possède une distribution normale dans chaque goupe, mais avec des moyennes et écarts-types différents dans chaque groupe. Le problème consiste alors à prendre un certain jeu de données et un nombre *a priori* de groupes et de déterminer la moyenne et la variance de chaque groupe, ainsi que la répartition de la population entre les différents groupes.

Plus précisément, supposons que la mixture soit composée de $K = 2$ distributions normales G_A et G_B, de moyenne et écart-type respectifs $\theta_A = (\mu_A, \sigma_A)$ et $\theta_B = (\mu_B, \sigma_B)$. La probabilité qu'une donnée appartienne à G_A est w_A et celle d'appartenir à G_B est w_B, avec $w_A + w_B = 1$. Le problème consiste à déterminer θ_A, θ_B et w_A étant donné l'ensemble de données.

Si on connaissait le groupe d'appartenance de chaque donnée, on pourrait aisément estimer la moyenne et l'écart-type de chaque groupe par les formules classiques. Pour estimer w_A, on pourrait utiliser la proportion de données appartenant à G_A.

D'un autre côté, si on connaissait ces paramètres, on pourrait déterminer la probabilité qu'une donnée x appartienne au groupe G_A par :

$$Pr[G_A|x] = \frac{Pr[x|G_A]\ Pr[G_A]}{Pr[x]} = \frac{Pr[x|G_A]\ w_A}{Pr[x]}$$

$Pr[x|G_A]$ est la probabilité que la donnée x résulte de la distribution G_A. Puisque l'on a supposé que les distributions sont gaussiennes, on a :

$$Pr[x|G_A] = Pr[x|\theta_A] = \frac{1}{\sqrt{2\pi}\sigma_A}\ e^{-\frac{x-\mu_A}{2\sigma_A^2}}$$

Le dénominateur $Pr[x]$ disparaît en calculant $Pr[G_B|x]$ et en normalisant les deux valeurs $Pr[G_A|x]$ et $Pr[G_B|x]$ par leur somme qui vaut 1 (cf chap. 4). On obtient ainsi la probabilité que x appartienne au groupe G_A ou au groupe G_B.

Plus généralement, si on a une mixture de K distributions, on estime la probabilité $Pr[G_i|x]$ par :

$$Pr[G_i|x] = \frac{Pr[x|G_i]w_i}{Pr[x]} = \frac{Pr[x|\theta_i]w_i}{\sum_{j=1}^{j=K} Pr[x|\theta_j]Pr[G_j]}$$

EM

Dans la réalité, on ne connaît pas toutes ces quantités : nombres de groupes (K), distributions des individus entre les différents groupes (w_i) et paramètres de ces distributions de probabilité (θ_i). En fait, on ne connaît même pas le type de distribution, mais on va supposer qu'elles sont d'un cretain type, par exemple, normal. EM est un algorithme qui estime ces paramètres. (EM n'est donc pas spécifiquement un algorithme de segmentation, mais il peut être utilisé pour faire de la segmentation.) On commence par choisir le nombre et le type de distributions composant la mixture. Ces hypothèses permettent de définir exactement le nombre et le type de paramètres décrivant la mixture. Dès lors, EM part d'une estimation initiale de ces paramètres et, itérativement, calcule les probabilités que chaque individu appartienne à tel ou tel groupe et utilise ces informations pour maximiser la probabilité que les paramètres des distributions collent aux données. EM trouve ainsi un jeu de paramètres de vraisemblance maximale (ML). C'est une méthode de gradient : EM maximise la vraisemblance que les données résulte d'une certaine mixture.

Considérons une mixture de K gaussiennes mono-dimensionnelles. Ayant estimé la probabilité pour chaque individu d'appartenir à chacune des distributions (on note $w_{i,k}$ l'estimation de la probabilité que x_i appartienne à la distribution k, *i.e.* l'estimation du terme $Pr[G_k|x_i]$), on peut estimer la moyenne et l'écart-type du groupe G_k par :

$$\mu_k = \frac{w_{1,k}x_1 + w_{2,k}x_2 + ...w_{n,k}x_n}{\sum_{i=1}^{i=N} w_{i,k}} \tag{10.5}$$

$$\sigma_k = \frac{\sqrt{w_{1,k}(x_1-\mu_k)^2 + w_{2,k}(x_2-\mu_k)^2 + ... + w_{n,k}(x_n-\mu_k)^2}}{\sum_{i=1}^{i=N} w_{i,k}} \tag{10.6}$$

$$\tag{10.7}$$

À partir de ces nouvelles estimations des paramètres des distributions de la mixture, on estime la nouvelle valeur des $w_{i,k}$, et on recommence l'estimation des paramètres.

On peut montrer que ce processus converge vers un optimum local (*cf.* [Bishop, 1995, p. 65-68] , MacLachlan and Krishnan [1997]).

On mesure la vraisemblance que le jeu de données $\mathcal{X}$ soit issu de la mixture dont les paramètres ont été estimés à une certaine itération. Cette vraisemblance est :

$$\mathcal{L}(\{w_k, \theta_k\}_{k\in\{1,...,K\}}|\mathcal{X}) = \Pi_{i=1}^{i=N} \sum_{k=1}^{k=K} (w_{i,k} Pr[x_i|\theta_k]) \tag{10.8}$$

Cette quantité mesure la qualité de la segmentation et augmente à chaque

itération de EM. Au bout d'un certain nombre d'itérations, cette vraisemblance ne varie presque plus ; on arrête alors les itérations.

Notons que dans les faits, on calcule le logarithme de la vraisemblance ; en effet, les probabilités intervenant dans son calcul sont généralement très petites et leur produit encore plus petit. Travailler avec des logarithmes amoindrit ce problème de petits nombres. On a donc la log-vraisemblance définie par :

$$\log(\mathcal{L}) = \sum_{i=1}^{i=N} \log\left(\sum_{k=1}^{k=K} w_{i,k} P(x_i|\theta_k)\right) \tag{10.9}$$

Pour diminuer le problème de convergence vers un optimum local, on peut relancer EM plusieurs fois avec des valeurs initiales des paramètres différentes puis utiliser la log-vraisemblance pour choisir le meilleur optimum : c'est celui pour lequel cette quantité est la plus grande.

On peut utiliser cette technique pour déterminer le nombre de groupes K. On exécute EM avec des valeurs de K croissantes et on s'arrête lorsque la log-vraisemblance ne varie plus significativement entre deux valeurs de K successives (*cf.* l'algorithme des centres mobiles, section 10.2.2).

Plus précisément, EM peut s'exprimer comme indiqué par l'algorithme 13 dans le cas d'une mixture de deux distributions de probabilités normales.

Si les données sont décrites par plusieurs attributs, on peut étendre aisément EM. Si les attributs sont indépendants entre-eux, on peut multiplier les probabilités concernant chacun des attributs comme dans le cas du raisonnement bayésien naïf. Sinon, on utilise la covariance entre les attributs.

Pour les attributs nominaux (par exemple, un attribut qui peut prendre ν valeurs différentes), on définit ν nombres, chacun associé à l'une des valeurs possibles de l'attribut. Chaque nombre représente la probabilité que l'attribut prenne cette valeur particulière. S'il y a K groupes, on définit en fait ν K nombres, chacun étant la probabilité que l'attribut ait cette valeur s'il appartient à ce groupe.

À moins d'en savoir plus sur la distribution de leurs valeurs, pour les attributs numériques :

- si sa valeur n'est pas bornée, on utilise une loi de distribution de probabilité normale (comme vu plus haut) ;
- si sa valeur possède une borne inférieure, on utilise une loi gaussienne log-normale (*cf.* fig. 10.5) : pour $x > 0$ (*i.e.*, la borne inférieure est 0),

$$Pr[x] = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x - \zeta)^2}{2\sigma^2}} dx$$

- si sa valeur est comprise dans l'intervalle $[x_{min}, x_{max}]$, on utilise une loi gaussienne *log-odds* :

$$Pr[x] = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\ln(\frac{x-x_{min}}{x_{max}-x}) - \zeta)^2}{2\sigma^2}} dx$$

Algorithme 13 Algorithme EM pour déterminer les paramètres d'une mixture de K distributions. Voir le texte pour les notations et pour les formules de calcul pour des distributions gaussiennes.

Nécessite: 2 paramètres : le jeu de données $\mathcal{X}$, une mixture de paramètres $\theta_{k,k\in\{1,...,K\}}$

initialiser, éventuellement aléatoirement, les paramètres de la mixture : les θ_k et les w_k

$\mathcal{L}_{\text{new}} \leftarrow -\infty$

répéter

  $\mathcal{L} \leftarrow \mathcal{L}_{\text{new}}$

  pour $k \in \{1, ...K\}$ **faire**

    pour $i \in \{1, ...N\}$ **faire**

      estimer $w_{i,k}$: la probabilité pour x_i d'appartenir à la distribution k : $w_{i,k} = Pr[G_k|x_i]$ à l'aide de la formule décrivant la distribution du groupe G_k.

    fin pour

    $w_k \leftarrow \sum_{i=1}^{i=N} w_{i,k}$

  fin pour

  pour $k \in \{1, ...K\}$ **faire**

    calculer θ_k

  fin pour

  $\mathcal{L}_{\text{new}} \leftarrow$ log-vraisemblance

jusque $|\mathcal{L} - \mathcal{L}_{\text{new}}| <$ seuil

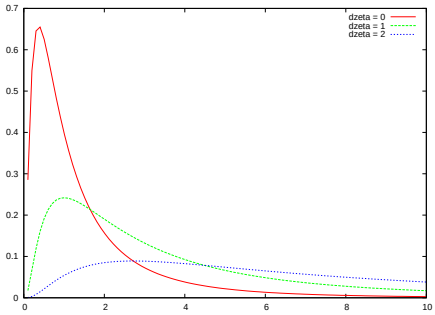


FIG. 10.5 – Exemple de distribution log-normale pour $\zeta = 0$, $\zeta = 1$ et $\zeta = 2$: quand ζ augmente, la distribution s'aplatit. L'écart-type est pris égal à 1 dans les 3 cas.

– si sa valeur est entière, on utilise une loi de Poisson :

$$Pr[x] = \frac{\lambda^x e^{-\lambda}}{x!}$$

pour $x \in \mathbb{N}$. La moyenne et la variance d'une loi de Poisson sont toutes deux égales à λ. Rappelons qu'une loi de Poisson tend vers une loi normale de variable centrée réduite $\frac{X-\lambda}{\sqrt{\lambda}}$ quand λ tend vers l'infini.

Illustration du fonctionnement d'EM

Voir la figure 10.6.

Comparaison expérimentale d'EM et centres mobiles

On reprend le même exemple que pour les centres mobiles : le jeu de données est constitué de 1000 points tirés au hasard dans le plan selon une mixture de 5 distributions normales équi-probables dont les paramètres sont donnés dans la table 10.1. Les résultats sont donnés dans cette même table. On observe la qualité de l'estimation ; la log-vraisemblance obtenue vaut -3.02.

EM et la classification

On peut utiliser EM dans le cadre de l'apprentissage semi-supervisé (*cf.* chap. 9). EM permet d'estimer la valeur la plus probable de variables cachées. On peut considérer que la classe de données non étiquetées est une variable cachée.

Supposons que les données soient des textes. Les exemples sont classés par catégories. Un classeur de Bayes naïf peut être utilisé pour estimer initialement $Pr[y_j|x_i]$ (*cf.* chap. 4). Avec ce modèle, on peut estimer $Pr[y_j|d_i]$ pour chaque donnée d_i et pour chaque classe y_j (phase E d'EM). On peut alors associer la classe MAP à chaque donnée (phase M d'EM). Il reste à itérer le processus. Cette approche a été décrite et expérimentée par Nigam and Ghani [2000].

10.2.5 Autres algorithmes de segmentation non hiérarchique

Les algorithmes de segmentation non hiérarchiques sont très nombreux. On en cite et discute brièvement quelques-uns ici.

autoclass

`autoclass` est une extension très enrichie d'EM : `autoclass` teste plusieurs nombre de segments (valeur de K), teste plusieurs initialisations des paramètres, prend en charge des attributs nominaux. `autoclass` ne s'applique qu'au cas où

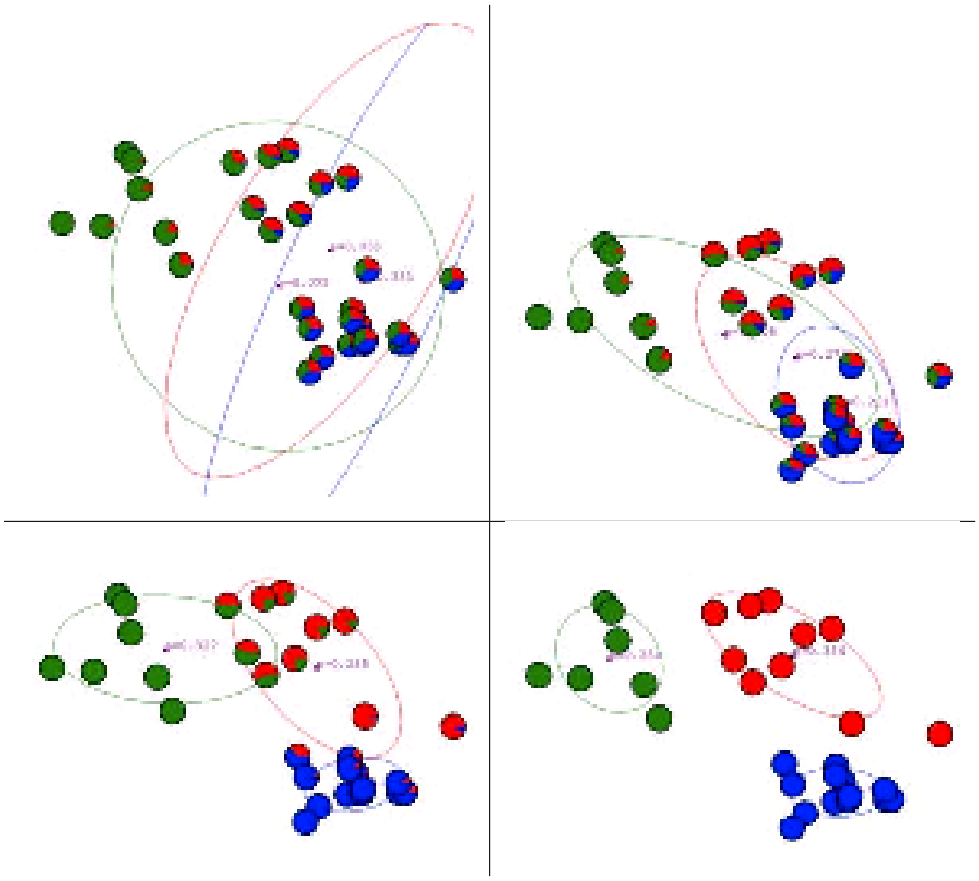


FIG. 10.6 – Ces quatre schémas illustrent graphiquement le fonctionnement d'EM. On suppose travailler sur une mixture de 3 gaussiennes en 2 dimensions. Pour 4 itérations de l'algorithme (en haut à gauche : l'initialisation, en haut à droite : après une itération ; en bas à gauche : après trois itérations ; en bas à droite : après 20 itérations et convergence), on indique par des ellipses les distributions normales estimées (une couleur pour chacune : pour chacune des 3 distributions, l'ellipse est centrée sur la moyenne estimée et ses petit et grand axes indiquent 2 écart-types) et, pour chaque donnée, un camembert indiquant la probabilité estimée qu'elle soit issue de chacune des 3 distributions. Au fur et à mesure des itérations, on voit les distributions se séparer et les camemberts devenir monochrome, signe que l'estimation d'appartenance de chaque donnée devient de plus en plus certaine.

chaque donnée est affectée à un et un seul segment [Cheeseman and Stutz, 1996].

Segmentation spectrale

Jusqu'à maintenant (depuis le début de ces notes de cours), on a toujours considéré que les données sont représentées par les valeurs de leurs attributs. C'est loin d'être la seule possibilité. Considérons le jeu de données représenté à la fig. 10.7(b), les couleurs indiquant les groupes. Pour nous humains, ces groupes sont naturels car chacun constitué de points à la suite des uns des autres, formant ces vermicelles. Si on représente chacun des points par ses coordonnées dans le plan, les algorithmes que nous avons vus jusqu'à maintenant sont incapables de reconstruire ces groupes, qui nous semblent si naturels.

Remarque

Avant de lire la suite, réflêchissez aux résultat que produirait les algorithmes de segmentation que nous avons rencontrés jusqu'à maintenant en supposant que les points sont représentés par leurs coorodnnées dans le plan.

Les groupes que nous voulons exhiber étant formés de points qui, pris par paires adéquates, sont très proches les uns des autres (remarquez bien que deux points quelconques d'un même vermicelle peuvent être très éloignés). Donc, une idée qui est tentante est de représenter chaque point par la distance entre lui et les autres points et d'utiliser ensuite une méthode capable de trouver les points qui sont reliés de proche en proche. De manière réciproque, on peut aussi utiliser non pas la distance entre deux points, mais un indice de proximité (l'inverse la distance par exemple) ; en particulier, on peut utiliser une fonction à base radiale (*cf.* 5.2.1) par exemple. Rappelons qu'avec ce genre de fonction noyau, dès que deux points sont relativement éloignés, leur proximité est très proche de 0 du fait de la décroissance exponentielle. Donc, quand deux points sont un peu éloigné, leur proximité est tellement petite que l'on peut la considérer comme nulle.

Poursuivons notre réflexion sur cette voie : on construit donc une représentation matricielle dont l'élément (i, j) est la distance entre la donnée x_i et la donnée x_j. Supposons maintenant que par un heureux hasard, les éléments d'indice inférieur à une certaine valeur i_b appartiennent tous au vermicelle bleu foncé, les éléments d'indice compris entre i_b et i_v soient ceux du vermicelle vert, ceux compris entre i_v et i_r soient ceux du vermicelle rose, ceux compris entre i_r et i_o soient ceux du vermicelle rouge et ceux d'indice supérieur à i_o soient ceux du vermicelle bleu clair.

Remarque

Ici, il est bon d'essayer de voir à quoi ressemble cette matrice avant de lire la suite.

Ce heureux hasard fait que si les considère les i_b premières lignes, les i_b premières colonnes seront non nulles et toutes les autres colonnes seront nulles,

car correspondant à des points éloignés de tout point du vermicelle bleu, de même pour tous les groupes de lignes suivant correspondants aux différents vermicelles. La matrice va donc avoir la structure suivante : des blocs carrés de valeurs non nulles situés sur la diagonale principale, des 0 ailleurs. Chaque bloc correspond à un groupe.

Dans la réalité, on ne peut pas compter sur le hasard : les valeurs nulles et les valeurs non nulles sont mélangées ; seulement, on sait que si les points étaient adéquatement ordonnés, la matrice aurait cette structure et, dès lors, il serait très simple d'extraire les groupes.

Par chance, il existe une opération mathématique qui permet d'extraire cette information : la décomposition spectrale, c'est-à-dire, le calcul des vecteurs propres. L'idée de la segmentation spectra[2]l est exactement celle-là : construire une matrice de dissimilarité du jeu de données puis en effectuer une décomposition spectrale.

Ces méthodes proviennent de deux sources :

1. approche de Spielman and Teng [1996] : on voit les points comme les nœuds d'un graphe et les étiquettes des arcs représentent la similarité entre les nœuds à chaque bout. La segmentation en deux groupes se ramène à un problème de partionnement de ce graphe, problème bien connu dans le champ de l'optimisation combinatoire (*cf.* l'annexe C). On a montré que le deuxième vecteur propre fournit une approximation de qualité garantie de la coupe optimale. Pour obtenir une K-partition, on peut soit répéter récursivement, soit utiliser les K vecteurs propres principaux (Malik et al. [2001]). Cette seconde solution a été montrée expérimentalement comme étant meilleure que la première. L'algorithme décrit ici suit cette idée.
2. approche de Perona and Freeman [1998] : on cherche le vecteur p tel que $\sum_{i,j=1}^{N}(A_{i,j} - p_i p_j)^2$ soit minimal où A est une matrice d'affinité (ou matrice de similarité : $A_{i,j}$ indique la similarité entre les points i et j ; A est symétrique et $A_{i,i} = 0$). A étant symétrique ($A = A^T$)), on sait que $p = \sqrt{\lambda_1}\overrightarrow{1}$.

Notons que Weiss [1999] a synthétisé différentes approches et tenté d'expliquer pourquoi elles fonctionnent.

L'algorithme proposé par Bolla [1991], repris par Ng et al. [2002], est donné en 14. Une illustration graphique de son utilisation est donnée à la figure 10.7.

Critiques :

\+ résultats assez impressionnants ;

o/- pas de mesure de la qualité de la segmentation obtenue : on « voit » que le résultat obtenu est bon ; de-là à le mesurer automatiquement...

\- numériquement lourd : extraction de vecteurs propres ce qui limite la taille du jeu de données ;

[2] *spectral clustering* en anglais

Algorithme 14 Un algorithme de segmentation spectrale, l'algorithme de Bolla [1991], Ng et al. [2002]. On dispose de N points notés x_i à regrouper en K groupes.

