RCP216

Version 1.0

Michel Crucianu et Raphaël Fournier-S'niehotta

22 mars 2017

Table des matières

CHAPITRE 1

Cours - Introduction

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/1introduction.pdf)]

La fouille de données peut être définie comme le « Processus d'extraction non triviale d'informations implicites, inconnues auparavant et potentiellement utiles (sous forme de règles, contraintes, régularités) à partir de données issues de bases de données » (Gregory Piatetsky-Shapiro). Si ce domaine est loin d'être nouveau, c'est seulement depuis quelques années que les praticiens se confrontent à de nouvelles difficultés liées à une augmentation significative du volume de données. Cette augmentation a été dans certains cas bien plus rapide que la croissance continue des capacités de calcul et de stockage des serveurs individuels, les volumes résultants étant alors incompatibles avec un traitement centralisé. On parle alors en général de « données massives » (*big data*).

Mais le volume n'est pas le seul défi (relativement) nouveau pour la fouille de données. Dans un rapport de 2001 (http ://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf) le META Group (actuellement Gartner) parlait de 3V : volume, variété, vélocité. En effet, les données massives ne sont plus simplement relationnelles mais se présentent souvent dans une grande diversité (variété) de formats semi-structurés (XML, JSON) ou non structurés (par ex. textes, images, vidéo), qui nécessitent la mise au point de méthodes de fouille (ou au moins de prétraitement) adaptées. Vous pouvez regarder les dernières enquêtes en date sur le site de KDnuggets (http ://www.kdnuggets.com/polls/index.html) (enquêtes à prendre avec des précautions car rien ne permet d'évaluer leur représentativité). La vélocité correspond au fait que dans de nombreux cas de nouvelles données arrivent sous forme de flux et doivent être traitées en temps réel (ou dans des délais cohérents avec les flux), ce qui impose de nouvelles exigences sur la latence des opérations de fouille. Enfin, différents intervenants dans ce domaine ajoutent volontiers d'autres V : véracité, valeur, visibilité, etc. Il faut mentionner aussi une autre caractéristique à ne pas négliger, la « faible densité en information », qui limite fortement l'utilité d'un travail réalisé sur un échantillon de taille limitée ou sur un nombre réduit de variables.

Les institutions et les entreprises qui possèdent des données (ou qui peuvent en récolter) concernant de près ou de loin leurs activités ne prennent conscience que très progressivement de la valeur que ces données peuvent avoir pour leurs activités. Au-delà des connaissances théoriques et pratiques concernant la collecte, la gestion et la fouille de données massives, il est très important de comprendre ce que peuvent apporter les données qu'on possède (et quelles autres données seraient potentiellement utiles). Les *success stories* (ou les éventuels échecs ou semi-échecs, lorsqu'ils ne sont pas passés sous silence) apportent des enseignements utiles, si on garde un regard critique pour éviter les pièges de la communication institutionnelle. Voir par exemple la note Analyse des big data. Quels usages, quels défis ? (http ://www.strategie.gouv.fr/publications/analyse-big-data-usages-defis).

Sujet du cours

Nous nous intéresserons dans ce cours aux trois défis mentionnés, le volume, la variété et la vélocité, en insistant toutefois sur le premier. Nous verrons quelles sont les approches actuelles pour faire *passer à l'échelle* la fouille de données et étudierons plus longuement des opérations de fouille en environnement distribué. Nous aborderons dans ce contexte certains problèmes fréquents dans la fouille de données massives. Enfin, nous examinerons le rôle de la visualisation et de l'interaction, non seulement dans la présentation des résultats mais aussi lors des opérations de fouille de données.

Nous considérons que les connaissances statistiques de base sont déjà acquises, que vous avez déjà suivi un enseignement sur les bases de données documentaires et distribuées, et que vous connaissez déjà des méthodes de fouille de données. Dans le cadre du certificat de spécialisation Analyste de données massives (http ://formation.cnam.fr/par-region/certificat-de-specialisation-analyste-de-donnees-massives-669531.kjsp), vous devez avoir suivi d'abord le cours Bases de données documentaires et distribuées (NFE204) (http ://www.bdpedia.fr/bases-documents/) et le cours Entreposage et fouille de données (STA211) (http ://maths.cnam.fr/spip.php ?article228).

Contenu et objectifs de l'enseignement

Dans la première partie du **cours** (2/3 du volume horaire) nous examinerons d'abord les principales approches permettant de faire de la fouille sur de très grands volumes de données. La première consiste à réduire le volume de données à traiter (et donc aussi de calculs à faire). On retrouve ici des méthodes classiques qui travaillent sur un échantillon des données et/ou qui réduisent le nombre de variables (la dimension des données). Nous examinerons les techniques d'échantillonnage et rappellerons des méthodes d'analyse factorielle des données (analyse en composantes principales, analyse factorielle discriminante, analyse des correspondances multiples).

Dans cette première approche on trouve également des méthodes (moins populaires dans le domaine de la fouille de données) qui cherchent à réduire l'*ordre* de complexité des calculs à faire en évitant les calculs à *faible potentiel* grâce, en général, à l'utilisation de structures d'index multidimensionnels ou métriques. Nous nous intéresserons ici principalement au hachage sensible à la similarité (*Locality Sensitive Hashing*, LSH) et à son utilisation dans des opérations de recherche par similarité ou de jointure par similarité.

La réduction de volume peut éventuellement permettre de faire le travail de fouille de données sur des plateformes matérielles standard, peu coûteuses. Lorsque cette première approche s'avère insuffisante ou ne peut pas être appliquée en raison de la « faible densité en information », la seconde approche, qui consiste à employer des plateformes parallèles, s'impose. Afin de maximiser le rapport entre la capacité de traitement et les coûts d'exploitation, la solution préférée aujourd'hui est d'utiliser des systèmes composés d'un nombre élevé d'ordinateurs standard, bon marché. Pour faciliter la mise en œuvre des méthodes de fouille de données sur ce type de plateforme, ainsi que pour maximiser la disponibilité malgré les pannes inhérentes d'ordinateurs individuels, un mécanisme d'exécution simple et relativement générique a été proposé, *MapReduce*. Les auditeurs ayant suivi le cours Bases de données documentaires et distribuées (NFE204) (http ://www.bdpedia.fr/bases-documents/) connaissent déjà ce mécanisme. Nous verrons comment écrire certains algorithmes de fouille de données en utilisant les primitives de *MapReduce*.

Dans la suite du cours nous nous focaliserons sur les particularités des opérations de fouille dans un contexte de données massives. Nous examinerons d'abord les méthodes employées dans les systèmes de recommandation, de plus en plus utilisés dans le marketing et surtout dans la vente en ligne. Nous regarderons ensuite la mise en œuvre de la classification automatique lorsque les volumes de données à traiter sont importants. Dans le développement et l'application de modèles décisionnels sur des données massives nous considérerons le cas des machines à vecteurs de support (*Support Vector Machines*, SVM) linéaires qui ne sont pas abordées dans le cours Entreposage et fouille de données (STA211) (http ://maths.cnam.fr/spip.php ?article228). Des informations intéressantes se trouvent souvent dans des données textuelles (messages, tweets, avis, blogs, etc.) et dans des réseaux sociaux, la fouille de textes et la fouille de réseaux sociaux sont donc des sujets très importants. Nous passerons en revue les objectifs de fouille possibles et les opérations associées. La vélocité étant une des caractéristiques clés des données massives, nous examinerons des opérations spécifiques au traitement de données qui arrivent en flux et doivent être traitées en temps réel.

La seconde partie du cours (1/3 du volume horaire) est une présentation synthétique mais concrète du domaine de la visualisation de données. La visualisation est importante à la fois pour faire comprendre les résultats et pour guider de façon interactive les opérations de fouille de données massives. Seront exposées à la fois des questions plus théoriques indispensables concernant notamment la perception humaine et des techniques de représentation (graphes, hiérarchies, lignes de temps) et d'interaction (association focus/contexte, distorsion, filtrage), avec leurs spécificités et domaines d'intérêt.

Les **travaux pratiques** (TP) sont une mise en œuvre directe de notions et méthodes vues en cours. Les TP qui concernent la fouille de données emploient le *framework open source* Spark (http ://spark.apache.org/) et le langage de programmation Scala. Une connaissance préalable du langage Scala n'est toutefois pas nécessaire, certaines notions indispensables seront vues lors des TP. Il est également possible d'utiliser les librairies de Spark

à partir de Java et de Python si un de ces langages vous est plus familier (noter toutefois que Java est bien plus « verbeux » que Scala ou Python). Enfin, SparkR devrait permettre d'utiliser l'environnement Spark à partir de code écrit en R (à partir de Spark 1.5). Les TP de visualisation et interaction seront réalisés avec Processing (http ://processing.org/), dans un langage qui peut être vu comme une simplification de Java (ou alors directement en Java).

Lors des séances de travaux pratiques, après une introduction à Spark et à Scala, vous apprendrez à manipuler des données numériques, à utiliser l'échantillonnage, à faire une analyse en composantes principales et une classification automatique, à travailler sur des données textuelles et sur des flux de données, à mettre au point un modèle décisionnel (SVM linéaire), ainsi qu'à faire des opérations de fouille sur des graphes (issus éventuellement de réseaux sociaux). Ensuite, après une introduction à Processing, vous apprendrez à faire des cartographies, des petits multiples (*small multiple* de Tufte), des *treemaps* et des nuages de mots.