Nécessite: 2 paramètres : le jeu de données $\mathcal{X}$, le nombre de groupes à constituer $k \in \mathbb{N}$

construire la matrice de Gram : $G = (g_{i,j} = e^{\frac{-||x_i - x_j||^2}{2\sigma^2}}$ pour $i \neq j, 0$ sinon) : G est une matrice de similarité utilisant le noyau gaussien

construire H, une matrice diagonale telle que $h_{i,i} = \sum_{j=1}^{j=N} g_{i,j}$; $h_{i,i}$ est donc la somme des similarités entre x_i et les autres points

calculer $L = H^{-1/2} G H^{-1/2}$: on obtient pour $i \neq j$:

$$l_{i,j} = \frac{g_{i,j}}{\sqrt{(\sum g_{i,k})(\sum g_{j,k})}}$$

et une diagonale nulle

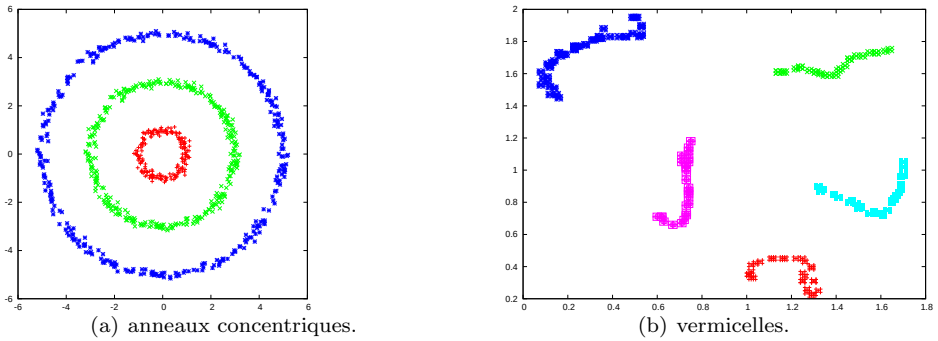
calculer les K premières vecteurs propres de L (associés aux K plus grandes valeurs propres)

construire la matrice E dont la i^{e} colonne est le i^{e} vecteur propre (ordonnés par valeur propre associée décroissante)

construire la matrice F en normant les lignes de E, soit : $F_{i,j} = \frac{E_{i,j}}{\sqrt{\sum_l E_{i,l}^2}}$

considérer chaque ligne de F comme un point et exécuter les centres mobiles sur ces points pour obtenir K groupes

affecter le point x_i au groupe j si et seulement si la ligne i de la matrice F est affectée au groupe j



(a) anneaux concentriques. (b) vermicelles.

FIG. 10.7 – Illustration des résultats de la segmentation spectrale. À gauche, 3 anneaux concentriques ; à droite : 5 formes vermicelliques. Les couleurs indiquent les regroupemets constitués par l'algorithme.

- dans l'algorithme de Ng, Jordan et Weiss, pas de possibilité de classer de nouveaux points une fois la segmentation effectuée.

Algorithmes de projection

Une autre approche, complémentaire, est constituée des algorithmes de projection. Du fait de son importance, elle fait l'objet du chap. 11.

10.3 Segmentation hiérarchique

En principe, il existe deux classes de méthodes de segmentation hiérarchique :

segmentation hiérarchique ascendante qui part des N points comme N groupes à partir desquels on construit une partition à $N-1$ classes par fusion de deux groupes, puis $N-2$ groupes par fusion de deux groupes, ... jusqu'à avoir rassemblé tous les points dans un seul groupe ;

segmentation hiérarchique descendante qui rassemble initialement les N points dans un seul groupe à partir duquel on construit 2 groupes, puis 3, ... puis N.

Les méthodes descendantes ne sont pas vraiment utilisées dans la pratique. On se concentre donc sur l'approche ascendante dans la suite.

10.3.1 Méthode ascendante

Donc, le principe d'une méthode de segmentation hiérarchique ascendante est le suivant : on part de N groupes, que l'on réduit à $N-1$, puis à $N-2$, ... On passe de $k+1$ à k groupes en regroupant deux groupes. On obtient ainsi une hiérarchie (ou arbre de segmentation : « dendogramme ») dans lequel on peut facilement retrouver k groupes en le coupant à un certain niveau (*cf.* fig. 10.8). Le principe de l'algorithme est donné en 15.

Algorithme 15 Segmentation hiérarchique

Nécessite: 1 paramètre : le jeu de données $\mathcal{X}$

initialiser les N groupes à raison d'une donnée par groupe : $G_i \leftarrow x_i$
marquer tous les G_i comme « prenable »
pour d de $N+1$ à $2N-1$ **faire**
  chercher les deux groupes prenables à fusionner : G_i et G_j
  les fusionner : $G_d \leftarrow G_i \cup G_j$
  marquer G_i et G_j comme « non prenable »
  marquer G_d comme « prenable »
fin pour

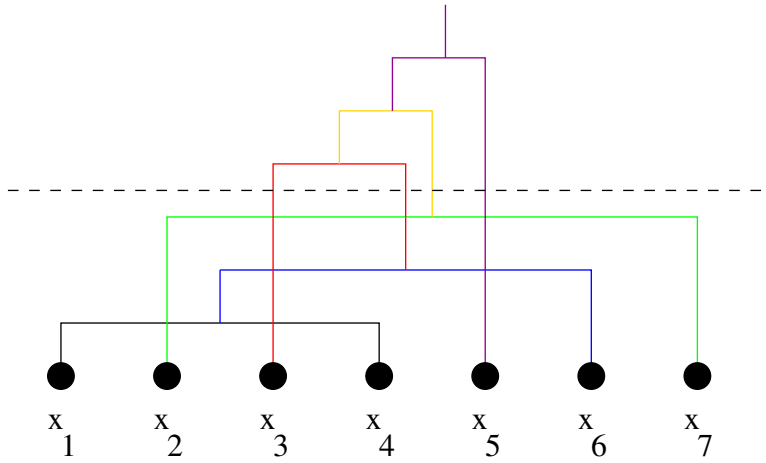


FIG. 10.8 – Illustration graphique de la méthode pour passer d'un dendogramme à une segmentation. On suppose avoir 7 données, les points notées x_1 à $_7$. L'algorithme construit la hiérarchie indiquée avec des couleurs : d'abord, on regroupe x_1 et x_4, puis ce groupe avec x_6, ... L'ordre dans lequel sont faits les regroupements est indiqué par l'altitude au-dessus des points. Pour aider à les distinguer, on a utilisé une couleur différente pour chaque nouveau regroupement : d'abord noir, puis bleu, vert, rouge, jaune et magenta. Une fois obtenue cette hiérarchie, on peut obtenir une segmentation en 4 groupes en « découpant » l'arbre là où il y a quatre branches verticales (découpage selon les pointillés). Les 4 morceaux qui tombent constituent les 4 groupes, soit : x_5 seul, x_3 seul, un groupe composé de x_2 et x_7 et un groupe composé des données restantes.

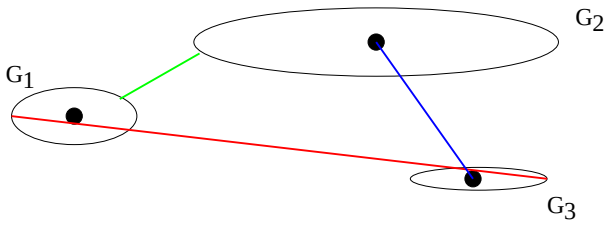


FIG. 10.9 – Une illustration des notions de 3 sauts définis dans le texte. On considère que l'on a trois groupes G_1, G_2 et G_3, chacun représenté par une ellipse. Le saut minimal (en vert) regroupe les groupes G_1 et G_2, le saut maximal (en rouge) regroupe G_1 et G_3 et le saut moyen (en bleu) regroupe G_2 et G_3. Le schéma est fait de telle manière que le regroupement soit différent dans chacun des cas de saut ; ceci n'est pas obligatoire. (Le schéma a été fait de manière approximative pour illustrer les 3 notions de saut, rien de plus.) À l'œil, quels groupes seront regroupés si l'on applique la méthode de Ward ?

Reste à expliciter comment on détermine les deux groupes à fusionner. Intuitivement, on veut fusionner les groupes dont les éléments sont les plus ressemblants. Donc, on utilise à nouveau une notion de dissimilarité entre données et entre groupes.

Différentes approches sont possibles (*cf.* fig. 10.9) :

- saut minimal : on fusionne les deux groupes entre lesquels la plus petite dissimilarité est minimale ;
- saut maximal : on fusionne les deux groupes entre lesquels la plus grande dissimilarité est la plus petite ;
- saut moyen : on fusionne les deux groupes entre lesquels la dissimilarité moyenne (donc, la dissimilarité entre leurs centres de gravité) est la plus petite ;
- méthode de Ward : on fusionne les deux groupes pour lesquels la perte d'inertie interclasse est la plus faible. De cette manière, on effectue la fusion qui entraîne la plus faible hétérogénéité.

La méthode de Ward

Dans la méthode de Ward, la perte d'inertie due à la fusion de deux groupes G_i et G_j est :

$$\delta_{G_i,G_j} = \frac{w_i w_j}{w_i + w_i} \, d^2(g_i, g_j) \tag{10.10}$$

où w_i et w_j sont respectivement la proportion de données s dans le groupe G_i ($\frac{|G_i|}{N}$) et dans le groupe G_j et g_i (resp. g_j) est le centre de gravité de G_i

(resp. G_j). On peut utiliser cette quantité en guise de dissimilarité entre deux groupes.

Quand on a fusionné deux groupes G_i et G_j, la dissimilarité entre ce regroupement et un autre groupe G_k peut se calculer par l'équation suivante :

$$\delta_{G_k,G_i\cup G_j} = \frac{(w_i + w_k)\ \delta_{G_i,G_k} + (w_j + w_k)\ \delta_{G_j,G_k} - w_k\ \delta_{G_i,G_j}}{w_i + w_j + w_k} \tag{10.11}$$

En utilisant cette distance, dans l'algorithme 15 de segmentation hiérarchique, on obtient « l'algorithme de Ward ».

Complexité temporelle et critique

Telle que la méthode de segmentation hiérarchique a été décrite ici, sa complexité temporelle est $O(N^3)$: c'est énorme. Aussi, cette méthode qui donne des résultats très statisfaisants ne peut-elle pas être employée dès que le nombre de données est un peu important.

Illustration de la segmentation hiérarchique

On a repris l'exemple des 5 groupes de points utilisés plus avec l'algorithme des centres mobiles puis EM (*cf.* table 10.1 et fig. 10.2).

On a effectué une segmentation hiérarchique ascendante sur ce jeu de données en utilisant le saut moyen pour décider des groupes à regrouper à chaque étape. Du dendogramme obtenu a été déduit 5 groupes. On obtient exactement les groupes attendus, les mêmes que ceux obtenus par les centres mobiles (*cf.* 10.4).

10.4 Application au jeu de données « iris »

On applique les différents algorithmes au jeu de données « iris ». Les données y étant regroupées par classe, l'attribut de classe est omis ; chaque donnée est donc un quadruplet, la classe permettant de voir si l'algorithme effectue une segmentation compatible des données avec leur classe connue.

10.4.1 Les centres mobiles sur les « iris »

Segmentation en 3 groupes

La distance entre deux données est la distance euclidienne.

On effectue 10 exécutions des centres mobiles et on retient la segmentation produisant l'inertie minimale intraclasse.

On obtient en fait des segmentations différentes ayant même inertie intraclasse (88.13). Par exemple, on a obtenu ces deux matrices de confusion :

	effectif	*setosa*	*versicolor*	*virginica*
groupe 0	60	0	76	23
groupe 1	51	98	1	0
groupe 2	39	0	7	92

et

	effectif	*setosa*	*versicolor*	*virginica*
groupe 0	58	0	79	20
groupe 1	50	100	0	0
groupe 2	42	0	9	90

Lecture de ces matrices de confusion : chaque valeur indique la proportion de données d'une certaine classe qui a été mise dans un groupe donné. Par exemple, dans la première matrice, la valeur 76 indique que 76 % des exemples de la classe *versicolor* sont regroupés dans le groupe 1.

Chaque ligne fait 100 % aux erreurs d'arrondi près.

La colonne effectif indique le nombre d'exemples appartenant aux groupes.

Dans les deux cas, les groupes sont assez équilibrés (*cf.* les effectifs). On constate que les groupes correspondent assez bien aux classes. En particulier, les *setosa* sont tous regroupés dans un seul groupe dans les deux cas ; cela n'est pas pour nous étonner cette classe étant linéairement séparable et éloignée des deux autres.

Détermination du nombre de groupes

On veut maintenant déterminer le nombre de groupe optimal. Pour cela, on effectue 10 exécutions des centres pour les valeurs de K comprises entre 2 et 10. La figure 10.10 illustre le résultat. On note que l'inflexion se produit vers $K = 3$, $K = 4$.

Du fait de ce résultant hésitant, on effectue une segmentation en 4 groupes par les centres mobiles. Comme précédemment, on construit 10 segmentations et on conserve celle dont l'inertie intraclasse est minimale.

La matrice de confusion est la suivante :

	effectif	*setosa*	*versicolor*	*virginica*
groupe 0	28	0	96	3
groupe 1	19	0	0	100
groupe 2	53	0	43	56
groupe 3	50	100	0	0

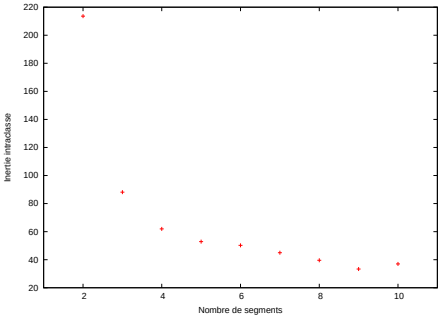
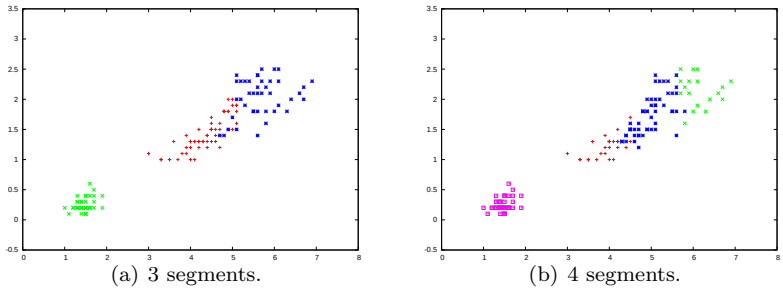


FIG. 10.10 – Inertie intraclasse de la segmentation construite par les centres mobiles pour différentes valeurs de K, sur le jeu de données « iris ».



(a) 3 segments. (b) 4 segments.

FIG. 10.11 – On a représenté les segmentations construites par les centres mobiles : à gauche, en 3 segments, à droite, en 4 segments. Les couleurs indiquent les groupes construits par l'algorithme. Les données sont représentées dans le plan (longueur des pétales, largeur des pétales).

Les *setosa* constituent un groupe homogène qui les contient tous, ce qui n'est pas étonnant puisqu'il se distingue nettement des deux autres classes.

Les *virginica* sont disséminés entre 3 groupes, l'un ne contenant que des *virginica*. Un groupe ne contient presque que des *versicolor*. Le dernier groupe (2) contient un mélange de *versicolor* et *virginica* : il correspond à l'intersection entre les deux classes.

On retrouve ces interprétations sur la figure 10.11.

10.4.2 EM sur les « iris »

On applique EM à la recherche d'une mixture de 3 gaussiennes 4-D sur le jeu de données « iris ».

Pour mémoire, rappelons les statistiques concernant ces 3 distributions mesurées sur le jeu de données :

classe	longueur des pétales		largeur des pétales	
	moyenne	écart-type	moyenne	écart-type
setosa	1.46	0.17	0.24	0.11
versicolor	4.26	0.47	1.33	0.20
virginica	5.55	0.55	2.03	0.27

Naturellement, le poids de chaque composante de la mixture est 1/3 puisqu'il y a autant de données dans chacune des 3 classes.

Les résultats suivants ont été obtenus par EM :

groupe	poids	longueur des pétales		largeur des pétales	
		moyenne	écart-type	moyenne	écart-type
0	0.30	4.22	0.47	1.30	0.19
1	0.33	1.46	0.17	0.24	0.11
2	0.36	5.48	0.57	1.99	0.29

La comparaison des deux tableaux indique que les *setosa* sont parfaitement retrouvés dans le groupe 1. Pour les deux autres groupes, les statistiques sont très proches.

Pour avoir plus de précisions, on peut construire la matrice de confusion :

	effectif	*setosa*	*versicolor*	*virginica*
groupe 0	45	0	43	2
groupe 1	50	50	0	0
groupe 2	55	0	7	48

dans laquelle toutes les valeurs sont des effectifs. Cette matrice ne fait que confirmer ce que l'on avait interprété précédemment.

10.4.3 Segmentation hiérarchique des « iris »

La distance entre deux données est la distance euclidienne.

On applique l'algorithme de segmentation vu plus haut. Le dendogramme est ensuite cisaillé pour fournir 3 segments.

La segmentation obtenue est représentée à la figure 10.12.

10.5 Comparaison de deux segmentations

Dans cette section, on aborde la question de la comparaison de segmentations obtenues par deux algorithmes différents.

On suppose disposer de deux segmentations S_1 et S_2 du jeu de données. S_1 partitionne le jeu de données en k_1 groupes notés $G_{1,i}$ et S_2 le partitionne en k_2 groupes notés $G_{2,j}$.

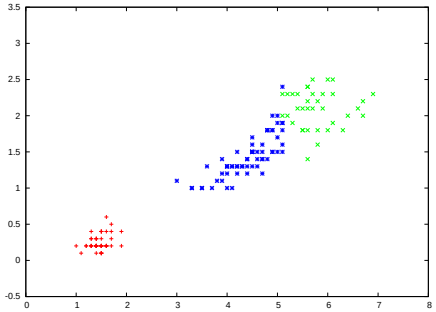


FIG. 10.12 – On a représenté la segmentation en 3 segments construite par l'algorithme de segmentation hiérarchique. Les couleurs indiquent les groupes construits par l'algorithme. Les données sont représentées dans le plan (longueur des pétales, largeur des pétales).

10.5.1 Analyse de tableau de contingence

On note $N_{i,j}$ le nombre de données qui appartiennent à $G_{1,i}$ et à $G_{2,j}$, soit $N_{i,j} = |G_{1,i} \cap G_{2,j}|$. $N_{i,j}$ mesure donc l'intersection entre deux groupes de deux segmentations différentes, ou encore leur « association ».

Cet ensemble de nombres $N_{i,j}$ remplit une matrice dénommée « tableau de contingence ».

Notons bien que l'on ne s'intéresse qu'aux effectifs de ces intersections, pas à leur composition.

Il y a alors deux approches pour étudier ce tableau : mesurer la significativité statistique de l'intersection entre deux groupes ; caractériser la force de cette association quand elle est significative.

Significativité d'une association

La significativité statistique d'une association est mesurée à l'aide de la statistique du χ^2.

...

Force d'une association significative

On ne s'intéresse ici qu'à des associations mesurées comme significatives par la méthode précédente.

Ayant établi qu'une association est significative, la valeur du χ^2 n'est pas un bon indicateur de sa force. La notion d'entropie permet de quantifier cette force en terme de mesure d'information : quelle quantité d'information est-elle gagnée quand j'apprends qu'un point du groupe $G_{1,i}$ appartient au groupe $G_{2,j}$?

...

10.5.2 Autre approche

De nombreux indicateurs se basent sur le décompte des paires de points qui font, ou ne font pas, partie des mêmes groupes dans les deux segmentations. (voir Meilă [2002]).

...

10.6 Critique

À compléter

Les méthodes de segmentation sont très nombreuses ; les travaux de recherche sur le sujet sont actuellement très nombreux. On consultera avec intérêt Candillier [2004] et Berkhin [2002] pour un tour plus complet sur la question, sans prétendre à l'exhaustivité.

10.7 Logiciels libres

- `weka` contient des algorithmes de segmentation : k-moyennes, EM en particulier : `http://www.cs.waikato.ac.nz/~ml`
- `autoclass` : `http://ic.arc.nasa.gov/ic/projects/bayes-group/autoclass/` ;
- une applet Java illustrant EM : `http://www.neurosci.aist.go.jp/~akaho/MixtureEM.html`
- `EMMIX` : `http://www.maths.uq.edu.au/~gjm/emmix/emmix.html` ;
- `crossbow` : logiciel accompagnant `rainbow` dédié à la segmentation de textes `http://www.cs.cmu.edu/~mccallum/bow` ;
- `cluster` : construit une segmentation hiérarchique. Je l'ai trouvé sur la Toile un jour et je ne sais plus d'où il vient. Il est accessible via mes pages à `http://www.grappa.univ-lille3.fr/~ppreux/Documents/cluster-2.9.tgz`.

10.8 Exercices

Exercice 24 Question 1 : Quel est le résultat de l'application de l'algorithme des centres mobiles au jeu de données représenté à la fig. fig :3anneauxConcentriques ?

Question 2 : Ce résultat vous paraît-il satisfaisant ?

Question 3 : Si non, imaginez une représentation des données (simple) qui ferait que l'application de l'algorithme des centres mobiles à ce jeu de données produirait les 3 groupes attendus.

Chapitre 11

Méthodes de projection

Contenu

À la suite du chapitre sur les méthodes de segmentation, on s'intéresse maintenant à des méthodes qui effectuent une projection des données depuis leur espace d'origine dans un espace généralement planaire. L'objectif est de visualiser ensuite cette projection afin de mieux comprendre un jeu de données, éventuellement d'en extraire une segmentation. Ici, il n'y a pas de construction automatique explicite des segments par les algorithmes qui vont être présentés ; cette projection fournit une aide à l'humain pour la compréhension de ses données.

Nous traitons de trois algorithmes : analyse en composantes principales, mise à l'échelle multi-dimensionnelle et réseau auto-organisé de Kohonen.