Les objectifs de cet enseignement sont :

— pour la fouille de données massives, vous faire comprendre les approches de passage à l'échelle, vous familiariser avec un logiciel permettant l'exécution distribuée efficace de méthodes de fouille de données, vous transmettre des méthodes de résolution de problèmes courants de la fouille de données et vous faire appliquer ces méthodes sur des données réelles ;
— pour la visualisation et l'interaction, vous faire comprendre les principes et les enjeux, vous transmettre des techniques usuelles ainsi que leurs utilisations et vous familiariser avec la programmation de ces techniques.

Après avoir suivi cet enseignement, les auditeurs doivent avoir la capacité à mettre œuvre, sur des données massives, des techniques de fouille de données et de visualisation interactive.

Organisation de l'enseignement

Chaque séance de cours est suivie par une séance de travaux pratiques (TP). La séance de TP a lieu immédiatement après le cours pour un des groupes de TP et (lorsque cela est possible) le lendemain soir pour l'autre groupe de TP. Il est envisageable de changer de groupe d'une semaine sur l'autre, en fonction de vos disponibilités. Il est en revanche nécessaire d'avoir des groupes assez équilibrés pour pouvoir travailler seul sur un ordinateur et pour que l'enseignant arrive à répondre rapidement à toutes vos questions. Il est possible d'apporter votre ordinateur portable pour travailler avec, l'enseignant pourra éventuellement vous assister avec l'installation de Spark lors de la première ou deuxième séance de TP.

Les horaires des cours et travaux pratiques, ainsi que les salles prévues, sont consultables sur http://emploi-du-temps.cnam.fr/emploidutemps2?FD=1 (choisir Saisie du code, entrer CERCP216 et choisir ensuite le semestre 1 ou 2). Attention, les salles peuvent parfois changer sans préavis, mieux vaut vérifier juste avant la séance. Il est également possible d'installer (sous Android seulement, pour l'instant) l'application Planni Cnam (https ://play.google.com/store/apps/details ?id=cnam.paris.plannicnam).

Les supports de cours sont disponibles ici ou à partir de http://cedric.cnam.fr/vertigo/Cours/RCP216/. Les transparents de tout le cours sont déjà en ligne, ainsi que les explications détaillées en HTML pour la partie fouille de données du cours. Pour la partie visualisation de ce cours, des vidéos avec toutes les explications seront accessibles à partir de http://lecnam.net pour les auditeurs du Cnam inscrits en formation à distance (FOD) à l'UE RCP216.

Pour les travaux pratiques, le contenu détaillé en HTML est déjà en ligne à partir de l'index situé en haut à droite de cette page ou de http://cedric.cnam.fr/vertigo/Cours/RCP216/. Les corrections et réponses aux questions sont mises en ligne après la dernière séance de la semaine (et retirées après l'examen).

Tous les supports de cours et de travaux pratiques mis en ligne peuvent évoluer à tout moment.

Dans les travaux pratiques et dans la réalisation des projets d'UE vous pouvez rencontrer des problèmes très divers, allant des fautes de frappe (!) à des anomalies de configuration de logiciels. Pour les résoudre vous avez la possibilité de vous adresser aux enseignants, mais ils ne sont pas toujours disponibles, ne peuvent pas répondre rapidement et peuvent avoir des difficultés à régler le problème à distance. Vous rencontrerez en général des problèmes auxquels d'autres se sont confrontés et vous trouverez les solutions sur des forums en ligne à l'aide d'un moteur de recherche. Dans d'autres cas vous avez aussi la possibilité d'utiliser le forum de l'UE http://deptmedia.cnam.fr/phpBB2/viewforum.php?f=50. L'inscription à ce forum est nécessaire pour y poster. Si vous avez une question urgente pour un enseignant, mieux vaut la lui envoyer par courriel directement. L'adresse de courriel est en général prénom.nom@cnam.fr.

Les enseignants qui interviendront lors des séances de cours et/ou de travaux pratiques sont : Michel Crucianu, Raphaël Fournier-S'niehotta, Marin Ferecatu, Pierre Cubaud.

L'évaluation est faite à travers un examen terminal sur table (pas sur ordinateur) et un projet d'UE. La note finale sera la moyenne entre la note obtenue à l'examen et la note de projet. Pour le premier semestre l'examen est en février avec rattrapage en avril, pour le second semestre l'examen est en juin avec rattrapage en septembre. La planification des examens doit être consultée sur http://www.cnam-paris.fr/suivre-ma-scolarite/examens-/ ou sur http://emploi-du-temps.cnam.fr/emploidutemps2?FD=1. Les enseignants ont ces mêmes sources d'information. Si la date de l'examen n'apparaît pas encore cela veut dire que la scolarité du Cnam ne l'a pas encore fixée.

Pendant l'examen vous aurez le droit de consulter vos propres documents écrits mais pas sur supports informatiques (ordinateur portable, tablette, smartphone...) car ils sont équipés de moyens de communication qui ne sont pas autorisés pendant les examens. Mieux vaut connaître (et avoir compris) le contenu du cours et se servir des documents écrits seulement pour se rappeler des détails précis ; découvrir le contenu du cours pendant l'examen, grâce aux documents écrits, n'est pas une bonne approche.

Quelques références bibliographiques

De nombreuses références, sur supports classiques ou électroniques, traitent les différents sujets abordés dans cet enseignement. Vous aurez des références spécifiques dans les différentes séances de cours ou de TP. Nous avons listé ci-dessous quelques références qui couvrent des parties plus larges du contenu de cet enseignement.

Sur la fouille de données massives :

J. Leskovec, A. Rajaraman, J. Ullman. Mining of Massive Datasets. Cambridge University Press. (http ://www.mmds.org)

Karau, H., A. Konwinski, P. Wendell et M. Zaharia. *Learning Spark - Lightning-Fast Big Data Analysis*. O'Reilly, 2015.

Ryza, S., U. Laserson, S. Owen and J. Wills. *Advanced Analytics with Spark - Patterns for Learning from Data at Scale*. O'Reilly, 2015.

http://www.kdnuggets.com/2015/09/free-data-science-books.html

Sur la visualisation de données et l'interaction :

Fry, B. *Visualizing Data*. O'Reilly. 2008.

Spence, R. *Information Visualization : Design for Interaction*. Prentice Hall. 2007.

Passage à l'échelle de la fouille de données

Dans le cadre de ce cours, par « passage à l'échelle » (*scalability*) nous entendrons la capacité à faire face à une augmentation forte du volume de données à traiter. Il faut remarquer que la notion de « passage à l'échelle » est plus large et, même dans un contexte de fouille de données massives, peut concerner d'autres aspects : capacité d'une méthode de fouille à traiter des données décrites par un très grand nombre de variables, capacité d'un outil à gérer ou à traiter des données malgré leur hétérogénéité, etc.

La **première approche** pour le passage à l'échelle consiste à réduire fortement le volume de calculs à faire. Une première méthode est la réduction (forte) du volume de données, en travaillant sur un échantillon et/ou sur un nombre bien plus faible de variables. Si les données disponibles présentent une « faible densité en information » alors cette méthode ne produira pas de bons résultats : si l'échantillon est trop petit, les « régularités » recherchées par la méthode de fouille ne se manifesteront pas suffisamment pour être détectables ; si le nombre de variables est trop faible, les « régularités » trouvées seront incomplètes et les capacités prédictives seront insuffisantes.

Une seconde famille de méthodes suivant cette approche de réduction travaille sur toutes les données (et toutes les variables) mais en exploitant leurs caractéristiques de similarité afin de diminuer l'ordre de complexité des calculs à faire. En effet, les modèles décisionnels sont souvent « locaux » : la décision pour une nouvelle donnée dépend exclusivement ou principalement des données étiquetées proches. Grâce à la construction préalable d'index il est parfois possible de réduire ainsi l'ordre de complexité des calculs à faire, par exemple $O(N) \rightarrow O(\log(N))$ ou

$O(cst.)$, $O(N^2) \rightarrow O(N \log(N))$, N étant le nombre total de données. Ces méthodes peuvent être exactes (fournir les mêmes résultats qu'un calcul exhaustif) ou, plus souvent, approximatives. En revanche, ces méthodes deviennent peu efficaces (réduction insuffisante de la complexité et même augmentation du coût) lorsque le nombre de variables décrivant les données est élevé.

La **seconde approche** pour le passage à l'échelle consiste à employer des plateformes de calcul parallèle. Pendant longtemps, les calculs parallèles étaient réalises sur des architectures parallèles dédiées. Non seulement ces architectures sont très coûteuses à réaliser (et à concevoir), mais elles présentent en général des points névralgiques et des « goulots d'étranglement », surtout au niveau des échanges entre les unités de calcul et le stockage de masse. Ces architectures sont aujourd'hui employées dans des secteurs spécifiques. Lorsque les volumes de données à traiter sont comparativement faibles mais les calculs à réaliser sont très lourds (par exemple, apprentissage d'un réseau de neurones profond), des solutions à base de processeurs graphiques surdimensionnés sont également disponibles.