Concernant le vocabulaire, on parle aussi de méthode de réduction de dimensionnalité : l'idée est que les données peuvent constituer un objet géométrique d'une certaine dimension plongé dans un espace de plus grande dimension (P dimensions en suivant notre notation)[1].

11.1 Analyse en composantes principales

L'analyse en composantes principales (ACP) est une méthode d'analyse de données ancienne et très utilisée, très connue en statistique et dans les sciences expérimentales et, malheureusement, à peu près inconnue des étudiants en informatique !

On suppose disposer d'un ensemble $\mathcal{X}$ de N données, chacune décrites par P attributs.

Si l'on considère les données comme des points dans un espace euclidien à P dimensions, l'objectif de l'ACP est de construire l'espace euclidien à P dimensions le plus caractéristique et le plus économique pour représenter ces points. L'objectif est donc de passer de l'espace des données à un espace de caractéristiques (*feature space*) ou espace factoriel.

Remarque

On peut présenter l'idée autrement : supposons que l'on doive transmettre les données mais que pour chacune, on ne soit autorisé à transmettre que $K < P$ nombres. Comment faire pour transmettre les données au mieux ? Ne transmettre que les K premiers attributs de chaque donnée n'est certainement pas une bonne idée. La bonne idée est de transformer l'espace des données initial en un espace de telle manière que chaque dimension soit caractérisée par la proportion de l'information qu'elle contient. En respectant cet ordre, en transmettant les K premières coordonnées des données dans cet espace, la perte d'information est garantie minimale[2].

On va donc transformer l'espace initial (des données) dans un espace à P dimensions de manière à ce que la variance observée dans chacune de ces nouvelles dimensions soit décroissante.

Dans le cadre de l'ACP, la transformation de l'espace est linéaire : chacune des dimensions de l'espace des caractéristiques est une combinaison linéaire des dimensions de l'espace de données. Plus récemment sont apparues de nouvelles techniques construisant l'espace des caractéristiques par des transformations

[1] en anglais, on parle aussi d'*embedding* et de *manifold embedding* en informatique, mais aussi de *ordination* en écologie.

[2] Voir la transformation de Karhunen-Loève en théorie de la communication laquelle est équivalente à l'ACP.

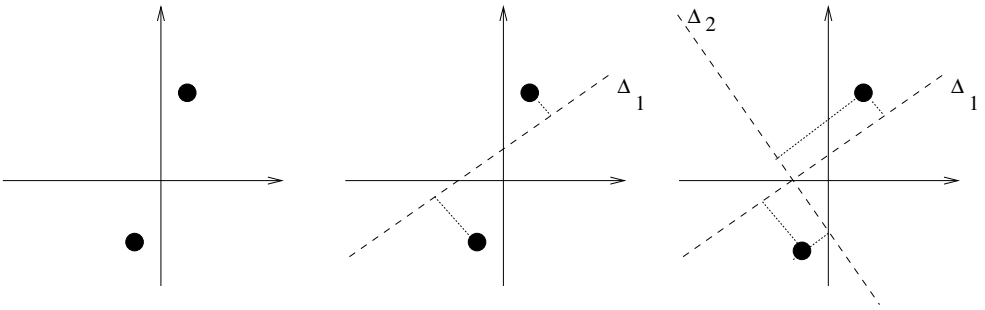


FIG. 11.1 – À gauche, on indique deux points dans le plan. Au milieu, on a construit la droite Δ_1 qui passe par l'origine et minimise la somme des distances au carré entre chaque point et elle (pointillés fins). À droite, on a construit Δ_2, orthgonale à Δ_1 et qui minimise également la somme des distances au carré à chaque point. Δ_1 et Δ_2 passent par l'origine.

non linéaires : analyse en composante curviligne, les réseaux de neurones non supervisés, l'« ACP noyau » (*cf.* chapitre 13).

11.1.1 L'Analyse en Composantes Principales

S'il n'y a que deux caractères a_j et a_k, on trace le graphe donnant a_j en fonction de a_k et l'on voit immédiatement s'il y a corrélation linéaire ou non. S'il y a plus de deux caractères, cette approche géométrique ne fonctionne plus, tout simplement parce nous (humains) sommes incapables de comprendre un espace à plus de 3 dimensions. Comme on s'intéresse typiquement à plus de 10 caractères, nos capacités sont mises en défaut.

L'idée fondamentale de l'ACP est la suivante : considérant le nuage de N points en P dimensions (dans cet espace, 1 point = 1 individu), on cherche à trouver le plan (donc, une représentation bi-dimensionnelle que l'on va pouvoir voir et comprendre) dans lequel la projection des points du nuage est la moins déformée possible, donc la plus fiable possible. Pour cela, on commence par rechercher la droite Δ_1 qui minimise la somme des distances au carré entre chaque point et la droite (*cf.* fig. 11.1). Cette droite a la propriété d'être la direction selon laquelle la variance (l'étalement) des points est la plus grande. Puis, on cherche une deuxième droite Δ_2 perpendiculaire à Δ_1 qui possède la même propriété ; et on continue jusqu'à obtenir P droites qui forment un repère orthogonal. C'est donc juste un problème de géométrie. Notons que comme toute méthode statistique, le nombre d'individus et le nombre d'attributs doivent être relativement importants pour que les conclusions de l'ACP aient un sens ; ainsi, il est bon que $N > 15$ et $P > 4$. Enfin, l'ACP s'applique à des données quantitatives.

Dans la suite, en guise d'exemple, on utilisera les données de la table 11.1.

TAB. 11.1 – Jeu de données utilisé pour illustrer l'ACP.

Individu	Poids	Taille	Âge	Note
1	45	1.5	13	14
2	50	1.6	13	16
3	50	1.65	13	15
4	60	1.75	15	9
5	60	1.7	14	10
6	60	1.7	14	7
7	70	1.6	14	8
8	65	1.6	13	13
9	60	1.55	15	17
10	65	1.7	14	11

Il décrit 10 élèves par certaines caractéristiques physiques et leur note moyenne annuelle[3].

Distance entre les individus

Abordant un problème de géométrie, il nous faut commencer par définir une notion de distance entre deux individus. S'inspirant de la géométrie euclidienne, on pourrait écrire :

$$d^2(x_i, x_j) = \sum_{k=1}^{k=P} (x_{i,k} - x_{j,k})^2$$

De l'importance de centrer et réduire les attributs

Considérons les individus x_4, x_5 et x_6. On a : $d^2(x_4, x_5) = 2.0$, $d^2(x_4, x_6) = 5.0$, $d^2(x_5, x_6) = 9.0$. Si l'on exprime maintenant les tailles en centimètres, les distances deviennent : $d^2(x_4, x_5) = 27$, $d^2(x_4, x_6) = 30$, $d^2(x_5, x_6) = 9$. On constate que selon l'unité utilisée, l'individu 6 est plus proche du 4$^{\text{e}}$ ou du 5$^{\text{e}}$!

Pour éviter ce problème de dépendance par rapport aux unités utilisées, il suffit d'utiliser des variables centrées réduites. Dans ce cas, on obtient $d^2(x_4, x_5) = 2.3$, $d^2(x_4, x_6) = 2.6$, $d^2(x_5, x_6) = 0.8$ quelles que soient les unités utilisées. Dans la suite, on considère que les données sont centrées et réduites.

Notons que la définition de distance donnée ci-dessus nécessite et signifie que tous les attributs ont la même importance dans la mesure de distance. On peut vouloir pondérer les attributs pour qu'ils aient plus ou moins d'importance avec une formule du type :

$$d^2(x_i, x_j) = \sum_{k=1}^{k=P} w_k (x_{i,k} - x_{j,k})^2$$

[3] cet exemple est totalement hypothétique.

TAB. 11.2 – Caractéristiques du centre de gravité pour le jeu de données de la table 11.1.

	Poids	Taille	Âge	Note
Moyenne (g)	58,5	1,64	13,8	12,0
Écart-type	7,84	0,08	0,79	3,50

TAB. 11.3 – Jeu de données de la table 11.1 centrées réduites.

Individu	Poids	Taille	Âge	Note
1	-1.72	-1.72	-1.01	0.57
2	-1.09	-0.45	-1.01	1.14
3	-1.09	0.19	-1.01	0.86
4	0.19	1.47	1.52	-0.86
5	0.19	0.83	0.25	-0.57
6	0.19	0.83	0.25	-1.43
7	1.47	-0.45	0.25	-1.14
8	0.83	-0.45	-1.01	0.29
9	0.19	-1.09	1.52	1.43
10	0.83	0.83	0.25	-0.29

où des poids $w_k \in \mathbb{R}$ pondère chaque composante de la distance pour chacun des attributs. Si tous les attributs ont la même importance, on prend des w_k tous égaux.

On suppose ici que tous les attributs ont la même importance et que $w_k = 1, \forall k$.

Remarque

Le lecteur curieux pourra adapter la suite décrivant l'ACP au cas général où les poids ne sont pas tous égaux.

On utilise la notion de « centre de gravité » introduite plus haut (*cf.* chap. 10, sec. 10.2).

On détermine les coordonnées centrées et réduites de chaque individu dans un nouveau repère dont l'origine est le centre de gravité et dont les axes sont parallèles aux axes originaux (celui des données non centrées réduites) : *cf.* table 11.3.

Présentation informelle des ACP

Quand on projette un ensemble de points dans un espace ayant moins de dimensions, la distance entre les points ne peut que diminuer (et en général, elle diminue effectivement). Pour trouver le meilleur plan de projection, on cherche

le plan dans lequel la distance entre les points projetés demeurent, en moyenne, maximale. Ce plan est qualifié de « principal ». Plus généralement, on définit P droites orthogonales les unes aux autres qui permettent de définir un repère orthonormé. Ces P droites sont les P « axes principaux » d'un repère dans lequel sont situés les individus de manière à les décrire de la façon la plus concise : l'objectif est que les coordonnées d'un individu (en moyenne) soient la plupart presque nulle et que seules quelques coordonnées aient une valeur importante. Les coordonnées d'un individu dans ce nouveau repère s'expriment par rapport à de nouveaux caractères appelés ses « composantes principales ». Notons $z_{j\in\{1,...P\}}$ les composantes principales, $z_{i,j}$ dénotant le j^{e} caractère principal de l'individu i. On a naturellement une relation reliant les attributs originaux aux attributs synthétiques : $z_j = u_{j,1}a_1 + u_{j,2}a_2 + ... + u_{j,P}a_P$. Les coefficients $(u_{j,1}, u_{j,2}, ..., u_{j,P})$ forment le j^{e} « facteur principal ». La meilleure représentation des données au moyen de $q < P$ caractères s'obtient en ne prenant en compte que les q premières composantes principales.

L'ACP est une méthode factorielle car elle construit de nouveaux caractères par combinaison des caractères initiaux. C'est une méthode linéaire car cette combinaison est linéaire.

Reste à savoir construire ces nouveaux caractères...

Méthode ACP

Dans tout ce qui suit, on suppose que l'on a centré et réduit les données, soit $\overline{a_j} = 0$ $\sigma(a_j) = 1, \forall j \in \{1, ...P\}$. Cette hypothèse simplifie les calculs et les notations.

Les N individus décrits par leur P caractères peuvent être mis sous forme d'une matrice ayant N lignes et P colonnes : 1 ligne décrit un individu et chaque colonne correspond à un attribut. Notons cette matrice $\mathbf{X}$.

La matrice de corrélation On définit la matrice de variance $\mathbf{V}$ comme suit :

$$\mathbf{V} = \begin{pmatrix} \text{var}(a_1) & \text{covar}(a_1, a_2) & \text{covar}(a_1, a_3) & ... & \text{covar}(a_1, a_P) \\ \text{covar}(a_2, a_1) & \text{var}(a_2) & \text{covar}(a_2, a_3) & ... & \text{covar}(a_2, a_P) \\ \text{covar}(a_3, a_1) & \text{covar}(a_3, a_2) & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \text{covar}(a_P, a_1) & \text{covar}(a_P, a_2) & \vdots & \vdots & \text{var}(a_P) \end{pmatrix} \tag{11.1}$$

et la matrice de corrélation $\mathbf{R}$:

$$\mathbf{R} = \begin{pmatrix} 1 & r(a_1,a_2) & r(a_1,a_3) & ... & r(a_1,a_P) \\ & 1 & r(a_2,a_3) & ... & r(a_2,a_P) \\ & & \ddots & & \vdots \\ & & & & 1 \end{pmatrix} \tag{11.2}$$

qui est symétrique puisque $r(a_j, a_k) = r(a_k, a_j)$ (le triangle inférieur gauche n'a pas été indiqué).

Pour des caractères centrés et réduits, ces deux matrices sont égales : $\mathbf{R} = \mathbf{V}$.

On a la relation : $\mathbf{R} = \frac{1}{P}\mathbf{X}^T\mathbf{X}$ où la notation $\mathbf{X}^T$ dénote la transposée de la matrice $\mathbf{X}$.

L'examen de la matrice de corrélation est intéressante : elle permet de repérer immédiatement les caractères fortement corrélés et les caractères qui ne sont pas du tout corrélés.

Analyse de la matrice de corrélation Le calcul des valeurs propres et des vecteurs propres de $\mathbf{R}$ fournit alors toutes les informations recherchées. Notons $\lambda_{i\in\{1,...P\}}$ les P valeurs propres de $\mathbf{R}$ ordonnées de manière décroissante : $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_P$, et leurs P vecteurs propres associés $\overrightarrow{V}_{i\in\{1,...P\}}$[4]. $\mathbf{R}$ étant par nature symétrique et définie positive[5], ses valeurs propres sont réelles et positives et ses vecteurs propres ont des coordonnées réelles. S'il existe une, ou des, valeurs propres nulles, cela signifie que les attributs ne sont pas linéairement indépendants les uns des autres : un, ou plusieurs, attributs sont obtenus par combinaison linéaire des autres. On suppose dans la suite que toutes les valeurs propres sont non nulles.

Ces vecteurs propres sont unitaires et orthogonaux deux à deux. Ils forment donc une base orthonormée. Ce sont les axes principaux recherchés ; de plus, l'axe principal est celui associé à la valeur propre la plus grande, ... ; donc, les q axes principaux sont les droites dont les vecteurs unitaires sont les vecteurs propres associés aux q valeurs propres les plus grandes et passant par g.

On obtient alors les coordonnées principales des individus en les projetant dans cet espace, c'est-à-dire en faisant le produit scalaire des coordonnées (centrées réduites) d'un individu par chacun des P vecteurs propres.

En prenant $q = 2$, on peut représenter les N individus dans le plan principal. La répartition des individus dans le plan principal doit être réalisée et étudier. La présence de formes particulières régulières (droite, courbe, regroupement par paquets, ...) est une information digne d'intérêt. Le problème est de savoir si la

[4] Le calcul des vecteurs propres et des valeurs propres est supposé connu : cf. cours de DEUG. On rappelle que si λ est une valeur propre et $\overrightarrow{V}$ son vecteur propre associé, on a $\mathbf{R}\ \overrightarrow{V} = \lambda\ \overrightarrow{V}$. Des logiciels libres calculent valeurs et vecteurs propres (voir par exemple le langage R)

[5] $\overrightarrow{v}\mathbf{R}\overrightarrow{v} > 0, \forall \overrightarrow{v}$

représentation dans le plan principal est fidèle au nuage de points initial. Nous abordons ce point ci-dessous.

Remarque

Quelques mots d'explication sur ce calcul de vecteurs et valeurs propres de la matrice de corrélation.

Que cherche-t-on ? Les directions de plus grand allongement du nuage de points, ces directions devant être orthogonales pour obtenir une base orthogonale.

Comment détermine-t-on l'allongement selon une certaine direction $\overrightarrow{v}$ du nuage de points ? Par la variance selon cette direction qui s'obtient simplement par $\overrightarrow{v}^T \mathbf{R} \overrightarrow{v}$.

Comment obtient-on la direction de plus grand allongement ? On cherche donc le vecteur $\overrightarrow{v}$ qui maximise cet allongement, *i.e.* le vecteur qui annule la dérivée de cette allongement.

Soit donc, on cherche $\overrightarrow{v}$ un vecteur de norme 1 qui annule $\frac{d\overrightarrow{v}^T \mathbf{R} \overrightarrow{v}}{d\overrightarrow{v}}$. Le calcul direct de cette dérivée ne donnant rien d'intéressant, le truc consiste à calculer la dérivée de cette même quantité divisée par 1 ; la norme de $\overrightarrow{v}$ vaut 1 (par hypothèse) ; cette norme vaut $\overrightarrow{v}^T \overrightarrow{v}$. Donc, on s'intéresse au $\overrightarrow{v}$ qui annule $\frac{\frac{d\overrightarrow{v}^T \mathbf{R} \overrightarrow{v}}{d\overrightarrow{v}}}{\overrightarrow{v}^T \overrightarrow{v}}$. Pour que cette dérivée soit nulle, on obtient la relation $\mathbf{R}\overrightarrow{v} = (\overrightarrow{v}^T \mathbf{R} \overrightarrow{v})\overrightarrow{v} = \lambda \overrightarrow{v}$. Donc, $\overrightarrow{v}$ doit être un vecteur propre de $\mathbf{R}$ associé à la valeur propre λ. D'autre part, puisque l'on cherche la direction associée à la plus grande variance (soit $\overrightarrow{v}^T \mathbf{R} \overrightarrow{v}$ maximal), on doit s'intéresser au vecteur propre associé à la plus grande valeur propre. Plus généralement, l'ensemble des vecteurs propres ordonnés selon leurs valeurs propres associées décroissantes fournira l'ensemble des directions correspondant aux allongements dans l'ordre décroissant du nuage de points. La matrice $\mathbf{R}$ étant symétrique et définie positive, on sait qu'elle ne possède que des valeurs propres réelles et positives ou nulles. De plus, on sait que ces vecteurs propres forment une base orthonormée.

Avant cela, l'ordre de grandeur des valeurs propres les unes par rapport aux autres indique leur importance. Si quelques valeurs propres ont des valeurs bien plus importantes que les autres, cela signifie que l'essentiel des informations est donné par ces axes principaux et que les autres axes donnent peu d'information. Au contraire, si toutes les valeurs propres ont sensiblement la même valeur, alors aucun axe principal ne peut être laissé de côté. Voulant extraire de l'information, la situation idéale est, bien entendu, lorsque 1 ou 2 valeurs propres sont très importantes par rapport aux autres. Remarquant que $\sum_{i \in \{1,\dots P\}} \lambda_i = P$, l'importance d'une valeur propre par rapport aux autres est bien mesurée par son inertie :

$$\mathcal{I}_i = \frac{\lambda_i}{P} \tag{11.3}$$

λ_i est la variance le long de l'axe principal porté par le vecteur propre $\overrightarrow{V_i}$, donc la variance de la composante principale z_i.

Notons que l'on a (théorème spectral) :

$$\mathbf{R} = \sum_{i=1}^{i=P} \lambda_i \overrightarrow{V_i} \overrightarrow{V_i}^T$$

Remarquons aussi que la matrice de covariance des données dans l'espace principal est diagonale : ainsi, par une ACP, on obtient une représentation dans un espace de données dans lequel les composantes sont non corrélées linéairement. C'est une propriété très intéressante : d'une manière générale, certains des algorithmes d'apprentissage que nous verrons par la suite fonctionnent mieux si les données ont leurs composantes non corrélées entre-elles. Aussi, une ACP peut-elle constituer un prétraitement intéressant.

Mesure de la déformation du nuage due à la projection dans l'espace principal

Donc, sachant que la projection dans le plan principal déforme la réalité en raccourcissant les distances (la répartition des individus dans l'espace de départ), il faut pouvoir estimer cette déformation. Pour cela, l'habitude veut que l'on calcule le cosinus de l'angle formé par la demi-droite allant de l'origine au point initial et la demi-droite allant de l'origine à la projection du point dans l'espace principal, soit un axe, soit un plan. On notera $\theta_{i\in\{1,...N\},j\in\{1,...P\}}$ l'angle entre la demi-droite passant par l'origine et le point initial x_i et l'axe principal j ; on notera $\theta_{i\in\{1,...N\},j\in\{1,...P\}\times k\in\{1,...P\}}$ l'angle entre la demi-droite passant par l'origine et le plan formé par les axes principaux j et k. Le cosinus carré de ces angles est :

$$\cos^2\theta_{i\in\{1,...N\},j\in\{1,...P\}} = \frac{z_{i,j}^2}{\sum_{j\in\{1,...P\}} x_{i,j}^2} \tag{11.4}$$

$$\cos^2\theta_{i\in\{1,...N\},j\in\{1,...P\}\times k\in\{1,...P\}} = \frac{z_{i,j}^2 + z_{i,k}^2}{\sum_{l\in\{1,...P\}} x_{i,l}^2} \tag{11.5}$$

où $z_{i,j}$ et $z_{i,k}$ sont les coordonnées sur les axes principaux j et k de l'individu x_i. Un cosinus carré inférieur à 0.3 indique une déformation importante ($\theta > 57^o$).

Exemple trivial

On prend volontairement un exemple trivial pour comprendre un peu mieux les choses. Plaçons-nous dans un espace 2D (donc $P = 2$) et considérons les $N = 3$ individus suivants : $(1, 1)$, $(2, 2.2)$, $(2.9, 3.1)$. Immédiatement, nous voyons que ces trois points sont presque disposés sur une droite à 45 degrés. Leur ACP va le confirmer.