Afin de maximiser le rapport entre la capacité de traitement et les coûts d'utilisation, la solution préférée aujourd'hui est de distribuer données et calculs sur un nombre élevé d'ordinateurs standard, bon marché, chacun avec son propre stockage de masse, reliés par un réseau local haut débit standard. La latence associée au transfert de données entre différents ordinateurs, ou entre ordinateurs et un stockage externe, incite à conserver les données sur les unités de stockage des ordinateurs individuels. Les données à traiter étant en général bien plus volumineuses que les programmes à exécuter, il est préférable de laisser le plus longtemps possible les données sur les unités de stockage locales des ordinateurs individuels et de transmettre à ces ordinateurs les différents programmes à exécuter sur les mêmes données. Les traitements à appliquer aux données seront ainsi partitionnés en tâches à exécuter par chaque ordinateur sur les données **locales**.

L'emploi d'un nombre (très) élevé d'ordinateurs standard augmente, naturellement, la probabilité de panne d'au moins un des ordinateurs participant à un traitement. Comment faire en sorte que la panne qui se produira de façon inévitable sur un (ou plusieurs) ordinateur(s) n'engendre pas une perte de données ? Un mécanisme de réplication permet de stocker chaque donnée sur plusieurs ordinateurs afin de réduire le risque de perte malgré des pannes individuelles. Comment éviter, en cas de panne sur un (ou plusieurs) ordinateur(s), l'interruption des calculs en cours dans le système distribué, avec obligation de reprise depuis le début (ou le dernier point de validation) ? En partitionnant les calculs en « grains » traçables et suffisamment fins pour qu'en cas de panne les grains abandonnés puissent être efficacement réaffectés aux ordinateurs encore actifs.

MapReduce et la fouille de données

Bien entendu, il ne faut pas laisser au programmeur applicatif le soin de mettre en œuvre les mécanismes de réplication, ni de gérer la traçabilité des grains et leur réaffectation. Ces tâches sont de la responsabilité du *framework* utilisé. Le programmeur applicatif a néanmoins la tâche importante (et parfois difficile) de reformuler les algorithmes qu'il souhaite mettre en œuvre sur la plateforme distribuée suivant le mécanisme d'exécution implémenté par le *framework*. Un mécanisme d'exécution très populaire, présent dans plusieurs *frameworks* (dont le *framework open source* Hadoop), est *MapReduce*. Nous l'examinerons très brièvement ici et vous êtes incités à (re)voir les explications particulièrement bien illustrées de MapReduce, premiers pas (http ://b3d.bdpedia.fr/mapreduce.html) (dans l'unité d'enseignement NFE204 du Cnam).

MapReduce propose de décomposer l'ensemble des opérations à réaliser en deux types de tâches élémentaires et uniformes, les *Map* et les *Reduce*. Chaque donnée passe d'abord par une tâche *Map* et ensuite, transformée par celle-ci, éventuellement par une tâche *Reduce*. En revanche, aucun ordre d'exécution particulier n'est attendu entre différentes tâches *Map* ou entre différentes tâches *Reduce*. Une ou plusieurs tâches *Map* et/ou *Reduce* peuvent être facilement assignées à chaque nœud de calcul. Par nœud de calcul nous entendons ici soit un ordinateur individuel, soit un cœur d'une unité centrale multi-cœur (dans ce dernier cas, la mémoire vive et le stockage de masse de l'ordinateur sont partagés entre les cœurs).

Le fonctionnement général de *MapReduce* est le suivant (voir aussi la figure *MapReduce* : exécution d'un programme) :

1. L'ensemble de données à traiter est découpé en fragments (*chunks*).
2. Chaque tâche *Map* est assignée à un nœud de calcul qui reçoit un ou plusieurs fragments que la tâche *Map* transforme en une séquence de paires [clé, valeur].
3. Chaque tâche *Reduce* est associée à une ou plusieurs clés et est assignée à un nœud de calcul.

4. Les paires (clé, valeur) produites par les *Map* sont groupées par clés et stockées sur les nœuds de calcul qui exécuteront les tâches *Reduce* respectives (étape *shuffle*).
5. Chaque tâche *Reduce* combine, pour chaque clé qui lui est associée, les valeurs des paires [clé, valeur] avec cette clé ; les résultats sont stockés et constituent le résultat du traitement.

Le programmeur écrit les fonctions *Map* et *Reduce*, le *framework* se charge de tout le reste.

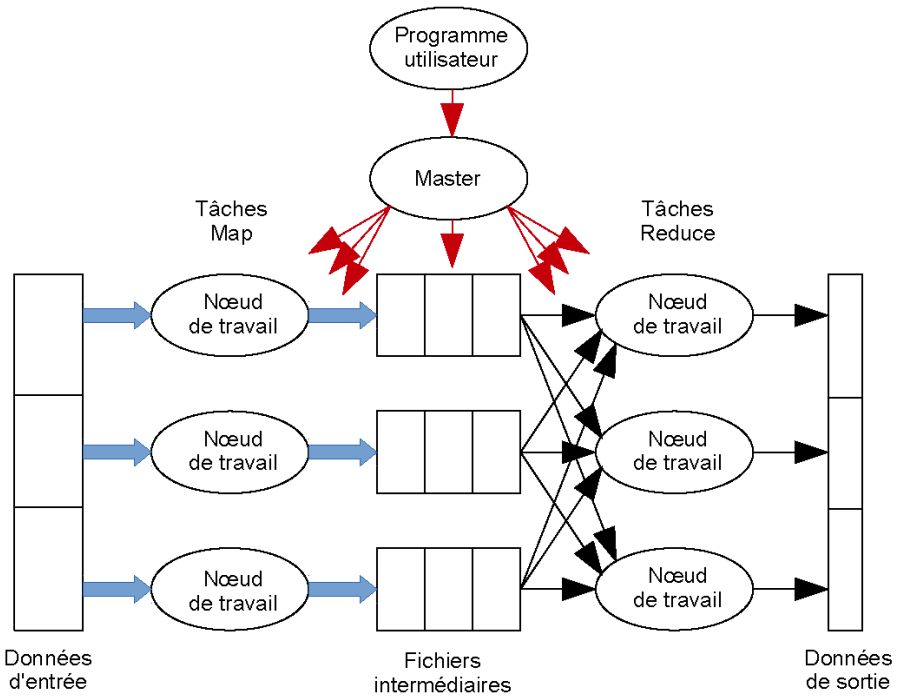


Fig. 1.1 – *MapReduce* : exécution d'un programme

On peut se demander si un mécanisme aussi simple est adapté à une classe suffisamment large d'algorithmes. Pour mieux illustrer l'écriture d'un algorithme en utilisant le modèle d'exécution MapReduce nous considérerons deux exemples classiques. Nous en verrons d'autres plus tard dans ce cours. Le premier exemple classique concerne le comptage du nombre d'occurrences de chaque mot dans une (grande) collection de documents textuels.

Chaque tâche *Map* est affectée à un nœud de calcul qui travaillera sur son fragment local de données (constitué d'un ou plusieurs documents textuels). Dans le cadre de la tâche *Map*, chaque nœud de calcul génère une paire [mot, 1] pour chaque occurrence d'un mot dans son fragment local. Les paires [mot, 1] sont ensuite groupées par mots et stockées (étape *shuffle*). Enfin, chaque tâche *Reduce* reçoit les paires correspondant à un mot, fait l'addition des valeurs qui lui sont associées dans toutes ces paires et stocke les résultats. Ce processus est également décrit dans l'algorithme suivant :

1. On considère, pour simplicité, que chaque document constitue un *fragment*.
2. Chaque tâche *Map* est assignée à un nœud de calcul qui reçoit un fragment, le découpe en mots et, pour chaque occurrence d'un mot, génère une paire [mot, 1] (la clé est ici le mot lui même, la valeur est égale à 1).
3. Chaque tâche *Reduce* est associée à une ou plusieurs clés (mots) et est assignée à un nœud de calcul.
4. Les paires [mot, 1] produites par les *Map* sont groupées par mots et stockées sur les nœuds de calcul qui exécuteront les tâches *Reduce* respectives (étape *shuffle*).
5. Chaque tâche *Reduce* reçoit les paires [mot, 1] correspondant à un mot, fait l'addition des valeurs 1 associées au mot et stocke le résultat.

Remarque : l'addition étant associative et commutative, il est possible de transférer dans les *Map* une partie des calculs des *Reduce* ; ainsi, une tâche *Map* peut en plus additionner le nombre d'occurrences de chaque mot dans le fragment qu'elle reçoit et générer plutôt des paires (mot, nombre d'occurrences du mot dans le fragment). Cela permet de minimiser les échanges de données car, au lieu de produire autant de paires [mot, 1] qu'il y a d'occurrences de ce mot dans le fragment, chaque tâche *Map* ne produira qu'une seule paire (mot, nombre d'occurrences du mot dans le fragment).