On a : $\overline{a_1} = 1.97$, $\overline{a_2} = 2.1$, $\sigma(a_1) = 0.95$, $\sigma(a_2) = 1.05$, d'où les coordonnées centrés réduites des individus :

1	-1.017	-1.044
2	0.035	0.095
3	0.982	0.949

La matrice de corrélation vaut :

$$\mathbf{R} = \begin{pmatrix} 1 & 0.999 \\ 0.999 & 1 \end{pmatrix}$$

dont les valeurs propres sont $\lambda_1 = 1.999$ et $\lambda_2 = 0.001$. La somme des valeurs propres est bien égale à $P = 2$. L'inertie des caractères principaux est $\mathcal{I}_1 > 0.999$ et $\mathcal{I}_2 < 0.001$. Les vecteurs propres sont $\overrightarrow{V_1} = (0.707, 0.707)$ et $\overrightarrow{V_2} = (-0.707, 0.707)$ (rappelons que $0.707 \approx \frac{\sqrt{2}}{2} = \sin(\pi/4) = \cos(\pi/4)$).

Tout cela signifie que les individus ont leurs caractères expliqués à plus de 99% par leur caractère principal qui est représenté par un axe orienté à 45 degrés : c'est normal puisque les individus sont presqu'alignés sur cette droite. L'ACP est bien cohérente avec ce que l'on avait vu.

On peut aussi calculer les coordonnées des individus dans leur nouveau repère : $z_{1,1} = -1.73, z_{1,2} = -0.08$, $z_{2,1} = 0.11, z_{2,2} = 0.06$, $z_{3,1} = 1.61, z_{3,2} = 0.03$.

On peut estimer la déformation : pour le premier individu,

$$\cos^2 \theta_{1,1\times 2} = \frac{1.73^2 + 0.08^2}{1.24^2 + 1.28^2} \approx 1.0$$

donc, $\theta_{1,1\times 2} = 13.6^o$, c'est-à-dire que la déformation est nulle pour le premier individu : c'est normal puisqu'il est situé sur la droite à 45 degrés.

Pour les autres caractères, les déformations sont également très faibles : $\theta_{2,1\times 2} \approx 7.9^o$ et $\theta_{3,1\times 2} \approx 15^o$.

Le cercle de corrélation On se propose d'étudier la corrélation entre les caractères initiaux et les caractères principaux.

Le coefficient de corrélation linéaire entre le j$^{\text{e}}$ caractère initial a_j et le k$^{\text{e}}$ caractère principal z_k est :

$$r_{j,k} = \overrightarrow{V_k}[j] \times \sqrt{\lambda_k} \tag{11.6}$$

où $\overrightarrow{V_k}[j]$ est la j$^{\text{e}}$ composante du vecteur $\overrightarrow{V_k}$.

On peut alors repérer s'il existe des caractères principaux fortement corrélés avec les caractères initiaux.

Remarque : $\sum_{k\in\{1,\dots P\}} r_{j,k}^2 = 1$.

On peut alors tracer les P points de coordonnées $(r_{j,1}, r_{j,2})_{j\in\{1,\dots P\}}$ dans le cercle unité. Si les points sont proches de la circonférence (ils sont alors significatifs), on peut éventuellement repérer des regroupements ou des oppositions de caractères.

En reprenant l'exemple trivial, on obtient les corrélations entre caractères initiaux et principaux comme suit :

	z_1	z_2
a_1	$r_{1,1} = 0.9996$	$r_{1,2} = -0.022$
a_2	$r_{2,1} = 0.9996$	$r_{2,2} = 0.022$

ce qui s'explique très bien puisque la première composante représente presque parfaitement les données (donc corrélation $r_{1,1} \approx 1$) et que a_1 et a_2 sont fortement corrélés (donc corrélation $r_{2,1} \approx 1$). Le tracé des points de coordonnées $(r_{2,1}, r_{2,2})$ et $(r_{1,1}, r_{1,2})$ les montrent situés sur le cercle de corrélation. Le premier caractère principal correspond à l'axe des abscisses tandis que le second correspond à l'axe des ordonnées.

Les autres axes principaux Nous n'avons parlé jusqu'ici que de l'utilisation des deux premiers axes principaux. Les autres axes peuvent également avoir leur utilité. Pour en juger, il faut comparer les différentes valeurs propres (cf. ci-dessus). Pour chacune, on peut mesurer leur inertie $\mathcal{I}_i$ et leur inertie cumulée :

$$\mathcal{C}_i = \sum_{j \leq i} \mathcal{I}_j \tag{11.7}$$

L'inertie indique la proportion de la distance moyenne conservée lorsque l'on passe des caractères initiaux aux caractères principaux. Si $\mathcal{C}_2 \approx 100$, cela signifie que les deux premiers caractères principaux ne déforment presque pas le nuage de points, donc, que ces deux caractères principaux décrivent presque complétement les individus initiaux. Au contraire, si toutes les inerties les plus importantes (ou un grand nombre) sont égales, cela signifie qu'il faut considérer non seulement les deux caractères principaux, mais aussi tous les autre caractères de même importance, donc étudier ces différents plans.

Quels axes principaux faut-il retenir ? Cette question est légitime. On présente deux critères :

- critère de Kaiser : on ne retient que les axes pour lesquels la valeur propre associée est > 1 ;
- $Scree - test$ de Cottell : on calcule les différences premières entre valeurs propres $\epsilon_i = \lambda_i - \lambda_{i+1}$ puis les différences secondes $\delta_i = \epsilon_i - \epsilon_{i+1}$. On retient les λ_i tant que $\delta_i > 0$.

Application

On reprend l'exemple donné plus haut avec des élèves et leur note.

$$\mathbf{R} = \begin{pmatrix} & \text{Poids} & \text{Taille} & \hat{\text{A}}\text{ge} & \text{Note} \\ \text{Poids} & 1 & 0.367 & 0.485 & -0.568 \\ \text{Taille} & 0.367 & 1 & 0.396 & -0.629 \\ \hat{\text{A}}\text{ge} & 0.485 & 0.396 & 1 & -0.322 \\ \text{Note} & -0.568 & -0.629 & -0.322 & 1 \end{pmatrix}$$

Les valeurs propres et les vecteurs propres sont :

$\lambda_1 = 2.391 \quad \overrightarrow{V_1} = (0.5080, 0.5038, 0.4453, -0.5383)$
$\lambda_2 = 0.750 \quad \overrightarrow{V_2} = (0.3065, -0.4641, 0.7058, 0.4381)$
$\lambda_3 = 0.584 \quad \overrightarrow{V_3} = (-0.6593, 0.5253, 0.4712, 0.2593)$
$\lambda_4 = 0.274 \quad \overrightarrow{V_4} = (-0.4619, -0.5042, 0.2855, -0.6715)$

On constate qu'aucune valeur propre n'est nulle ; on peut vérifier que les 4 vecteurs propres sont unitaires et orthogonaux deux à deux.

On peut calculer l'inertie et l'inertie cumulée de chacune des valeurs propres :

λ	$\mathcal{I}$	$\mathcal{C}$
2.391	0.59775	0.59775
0.750	0.1875	0.78525
0.584	0.146	0.93125
0.274	0.0685	1.0

Les coordonnées des individus dans le nouveau repère se calculent comme suit : par exemple, pour le premier individu, ses coordonnées sont :

$$\begin{pmatrix} (x_{1,1}x_{1,2}x_{1,3}x_{1,4})\ \overrightarrow{V_1} \\ (x_{1,1}x_{1,2}x_{1,3}x_{1,4})\ \overrightarrow{V_2} \\ (x_{1,1}x_{1,2}x_{1,3}x_{1,4})\ \overrightarrow{V_3} \\ (x_{1,1}x_{1,2}x_{1,3}x_{1,4})\ \overrightarrow{V_4} \end{pmatrix}$$

soit $(-2.64, -0.20, -0.10, 1.04)$. Pour les N individus, leurs coordonnées sur les axes principaux sont :

Individu	Coordonnées (z_1, z_2, z_3, z_4)
1	$(-2.64, -0.20, -0.10, 1.04)$
2	$(-1.94, -0.36, 0.32, -0.35)$
3	$(-1.44, -0.80, 0.59, -0.49)$
4	$(2.08, 0.08, 1.20, 0.19)$
5	$(0.99, -0.42, 0.30, -0.05)$
6	$(1.47, -0.82, 0.06, 0.56)$
7	$(1.32, 0.35, -1.45, 0.41)$
8	$(-0.43, -0.14, -1.25, -0.67)$
9	$(-0.57, 2.39, 0.41, -0.07)$
10	$(1.17, -0.08, -0.07, -0.57)$

Remarques : notons :

$$z_1^2 + z_2^2 + z_3^2 + z_4^2 = x_1^2 + x_2^2 + x_3^2 + x_4^2 = gy$$

On a alors :

$$z_1^2 = gy \times \mathcal{C}_1$$
$$z_1^2 + z_2^2 = gy \times \mathcal{C}_2$$
$$z_1^2 + z_2^2 + z_3^2 = gy \times \mathcal{C}_3$$

On peut représenter alors les individus dans le plan principal $z_1 \times z_2$:

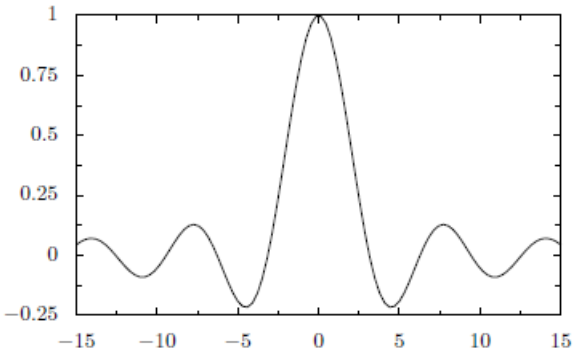
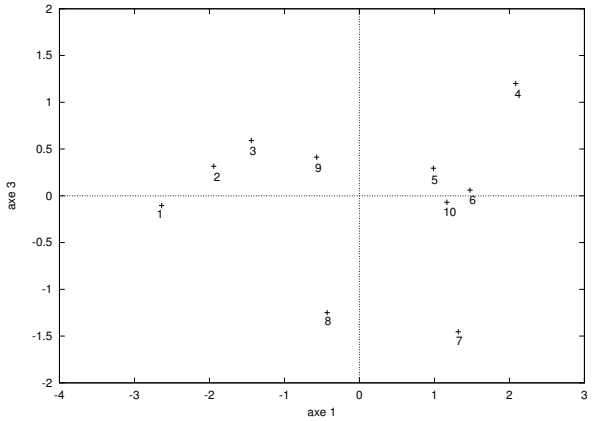


Figure 1: $y(x) = \frac{\sin(x)}{x}$

On repère deux regroupements, l'un de 3 individus, l'autre de 5 ; les deux autres individus sont isolés. Il faut avoir à l'esprit que la distance entre les points est plus fidélement conservée, entre l'espace initial et le plan principal, pour des points éloignés des axes.

Dans le plan $z_1 \times z_3$, on a la représentation suivante :

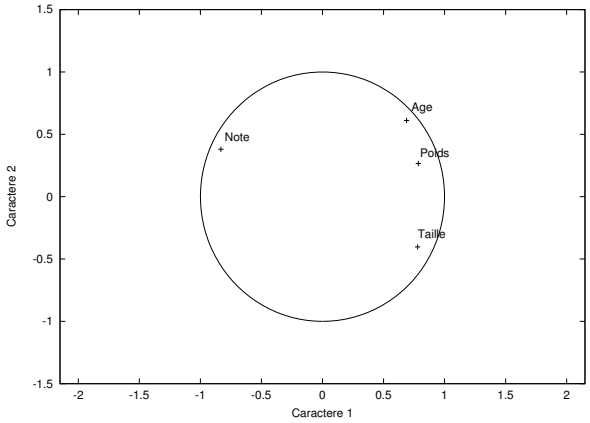


On peut aussi s'intéresser au plan $z_2 \times z_3$.

On peut calculer la corrélation entre caractères initiaux et caractères principaux :

	z_1	z_2	z_3	z_4
(Poids) a_1	0.785	0.266	-0.504	-0.242
(Taille) a_2	0.779	-0.403	0.402	-0.264
(Âge) a_3	0.689	0.611	0.360	0.150
(Note) a_4	-0.832	0.380	0.198	-0.352

À l'aide des deux premières colonnes, on peut tracer le cercle de corrélation :



On note que :

- les 4 caractères sont fortement corrélés (en valeur absolue) avec le premier axe principal ;
- le caractère Note est corrélé négativement avec les trois autres caractères ;
- le groupe 1, 2 et 3 comprend donc des élèves moins développés physiquement (plus jeunes, plus légers, plus petits) que la moyenne mais qui ont de bonnes notes ;
- le groupe 4, 5, 6, 7 et 10 regroupent des élèves plus développés que la moyenne mais qui ont une note faible ;
- 8 et 9 contredisent ces deux tendances générales ;
- en particulier, 9 a une très bonne note en dépit de ses caractéristiques physiques.

Pour estimer la déformation due à la projection dans le plan principal, on calcule le $\cos^2$:

Individu	$\cos^2\theta_{i,1}$	$\cos^2\theta_{i,2}$	$\cos^2\theta_{i,1\times2}$	$\cos^2\theta_{i,1\times3}$
1	0.859	0.005	0.82	0.82
2	0.916	0.031	0.95	0.94
3	0.628	0.195	0.83	0.74
4	0.745	0.001	0.75	0.99
5	0.785	0.142	0.88	0.79
6	0.690	0.212	0.87	0.67
7	0.419	0.030	0.45	0.93
8	0.084	0.008	0.09	0.79
9	0.053	0.919	0.97	0.08
10	0.804	0.004	0.80	0.80

L'individu 8 est donc très mal représenté dans le plan principal. Dans le plan 1×3, il est par contre bien représenté ; dans ce plan, c'est 9 qui est très mal représenté ($\cos^2 \approx 0.08$).

Application aux iris

Pour poursuivre l'illustration de l'ACP, on effectue une ACP sur le jeu de données « iris ».

La matrice de corrélation est :

$$\mathbf{R} = \begin{pmatrix} & \text{longueur sépales} & \text{largeur sépales} & \text{longueur pétales} & \text{largeur pétales} \\ \text{longueur sépales} & 1 & -0.109369 & 0.871754 & 0.817954 \\ \text{largeur sépales} & & 1 & -0.420516 & -0.356544 \\ \text{longueur pétales} & & & 1 & 0.962757 \end{pmatrix}$$

On constate des corrélations faibles pour les attributs concernant les sépales, mis à part entre la longueur des sépales et celle des pétales.

Les valeurs propres sont :

valeur propre	inertie
2.91082	73 %
0.921221	23 %
0.147353	4 %
0.0206077	1%

La somme des inerties ne fait pas 1, à cause des arrondis.

Très clairement, deux attributs principaux seront suffisants pour représenter les données puisqu'ils cumulent 96 % d'inertie. La figure 11.2 représente le jeu de données dans le plan principal.

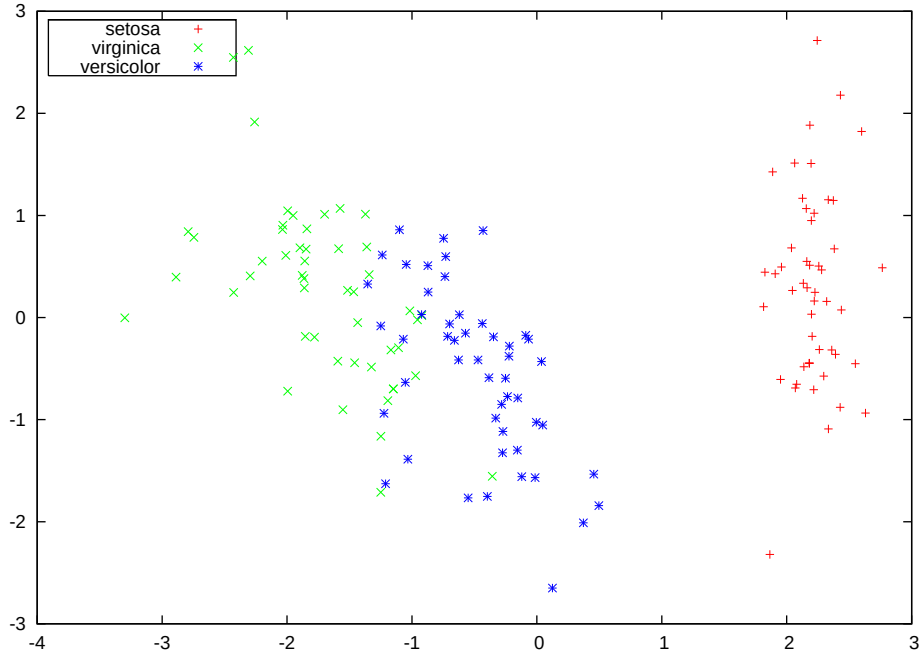


FIG. 11.2 – Projection dans le plan factoriel principal des iris. On a indiqué les 3 classes par des couleurs différentes.

Corrélation entre attributs et facteurs :

	z_1	z_2	z_3	z_4
(longueur sépales) a_1	-0.89	0.36	0.28	0.04
(largeur sépales) a_2	0.45	0.89	-0.09	-0.02
(longueur pétales) a_3	-0.99	0.02	-0.05	-0.12
(largeur pétales) a_4	-0.96	0.06	-0.24	0.08

On voit que les corrélations sont très fortes entre les deux premiers facteurs z_1 et z_2 et les 4 attributs. On voit même que z_1 est très fortement corrélé avec la longueur et la largeur des pétales.

11.1.2 La (grande) famille des ACP

L'ACP a été inventée par Hotelling dans les années 1930. Cette méthode a été complétée par d'autres qui en sont dérivées.

Analyse factorielle des correspondances

L'analyse factorielle des correspondances (AFC) est une ACP pour étudier des tableaux de contingence : on considère des individus décrits par deux caractères nominaux. On construit le tableau de contingence dans lequel les lignes représentent les valeurs prises par l'un des caractères, les colonnes représentent les valeurs prises par l'autre caractère. À l'intersection, on trouve le nombre d'individus pour lesquels on a conjonction de ces deux valeurs de caractères. Cette méthode a été proposée dans les années 1960 par Benzécri.

L'AFC a été étendue à plus de deux caractères, ce qui a donné l'analyse factorielle des correspondances multiples (AFCM). Cette méthode est très utilisée pour l'analyse d'enquêtes auprès de l'opinion (sondage).

Analyse factorielle discriminante

Dans ses objectifs, l'analyse factorielle discriminante (AFD) est proche des méthodes de classification : elle traite un jeu d'exemples (données classées) et permet de classer de nouvelles données. Pour cela, elle effectue une description statistique de chaque groupe d'exemples ; à partir de cette description, elle permet de déterminer la classe la plus probable d'une donnée.

11.1.3 ACP et textes : l'indexation par la sémantique latente

L'indexation par la sémantique latente (ISL) est l'ACP adaptée aux textes.

Comme précédemment, un texte est décrit par un vecteur dont chaque composante correspond à un mot d'un vocabulaire $\mathcal{V}$. L'ISL effectue une projection

des textes et des mots dans un espace factoriel. Plus haut, on a décrit l'ACP qui s'applique à un ensemble de données décrites par des caractères. Ici, on décrit l'ACP qui analyse un jeu de données constitué de deux types de données, les textes et les mots.

On part de l'hypothèse que la similarité entre deux vecteurs-documents implique une proximité sémantique des documents correspondants et que une forte similarité entre deux vecteurs-mots indique la synonymie des deux mots.

Principe de la méthode de décomposition

Soit la matrice $\mathbf{A}$ dont chaque colonne est un vecteur-document (donc autant de colonnes qu'il y a de documents : notons ce nombre N). Chaque ligne correspond à un mot du vocabulaire que l'on suppose de taille $P = |\mathcal{V}|$. Les composantes $a_{m,t}$ de la matrice $\mathbf{A}$ sont en relation avec le nombre d'occurences du mot m dans le texte t. En général, on les prend de la forme :

$$a_{m,t} = L(m,t) \times G(m)$$

où $L(m,t)$ est une pondération dite « locale », c'est-à-dire affectant uniquement le mot m du texte t, tandis que $G(m)$ est une pondération dite « globale », ce qui signifie que ce poids est le même pour tous les textes où apparaît le mot m.

Les pondérations courantes sont :

- pour $L(m,t)$:
  - fréquence d'occurence[6] du mot m dans le texte t : $FT(m,t) = \frac{n(m,t)}{|t|_1}$ où la notation $|t|_1$ indique le nombre de mots différents présents dans le texte t ;
  - le log de cette quantité :$\log(FT(m,t)+1)$? Cette quantité est précisèment ce que l'on définit comme la fréquence du terme au chap. 5 ;
- pour $G(m)$:
  - $G(m) = \frac{1}{\sqrt{\sum_j FT(m,j)^2}}$, ce qui permet de normer les lignes : tous les mots sont ainsi représentés par un vecteur de même norme ;
  - poids GfIdF : $G(m) = \frac{f(m)}{n(m)}$ où $f(m)$ est la fréquence d'occurences du mot m dans l'ensemble des textes et $n(m)$ est le nombre de textes dans lesquels apparaît le mot m ;
  - poids Idf : $G(m) = \log_2 N/n(m)$ que l'on a déjà rencontré au chap. 5 ;
  - entropie : $G(m) = 1 - \sum_t \frac{p_{m,t} \log p_{m,t}}{\log_2 N}$ où $p_{m,t} = \frac{FT(m,t)}{f(m)}$.

[6] attention : ne pas confondre avec la notion présentée au chap. 5. Ici, c'est simplement le nombre d'occurences du mot divisé par le nombre de mots apparaissant dans le texte.

On effectue ensuite l'ACP de la matrice $\mathbf{A}$. Rappelons que l'on a ici deux types de données mêlées dans cette matrice (des mots et des textes). On veut donc extraire des axes principaux pour ces deux types de données[7].