Le second exemple concerne la multiplication (grande) matrice x vecteur : $\mathbf{y} = \mathbf{M} \times \mathbf{x}$ (ou $y_i = \sum_{j=1}^{n} m_{ij} x_j$)

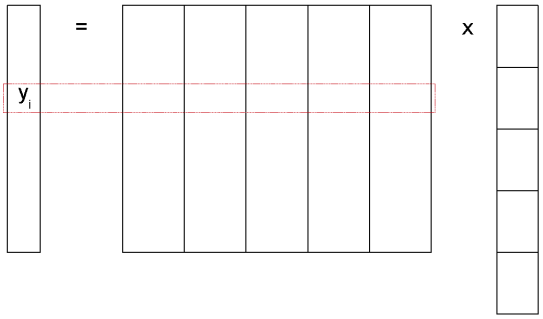


Fig. 1.2 – Multiplication matrice x vecteur

1. La (très grande) matrice $\mathbf{M}$ est découpée en fragments qui sont des groupes de colonnes ; un fragment doit tenir dans la mémoire d'un nœud de calcul. Le vecteur $\mathbf{x}$ est aussi découpé en groupes de lignes de façon correspondante.
2. Chaque tâche *Map* est assignée à un nœud de calcul qui reçoit un fragment de la matrice $\mathbf{M}$ et le fragment correspondant de $\mathbf{x}$; pour chaque élément m_{ij} de son fragment, calcule $m_{ij}x_j$ et génère $(i, m_{ij}x_j)$.
3. Chaque tâche *Reduce* est associée à une ou plusieurs valeurs de i et est assignée à un nœud de calcul.
4. Les paires $(i, m_{ij}x_j)$ sont groupées par i et stockées sur les nœuds de calcul qui exécuteront les tâches *Reduce* respectives (étape *shuffle*).
5. Chaque tâche *Reduce* reçoit les paires correspondant une valeur de la clé i, fait l'addition des valeurs qui lui sont associées et stocke les résultats.

Bien qu'intéressant car il permet de tirer profit du *framework open source* Hadoop, le mécanisme d'exécution *MapReduce* est aussi assez contraignant. D'abord, même si la définition des *Map* et *Reduce* est assez générique, le niveau d'abstraction reste assez bas ; les algorithmes rencontrés dans la fouille de données sont souvent itératifs, le programmeur d'un tel algorithme doit définir les opérations des *Map* et *Reduce* dans le cadre d'une même itération, mais aussi relier entre elles les itérations successives. Ensuite, le mécanisme de reprise sur panne de *MapReduce* impose le stockage des résultats des *Reduce*, or pour un algorithme itératif ces résultats ne sont qu'intermédiaires : l'itération suivante doit prendre ces résultats comme données d'entrée. Ce stockage des résultats intermédiaires engendre un ralentissement très significatif des algorithmes.

La comparaison des débits entre les différents niveaux de la hiérarchie de stockage, voir la figure suivante, montre bien que la sauvegarde des résultats intermédiaires, à chaque itération, sur un stockage de masse (disques classiques ou SSD) ralentit inévitablement et significativement le déroulement des algorithmes itératifs par rapport à un maintien des résultats intermédiaires en mémoire.

Spark et la fouille de données

La solution de *Spark* (https://spark.apache.org) est de **conserver les résultats intermédiaires dans la mémoire vive** des nœuds de calcul et de **garder l'historique des opérations** ayant permis d'obtenir ces données. En cas de panne, l'historique permet de recalculer les données perdues (car stockées dans la mémoire vive d'un nœud

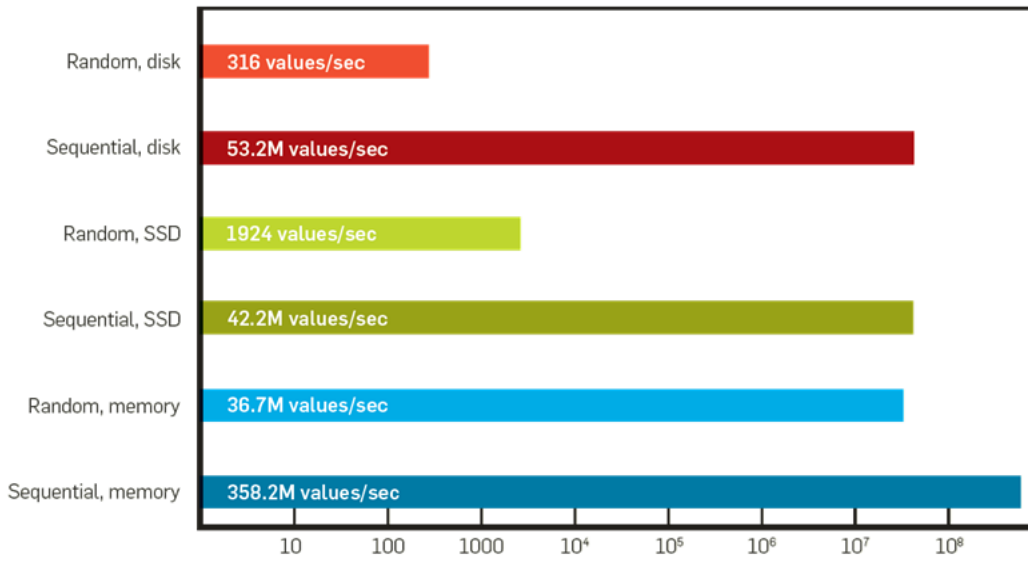


Fig. 1.3 – Hiérarchie de stockage (issue de *Communications of the ACM* 2009 (8) : 36-44)

de calcul tombé en panne) à partir des dernières données encore disponibles. L'impact de cette évolution sur les algorithmes itératifs est très significatif, une accélération de 10 fois par rapport au mécanisme *MapReduce* de départ peut être observée. A la base de ce fonctionnement on trouve les *Resilient Distributed Dataset* (RDD) qui sont des collections de données partitionnées en fragments (ou partitions, *chunks*) distribués sur les nœuds de calcul et conservés (autant que possible) en mémoire. Vous êtes incités à (re)voir aussi les explications de la séance Systèmes itératifs : Spark (http ://b3d.bdpedia.fr/calculdistr.html#s3-systemes-iteratifs-spark) de l'unité d'enseignement NFE204 du Cnam, ainsi que la première séance de travaux pratiques de notre cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/tpSparkScala.html).

Il y a aujourd'hui d'assez nombreux environnements de fouille de données utilisant le mécanisme d'exécution MapReduce, parfois amélioré (comme dans le cas de Spark) et/ou complété. Avant de revenir à Spark, qui a été choisi pour les travaux pratiques de notre enseignement, nous passons en revue les environnements les mieux connus disponibles en *open source*. Ce secteur est caractérisé par un développement rapide et des mutations sont à attendre.

— **H2O** (0xdata) : librairie assez large et en développement rapide, orientée apprentissage statistique, exécution efficace, interface avec R ; support commercial de 0xdata ;
— **MLlib / Spark** (Apache, Databricks) : librairie en développement rapide, utilisant le support efficace de Spark, interface récente avec R (SparkR) ; distribuée par Cloudera (entre autres) ;
— **Mahout** (Apache) : librairie large mais très hétérogène, latence due à un fonctionnement MapReduce classique (stockage résultats intermédiaires), sans support commercial ;
— **Vowpal Wabbit** (John Langford) : exécution efficace, inclut quelques algorithmes clé, mais sans support commercial ;
— **RHadoop** (Revolution Analytics) : pour écrire des jobs MapReduce en R ; support commercial pour la licence Enterprise ;
— **RHIPE** (Suptarshi Guha) : pour écrire des jobs MapReduce en R ; sans support commercial.

Comment choisir entre ces différentes solutions ? Nous pouvons mentionner ici quelques critères de choix :

1. L'efficacité : l'environnement est basé sur un suivi strict de *MapReduce* ou propose des solutions pour réduire la latence (comme par ex. conserver les données intermédiaires en mémoire et garder l'historique des opérations pour pouvoir les recalculer) ?
2. La facilité d'utilisation : l'environnement exige une gestion explicite ou plutôt implicite du parallélisme (par exemple, partitionnement automatique en fragments, en fonction du nombre de nœuds de calcul disponibles, et application transparente des opérations aux fragments quels qu'ils soient) ?
3. Les fonctionnalités actuelles et les perspectives de développement : certaines solutions *open source* permettent de rassembler une communauté de développeurs assez large.

4. La gestion de la distribution des tâches sur les nœuds de calcul : est-elle explicite ou implicite, intégrable avec d'autres opérations de type MapReduce et non-MapReduce ? La compatibilité avec Hadoop YARN est une garantie de grande flexibilité.
5. Quels formats de données sont acceptés de façon « native » ? HDFS, Bzip/Gzip, Apache Avro, HBase, Hive, Impala, etc. ?
6. La présence d'un support commercial : important pour certains développements industriels où une bonne maîtrise des délais de mise en œuvre est nécessaire, il est disponible pour les solutions propriétaires mais aussi pour certaines solutions *open source*.

Notre choix d'utiliser Spark pour les travaux pratiques de cet enseignement est justifié par l'efficacité de la solution proposée, la gestion globalement implicite du parallélisme, les fonctionnalités actuelles (mais aussi les perspectives de développement) et l'existence de librairies qui couvrent non seulement la fouille de données mais aussi le traitement des graphes et des flux de données, donc la totalité des sujets abordés dans la première partie du cours. Enfin, Spark est un projet très actif de la fondation Apache.

Parmi les librairies existantes de Spark, dans le cadre des travaux pratiques nous nous servirons de :
— MLlib : analyse des données, fouille de données, apprentissage statistique ;
— Spark Streaming : traitement de flux de données (par ex. Twitter, ZeroMQ, Kinesis, *sockets* TCP) ;
— GraphX : calculs sur les graphes (*data-parallel* et *graph-parallel*) ; remplace Pregel.

Spark permet d'exécuter des programmes en Scala, en Java et en Python. Un programme Spark tourne dans une machine virtuelle Java. Le code Scala peut appeler de façon **native** des librairies écrites en Java (et inversement). Depuis la version 1.5, il est possible d'exécuter du code R en bénéficiant de la plateforme Spark (avec SparkR, mais le spectre des méthodes accessibles est encore assez restreint).

Pour finir, quelques mots concernant le déploiement de Spark (voir aussi https://spark.apache.org/docs/latest/cluster-overview.html). La figure suivante présente le déploiement sur un *cluster* d'ordinateurs (ensemble d'ordinateurs interconnectés et gérés de façon unifiée).

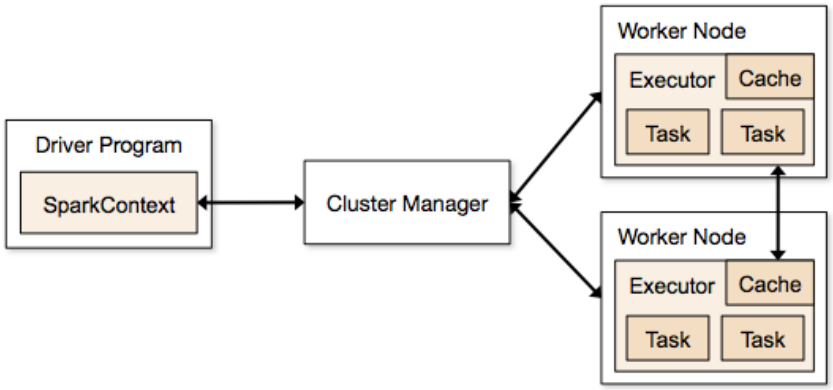


Fig. 1.4 – Déploiement sur *cluster* (figure issue de la documentation de Spark)

Le programme *driver* contrôle la logique de l'application et distribue les tâches aux nœuds de calcul (*worker nodes*). En raison des communications nécessaires entre le *driver* et les *workers*, mieux vaut que le *driver* tourne sur un des ordinateurs du *cluster*.

Le gestionnaire de *cluster* (*cluster manager*) veille à la bonne gestion des ressources du *cluster*. Comme gestionnaire de *cluster*, Spark propose le sien (mode autonome ou *standalone*, pas d'intégration avec d'autres opérations), mais permet d'utiliser Apache Mesos (intégration possible avec d'autres opérations basées sur MapReduce) ou encore Hadoop YARN (intégration possible avec d'autres opérations basées sur MapReduce ou non-MapReduce).

Plusieurs applications Spark différentes exécutées sur les mêmes nœuds de calcul sont complètement isolées entre elles car chacune tourne dans une machine virtuelle Java (JVM) spécifique et est donc isolée des autres. Ces applications peuvent communiquer entre elles uniquement à travers des fichiers. Pour des raisons d'efficacité, le *driver* devrait tourner sur un nœud du cluster.

CHAPITRE 2

Cours - Réduction du volume de données

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/2reductionVolume.pdf)]

Nous nous intéressons ici à la première approche de passage à l'échelle, qui consiste à réduire fortement le volume de calculs à réaliser. Une première méthode est la diminution forte du volume de données, en choisissant un échantillon de données et/ou un nombre réduit de variables. Les résultats obtenus ainsi sont en général des *approximations* des résultats qui seraient issus des données complètes. Une autre méthode, examinée dans la séance de cours suivante, consiste à travailler sur toutes les données (et toutes les variables) mais en exploitant leurs caractéristiques de similarité afin de diminuer *l'ordre de complexité* des calculs à réaliser. Des méthodes hybrides peuvent également être employées, comme la réduction de l'ordre de complexité *après* réduction de dimension (permettant de diminuer l'impact de la *malédiction de la dimension*, cf. cette partie (http ://cedric.cnam.fr/vertigo/Cours/RCP216/coursSimilariteRecommandation.html#malediction-de-la-dimension) d'une séance suivante). Nous verrons que chacune de ces approches peut présenter dans certains cas des inconvénients majeurs, la seule solution étant alors le recours au calcul parallèle.

Si la diminution du volume de données à traiter est assez forte, une architecture classique, centralisée, peut s'avérer suffisante. Lorsque les données disponibles présentent une « faible densité en information » alors cette approche ne produira pas de bons résultats : avec un échantillon trop petit, les « régularités » recherchées par la méthode de fouille ne se manifesteront pas suffisamment pour être détectables ; avec trop peu de variables, les « régularités » trouvées seront très incomplètes ou les capacités prédictives insuffisantes.

Calculs sur un échantillon

Nous examinons d'abord brièvement l'échantillonnage, qui vise à inférer des propriétés concernant toute la « population » de N données à partir d'un sous-ensemble (échantillon) de seulement $n \ll N$ données. Une présentation détaillée peut être trouvée par exemple dans *[Til01]* (page 131).

Une question importante abordée par la théorie de l'échantillonnage est la construction des échantillons, basée sur des méthodes non aléatoires ou des méthodes aléatoires. Dans la première catégorie on peut trouver la construction d'un échantillon par choix d'expert (une bonne connaissance du problème est supposé permettre à un expert de désigner les données ou les observations à retenir dans l'échantillon), par le volontariat (les observations concernent des individus qui se portent volontaires), etc. Ces méthodes répondent souvent à des considérations pragmatiques mais il est difficile de qualifier la représentativité de l'échantillon, donc la qualité de l'inférence qui est réalisée.

Parmi les méthodes aléatoires de construction d'échantillon nous mentionnons ici l'échantillonnage simple, l'échantillonnage stratifié et l'échantillonage en grappes (voir par ex. *[Til01]* (page 131) pour d'autres méthodes et des présentations plus détaillées).

L'échantillonnage simple consiste à faire des tirages indépendants, habituellement sans remise, chaque donnée (ou observation) ayant la même probabilité d'être sélectionnée, $p_s = \frac{n}{N}$.

L'échantillonnage stratifié considère que l'ensemble de données est constitué de sous-ensembles (ou *strates*) qui présentent une certaine *homogénéité* interne par rapport à l'étude. Un échantillonnage simple est ensuite appliqué dans chaque strate. Par rapport à l'application directe d'un échantillonnage simple à l'ensemble de la population, l'application d'un échantillonnage stratifié augmente la précision pour une même valeur de n (ou conserve la

précision avec n plus faible). Il est par ailleurs possible de moduler la représentation des différents strates dans l'échantillon en choisissant une valeur de p_s adéquate dans chaque strate.

Exemple :

Pour une étude des pratiques des clients du commerce en ligne, on considère qu'il y a une certaine homogénéité à l'intérieur de chaque tranche de revenus. Si un échantillonnage simple est appliqué directement sur la population entière, la proportion relative de chaque tranche de revenus n'est pas bien conservée dans l'échantillon ; plus la population d'une tranche de revenus est faible, plus son taux de présence dans l'échantillon s'éloignera en général de p_s. On applique alors un échantillonnage stratifié : on partitionne la population en tranches de revenus (1 tranche = 1 strate) et on applique un échantillonnage simple en imposant un même taux de sélection dans chaque tranche.

L'échantillonnage en grappes vise en général à simplifier la réalisation de l'étude. On considère que l'ensemble de données est constitué de sous-ensembles (ou « grappes ») tels que les différences intra-grappe sont plus fortes que les différences inter-grappe. On sélectionne alors au hasard des « grappes » et on examine tous les individus de chaque grappe choisie. Par rapport à un échantillonnage simple appliqué à l'ensemble de la population, l'échantillonnage en grappes facilite la réalisation de l'étude s'il est plus simple d'obtenir les données par grappes, tout en conservant la précision (si les différences sont plus fortes à l'intérieur de chaque grappe qu'entre les grappes).

Exemple :

Pour une étude des pratiques sportives des élèves de troisième en zone urbaine, on considère qu'il y a autant de diversité à l'intérieur d'un même collège qu'entre collèges. Avec un échantillonnage simple il faudrait vraisemblablement interroger quelques élèves de chaque collège de chaque zone urbaine. Un échantillonnage en grappes, avec 1 grappe = 1 collège, nous amènerait plutôt à sélectionner quelques collèges et à interroger tous les élèves de troisième de ces collèges, solution bien plus facile à mettre en œuvre.

Quelle que soit la méthode de construction d'échantillon utilisée, il est important de garder à l'esprit que la taille de l'échantillon doit être suffisante pour que les régularités supposées (qui seront recherchées) y trouvent un support satisfaisant.

Échantillonnage dans Spark

Dans Spark (http ://spark.apache.org/docs/latest/) on trouve l'échantillonnage simple et l'échantillonnage stratifié commes méthodes de base (MLlib n'est pas nécessaire) :

Échantillonnage simple, qui peut s'appliquer à tout RDD et retourne un RDD de même type : méthode `sample(withReplacement: Boolean, fraction: Double, seed: Long): RDD[T]` ; `fraction` indique la probabilité de choisir chaque donnée. La taille de l'échantillon n ne peut pas être contrôlée de façon précise, chaque donnée étant sélectionnée avec une probabilité p_s (plus N sera grand, plus n approchera en général $p_s \cdot N$).