Pour cela, on effectue une décomposition en valeurs singulières de $\mathbf{A}$. $\mathbf{A}$ est donc une matrice de taille $M \times N$. La décomposition en valeurs singulières fournit les matrices U, Σ et V telles que :

$$\mathbf{A} = U\Sigma V^T$$

U est une matrice $M \times N$ alors que Σ et V sont des matrices $N \times N$.

On a (propriétés de la décomposition en valeurs singulières) :

- $U^T U = V^T V = I_N$ où I_N est la matrice identité $N \times N$;
- $\Sigma = diag(\sigma_1, ...\sigma_N)$ avec $\sigma_i > 0$ pour $1 \leq i \leq r$ et $\sigma_i = 0$ pour $i > r$. r est nommé le rang de la matrice $\mathbf{A}$. Si $r < N$, on dit que la matrice $\mathbf{A}$ est singulière ;
- les σ_i sont les racines carrées des valeurs propres de $\mathbf{A}^T\mathbf{A}$ et $\mathbf{A}\mathbf{A}^T$;
- les vecteurs propres de $\mathbf{A}\mathbf{A}^T$ constituent les colonnes de U et se nomment les vecteurs singuliers à gauche de $\mathbf{A}$;
- les vecteurs propres de $\mathbf{A}^T\mathbf{A}$ constituent les colonnes de V, lesquels sont dénommés les vecteurs singuliers à droite de $\mathbf{A}$.

Pour comprendre l'intérêt de ce qui précède et le lien avec une ACP, il est important de noter que :

1. $\mathbf{A}\mathbf{A}^T$ est une matrice dont chaque terme correspond au produit scalaire entre une ligne de $\mathbf{A}$ (un mot) et une ligne de $\mathbf{A}$ (un autre mot). Donc, chacun de ses termes représente le cosinus entre ces deux mots, si l'on a pris soit de normaliser les lignes ou est proportionnelle à ce cosinus sinon. Donc, chacun des éléments de cette matrice représente la similarité entre les deux mots, la similarité étant mesurée par la répartition de ces deux mots parmi les textes ;
2. $\mathbf{A}^T\mathbf{A}$ est une matrice dont chaque terme correspond au produit scalaire entre une colonne de $\mathbf{A}$ (un texte) et une colonne de $\mathbf{A}$ (un autre texte). Donc, chacun de ces termes mesure la similarité entre deux textes ;
3. donc, $\mathbf{A}\mathbf{A}^T$ et $\mathbf{A}^T\mathbf{A}$ sont des matrices de similarité. La décomposition en vecteurs propres de ces matrices est connue pour apporter des informations très intéressantes (*cf.* les méthodes de segmentation spectrale au chap. 10). Ces vecteurs propres sont dans les matrices U et V obtenues par décomposition en valeurs singulières. Quant à elle, les valeurs singulières (valeur absolue des valeurs propres) sont dans Σ.

[7] on pourrait ne faire qu'une simple ACP, mais on n'aurait alors des informations que sur les textes ou sur les mots, pas sur les deux.

On a vu dans la partie sur l'ACP que, généralement, les premières valeurs propres ont des valeurs importantes alors que les autres ont des valeurs négligeables. Nous retrouvons cette propriété ici.

Ainsi, en n'utilisant que les k premières valeurs singulières avec leur vecteurs propres à gauche et à droite respectifs, on obtient une approximation A_k de la matrice $\mathbf{A}$ de départ :

$$A_k = U_k \Sigma_k V_k^T$$

où U_k et V_k sont composés des k premières colonnes de U et V et $\Sigma_k = diag(\sigma_1, ..., \sigma_k)$. A_k est la meilleure approximation de rang k de $\mathbf{A}$ au sens des moindres carrés : $A_k = \arg\min_{B \in \text{matrices de rang } k} ||A_k - B||^2$.

Ainsi, d'une certaine manière, A_k capture les particularités les plus importantes de la matrice $\mathbf{A}$ initiale. Le fait de laisser de côté un certain nombre de dimensions permet de se débarasser (au moins en partie) des problèmes liés à la polysémie et à la synonymie.

Projection dans l'espace factoriel

Soit un texte t que l'on souhaite projeter dans l'espace factoriel. Ce texte est décrit à partir du vocabulaire $\mathcal{V}$. Sa projection est $\hat{t} = t^T U_k \Sigma_k^{-1}$.

Interprétation

L'exploitation est riche. On peut s'intéresser à différentes questions :

- pour un mot donné m, quels sont les mots « proches » *i.e.* dont l'apparition dans les textes est à peu près la même. Cela va permettre d'exhiber des synonymes ;
- pour un texte donné t, quels sont les textes proches ? Cela va permettre de trouver des textes traitant du même sujet ;
- pour un mot ou un ensemble de mots donnés, quels sont les textes pour lesquels ce(s) mot(s) sont les plus significatifs ? Cela va permettre une interrogation par mots-clés.

Recherche de synonymes

Soit un mot $m \in \mathcal{V}$ pour lequel on veut trouver les mots qui sont utilisés de la même manière. On constitue un texte qui ne contient qu'un seul mot, m et on projette ce texte dans l'espace factoriel.

Il ne reste plus qu'à chercher les mots de $\mathcal{V}$ dont la projection est proche par une approche « plus proches voisins ».

Recherche de textes identiques

Soit un texte t, qui fait partie de l'ensemble des textes initiaux ou pas. On décrit ce texte par sa composition en mots du vocabulaire $\mathcal{V}$ et on le projette. De même, on calcule la projection de chacun des N textes et on détermine les plus proches.

Interrogation par mots-clés

On suppose que les mots-clés font partie de $\mathcal{V}$. Dès lors, on constitue un texte t comprenant ces mots, on le projette et, à nouveau, on recherche les textes dont la projection est la plus proche.

Bien entendu, les composantes des pseudo-textes sont pondérées si nécessaire, comme l'ont été les composantes de $\mathbf{A}$.

Mots et leur traduction

On dispose d'un même ensemble de textes dans deux langues. L'ISL va projeter chaque mot d'une langue près de sa traduction dans l'autre langue, ainsi que chaque paire de textes l'un près de l'autre. Cela permet de trouver la traduction d'un terme et d'effectuer une interrogation dans une langue et d'obtenir le résultat dans l'autre langue.

11.1.4 Critique de l'ACP

+ méthode étudiée depuis des dizaines d'années ; donc, elle a été mise à l'épreuve maintes fois ; donc, il est essentiel de la connaître et de savoir l'utiliser ;
o c'est une méthode linéaire. Aussi, seules les corrélations linéaires sont-elles détectées et exploitables ;
- calculer les valeurs et vecteurs propres avec précision est techniquement relativement lourd, notamment si le nombre d'attributs est très élevé.

11.1.5 ACP non linéaire

Les ACP ont été étendues pour attaquer le cas où les relations entre les caractères sont non linéaires. Il y a au moins trois classes de méthodes :

– méthodes de projection non linéaire (par exemple, l'analyse en composante curviligne (*cf.* Demartines and Hérault [1997])) ;
– réseaux de neurones non supervisés avec lesquels il y a de très forts liens (*cf.* sec. 11.3) ;
– ACP noyau reposant sur les machines à vecteurs supports

11.2 La mise à l'échelle multi-dimensionnelle

La mise à l'échelle multi-dimensionnelle (MDS) est une méthode très classique de projection d'un ensemble de données dans un espace plus petit. Elle a été introduite en psychologie dans les années 1950.

La méthode repose sur une matrice de dissimilarité qui donne la distance entre tout couple de données (x_i, x_j) : $D_{i,j}$. On suppose que D est symétrique ($D_{i,j} = D_{j,i}$) et que $D_{i,i} = 0$. D'une manière générale, cette matrice D produit un objet géométrique ayant $N - 1$ dimensions. L'objectif de la MDS est de le projeter dans un espace planaire en le déformant le moins possible. Chaque point x_i est projeté en un point noté z_i.

Remarque

Quand on projette un objet géométrique dans un espace de plus petite dimension, on le déforme forcément (*cf.* la projection d'un cube en 2D par exemple). On cherche donc à projeter en minimisant la déformation d'une part, à mesurer cette déformation d'autre part. Dans le cas de l'ACP, le $\cos^2$ mesure cette déformation. Dans la MDS, la fonction Stress joue ce rôle.

La MDS est utilisée dans plusieurs cas :
- réduction de la dimension d'un objet géométrique ;
- détection de regroupements ;
- représentation planaire de graphes ;
- en chimie, pour la représentation de configurations moléculaires.

Pour cela, on définit une fonction dite « fonction de stress » qui a une forme du genre :

$$\text{Stress } (z_1, z_2, ...z_k) = \sqrt{\sum_{i \neq j} (D_{i,j} - ||z_i - z_j||)^2}$$

dans laquelle on associe z_i, la projection de la donnée x_i.

On veut minimiser cette fonction de stress : en effet, en la minimisant, on minimise l'écart entre la distance entre deux points dans l'espace original $D_{i,j}$ et la distance entre leurs projections $||z_i - z_j||)^2$.

Il existe plusieurs types de MDS. On distinguera ici :
- échelle directe ou indirecte :
  - échelle de distance de Kruskal-Shepard dans laquelle la distance entre les z_i est directement liée à la dissimilarité entre les x_i correspondants ;
  - échelle de Torgerson-Gower (échelle dite classique) dans laquelle on minimise la fonction :

$$\text{Strain } (z_1, z_2, ...z_k) = \sqrt{\sum_{i \neq j} (D_{i,j} - < z_i, z_j >)^2}$$

Conceptuellement, la différence entre les deux est que l'échelle classique est liée à une notion d'origine, alors que celle basée sur la distance n'est

pas liée à une origine. On peut contraindre l'échelle classique pour que l'origine soit le centre de gravité.
D'un point de vue calcul, l'échelle classique se résoud par une décomposition en vecteurs propres et on atteint un optimum global. L'échelle de distance nécessite un processus itératif, une descente de gradient, et on atteint un optimum local.

- échelle métrique ou non :
  - échelle métrique : la distance entre les z_i est directement liée à la dissmilarité entre les x_i ;
  - échelle non métrique : la distance est ici basée sur le rang des dissimilarités.

11.3 Réseaux de Kohonen

On présente tout d'abord la méthode des cartes auto-organisatrices de Kohonen. C'est un type de réseau de neurones particulier dit non supervisé car on ne spécifie pas au réseau ce qu'il doit apprendre : on lui fournit des données, à charge pour lui de les organiser. Cet algorithme possède de très jolies propriétés qui ont fait son succès.

11.3.1 Introduction

On suppose comme d'habitude que l'on dispose de N données $\mathcal{X} = \{x_i\}$, chacune décrite par ses P attributs.

On a un réseau constitué de K neurones (ou unités) et de P entrées. Les unités sont organisées topologiquement. Cette organisation peut être en 1, 2, 3 dimensions. En général, on les place sur une grille bi-dimensionnelle ce qui en permet une visualisation aisée. Chaque unité i reçoit les P entrées par une connexion de poids synaptique $w_{e,i}$, où e indique l'une des P entrées (*cf.* fig. 11.3).

Il y a donc $k \times P$ poids à apprendre. Ceux-ci sont appris à partir des données composant $\mathcal{X}$. Une fois l'apprentissage effectué :

- on obtient une « carte topologique » qui est une projection bi-dimensionnelle des données. Elle représente une organisation des données ;
- on peut projeter de nouvelles données et déterminer les plus ressemblantes parmi $\mathcal{X}$ par une méthode de type plus proches voisins.

11.3.2 Algorithme d'apprentissage

L'algorithme d'apprentissage est donné en 16 sous forme schématique.

On décrit ci-après les trois phases de l'algorithme qui apparaissent dans la boucle **répéter** ci-dessous.

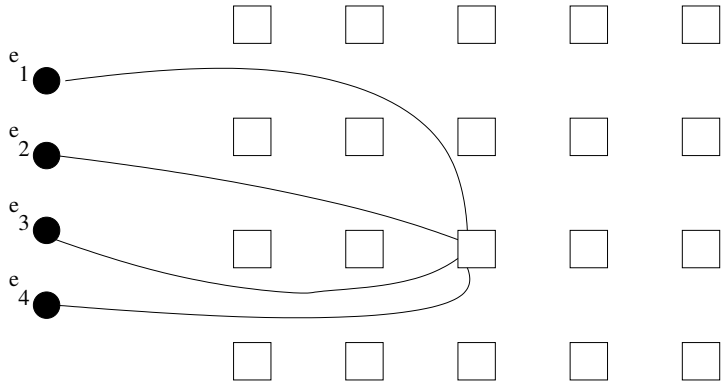


FIG. 11.3 – Schéma d'un réseau de Kohonen. À gauche, on note les entrées (on a supposé que $P = 4$). À droite, la grille des $k = 20$ unités. Pour l'une d'entre-elles, on a représenté les liaisons avec les entrées. Pour chaque unité, la notion de « voisinage » est très intuitive : ce sont les unités topologiquement proches : selon le cas, on considère 4, 6 ou 8 voisins.

Algorithme 16 Apprentissage dans un réseau de Kohonen

Nécessite: les N données d'apprentissage $\mathcal{X}$
  initialiser les $w_{e,i}$ aléatoirement
  répéter
    mélanger les exemples
    pour tout exemple $x_i \in \mathcal{X}$ **faire**
      placer x_i en entrée
      déterminer l'unité la plus activée (unité vainqueur) i_v
      déterminer les unités voisines de l'unité vainqueur i_v
      mettre à jour les poids des unités en fonction de leur proximité à l'unité vainqueur i_v
    fin pour
  jusque variation des poids est faible

Auparavant, brièvement, on donne l'idée générale de ce que fait cet algorithme. L'objectif est que les poids entre les entrées et chaque neurone entraîne la propriété suivante : deux données ressemblantes activent des unités proches et, au contraire, deux données dissemblables activent des unités éloignées. Pour cela, quand une donnée est présentée en entrée, on s'arrange pour créer un gradient d'activation décroissant autour de l'unité la plus activée. La correction apportée à chaque poids dépend donc de la distance à l'unité vainqueur. Au début, ce forçage a lieu sur l'ensemble du réseau ; petit à petit, la modification des poids ne concerne plus que les unités très proches de l'unité vainqueur.

On reprend maintenant les 3 étapes composant le corps de la boucle **répéter** de l'algorithme 16.

Déterminer l'unité la plus activée :

1. pour chaque neurone i, on calcule son activation $a_i =< \overrightarrow{w_i}, \overrightarrow{x} >$ où $\overrightarrow{x}$ est la donnée en entrée vue comme un vecteur ayant P composantes, $\overrightarrow{w_i}$ est le vecteur des poids synaptiques reliant les P entrées au neurone i :

$$\overrightarrow{w_i} = \begin{pmatrix} w_{1,i} \\ w_{2,i} \\ ... \\ w_{P,i} \end{pmatrix}$$

2. on détermine le neurone vainqueur dont l'activation est la plus forte : cela peut se faire par : $i_v = \arg\max_i \|a_i\|$; cela peut aussi se faire par : $i_v = \arg\min_i \|\overrightarrow{x} - \overrightarrow{w_i}\|$ ce qui permet d'éviter le calcul de l'activation de tous les neurones.

Déterminer les unités voisines de l'unité vainqueur : le neurone vainqueur est au centre d'un voisinage. Ce voisinage est mesuré par une fonction $h_{i_v,j}$ où i_v est le neurone vainqueur, j un autre neurone. Ce voisinage est défini grâce à une fonction distance $d_{i,j}$ qui donne la distance entre le neurone i et le neurone j. $h_{i,j}$ doit être symétrique autour du neurone vainqueur. De plus, $h_{i,j}$ doit décroître de manière monotone quand on s'éloigne du neurone vainqueur. Aussi, un choix typique est une gaussienne :

$$h_{i,j} = e^{-\frac{d_{i,j}^2}{2\ \sigma^2}}$$

dont l'écart-type σ caractérise la largeur du voisinage.
Pour la distance, on peut prendre :
– si les neurones sont disposés en ligne : $d_{i,j} = |i - j|$;
– si les neurones sont disposés sur une grille : leur distance euclidienne.

Il est bon que la taille du voisinage décroisse au cours du temps. On y arrive en prenant $\sigma(t) = \sigma(0)e^{-t/\tau_1}$, $\sigma(0)$ et τ_1 étant des constantes, t étant le numéro de l'itération courante dans l'algorithme 16.

Mise à jour des poids : sous forme vectorielle, on peut écrire la mise à jour des poids synpatiques du neurone i comme suit :

$$\Delta\overrightarrow{w_i} = \alpha f(i)\overrightarrow{x} - g(f(i))\overrightarrow{w_i}$$

où :
- α est le taux d'apprentissage ;
- $f(i)$ peut être pris égal à h_{j,i_v} ;
- $g(f(i))$ doit être nul si $f(i) = 0$ et positive sinon ; on peut prendre $g(f(i)) = \alpha f(i)$.

D'où :

$$\Delta w_i = \alpha h_{i,i_v}(x - w_i)$$

On peut prendre $\alpha(t) = \alpha_0 e^{-t/\tau_2}$.

11.3.3 Déroulement de l'apprentissage

D'un point de vue qualitatif, partant d'un état de désordre complet, l'apprentissage s'effectue en deux phases :
- auto-organisation : environ un millier d'itérations au cours desquels est créée une topologie (des relations de voisinage) grossière sur les K unités ;
- convergence au cours de laquelle la valeur des poids est affinée (nombre d'itérations = environ 500 fois le nombre de neurones).

Durant la phase d'auto-organisation, on peut prendre $\alpha_0 \approx 0.1$ et $\tau_2 = 1000$. Concernant la fonction h, celle-ci doit être telle qu'au début, le voisinage d'un neurone est constitué de tous les neurones, puis ce voisinage diminue jusqu'à ne contenir que très peu de neurones (voire 0 ou 1). On peut prendre $\tau_1 = \frac{1000}{\log(\sigma_0)}$.

Durant la phase de convergence, on peut conserver $\alpha(t) \approx 0.01$ constant. h ne doit contenir que très peu de neurones (voire 0 ou 1).

Ces deux phases ne sont pas explicitement séparées : elles s'enchaînent d'elles-mêmes au cours de l'exécution de l'apprentissage.

11.3.4 Exploitation d'un apprentissage

Une fois l'apprentissage effectué, le réseau s'est auto-organisé. On peut l'utiliser pour projeter de nouvelles données et déterminer les données d'apprentissage qui en sont les plus proches (*cf.* k-plus proches voisins au chap. 5) : ce sont les données les plus ressemblantes.

chien	.	.	renard	.	.	chat	.	.	aigle
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	hibou
.	.	.	.	.	.	tigre	.	.	.
loup	.	.	.	.	.	.	.	.	faucon
.	.	.	lion	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	colombe
cheval	.	.	.	.	.	.	coq	.	.
.	.	.	.	vache	.	.	.	.	oie
zèbre	.	.	.	.	.	.	canard	.	.

FIG. 11.4 – Pour chaque neurone de la carte de Kohonen, on indique le neurone le plus actif pour chacun des exemples présenté en entrée, une fois l'apprentissage effectué.

On prend l'exemple suivant : un ensemble d'animaux, chacun est décrit par 13 caractères (*cf.* table 11.4).

On utilise une grille de 10×10 neurones. En entrée, on place un vecteur (normalisé) ayant 13 composantes (les 13 caractères). On effectue 2000 itérations pour que l'apprentissage soit effectué.

Ensuite, on place en entrée chacun des exemples. Pour chaque exemple, on note le neurone le plus activé. Par exemple, on obtient la carte indiquée à la figure 11.4.

On peut aussi représenter une carte sur laquelle chaque neurone est étiqueté avec l'entrée qui le rend le plus actif 11.5. On constate des regroupements tout à fait significatifs entre différentes catégories d'animaux qui sont intuitivement ressemblantes.

11.3.5 Application des réseaux de Kohonen à des textes

On présente ici Websom, une application grandeur réelle des réseaux de Kohonen à la classification de textes [Kohonen et al., 2000, websom@websom.hut.fi, 1999, Lagus et al., 1999].

Il s'agît d'organiser des documents : il y a notamment eu des applications à des *news* (WEBSOM) et à des résumés de brevets pour WEBSOM2. Le réseau est bi-dimensionnel. Chaque document est associé à un neurone. Plus les neurones associés à deux documents sont proches, plus les textes sont proches. Initialement, les textes sont classés en un certain nombre de classes.