Échantillonnage stratifié, peut s'appliquer à tout RDD de type (clé, valeur), retourne un RDD de même type :

1. `sampleByKey(withReplacement: Boolean, fractions: Map[K, Double], seed: Long): RDD[(K, V)]` : méthode de base (une seule passe sur les données) qui fait un tirage pour chaque donnée et respecte de façon *approximative* la taille d'échantillon visée ($f_k \cdot N_k$) ; `fractions` indique, pour chaque clé k, la probabilité f_k de choisir chaque donnée et N_k est le nombre de données de clé k dans le RDD initial.
2. `sampleByKeyExact(withReplacement: Boolean, fractions: Map[K, Double], seed: Long): RDD[(K, V)]` : méthode plus coûteuse (deux passes sur les données) qui retourne $\lceil f_k \cdot N_k \rceil$ données pour chaque clé k (mêmes notations que pour `sampleByKey()` ci-dessus).

Réduction de dimension

Considérons N données (observations) définies dans $\mathbb{R}^m$. Une réduction de dimension consiste à obtenir une représentation des N données dans $\mathbb{R}^k$, avec $k \ll m$. Cette diminution du nombre de variables décrivant les données peut avoir plusieurs objectifs :

1. Réduire le volume de données à traiter, tout en conservant au mieux « l'information utile ». Il est nécessaire de définir d'abord ce qu'est information utile.
2. Améliorer le rapport signal / bruit et supprimant des variables non pertinentes. Il est nécessaire de définir d'abord ce qu'est une variable non pertinente.
3. Améliorer la « lisibilité » des données en mettant en évidence des relations entre variables ou groupes de variables ou en facilitant la visualisation. Il est nécessaire de définir d'abord ce qu'il faut mettre en évidence.
4. Atténuer la « malédiction de la dimension » (*curse of dimensionality*, voir cette partie (http ://cedric.cnam.fr/vertigo/Cours/RCP216/coursSimilariteRecommandation.html#malediction-de-la-dimension) d'une séance suivante).

Vu la multiplicité des objectifs et des critères associés, de nombreuses méthodes de réduction de dimension ont été définies.

Approches de réduction de dimension

La construction de méthodes de réduction de dimension suit une des deux approches suivantes : la sélection de variables et la transformation de variables.

La **sélection de variables** (*feature selection*, voir par ex. la synthèse de *[TAL14]* (page 131)) consiste à choisir un sous-ensemble de k variables parmi les m variables initiales. Les variables sélectionnées gardent ainsi leur signification initiale, ce qui contribue à la lisibilité des modèles construits ultérieurement. Cette approche est potentiellement sous-optimale par rapport à la seconde approche qui est la construction de nouvelles variables.

Chaque méthode de sélection de variables fait partie d'une des catégories suivantes :

1. Méthodes de filtrage : basées sur des critères (par ex. minimisation de la redondance entre variables, maximisation de l'information mutuelle avec la classe à prédire) qui ne tiennent pas compte des résultats du modèle décisionnel ultérieur.
2. *Wrappers* : basées sur des mesures des performances du modèle décisionnel qui emploie les variables sélectionnées.
3. Méthodes intégrées (*embedded*) : l'opération de sélection est indissociable de la méthode de modélisation décisionnelle.

La sélection de variables est confrontée à un problème de complexité algorithmique : pour choisir k variables parmi m, l'espace de recherche contient C_m^k possibilités. Afin d'éviter une recherche exhaustive dans cet espace, des méthodes approximatives sont adoptées (la solution est en général sous-optimale par rapport à celle d'une recherche exhaustive). Nous pouvons mentionner les approches suivantes :

1. Tri des m *variables initiales* par rapport à un critère de « pertinence » exprimable par variable individuelle, indépendamment des autres, puis sélection des k premières (par ex., dans Spark, sélection sur la base du test du χ^2 avec `ChiSqSelector`).
2. Construction incrémentale de l'ensemble de k variables : à chaque itération on ajoute la variable qui forme le meilleur ensemble avec celles déjà sélectionnées lors des itérations précédentes.

La **transformation de variables** (*feature extraction*, expression qui peut avoir une signification plus large) consiste à construire de *nouvelles* variables à partir des variables initiales. Cette approche présente plus de flexibilité par rapport à la seule sélection de variables. En revanche, si les variables initiales ont une signification précise, les nouvelles variables sont rarement interprétables.

Les nouvelles variables sont obtenues par des méthodes qui sont (voir les figures suivantes)

1. Linéaires : trouver un sous-espace linéaire de dimension k dans l'espace initial $\mathbb{R}^m$.
2. Non linéaires : trouver un sous-espace non linéaire de dimension k dans l'espace initial.

Nous rappellerons brièvement dans la suite trois méthodes factorielles linéaires :

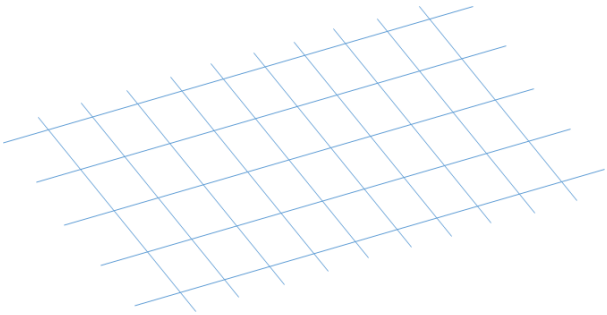


Fig. 2.1 – Sous-espace bidimensionnel linéaire dans l'espace tridimensionnel

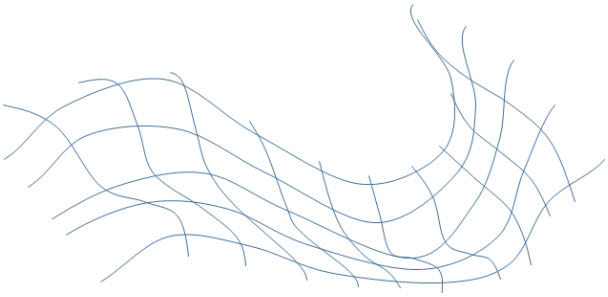


Fig. 2.2 – Sous-espace bidimensionnel non linéaire dans l'espace tridimensionnel

1. L'analyse en composantes principales (ACP), méthode à caractère exploratoire, adaptée à des données décrites par des variables quantitatives.
2. L'analyse factorielle discriminante (AFD), méthode à caractère exploratoire et décisionnel, adaptée à des données décrites par des variables quantitatives et appartenant à plusieurs classes.
3. L'analyse des correspondances multiples (ACM), méthode à caractère exploratoire, adaptée à des données décrites par des variables nominales.

Pour plus de détails concernant ces méthodes classiques vous êtes invités à consulter des sources externes (par ex. *[CABB04]* (page 131), *[Sap11]* (page 131)).

Analyse en composantes principales

L'ACP est une méthode d'analyse exploratoire des données : à partir d'un ensemble de N observations caractérisées par m variables quantitatives initiales, on cherche à condenser la représentation des données en conservant au mieux leur organisation globale. Pour cela, on représente les données sur k nouvelles variables (les composantes principales, $k < m$) obtenues comme des combinaisons linéaires des variables initiales, en conservant le plus de variance possible.

Dans l'analyse de données massives, l'ACP est principalement utilisée pour :

1. Condenser les données en conservant au mieux leur organisation globale.
2. Visualiser en faible dimension l'organisation prépondérante des données.
3. Interpréter les corrélations ou anti-corrélations entre multiples variables.
4. Interpréter les projections des prototypes de classes d'observations par rapport aux variables.
5. Préparer des analyses ultérieures en éliminant les sous-espaces dans lesquels la variance des données est très faible (assimilés, parfois à tort, à des sous-espaces de bruit, sans information utile).

Dans la matrice $\mathbf{R}$ des données brutes chaque ligne correspond à une observation et chaque colonne à une variable initiale. L'ACP connaît plusieurs variantes, selon le pré-traitement appliqué à la matrice $\mathbf{R}$:

1. ACP générale : appliquée directement sur $\mathbf{X} = \mathbf{R}$. Interviennent dans l'analyse à la fois la *position* du nuage d'observations par rapport à l'origine et la *forme* du nuage. Cette variante est utilisée rarement, essentiellement pour tenir compte du zéro naturel de certaines variables.

2. ACP centrée : centrage préalable des variables. La matrice analysée, $\mathbf{X}$, est obtenue en transformant $\mathbf{R}$ pour que chaque variable (chaque colonne) soit de moyenne nulle. Cela revient à s'intéresser à la *forme* du nuage d'individus par rapport à son centre de gravité. Cette variante est utilisée lorsque les variables initiales sont directement comparables (de même nature, intervalles de variation comparables).
3. ACP normée : réduction préalable des variables. La matrice analysée, $\mathbf{X}$, est obtenue en transformant $\mathbf{R}$ pour que chaque variable (chaque colonne) soit de moyenne nulle et d'écart-type unitaire. On s'intéresse donc à la forme du nuage d'individus après normalisation des variables. Cette variante (la plus fréquemment rencontrée) est employée lorsque les variables (toutes quantitatives) sont de nature différente ou présentent des intervalles de variation très différents.

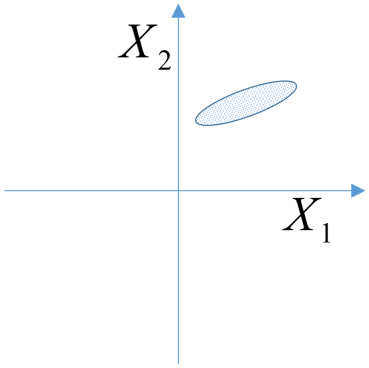


Fig. 2.3 – ACP générale

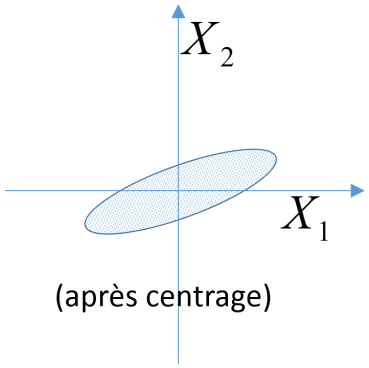


Fig. 2.4 – ACP centrée

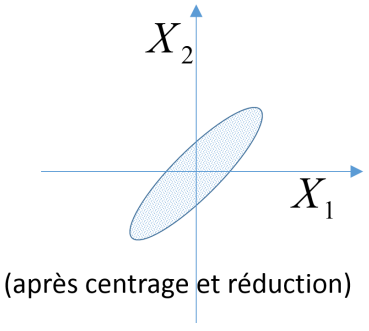


Fig. 2.5 – ACP normée

Les lignes de la matrice analysée, $\mathbf{X}$, décrivent les observations dans l'espace des variables initiales, alors que les colonnes de $\mathbf{X}$ décrivent les variables dans l'espace des observations. Deux analyses sont donc possibles, l'analyse du nuage des observations et l'analyse du nuage des variables.

Dans l'analyse du nuage des observations nous cherchons k nouvelles variables obtenues comme des combinaisons linéaires des variables initiales (c'est à dire, un sous-espace linéaire de dimension k de l'espace initial) telles que la projection des observations sur ces variables conserve le plus de *variance* (ou « dispersion »).

Il est facile de montrer (voir par ex. *[CABB04]* (page 131), *[Sap11]* (page 131)) que le sous-espace de dimension k recherché est généré par les k vecteurs propres $\mathbf{u}_\alpha$ associés aux k plus grandes valeurs propres λ_α de la matrice $\mathbf{X}^T\mathbf{X}$, satisfaisant donc l'équation $\mathbf{X}^T\mathbf{X}\mathbf{u}_\alpha = \lambda_\alpha\mathbf{u}_\alpha$, $\alpha \in \{1, \ldots, k\}$. Les vecteurs $\beta\mathbf{u}$ sont en général choisis de norme égale à 1. Rappelons ici que si $\mathbf{u}_\alpha$ satisfait $\mathbf{X}^T\mathbf{X}\mathbf{u}_\alpha = \lambda_\alpha\mathbf{u}_\alpha$ alors pour tout $\beta \in \mathbb{R}$, $\beta\mathbf{u}_\alpha$ satisfait la même relation.

Nous remarquerons que pour l'ACP centrée $\Rightarrow$ $\mathbf{X}^T\mathbf{X}$ est la matrice des *covariances empiriques* alors que pour l'ACP normée $\Rightarrow$ $\mathbf{X}^T\mathbf{X}$ est la matrice des *corrélations empiriques*.

La matrice $\mathbf{X}^T\mathbf{X}$ est symétrique et (semi-)définie positive, donc toutes ses valeurs propres sont réelles, positives (certaines nulles si la matrice est semi-définie) et les vecteurs propres associés à des valeurs propres différentes sont orthogonaux.

La figure suivante montre un exemple d'analyse du nuage des observations. Les $N = 5500$ données initiales de dimension $m = 40$ sont projetées sur les deux premières composantes principales ($k = 2$) de l'analyse du nuage des observations. Les observations correspondent à des descripteurs visuels de pixels extraits d'images de textures [1]. En plus des 40 variables quantitatives initiales, chaque pixel est caractérisé par la classe de texture (11 classes au total) à laquelle il appartient, mais l'ACP ne tient pas compte de cette variable nominale. Cette classe a néanmoins été représentée par une couleur et une forme spécifique du point projection (11 classes au total) afin de nous permettre de voir dans quelle mesure les directions de variances maximales (deux premiers axes principaux dans cette projection bidimensionnelle) permettent de séparer les classes.

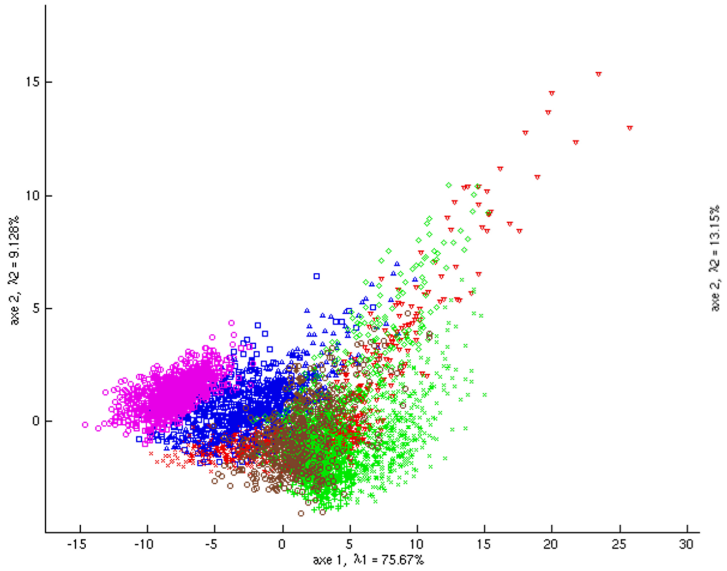


Fig. 2.6 – Exemple 1 ACP : projection des observations sur le premier plan factoriel

L'analyse du nuage des variables se déroule de façon symétrique. La matrice à diagonaliser est dans ce cas $\mathbf{X}\mathbf{X}^T$ et ses valeurs propres non nulles sont les mêmes que celles de $\mathbf{X}^T\mathbf{X}$. Le lien entre les matrices analysées mène à des relations de transition entre les deux analyses (du nuage des observations et du nuage des variables) qui permettent d'obtenir les résultats de l'une directement à partir des résultats de l'autre (voir par ex. *[CABB04]* (page 131), *[Sap11]* (page 131)). Comme en général $N \gg m$ (nombre d'observations $\gg$ nombre de variables initiales), il est préférable de traiter la matrice $\mathbf{X}^T\mathbf{X}$ de dimension $m \times m$ plutôt que $\mathbf{X}\mathbf{X}^T$ de dimension $N \times N$.

La figure suivante montre un exemple d'analyse du nuage des variables. Les 10 variables initiales sont projetées sur les deux premières composantes principales de l'analyse du nuage des variables. Chaque observation correspond à un représentant typique d'une espèce de mammifères ; il y a $N = 62$ observations décrites par $m = 10$ variables initiales [2].

Le variables de départ étant de nature diverse (poids, durées, indices), l'ACP normée a été employée. Les variables initiales ont donc d'abord été normalisées avant l'analyse. Cela explique pourquoi chaque variable initiale est re-

1. Données mises à disposition par le Laboratoire de Traitement d'Image et de Reconnaissance de Formes (LTIRF) de l'Institut National Polytechnique de Grenoble (INPG) dans le cadre du projet ESPRIT III ELENA (No. 6891) et du groupe de travail ESPRIT ATHOS (No. 6620).
2. Données issues de Allison T., Cicchetti D., *Sleep in mammals : ecological and constitutional correlates*, Science, vol. 194, pp. 732-734.

présentée par un vecteur de norme égale à 1, situé donc sur une hyper-sphère centrée dans l'origine. L'intersection de cette hyper-sphère avec le plan de projection illustré est donc un cercle et les vecteurs sont projetés à l'intérieur de ce cercle. Plus la projection d'une variable initiale est proche du cercle, plus cette variable est proche du plan de projection, donc bien représentée par ce plan.

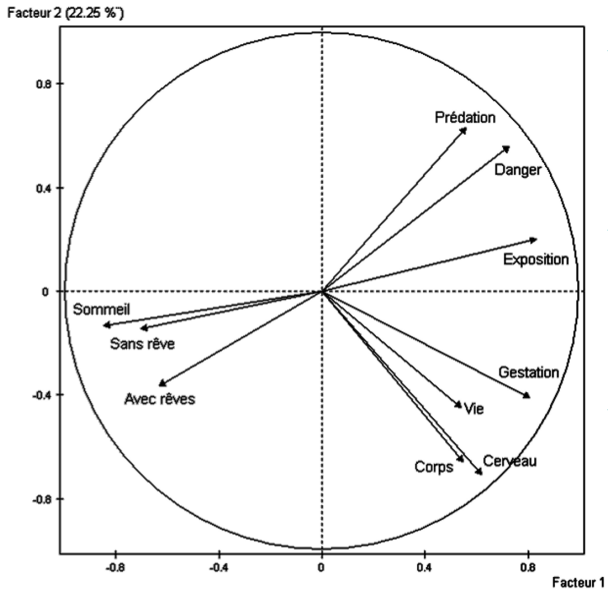


Fig. 2.7 – Exemple 2 ACP : projection des variables sur le premier plan factoriel

Une visualisation simple permet d'identifier trois groupes de variables initiales (corrélation forte des variables à l'intérieur de chaque groupe) : le groupe « sommeil », le groupe « danger » et le groupe « corps, cerveau, vie, gestation » (CCVG). On observe une forte opposition entre le groupe « sommeil » et le groupe « danger » (anti-corrélation des variables entre ces deux groupes) et une opposition plus faible entre le groupe « sommeil » et le groupe CCVG.