TAB. 11.4 – Des animaux et leurs caractères

		colombe	coq	canard	oie	hibou	faucon	aigle	renard	chien	loup	chat	tigre	lion	cheval	zèbre	vache
est	petit	x	x	x	x	x	x	.	.	.	.	x	.	.	.	.	.
	moyen	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.
	grand	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x
a	2 pattes	x	x	x	x	x	x	x	.	.	.	.	.	.	.	.	.
	4 pattes	.	.	.	.	.	.	.	x	x	x	x	x	.	x	x	x
	poils	.	.	.	.	.	.	.	x	x	x	x	x	x	x	x	x
	sabots	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x
	crinière	.	.	.	.	.	.	.	.	.	x	.	.	x	x	x	.
	plumes	x	x	x	x	x	x	x	.	.	.	.	.	.	.	.	.
aime	chasser	.	.	.	.	x	x	x	x	.	x	x	x	x	.	.	.
	courir	.	.	.	.	.	.	.	.	x	x	.	x	x	x	x	.
	voler	x	.	.	x	x	x	.	.	.	.	.	.	.	.	.	
	nager	.	.	x	x	.	.	.	.	.	.	.	.	.	.	.	.

chien	chien	renard	renard	renard	chat	chat	chat	aigle	aigle
chien	chien	renard	renard	renard	chat	chat	chat	aigle	aigle
loup	loup	lion	lion	lion	tigre	tigre	tigre	hibou	hibou
loup	loup	lion	lion	lion	tigre	tigre	tigre	faucon	faucon
loup	loup	lion	lion	lion	tigre	tigre	tigre	faucon	faucon
loup	loup	lion	lion	lion	hibou	colombe	faucon	colombe	colombe
cheval	cheval	lion	lion	lion	colombe	coq	coq	colombe	colombe
cheval	cheval	zèbre	vache	vache	vache	coq	coq	colombe	colombe
zèbre	zèbre	zèbre	vache	vache	vache	coq	coq	canard	oie
zèbre	zèbre	zèbre	vache	vache	vache	canard	canard	canard	oie

FIG. 11.5 – Pour chaque neurone de la carte de Kohonen, on indique la valeur en entrée qui produit la plus grande activation. On peut partionner les neurones en deux groupes connexes : les oiseaux et les mammifères. On constate également que les prédateurs sont concentrés en haut et les proies vers le bas.

Représentation d'un texte

Chaque texte est représenté par un sac de mot (*cf.* chap. 4, section 4.5.1). Dans le cas de WEBSOM2, il y a $N = 7.10^6$ documents. On constitue donc le vocabulaire de ces documents dont on retire les mots qui apparaissent moins de 50 fois et les 1335 mots les plus communs ont été éliminés. Cela fournit un corpus de 43222 mots. Chaque texte de brevet peut ainsi être représenté par un vecteur ayant 43222 composantes. Parmi ces 43222 mots potentiels, chaque texte contient 132 mots différents en moyenne.

Pondération des mots du sac Chaque composante du vecteur document est, en gros, le nombre d'occurences du terme dans le documents (histogramme). En fait, chaque mot est pondéré par son entropie de Shannon.

Notons $|\mathcal{Y}|$ le nombre de catégorie de textes. On calcule le poids $w(m, y)$ du mot m dans la catégorie y par :

$$w(m, y) = \frac{\text{nombre d'occurences de } m \text{ dans les textes de catégorie } y}{\text{nombre d'occurences de } m \text{ dans tous les textes}}$$

L'entropie de Shannon de m est alors :

$$H(m) = -\sum_{y \in \mathcal{Y}} w(m, y) \log\left(w(m, y)\right)$$

(*cf.* l'entropie rencontrée dans les arbres de décision au chap. 3) d'où l'on déduit le poids du mot m :

$$w(m) = H_{\max} - H(m) = \log\left(|\mathcal{Y}|\right) - H(m)$$

Représentation caractéristique d'un texte Cependant, 43222 composantes est trop. De plus, deux documents ayant même histogramme de mots peuvent avoir des sens très différents. Hors, on est intéressé par des documents proches au sens de leur sémantique, pas du vocabulaire qui s'y trouve. Aussi, on construit une représentation caractéristique des textes. Cette représentation abstraite devra être proche si les documents qu'elle représente sont proches.

Une possibilité couramment utilisée consiste à réduire le nombre de composantes. Plusieurs techniques sont possibles consistant à projeter ce vecteur dans un espace de plus petite dimension : indexation par la sémantique latente (*cf.* 11.1.3) par exemple, ou projection aléatoire. Cette dernière technique est très simple à mettre en œuvre et elle donne des résultats étonnemment bons dans l'application WEBSOM2. La projection aléatoire consiste simplement à sélectionner aléatoirement un certain nombre de dimensions (500 dans WEBSOM2) et à n'utiliser que ces dimensions des vecteurs documents.

On construit une matrice d (nombre de dimensions de l'espace projeté) lignes sur n colonnes (dimension initiale du vecteur à projeter). Chaque colonne est normée. Chaque composante de la matrice est tirée selon une distribution normale centrée réduite.

Une fois cette matrice de projection constituée, on multiplie simplement chaque vecteur document par cette matrice pour obtenir sa projection dans l'espace d dimensionel.

Sur des échantillons, dès que $d > 100$, on a constaté expérimentalement que le résultat de la classification est aussi bon que si l'on prend les vecteurs documents complets.

Ainsi, le codage des documents est de complexité $O(NL) + O(n)$ où N est le nombre de documents, L est le nombre moyen de mots différents par texte et n est la dimension initiale du vecteur document (43222 ici). L'indexation par la sémantique latente aurait entraîné une complexité du codage de $O(NLd)$ où d est la dimension résultante (500 ici), clairement bien supérieure. Par ailleurs, cette technique nécessite une décomposition en valeurs singulières qui est une opération numérique coûteuse, évidemment bien plus coûteuse que la génération de d nombres aléatoires.

Apprentissage de la carte

La carte terminale est constituée de 1.002.240 neurones. Avec 500 entrées, l'apprentissage consiste à trouver 500.10^6 poids. L'entraînement d'un tel nombre de poids serait extrêmement coûteuse, voire peut être même impossible dans la pratique avec les ordinateurs actuels.

Aussi, l'apprentissage s'est fait en 4 étapes en commençant avec un petit nombre de neurones et en augmentant ce nombre une fois l'apprentissage effectué. Initialement, on utilise 435 neurones ; après apprentissage des poids, on

multiplie ce nombre par 16, puis par 16 et enfin par 9. À chaque étape, l'apprentissage est refait ; en dehors de la toute première étape, l'apprentissage est effectué en s'appuyant sur l'apprentissage précédent. Une fois un apprentissage effectué, on augmente le nombre de neurones, donc le nombre de poids. Les poids d'un nouveau neurone sont initialisés en effectuant une interpolation linéaire avec les 3 neurones les plus proches de la carte précédente.

Ainsi, l'apprentissage complet coûte $O(dM^2) + O(dM) + O(M_0^2)$ où d est le nombre d'entrées (500), M est le nombre final de neurones (1 million) et M_0 est le nombre initial de neurones (435). Tout cela a demandé 300.000 pas d'apprentissage. Par entraînement direct du million de neurones, le coût aurait été $O(dM^2)$. Le gain est donc d'environ M/M_0, soit environ 20.000 ; hors, ainsi réalisé, l'apprentissage a nécessité 6 semaines de calcul sur une machine SGI O2000 équipée de 6 processeurs !

Une fois terminé l'apprentissage, la classification de documents dans la carte obtenue entraîne 64 % de classements corrects (mesuré sur un jeu de données étiqueté).

Visualisation de la carte Une interface graphique de type hyper-texte a été construite pour exploiter l'apprentissage ainsi effectué. L'ensemble des classes est localisé dans un espace bi-dimensionnel dans lequel on peut naviguer en cliquant sur les zones qui nous intéressent. On obtient alors des informations de plus en plus détaillées jusqu'à atteindre les documents eux-mêmes ; on peut aussi naviguer de proche en proche.

Étiquetage automatique de la carte

Utilisation de la carte

- analyse et compréhension d'un ensemble de textes ;
- classification de nouveaux textes ;
- recherche des documents proches d'un certain document (*cf.* k-plus-proches-voisins, chap. 5) ;

Notons pour finir que les cartes de Kohonen peuvent être appliquées à d'autres types de données. Il existe notamment une application du même genre que websom pour la musique.

11.3.6 Critique des cartes de Kohonen

+ bonne capacité de généralisation ;
+ très utilisé pour des tâches difficiles : analyse signal (caractères écrits, analyse signal, ...) ;
+ fonctionne bien même en présence de données bruitées ;
+ visualisation du résultat ;

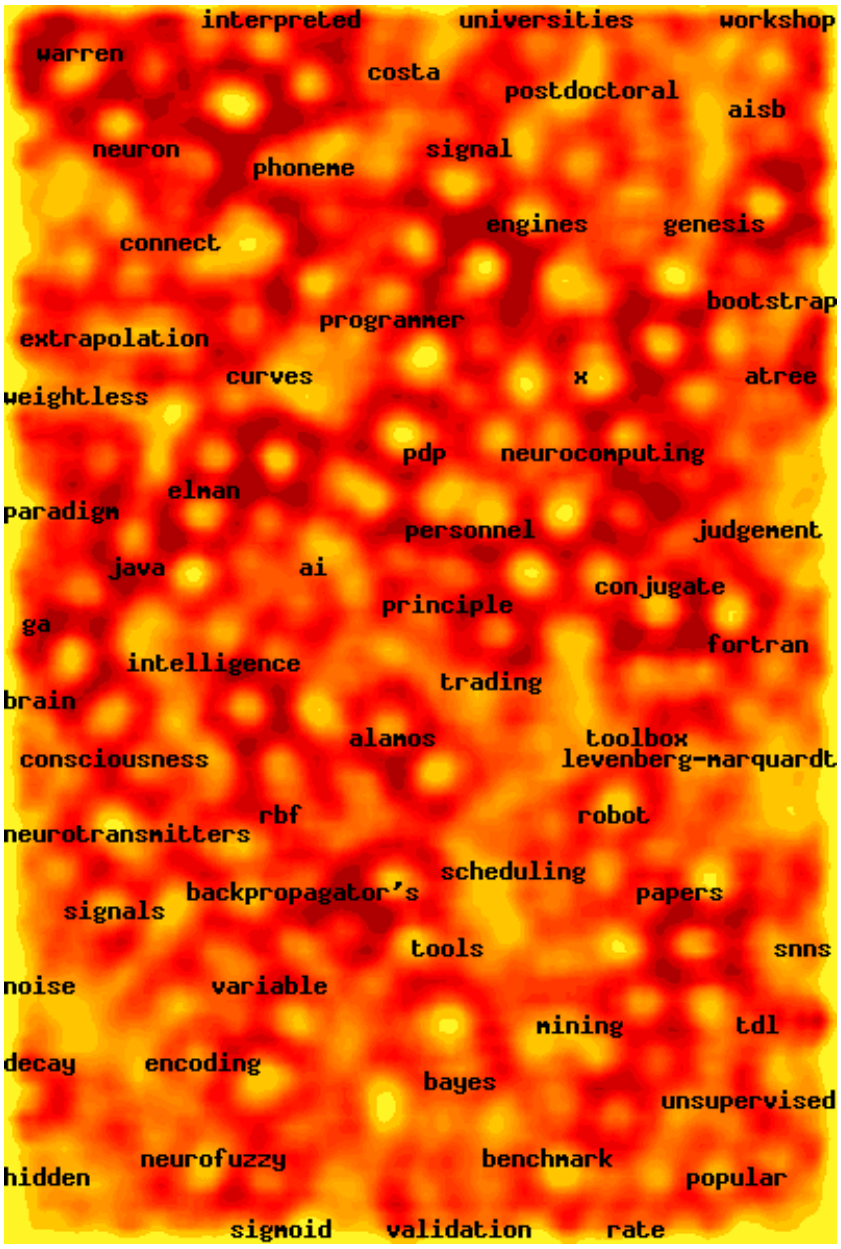


FIG. 11.6 – Exemple de carte construite par Websom.

– détermination du nombre d'unités pas forcément évident ;
- trouve un optimum local et non global ;
- difficile d'extraire un modèle de l'apprentissage effectué par le réseau.

11.4 Conclusion

Il existe de très nombreuses autres techniques de projections linéaires ou non linéaires. L'algorithme des centres mobiles (chap. 10) peut lui aussi être vu comme une méthode de projection : chaque donnée est projetée sur l'un des K centres. L'analyse en composantes curvilignes (Demartines and Hérault [1997]), les gas de neurones (Fritzke [1994]), LLE (Saul and Roweis [2003]), Isomap (Tenebaum et al. [2000]), SNE (Hinton and Roweis [200]) sont aussi à ranger dans les méthodes de projection.

11.5 Les logiciels libres

– *SOM_PAK : the self-organizing map program package* : `http://www.cis.hut.fi/nnrc/papers/som_tr96.ps.Z`
– ggobi est un magnifique logiciel libre pour l'exploration visuelle et interactive de données. Il contient un module pour la mise à l'échelle multidimensionnelle : `http://www.ggobi.org`
– DemoGNG : joli logiciel qui permet de tester de nombreuses algorithmes de projection non linéaire. `http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html`

11.6 Références

Deux livres qui font une présentation générale de l'ACP, AFC et AFD : le « Que-sais je ? » [Bouroche and Saporta, 1980] tout simplement et [Lebart et al., 1997]. [Foucart, 1997] est moins académique (en particulier, l'auteur ne présente pas les motivations des calculs et présente de nombreuses applications).

11.7 Exercices

Chapitre 12

Les règles d'association

Contenu

On dipose de N données x_i, chacune décrites par P attributs ; $x_{i,j}$ dénote la valeur de l'attribut a_j de la donnée i.

Dans de nombreuses applications, chaque attribut correspond à un item et la valeur de cet attribut dans une donnée particulière indique sa quantité dans cette donnée. Un cas particulier est celui où les attributs sont à valeur binaire et indiquent la présence ou l'absence d'un item.

Une règle d'association est de la forme $a_i = v_i, a_j = v_j, ...a_m = v_m \Rightarrow a_\alpha = v_\alpha, a_\beta = v_\beta, ...$, ce qui s'interprête par : « si les attributs a_i, a_j, ... a_m ont une certaine valeur, alors l'attribut a_α prend généralement une certaine valeur v_α, a_β une certaine valeur v_β, ... ».

La difficulté consiste notamment à trouver des règles qui soient significatives et non seulement le résultat du hasard. Les valeurs de N et P sont généralement

très grandes ($N = 10^6$ et $P = 10^5$ par exemple).

On s'intéresse au cas où les attributs prennent une valeur binaire, indiquant donc la présence ou l'absence d'un item. On présente une approche qui s'appuie sur la notion d'ensemble d'items fréquents (EIF), c'est-à-dire, des items qui sont souvent présents ensemble dans une même donnée. Après avoir détecté ces EIF, on génère ensuite des règles d'association.

12.1 Définitions

Dans ce qui suit, pour illustrer le discours, on part des individus suivants :

	Item A	Item B	Item C	Item D
individu 1	x	x		
individu 2	x		x	
individu 3		x		
individu 4	x		x	x
individu 5		x		

Le tableau de co-occurences est un tableau indiquant pour chaque paire d'items le nombre de co-occurences dans l'ensemble des individus :

	Item A	Item B	Item C	Item D
Item A	3	1	2	1
Item B	1	3	0	0
Item C	2	0	2	1
Item D	1	0	1	1

Définition 15 *On définit le « support » d'un ensemble d'items comme la fréquence d'apparition simultanée des items figurant dans l'ensemble.*

Un synonyme à « support » est « couverture ».

Exemples :

- support (A, B) = $\frac{1}{5}$ car A et B n'apparaissent simultanément que dans l'individu 1 ;
- support (A, C) = $\frac{2}{5}$ car A et C apparaissent simultanément dans les individus 2 et 4.

On dit qu'un ensemble d'items est un ensemble d'items fréquents si le support de cet ensemble d'items est supérieur à un certain seuil ($> 1\%$ par exemple).

Propriété 6 *Si S est un ensemble d'items fréquents, alors tout sous-ensemble de S est également un ensemble d'items fréquents.*

Définition 16 *Un ensemble d'items fréquents S est maximal si tout sur-ensemble de S n'est pas un EIF.*

Définition 17 *La « confiance » d'une règle « si condition alors conclusion » est le rapport :*

$$\frac{\textit{nombre de données où les items de la condition et de la conclusion apparaissent simultanément}}{\textit{nombre de données où les items de la condition apparaissent simultanément}}$$

Exemples :

- confiance (si A, alors B) $= \frac{1}{3}$ car A et B apparaissent simultanément dans 1 individu et A apparaît dans 3 individus ;
- confiance (si A, alors C) $= \frac{2}{3}$ car A et C apparaissent simultanément dans 2 individus et A apparaît dans 3 individus.

On définit un seuil de confiance comme la valeur minimale que la confiance doit avoir pour que l'apparition simultanée des items considérés ne puisse pas être simplement due au hasard, dans le cas où l'item C est très fréquent dans les individus.

On ne s'intéresse qu'aux règles ayant une confiance maximale.

Propriété 7 *Si la règle « si a et b alors c et d » a une confiance supérieure à un seuil fixé, alors les deux règles :*

- *« si a et b et d alors c »*
- *« si a et b et c alors d »*

ont une confiance supérieure à ce même seuil.

Attention, si la règle « si a et b alors c et d » a une confiance supérieure au seuil fixé, on ne peut rien dire quant aux règles « si a et b alors c » et « si a et b alors d ».

12.2 Algorithme A-Priori

On va maintenant présenter un algorithme qui détermine les règles d'association présentes dans un jeu de données, pour un seuil de support et un seuil de confiance fixés.

Cet algorithme fonctionne en deux phases : tout d'abord on recherche les ensembles d'items fréquents ; ensuite, on utilise ces EIF pour déterminer les règles d'association dont la confiance est supérieure au seuil fixé.

12.2.1 Construction des ensembles d'items fréquents

La construction des EIF est itérative : on va construire les EIF contenant un seul item, puis ceux contenant 2 items, puis ceux contenant 3 items, ...

L'idée est donc de ne retenir que les items dont le support est supérieur au seuil fixé, puis les paires d'items, puis les triplets, ... Pour cela, on pourrait considérer successivement toutes les paires d'items possibles, compter le nombre d'occurences de chacune et déterminer si leur support est supérieur au seuil ; on ferait ensuite de même pour les triplets, ... Cependant, cette approche n'est pas envisageable dans la pratique où le nombre de données et surtout d'attributs est très grand : cette approche naïve demanderait des temps de calcul beaucoup trop grands (complexité temporelle non polynomiale : *cf.* annexe C).

On va en fait s'appuyer sur la propriété mentionnée plus haut (*cf.* proposition 6) : pour qu'un ensemble d'items soit fréquents, il faut que tous ses sous-ensembles soient des EIF. Ainsi, ce n'est pas la peine de s'intéresser à des ensembles d'items dont tous les sous-ensembles ne sont pas eux-mêmes des EIF.

Ces remarques nous fournissent l'algorithme pour engendrer tous les EIF : on commence par déterminer les EIF de taille 1 ; on note cet ensemble L_1. Ensuite, on construit l'ensemble C_2 des EIF candidats de taille 2 : ce sont tous les couples construits à partir des EIF de taille 1. On obtient la liste des EIF de taille 2 L_2 en ne conservant que les éléments de C_2 dont le support est supérieur au seuil. On construit alors C_3, l'ensemble des triplets d'items dont les 3 sous-paires sont dans L_2 et on ne retient que ceux dont le support est supérieur au seuil, ce qui produit L_3. Et ainsi de suite, tant que L_i n'est pas vide. Plus formellement, on obtient l'algorithme 17.

Algorithme 17 Algorithme A-priori qui construit tous les EIF. Ceux-ci sont ensuite disponibles dans les variables L_i.

Nécessite: un support seuil s
  $L_1 \leftarrow$ liste des items dont le support est $> s$
  $i \leftarrow 1$
  répéter
    $i++$
    à partir de L_{i-1}, déterminer l'ensemble C_i des EIF candidats comprenant i items.
    $L_i \leftarrow \emptyset$
    pour tout élément $e \in C_i$ **faire**
      si support $(e) >$ seuil **alors**
        aouter e à L_i
      fin si
    fin pour
  jusque $L_i \neq \emptyset$

12.2.2 Génération des règles d'association à partir des EIF

Disposant des EIF, il nous faut maintenant les transformer en règles. Avec les notations précédentes, on dispose d'un ensemble de L_i pour des i croissant de 1 à une certaine valeur, chaque L_i étant une liste de i items dont la fréquence d'apparitions est supérieure à un certain seuil.

Supposons que L_3 contienne le triplet (a, b, c). Plusieurs règles d'association peuvent être engendrées par ce triplet :

1. si a et b alors c
2. si a alors b et c
3. si b et c alors a
4. si b alors a et c
5. si a et c alors b
6. si c alors a et b

Parmi ces 6 règles candidates, on ne doit retenir que celles dont la confiance est supérieure au seuil fixé.

Si l'énumération des règles candidates est possible sur ce petit exemple, il faut bien être conscient que sur dans une application réelle cela est impossible : à nouveau, le nombre de règles candidates est beaucoup trop grand. Il nous faut donc une méthode plus judicieuse.

Remarque

Exercice : soit un EIF contenant i items. Combien y-a-t'il de règles candidates ? Qu'en pensez-vous ?

À nouveau, on va adopter une approche itérative. En effet, il est tout à fait envisageable de tester toutes les règles candidates dont la conclusion ne contient qu'un seul item.

Remarque

Exercice : soit un EIF contenant i items. Combien y-a-t'il de règles candidates ayant un seul item en conclusion ? Qu'en pensez-vous ?

Ensuite, on va appliquer la proposition 7 : ainsi, supposons que $L_4 = \{(a, b, c, d)\}$. Supposons que la confiance des règles « si a et b et c alors d » et « si a et b et d alors c » soit supérieure au seuil. Dans ce cas, on peut affimer que la confiance de la règle « si a et b alors c et d » est également supérieure au seuil. Ainsi, à partir des règles ayant un seul item en conclusion et dont la confiance est supérieure au seuil, on peut engendrer les règles ayant deux items en conclusion dont la confiance est supérieure au seuil. Bien entendu, on peut ensuite itérer vers les règles ayant 3 items en conclusion, puis 4, ...

12.2.3 Discussion

Le nombre d'items étant typiquement grand, le nombre de règles à évaluer croît très vite en fonction de i. Il faut donc optimiser. On peut par exemple :

- élaguer par support minimum : pour un n donné, on ne considère que les règles ayant un support supérieur à un seuil fixé : ceci élimine les items trop peu fréquents pour générer des règles intéressantes ;
- ne s'intéresser qu'à certaines règles, par exemple, celles dont la conclusion correspond à un certain item ;
- faire des regroupements d'items.

Avantages de la méthode :

- liste d'items de taille variable
- résultats clairs

Défauts :

- produit énormément de règles parmi lesquels peu sont véritablement intéressantes : post-traitement humain important ;
- peut produire des règles triviales ;
- coût en temps de calcul important : complexité non polynomiale ;
- problème dans le cas où les données sont bruitées.

Différents algorithmes ont été proposés qui ont des performances bien supérieures à A-Priori : voir par exemple Fp-Growth [Han et al., 2000] ou PCY.

12.2.4 Application sur l'exemple

Sur l'exemple vu plus haut, on prend un support minimal de 2. On a :

- $L_1 = \{A, B, C\}$
- $C_2 = \{(A, C)\}$
- $L_2 = \{(A, C)\}$
- $C_3 = \emptyset$

Les règles suivantes sont examinées :

- « si A et B alors C »
- « si A et C alors B »
- « si B et C alors A »

dont les confiances sont nulles. On n'examine donc pas de règles ayant deux items en conclusion.

12.3 Applications

L'algorithme A-priori est notamment appliqué au problème de l'analyse du panier de la ménagère. Dans ce cas, chaque individu est un « panier » d'un client de supermarché et chaque item est un produit qui s'y trouve. Le gérant

du supermarché est intéressé par savoir quels articles sont achetés ensemble de manière fréquente.

Pour limiter le nombre d'attributs (un attribut = un type de produit vendu), on peut effectuer des regroupements d'items comme « conserves », « conserves de légumes », « conserves de légumes de telle marque », « conserves de légumes de telle taille », ...

On peut aussi ajouter d'autres attributs virtuels comme « fin de semaine » ou « période de vacances scolaires » qui permettent d'extraire des règles de la forme : « si produits a, b et c et fin de semaine alors produit l ».

Dans d'autres applications d'extraction de règles d'association, les individus sont des textes et les items des mots, ou encore les individus sont être des phrases et les items des documents.

12.4 Les paires rares mais importantes

On s'intéresse maintenant à la recherche de paires d'items dont le support est faible mais dont la présence est fortement corrélée.

Les données peuvent être vues comme une matrice $\mathbf{X} = (x_{i,j})$ dont chaque ligne représente un individu et chaque colonne représente un item. Typiquement la matrice $x_{i,j}$ est très creuse : pour chaque individu, il n'y qu'une faible proportion d'items présents (souvent bien moins qu'1 %). On cherche des paires de produits souvent présents ensemble, *i.e.* des colonnes similaires. Le problème à résoudre est donc de trouver le plus efficacement possibles ces paires de colonnes similaires, en faisant l'hypothèse que les items apparaissent rarement (support faible). On peut essayer d'utiliser l'algorithme A-Priori vu plus haut mais il est mal adapté à la recherche d'items de support faible ; on peut aussi essayer d'imaginer des méthodes plus spécifiques. C'est ce que nous décrivons ci-dessous.

On suppose par ailleurs que :

- P est suffisamment petit pour que l'on puisse stocker une information (un nombre) concernant chaque colonne en mémoire centrale, mais on ne peut pas stocker en mémoire centrale pour chaque paire d'attributs ;
- N est tellement grand que l'on ne peut pas stocker toute la matrice en mémoire centrale, même en tenant compte du fait que la matrice est creuse, et même en la compressant ;
- on cherche une alternative à l'utilisation des critères utilisés plus haut (support, confiance) car l'ensemble de paires candidates est trop grand pour être énuméré en un temps raisonnable.

Les applications de cette recherche de paires fortement corrélées à faible support sont :

- lignes et colonnes sont des pages web et $x_{i,j} = 1$ si la page i pointe sur la page j. Dans ce cas, des colonnes similaires peuvent être des pages traitant

d'un même sujet : ces pages sont pointées par les mêmes pages ;
- lignes et colonnes sont des pages web et $x_{i,j} = 1$ si la page j pointe sur la page i. Dans ce cas, des colonnes similaires sont des pages miroirs ;
- les lignes sont des pages web ou des documents, les colonnes sont des mots. Des colonnes similaires indiquent des mots qui apparaissent souvent ensemble dans les mêmes pages ;
- les lignes sont des pages web ou des documents, les colonnes sont des phrases. Les colonnes similaires indiquent des pages miroir ou des plagiats.

12.4.1 Similarité

On commence par préciser ce que l'on entend par « similarité » de deux colonnes. Informellement, deux colonnes sont similaires si elles ont généralement des 1 aux mêmes lignes.

Définition 18 *On mesure la similarité de deux colonnes $x_{.,j}$ et $x_{.,k}$ par le rapport entre le nombre de lignes où $x_{i,j}$ et $x_{i,k}$ sont égaux à 1 en même temps par le nombre de lignes où l'un des deux seulement vaut 1 :*

$$Sim(x_{.j}, x_{.k}) = \frac{|x_{i,j} \wedge x_{i,k}|}{|x_{i,j} \vee x_{i,k}|} \tag{12.1}$$

Complexité du calcul de Sim : $O(N)$ pour deux colonnes données ; il y a $O(P^2)$ paires de colonnes ; donc $O(N \times P^2)$ pour l'ensemble des données.

En prenant $N = 10^6$ et $P = 10^5$, $NP^2 = 10^{16}$. Sur un ordinateur capable d'effectuer 1 million de comparaisons par seconde, le calcul de Sim demande 10^{10} secondes, soit 10 milliards de secondes, soit ... 300 ans !

On va donc essayer de trouver des manières de diminuer largement le temps de calcul. Pour cela, on va adopter une approche statistique : au lieu de calculer Sim, on va calculer une autre quantité qui aura la même valeur statistiquement (un estimateur de Sim autrement dit), mais moins coûteuse en temps de calcul.

12.4.2 Signature

On va définir la signature d'une colonne $Sig(a_j)$ de telle manière que :
- complexité spatiale : $Sig(a_j)$ est suffisamment petit pour que l'on puisse la stocker en mémoire centrale pour toutes les colonnes ;
- cohérence des notions de signature et de similarité : deux colonnes $x_{.,j}$ et $x_{.,k}$ sont très similaires si leurs signatures sont très similaires.

Une idée qui ne marche pas : sélectionner au hasard 100 lignes ; considérer la colonne qui nous intéresse et la valeur des bits de cette colonne uniquement sur ces 100 lignes ; la chaîne de 100 bits obtenue est la signature de la colonne.

Au lieu de faire 10^6 comparaison par paire de colonnes, on n'en ferait plus que 100 : on gagne donc un facteur 10^4, soit un temps de calcul global d'environ

10^6 secondes, soit 12 jours : de ce point de vue, c'est beaucoup mieux que 300 ans.

Cependant, cette idée ne marche pas car, en général, la signature d'une colonne ne sera composée que de 0 ; donc, deux colonnes quelconques seront généralement considérées comme similaires mêmes si leurs 1 sont placés sur des lignes différentes.

On définit le type de deux colonnes pour une ligne donnée par a, b, c ou d comme suit :

Type	$x_{i,j}$	$x_{i,k}$
a	1	1
b	1	0
c	0	1
d	0	0

Pour une paire de colonnes, On note respectivement a, b, c et d le nombre de lignes de type a, b, c et d.

On a :

$$Sim(x_{.,j}, x_{.,k}) = \frac{|x_{i,j} \wedge x_{i,k}|}{|x_{i,j} \vee x_{i,k}|} = \frac{a}{a+b+c} \tag{12.2}$$

Il nous reste à trouver une bonne signature, bonne dans le sens où deux colonnes ayant des signatures similaires sont des colonnes effectivement similaires, et dans le sens où son calcul n'est pas trop coûteux ? C'est la question abordée dans les 4 prochaines sections.

12.4.3 Signature par hashage min

Cette première approche consiste tout d'abord à sélectionner 100 numéros de lignes (100 nombres compris entre 1 et N donc).

Ensuite, pour chaque colonne a_j, on calcule sa signature de la manière suivante : pour chaque numéro de ligne i, on note $h(a_j)(i)$ le plus petit numéro de ligne $k \geq i$ tel que $x_{i,j} = 1$.

L'ensemble des 100 nombres $h(a_j)$ constitue la signature de la colonne a_j.

Considérons deux colonnes a_j et a_k de la matrice $\mathbf{X}$. $Pr[h(a_j)(i) = h(a_k)(i)] = \frac{a}{a+b+c}, \forall i \in \{1, ...100\}$, avec a, b et c définis comme précédemment selon le type des deux colonnes. On constate que cette probabilité est égale à la similarité des deux colonnes.

Donc, la similarité des deux listes $h(a_j)$ et $h(a_k)$ est une bonne mesure de la similarité des deux colonnes ; elle est statistiquement significative. Son coût de calcul est faible.

12.4.4 Hashage localement sensible (LSH)

12.4.5 Hashage k-min

12.5 Logiciels libres

- `apriori` de Ch. Borgelt : `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html#assoc` . Voir aussi le logiciel `eclat` sur la même page ;
- `weka` contient une implantation de l'algorithme A-Priori ; `http://www.cs.waikato.ac.nz/~ml`
- un générateur de jeux de données : `quest` sur le site `http://www.ibm.com` ;

- `Magnum Opus` : `http://www.rulequest.com` : logiciel commercial dont une version de démonstration est accessible gratuitement en ligne ; les algorithmes sont décrits dans Webb [2000]

Chapitre 13

Prédiction numérique

Contenu

Ce chapitre aborde un problème très général où il s'agît de déterminer la classe d'instances, cette classe étant cette fois une variable continue, quantitative. Quittant le discret pour le continu, on arrive dans le domaine des mathématiques plutôt que celui de l'informatique. De nombreuses méthodes ont été proposées. Voir par exemple Motulsky and Christopoulos [2003], Hilborn and Mangel [1997] et le chapitre 15 de [Press et al., 1988][1]. Relevant donc plus d'un cours de mathématiques, on effectue un bref survol, mettant essentiellement l'accent sur les approches plus purement informatiques.

Comme dans les autres chapitres, on dispose d'un jeu de N données noté $\mathcal{X}$, chacune étant décrite par P attributs. On suppose ces attributs quantitatifs.

13.1 Régression linéaire simple et multiple

13.2 Arbres de régression

Cette technique produit un modèle linéaire par morceaux.

[1] il est peut-être bon de rappeler ici que les programmes en C reproduits dans ce livre sont tous faux ! Il n'en demeure pas moins que les explications sont correctes et très intéressantes.

Principe : on construit un arbre de décision dans lequel chaque feuille contient une formule de régression linéaire. Selon la feuille dans laquelle une donnée est classée, telle ou telle fonction est utilisée pour estimer la classe de la donnée.

13.3 Réseau de neurones

On peut appliquer ce que nous avons dit concernant la classification par réseau de neurones supervisé (*cf.* chap. 7) à la régression non linéaire. On a montré qu'un réseau de neurones supervisé constitué d'une couche cachée d'unités sigmoïdes et d'une sortie linéaire est un approximateur de fonctions universel Hornik [1991].Donc, au lieu d'apprendre la classe à valeur discrète associée à chaque donnée, on apprend au réseau une valeur continue.

Tout ce qui a été dit sur la classification par réseau de neurones supervisé s'applique ici (*cf.* chap. 7).

13.4 Régression à vecteur support : RVS

Les machines à vecteurs supports ont été adaptées au problème de régression (régression à vecteurs supports : RVS).

13.5 Régression locale pondérée

13.6 Logiciels libres

- `weka` contient une implantation de l'algorithme de construction d'arbre de régression (M5), de réseaux de neurones supervisés (avec les défauts déjà mentionnés : ce n'est pas un outil performant, mais pédagogique) : `http://www.cs.waikato.ac.nz/~ml`
- `JavaNNS`, boîte à outil de réseaux de neurones, permet de faire de la régression : `http://www-ra.informatik.uni-tuebingen.de/SNNS`
- `svmTorch` est un outil sérieux pour faire de la régression par machine à vecteurs supports : `http://www.idiap.ch/learning/SVMTorch.html`

Chapitre 14

Pré- et post-traitement

Chapitre 15

Applications à la fouille de données

Contenu

15.1 Fouille de textes

15.2 Fouille de données sur le web

Voir [Baldi et al., 2003].

15.3 Commerce électronique

Annexe A

Rappels de statistiques

Valeur moyenne d'un attribut

La moyenne d'un attribut a_j sur l'ensemble des N individus est :

$$\overline{a_j} = \frac{\sum_{i=1}^{i=N} x_{i,j}}{N} \tag{A.1}$$

Il faut distinguer cette valeur moyenne estimée sur un ensemble d'individus $\mathcal{X}$ de la « vraie » moyenne qui serait mesurée sur l'ensemble de toutes les données $\mathcal{D}$.

Variance d'un attribut

La variance[1] est :

$$\text{var}(a_j) = \frac{\sum_{i=1}^{i=N} (x_{i,j} - \overline{a_j})^2}{N-1} \tag{A.2}$$

Écart-type d'un attribut

et l'écart-type :

$$\sigma(a_j) = \sqrt{\text{var}(a_j)} \tag{A.3}$$

Là aussi, il faut bien distinguer l'estimation de la variance, ou de l'écart-type, de sa « vraie » valeur.

On a aussi :

$$\sigma(a_j) = \sqrt{\overline{a_j^2} - \overline{a_j}^2} \tag{A.4}$$

où $\overline{a_j^2}$ est la moyenne sur i des $x_{i,j}^2$:

$$\overline{a_j^2} = \frac{x_{i,j}^2}{N}$$

et $\overline{a_j}^2$ est le carré de la moyenne sur i des $x_{i,j}$:

$$\overline{a_j}^2 = (\frac{x_{i,j}}{N})^2$$

[1] le dénominateur est $N-1$ et non N car la moyenne est estimée à partir des données en présence. Si l'on connaissait *a priori* la valeur de la moyenne, le dénominateur serait N. C'est une question de biais : *cf.* un cours de stats.

Intuitivement, variance et écart-type indiquent si les valeurs prises par une variable sont proches de la moyenne ou plutôt dispersées. Plus la variance est faible, plus les valeurs sont concentrées autour de la moyenne. On peut voir la moyenne comme un moyen de résumer les $x_{i,j}$ (j fixé) ; de ce point de vue, l'écart-type est une information complémentaire à la moyenne permettant de caractériser plus précisément la répartition de la valeur d'un attribut.

Remarque

Si l'on a N nombres et que l'on veut fournir 2 nombres qui permettent de les représenter « au mieux », il est clair que prendre deux nombres parmi les N au hasard n'apporte pas grand chose... Par contre, le couple composé de la moyenne et de l'écart-type (ou la variance) contient une information beaucoup plus précise sur ces N nombres.

On peut construire un histogramme, représentation graphique de la distribution des valeurs prises par le caractère considéré. Plus la variance est faible, plus l'histogramme est resserré et pointu.

Attribut centré

On dit qu'un caractère est centré si sa moyenne est nulle. Donc, on centre un attribut a_j en lui retirant sa valeur moyenne $\overline{a_j}$.

Attribut réduit

On dit qu'un attribut est réduit si son écart-type vaut 1. Donc, on réduit un attribut en divisant sa valeur par son écart-type.

Attribut centré réduit

La notion de variable centrée réduite est importante dans la mesure où le centrage et la réduction entraînent une espèce de normalisation des données en éliminant l'influence de l'unité de mesure (*cf.* section 11.1.1). De plus, cela simplifie les calculs et les équations. Considérons l'attribut a ; $\frac{a-\overline{a}}{\sigma(a)}$ est l'attribut centré réduit associé à a.

Coefficient de corrélation linéaire

On définit alors le coefficient de corrélation linéaire entre deux attributs a_j et a_k (centrés ou non) :

$$r(a_j, a_k) = \frac{\sum_{i=1}^{i=N} x_{i,j}x_{i,k}}{\sqrt{(\sum_{i=1}^{i=N} x_{i,j})^2 - (\sum_{i=1}^{i=N} x_{i,k})^2}} \tag{A.5}$$

Le coefficient de corrélation linéaire prend toujours une valeur dans l'intervalle $[-1, 1]$. Il indique si les deux variables ont leur valeurs liées de manière linéaire (*cf.* fig. A.1).

Remarque

À nouveau, on peut voir le coefficient de corrélation linéaire comme un moyen de résumer l'information présente dans une série de couples de valeurs.

covariance

On définit alors la covariance entre deux attributs centrés a_j et a_k :

$$\mathrm{cov}(a_j, a_k) = \frac{\sum_{i=1}^{i=N} x_{i,j}x_{i,k}}{N-1} \tag{A.6}$$

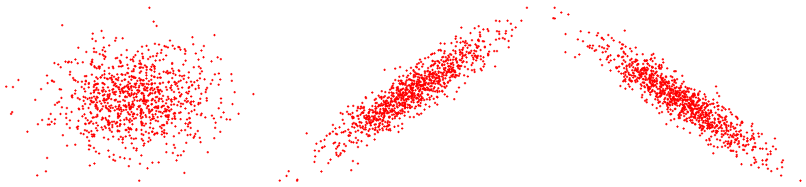


FIG. A.1 – À gauche, pas de corrélation linéaire ($r \approx 0$) ; au centre, une corrélation linéaire forte positive ($r \approx +1$) ; à droite, une corrélation linéaire négative ($r \approx -1$).

D'une manière générale, le coefficient de corrélation linéaire peut également s'exprimer comme suit :

$$r(a_j, a_k) = \frac{\text{cov}(a_j, a_k)}{\sigma(a_j)\ \sigma(a_k)} \tag{A.7}$$

soit, si les attributs sont centrés et réduits :

$$r(a_j, a_k) = \text{cov}(a_j, a_k) \tag{A.8}$$

Notons que dans tout ce qui précède, les rôles des deux caractères sont symétriques. Aussi, on a en particulier : $r(a_j, a_k) = r(a_k, a_j)$.

Bien entendu, on peut parler de corrélation non linéaire (parabolique, ...) qui indique l'existence d'une relation entre deux caractères simplement un peu plus complexe. Nous ne nous intéresserons ici qu'aux corrélations linéaires.

Droite des moindres carrés

Portons chaque point $(x_{i,j}, x_{i,k})$ sur un graphique planaire, les $x_{i,k}$ étant placés en abscisse, les $x_{i,j}$ en ordonnées. On peut ensuite définir la droite qui passe « au mieux » dans le nuage de points, c'est-à-dire la droite qui minimise la somme des distances entre chaque point du nuage à la droite ; cette droite est dite des « moindres carrés ». Son équation est donnée par :

$$Y = \frac{r(a_j, a_k)\ \sigma(a_j)}{\sigma(a_k)}\ (X - \overline{a_k}) + \overline{a_j} \tag{A.9}$$

Remarque

L'équation de cette droite s'obtient en calculant la somme des distances vecticales entre chaque point $(x_{i,j}, x_{i,k})$ et une droite $Y = aX + b$ puis, en calculant les dérivées partielles de cette distance par rapport au coefficient directeur a et à b que l'on égalise à 0 — pour minimiser cette distance — pour en déduire les valeurs de a et b.

Par « distance verticale » entre un point $(x_{i,j}, x_{i,k})$ et la droite $Y = aX + b$, on entend $(x_{i,k} - (ax_{i,j} + b))^2$.

Prenons un petit exemple pour illustrer un point. Soient les trois séries de nombres suivantes :

0.5	1	0.7
1	1.5	1
1.5	1.4	1.4
1.5	1.2	1.5
1	0.9	1.1
0.75	0.8	0.8
1.25	0.7	1.2
0.75	0.9	0.9
1	1	1
0.75	1.1	0.9

On a :
- pas de corrélation entre les colonnes 1 et 2 ($r_{1,2} = 0.36$) ;
- forte corrélation linéaire entre les colonnes 1 et 3 ($r_{1,3} = 0.98$) ;
- les données dans les colonnes 2 et 3 sont les mêmes, pas dans le même ordre : même moyenne (1.05) et même écart-type (0.24).

Autre cas : regroupements : on peut avoir même moyennes et écarts-types mais avoir des distributions différentes. Exemple :

0.7	0.9
0.75	1
1.63	1.4
1.58	1.5
0.8	1.1
0.9	0.8
1.55	1.2
0.85	0.9
0.83	1
0.88	0.9

On a :
- moyenne presqu'égales : 1.05 et 1.07 ;
- des écarts-types proches : 0.38 et 0.23 ;
- les valeurs de la première colonne sont réparties selon deux groupes (autour de 0.8 et de 1.6), alors que celles de la seconde colonne sont toutes concentrées autour de 1.

Cette structuration doit être détectée.

Annexe B

Théorème de Bayes

Soient S un ensemble d'événements, $A \subset S$ et $B \subset S$ des (ensembles d') événéments (*cf.* fig. B). La probabilité d'un événement est :

$$Pr[A] = \frac{\text{surface de } A}{\text{surface de } S} \tag{B.1}$$

La probabilité d'occurence simultanée de deux événements est :

$$Pr[A \text{ et } B] = Pr[A \wedge B] = \frac{\text{surface de } A \cap B}{\text{surface de } S}$$

La probabilité d'occurence d'un événement ou d'un autre est :

$$Pr[A \text{ ou } B] = Pr[A \vee B] = \frac{\text{surface de } A + \text{ surface de } B - \text{ surface de } A \text{ et } B}{\text{surface de } S}$$

Autrement dit, la probabilité que A ou B survienne est :

$$Pr[A \vee B] = Pr[A] + Pr[B] - Pr[A \wedge B] \tag{B.2}$$

Enfin, la probabilité d'occurence d'un événement A si un autre événement B a lieu est :

$$Pr[A|B] = \frac{\text{surface de } A}{\text{surface de } B}$$

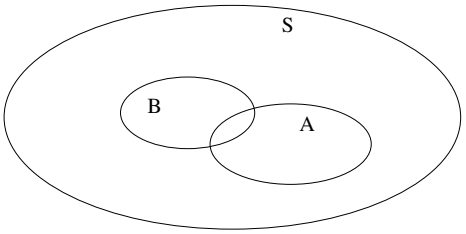


FIG. B.1 – L'ensemble des événements possibles S et deux événements particuliers A et B.

$Pr[A|B]$ signifie : probabilité que l'événement A survienne si l'événement B survient également. C'est-à-dire, on suppose que B est vrai et on cherche la probabilité que A survienne. Ici, on restreint donc l'ensemble des événements possibles à B et on cherche $Pr[A]$ sous cette condition. Par analogie avec (B.1), on a :

$$Pr[A|B] = \frac{\text{surface de } A \text{ et } B}{\text{surface de } B} \tag{B.3}$$

soit :

$$Pr[A|B] = \frac{Pr[A \wedge B]}{Pr[B]} \tag{B.4}$$

Par symétrie, on peut également écrire :

$$Pr[B|A] = \frac{Pr[B \wedge A]}{Pr[A]} \tag{B.5}$$

soit :

$$Pr[A|B]Pr[B] = Pr[B|A]Pr[A] \tag{B.6}$$

que l'on ré-écrit :

$$Pr[A|B] = \frac{Pr[B|A]Pr[A]}{Pr[B]} \tag{B.7}$$

ce qui démontre le théorème de Bayes.

Annexe C

Complexité algorithmique

Complexité algorithmique

La « complexité algorithmique » permet de caractériser les ressources qui sont utilisées par un ordinateur pour exécuter un algorithme particulier. Ces ressources sont de deux ordres :

- spatiales : c'est la quantité de mémoire qui est nécessaire pour son exécution ;
- temporelles : c'est la quantité de temps qui est nécessaire pour son exécution.

Concernant la complexité temporelle, précisons qu'il ne s'agît pas de calculer combien de secondes ou d'années un certain algorithme va fonctionner avant de donner son résultat. Il s'agît de mesurer le nombre d'opérations élémentaires qui doivent être exécutées ; ensuite, sachant combien de temps un certain ordinateur met pour exécuter cette opération élémentaire, on peut en déduire à peu près le temps d'exécution. Mais en fait, le but n'est pas du tout là ; le but est de comparer des algorithmes entre-eux selon un critère quantitatif précis et de pouvoir déterminer en (presqu') un clin d'œil si un algorithme est utilisable ou non dans la pratique : un algorithme est utilisable si on estime qu'il ne va pas avoir tendance à utiliser une quantité de mémoire ou de temps que l'on ne pense pas envisageable avec les ordinateurs qui fonctionnent selon leur mode de fonctionnement actuel. Là encore, on attire l'attention du lecteur : par l'expression « les ordinateurs qui fonctionnent selon leur mode de fonctionnement actuel », nous ne voulons pas signifier les ordinateurs avec leurs caractéristiques de juillet 2005 ; nous faisons appel ici à la notion de « machine de Turing », proposée dans les années 1930 par Alan Turing pour décrire le fonctionnement des ordinateurs de manière abstraite. 70 ans plus tard, tous les ordinateurs qui ont existé et qui existent[1] fonctionnent comme des machines de Turing (et ils sont même toujours moins puissants qu'une machine de Turing qui possède une

[1] quand on dit « tous », c'est vraiment tous : de la minuscule calculatrice de poche aux plus gros des supercalculateurs du Pentagone.

mémoire infinie).

La chose que l'on veut capturer ici est la suivante : j'ai un jeu de N données et un algorithme $\mathcal{A}$. Combien d'opérations seront réalisées par $\mathcal{A}$ sur le jeu de données en fonction de N ? et quelle quantité de mémoire cela va-t-il nécessiter ? toujours en fonction de N.

Quelques exemples (en considérant que la variable importante est le nombre de données N ; on pourrait aussi exprimer la complexité en fonction de N et du nombre d'attributs P) :

- lire N données et les placer dans un tableau : le nombre d'opérations est de l'ordre de N (*i.e.*, linéaire en N), ce que l'on note $O(N)$;
- afficher les N données stockées dans un tableau : $O(N)$;
- faire la somme des N données : $O(N)$;
- chercher la plus petite valeur parmi N nombres : on doit comparer chaque nombre à tous les autres, soit $N \times (N-1)$ comparaisons. Par symétrie, on peut diviser ce nombre par 2. Mais, on ne tient pas compte des constantes, d'où : $O(N^2)$;
- trier N nombres : algorithmes simples : $O(N^2)$; au mieux, dans le cas général, on fait $O(N \log N)$.

Complexité linéaire

Les algorithmes en $O(N)$ sont dits linéaires. C'est en général ce que l'on peut faire de mieux pour résoudre un problème exactement. On peut envisager de traiter ainsi des volumes de l'ordre de $N = 10^{10}$, voire quelques ordres de grandeur de plus. (Les ordinateurs actuels exécutent environ 1 milliard d'opérations élémentaires par seconde, d'où cet ordre de grandeur.)

Complexité polynomiale

Les algorithmes en $O(N^i)$ sont dits polynomiaux. Dès que l'exposant i dépasse 2 ou 3, l'algorithme n'est plus exécutable en temps raisonnable pour un nombre de données un peu important. Il n'en demeure pas moins que l'algorithme s'arrête au bout d'un temps fini. À raison, de 10 opérations élémentaires à effectuer sur chaque donnée, un algorithme en $O(N^2)$ traite 30000 données en 10 secondes ; un algorithme en $O(N^3)$ n'en traite que 1000 dans le même temps, soit 30 fois moins.

Cas intermédiaire, les algorithmes en $O(N \log N)$ sont beaucoup moins coûteux qu'un algorithme en $O(N^2)$. Leur exécution est envisageable pour des N grands. À raison, de 10 opérations élémentaires par donnée, un algorithme en $O(N \log N)$ traite 200000 données en 10 secondes.

Les algorithmes dont la complexité est polynomiale font tous parti d'une classe dite, classe $\mathcal{P}$.

Complexité non polynomiale

Malheureusement, les problèmes intéressants ont généralement une complexité non pas polynomiale, mais exponentielle, *i.e.* en $O(e^N)$. À raison, de 10 opérations élémentaires par donnée, un algorithme en $O(e^N)$ traite 19 données en 10 secondes ! Les problèmes de classification et de segmentation font parti de ces problèmes... Fort logiquement, ces problèmes non polynomiaux font parti

de la classe $\mathcal{NP}$. Parmi tous les problèmes non polynomiaux, il y en a tout un ensemble dont la complexité est équivalente ; on dénomme la classe $\mathcal{NP}$-complets et les problèmes qui y appartiennent, les problèmes $\mathcal{NP}$-complets. En fait, à l'heure actuelle, on ne sait pas si ces problèmes sont vraiment non polynomiaux ; le fait est que l'on n'a jamais réussi à trouver un algorithme polynomial capable de résoudre exactement l'un d'eux. Cette question, $\mathcal{P}$ est-elle égale à $\mathcal{NP}$, est l'une des plus importantes en informatique à l'heure actuelle. À peu près tout le monde pense que la réponse est non et que cela ne vaut même pas la peine d'essayer de montrer le contraire. Si on montre un jour le contraire, une véritable révolution s'ensuivra en algorithmique, avec des conséquences inimaginables dans toutes les applications de l'informatique.

Algorithme non exact, approché et heuristique

Dans tout ce qui précède, on a parlé d'algorithmes résolvant exactement un problème. Les problèmes intéressants étant de complexité non polynomiale, une démarche habituelle consiste à résoudre le problème de manière approchée, ou « non exacte ». Par exemple, prenons le problème de calculer la moyenne de N nombres. Comme on l'a dit plus haut, l'algorithme correspondant est en $O(N)$. Dans certains cas, c'est encore beaucoup trop (par exemple, si $N = 10^{15}$). On peut alors utiliser une approche non exacte comme suit : parmi les N nombres, en sélectionner au hasard 10^9 et calculer leur moyenne. Sous réserve de quelques hypothèses sur ces nombres, on sait que la moyenne ainsi évaluée sera proche de la moyenne des 10^{15} nombres initiaux ; on a ainsi adopté un algorithme non exact pour calculer leur moyenne et transformer un calcul de durée très longue en un calcul très court. Ce type d'algorithmes non exacts est souvent dénommé « heuristique » ou encore, algorithme approché. On parle aussi d'une part d'algorithme d'optimisation globale pour un algorithme résolvant exactement un problème et trouvant donc un optimum global, d'autre part d'algorithme d'optimisation locale pour un algorithme qui ne fournit qu'un optimum local, sans garantie particulière par rapport à l'optimum global.

Décidabilité, semi-décidabilité et indécidabilité

Notons pour terminer que tous les algorithmes dont nous avons parlé s'arrêtent au bout d'un temps fini, même s'il est éventuellement très long, même par rapport à l'âge de l'univers. Il y a pire. Certains algorithmes ne s'arrêtent jamais ; un problème pour lequel il n'existe pas d'algorithme qui s'arrête au bout d'un temps fini (ou pour être plus plus précis, dont on ne puisse pas démontrer qu'il s'arrête au bout d'un temps fini) est dit « indécidable ». Enfin, certains algorithmes s'arrêtent dans certains cas, pas d'en d'autres[2] ; un problème pour lequel le meilleur algorithme dont on dispose s'arrête au bout d'un temps fini dans certains cas, ne s'arrête jamais dans d'autres, est dit « semi-décidable ».

Algorithmes déterministes et algorithmes stochastiques

Pour clore cette courte annexe, notons qu'il ne faut pas confondre ce qui vient d'être dit avec la notion d'algorithme « déterministe ». Un algorithme

[2] remarquons qu'il s'agît bien d'algorithmes déterministes.

déterministe est un algorithme qui, à chaque fois qu'il est exécuté sur une machine donnée dont la configuration ne change pas, sur les mêmes entrées (jeu de données, paramètres, ...), produit exactement[3] le même résultat. Un algorithme dont le résultat peut varier entre deux exécutions, bien que ses entrées et paramètres soient demeurées les mêmes, est dit « non déterministe » ou « stochastique ».

En guise d'exercice, on réfléchira à chacun des algorithmes rencontré dans ce poly, et on déterminera s'il est déterministe ou stochastique. On pourra aussi essayer de déterminer leur complexité.

fonction calculable

D'un point de vue théorique, on peut considérer que tout algorithme calcule une certaine fonction de ses entrées. On peut donc définir l'ensemble des fonctions pour lesquelles il existe un algorithme déterministe qui la calcule en un temps fini. Cela définit l'ensemble des fonctions dites « calculables ». On peut aussi définir l'ensemble des fonctions pour lesquelles il existe un algorithme stochastique qui la calcule en temps fini. On montre que ces deux ensembles de fonctions sont les mêmes. Ainsi, contrairement à ce que l'on pourrait penser, les algorithmes non déterministes ne permettent pas de calculer plus de fonctions, ou d'autres fonctions, que les algorithmes déterministes.

[3] par « exactement », on entend vraiment exactement, *i.e.* au bit près.

Annexe D

Programmation non linéaire

Source : Fourer [2004]

Un « programme non linéaire » est un problème qui peut être mis sous la forme suivante : minimiser $F(\overrightarrow{x})$ en respectant les contraintes :

- $g_i(\overrightarrow{x}) = 0$ pour $i = 1, ..., m1$, $m1 \geq 0$
- $h_j(\overrightarrow{x}) >= 0$ pour $j = m1 + 1, ..., m$, $m \geq m1$

Nous avons donc une fonction à valeur scalaire F de plusieurs variables $\overrightarrow{x}$ que nous cherchons à minimiser tout en respectant un certain nombre de contraintes qui délimitent l'ensemble de données acceptables. F est nommée la « fonction objectif ». (Remarquons que si l'on veut maximiser F, il suffit de multiplier F par -1.)

En toute généralité, la programmation non linéaire est un domaine difficile. Un cas bien étudié est celui où les contraintes sont linéaires, c'est-à-dire, les g_i et h_j sont des fonctions linéaires de $\overrightarrow{x}$. De plus, si F est au plus quadratique, ce problème se nomme un « programme quadratique ». Quand F est linéaire, on tombe sur un problème de « programmation linéaire » qui est un problème lui-aussi très connu et très étudié. Un autre cas simple très étudié est celui où il n'y a pas de contraintes.

Pour un problème d'optimisation quadratique avec contraintes, Lagrange a proposé une méthode qui est présentée et mise en application dans le chapitre sur les machines à vecteurs supports (*cf.* chap. 8).

Index

Bibliographie

P. Baldi, X. Frasconi, and P. Smyth. *Modeling the internet and the web*. Wiley, 2003. Non cité

P. Berkhin. survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002. URL `http://citeseer.ist.psu.edu/berkhin02survey.html`. Non cité

C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995. Non cité

M. Bolla. Relations between spectral and classificaiton properties of multigraphs. Technical Report 91-27, DIMACS, Rutgers University, NJ, USA, 1991. Non cité

J-M. Bouroche and G. Saporta. *L'analyse des données*. Que sais-je ? 1854. Presses Universitaires de France, 1980. Non cité

L. Breiman. Random forests. *Machine Learning*, 2001. URL `http://www.stat.berkeley.edu/users/breiman/RandomForests`. Non cité

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. Non cité

L. Candillier. La classification non supervisée. Technical report, GRAPPA, Université de Lille 3, Villeneuve d'Ascq, France, 2004. URL `http://www.grappa.univ-lille3.fr/~candillier/rapports/prstClustering.ps`. Non cité

J. Cendrowska. Prism : An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4) :349–370, 1987. Non cité

P. Cheeseman and J. Stutz. Bayesian classification (autoclass) : theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. MIT Press, 1996. URL `http://ic.arc.nasa.gov/projects/bayes-group/images/kdd-95.ps`. Non cité

W. W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995. URL `http://citeseer.nj.nec.com/cohen95fast.html`. Non cité

W.W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Proc. of AAAI conference*, pages 335–342, 1999. URL `http://www-2.cs.cmu.edu/~wcohen/slipper/`. Non cité

T.M. Cover. Geometrical and statistical properties of systems of linear inqequalities with application in pattern recognition. *IEEE Transactions on Electronic Computers*, 14 :326–334, 1965. Non cité

P. Demartines and J. Hérault. Curvilinear component analysis : A self-organizing neural network for non linear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1) :148–154, 1997. Non cité

F. Denis and R. Gilleron. Apprentissage à partir d'exemples, 2000. URL `http://www.grappa.univ-lille3.fr`. Non cité

T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees : bagging, boosting, randomization. *Machine Learning*, 40(2), 1999. URL `http://citeseer.ist.psu.edu/dietterich98experimental.html`. Non cité

R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973. Non cité

Th. Foucart. *L'analyse des données*. Presses Universitaires de Rennes, 1997. ISBN : 2-86847-260-5. Non cité

R. Fourer. Nonlinear programming frequently asked questions, 2004. URL `www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html`. Non cité

B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9), 1994. Non cité

J. Gibb. Back propagation family album. Technical Report C/TR96-05, Macquarie University, Dpt of Computing, 1996. Non cité

R. Gilleron and M. Tommasi. Découverte de connaissances à partir de données, 2000. URL `http://www.grappa.univ-lille3.fr`. Non cité

F. Girosi and T. Poggio. Networks and the best approximation property. Technical Report AI Memo 1164, MIT AI Lab, October 1989. URL `http://citeseer.ist.psu.edu/52821.html`. Non cité

J. Han and M. Kamber. *Data mining, Concepts and techniques*. Morgan-Kaufmann, 2001. Non cité

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM Int'l Conf. on the Management of Data (SIGMOD)*, 2000. Non cité

E. Hartman, J. Keeler, and J. Kowalski. layered neural network with gaussian hidden units as universal approximators. *neural computation*, 2(2) :210–215, 1990. Non cité

R. Hilborn and M. Mangel. *the ecologist detective — confronting models with data*. Princeton University Press, 1997. Non cité

G. Hinton and S. Roweis. Stochastic neighbor embedding. In *Proc. NIPS*, 200. Non cité

T.K. Ho. The random subspace method for constructing decision forests. *IEEE PAMI*, 20(8) :832–844, 1998. URL `http://citeseer.ist.psu.edu/ho98random.html`. Non cité

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 :251–257, 1991. Non cité

Information Technology Service. Handling missing or incomplete data, 2004. URL `http://www.utexas.edu/its/rc/answers/general/gen25.html`. Non cité

T. Joachims. A probabilistic analysis of the rocchio algorithm with TD.IDF for text categorization. Technical Report CMU-CS-96-118, Carnegie-Mellon Institute, 1996. Non cité

T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, and A. Saarela. Self-organization of a massive document collection. *IEEE Transactions on neural networks*, 11(3), 2000. Non cité

K. Lagus, T. Honkela, S. Kaski, and T. Kohonen. WEBSOM for textual data mining. *Artificial Intelligence review*, 13 :345–364, 1999. Non cité

Y. Le Cun, L. Bottou, G.B. Orr, and K-R. Müller. Efficient backprop. In G.B. Orr and K-R. Müller, editors, *Neural networks : tricks of the trade*. springer, 1998. URL `http://citeseer.nj.nec.com/lecun98efficient.html`. Non cité

L. Lebart, A. Morineau, and M. Piron. *Statistique exploratoire multidimensionnelle*. Dunod, 1997. ISBN : 2-10-004001-4. Non cité

G.J. MacLachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley, 1997. Non cité

J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1) :7–27, 2001. URL `http://citeseer.ist.psu.edu/malik01contour.html`. Non cité

M. Meilă. Comparing clusterings. Technical Report 419, University of Washington, Seattle, WA, USA, 2002. URL `http://www.stat.washington.edu/www/research/reports/2002/tr418.ps`. Non cité

T. Mitchell. *Machine Learning*. Mc Graw-Hill, 1997. 0-07-115467-1. Non cité

H. Motulsky and A. Christopoulos. *Fitting models to biological data using linear and nonlinear regression — A practical guide to curve fitting*. GraphPad Software Inc., 2003. URL `http://www.graphpad.com`. Non cité

A. Ng, M. Jordan, and Y. Weiss. on spectral clustering : Analysis and an algorithm. In *Proc. NIPS*. MIT Press, 2002. Non cité

K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. of Information and Knowledge Management*, pages 86–93, 2000. URL `http://citeseer.ist.psu.edu/699622.html`. Non cité

P. Perona and W.T. Freeman. A factorization approach to grouping. In *Proc. of the European Conference on Computer Vision*, volume 1406 of *Lecture Notes in Computer Science*, page 650 :670. Springer-Verlag, 1998. URL `http://citeseer.ist.psu.edu/perona98factorization.html`. Non cité

C. Peterson, T. Rögnvaldsson, and L. Lönnblad. Jetnet. Technical Report CERN-TH.7135/94, CERN, 1993. URL `http://www-dapnia.cea.fr/Spp/Experiences/OPAL/opalcern/nnbegin.html`. Non cité

W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *numerical recipes in C*. Cambridge University Press, 1988. URL `http://www.nr.com`. Non cité

R. Quinlan. *C4.5 : programs for machine learning*. Morgan-Kaufmann, 1993. Non cité

W.S. Sarle. Prediction with missing inputs, 1998. URL `http://ftp.sas.com/pub/neural/missing.ps`. Non cité

W.S. Sarle. Neural network FAQ, 1997a. URL `ftp://ftp.sas.com/pub/neural/FAQ.html`. Non cité

W.S. Sarle. Measurement theory : Frequently asked questions, 1997b. URL `http://ftp.sas.com/pub/neural/measurement.html`. Non cité

L.K. Saul and S.T. Roweis. Think globally, fit locally : unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4 : 119–155, 2003. URL `http://www.jmlr.org`. Non cité

R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2) : 197–227, 1989. Non cité

C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 1948. URL `http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf`. Non cité

J.R. Shewchuk. an introduction to the conjugate gradient method without the agonizing pain, 1994. URL `http://www-2.cs.cmu.edu/~jrs/jrspapers.html`. Non cité

D.A. Spielman and S-H. Teng. Spectral partitioning works : planar graphs and finite element meshes. In *IEEE Symposium on Foundations of Computer Science*, pages 96–105, 1996. URL `http://citeseer.ist.psu.edu/spielman96spectral.html`. Non cité

J.B. Tenebaum, V. de Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290 :2319–2323, 2000. Non cité

D. Ullman. Data mining lecture notes, 2000. URL `http://www-db.stanford.edu/~ullman/mining/mining.html`. Non cité

G.I. Webb. Efficient search for association rules. In *Knowledge Discovery and Data Mining*, pages 99–107, 2000. URL `http://citeseer.nj.nec.com/webb00efficient.html`. Non cité

websom@websom.hut.fi. Websom web site, 1999. URL `http://websom.hut.fi/websom`. Non cité

Y. Weiss. segmentation using eigenvectors : a unifying view. In *Proc. of the International Conference on Computer Vision*, 1999. URL `http://www.cs.huji.ac.il/~yweiss/iccv99.pdf`. Non cité

I. Witten and E. Franck. *Data mining, Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan-Kaufmann, 2000. Non cité

Y. Yang. an evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1/2) :67–88, 1999. URL `http://www-2.cs.cmu.edu/~yiming/publications.html`. Non cité