Le choix du nombre de composantes à retenir (valeur de k) dépend de l'objectif de l'analyse :

1. Analyse descriptive avec visualisation : déterminer à partir de quel ordre les différences entre les pourcentages d'inertie expliquée ne sont plus significatives. Un test statistique d'égalité entre valeurs propres successives peut être envisagé si la distribution des données est proche d'une distribution normale. Une méthode heuristique est souvent utilisée, voir la figure suivante : on construit le graphique des valeurs propres triées par ordre décroissant et on conserve les valeurs propres (et les composantes principales associées) qui précèdent le premier « coude ».
2. Réduction du volume de données : on impose la conservation d'une qualité d'approximation des données initiales, mesurée par le taux d'inertie expliquée (par ex. 85%). Le taux de réduction du volume interviendra, en général, dans ce choix.
3. Prétraitement avant application de méthodes décisionnelles : on utilise souvent un simple critère de bon conditionnement de la matrice des covariances empiriques (ou corrélations empiriques si analyse normée) ; attention, ce choix peut s'avérer nocif lorsque les directions qui présentent la plus faible variance sont des directions prédictives (par exemple, dans un problème de classement, des directions discriminantes ou qui aident à séparer les classes). Il est également possible de considérer le nombre d'axes comme un paramètre supplémentaire de la méthode décisionnelle et d'employer ensuite une technique de sélection de modèle décisionnel.

Les valeurs et vecteurs propres de $\mathbf{M} = \mathbf{X}^T\mathbf{X}$ (en général $N \gg m$, on s'intéresse donc à $\mathbf{X}^T\mathbf{X}$ plutôt qu'à $\mathbf{X}\mathbf{X}^T$) peuvent être obtenus de différentes façons, selon qu'on souhaite déterminer *toutes* les valeurs propres de $\mathbf{M}$ (de dimension $m \times m$) ou seulement les k plus grandes, avec les vecteurs propres unitaires associés.

Pour obtenir toutes les valeurs propres, l'algorithme est de complexité $O(m^3)$: on commence par calculer le déterminant de $(\mathbf{M} - \lambda\mathbf{I}_m)$ afin de trouver les valeurs propres, ensuite on résout pour chaque valeur propre un système linéaire de m équations avec m inconnues pour déterminer le vecteur propre associé. Si m est très élevé

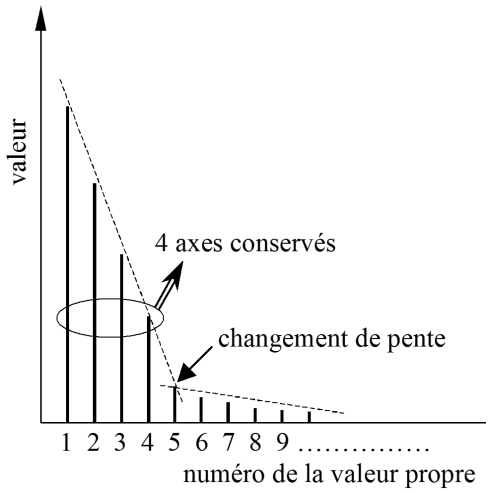


Fig. 2.8 – Choix du nombre d'axes en ACP : lorsque les différences entre valeurs propres successives deviennent faibles, les axes principaux correspondants ne sont plus significatifs (deviennent très instables par rapport au choix de l'échantillon analysé)

et on souhaite conserver $k \ll m$ composantes, il est inutile de chercher à obtenir toutes les valeurs propres et on préfère la solution suivante.

Pour obtenir les k plus grandes valeurs propres, un algorithme itératif de complexité $O(Nmk)$ peut être employé. On obtient les valeurs propres une par une (avec le vecteur propre associé), à partir de la plus grande. Pour cela, on itère $\mathbf{x}_{i+1} = \frac{\mathbf{M}\cdot\mathbf{x}_i}{\|\mathbf{M}\cdot\mathbf{x}_i\|}$ à partir d'un vecteur $\mathbf{x}_0$ quelconque non nul. A convergence, $\lambda_1 = \mathbf{x}^T \cdot \mathbf{M} \cdot \mathbf{x}$ sera la plus grande valeur propre et le $\mathbf{x}$ obtenu le vecteur propre associé. On calcule $\mathbf{M}_2 = \mathbf{M} - \lambda_1 \mathbf{x} \cdot \mathbf{x}^T$ et on applique le même processus itératif sur $\mathbf{M}_2$ pour obtenir λ_2, etc. Cette solution est encore plus intéressante pour des variables initiales « creuses » : si chaque variable prend des valeurs non nulles pour au maximum p observations, avec $p \ll N$, alors p remplace N dans la complexité.

Analyse factorielle discriminante

L'AFD est une méthode d'analyse de données multidimensionnelles qui présente à la fois une composante descriptive et une composante décisionnelle. On considère N observations caractérisées par m variables quantitatives initiales (matrice de données $\mathbf{X}$) *et* une variable nominale de « classe » $Y \in \{1, \ldots, q\}$. Lors de l'étape descriptive on cherche à identifier k « facteurs discriminants » ($k < m$, $k < q$!) qui permettent de différencier au mieux les classes ; ces facteurs discriminants sont des combinaisons linéaires des variables initiales. Lors de l'étape décisionnelle on construit un modèle de *discrimination* (ou de *classement*) permettant de décider à quelle classe affecter une nouvelle observation à partir des valeurs prises par les variables quantitatives (donc implicitement par les facteurs discriminants).

Les principales utilisations de l'AFD sont :

1. Descriptive : condenser la représentation des données en conservant au mieux la séparation entre les classes.
2. Décisionnelle : classer de nouvelles observations à partir du sous-espace linéaire qui optimise la séparation.

Nous nous intéresserons ici exclusivement à la composante descriptive. Plus d'explications sur l'AFD (à la fois sur les aspects descriptifs et décisionnels) peuvent être trouvées par ex. dans *[CABB04]* (page 131), *[Sap11]* (page 131).

La figure suivante montre un exemple d'AFD. Les $N = 5500$ données initiales de dimension $m = 40$ issues de [1], avec $q = 11$, sont projetées sur les deux premières composantes discriminantes ($k = 2$).

En comparant le résultat avec la projection des mêmes données sur les deux premières composantes principales (voir plus haut) on constate que la séparation entre les classes est naturellement bien meilleure avec l'AFD. On voit aussi que cette séparation entre les 11 classes n'est pas parfaite sur le premier plan discriminant ; la séparation

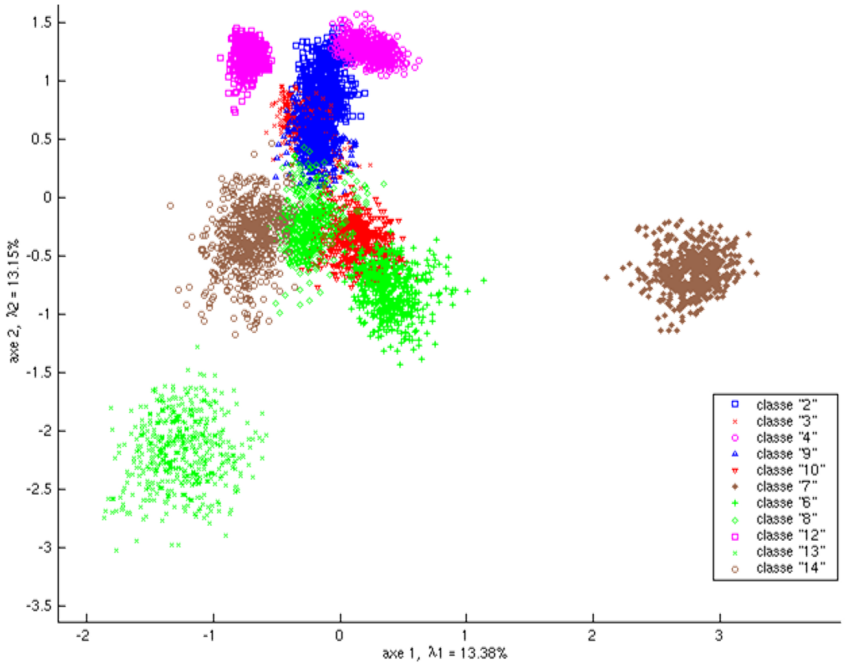


Fig. 2.9 – Exemple 1 AFD : projection des observations sur le premier plan factoriel

est en revanche presque parfaite dans l'espace tridimensionnel correspondant aux trois premières composantes discriminantes (non représenté ici).

La figure suivante illustre, pour un exemple simple (deux classes de forme allongée dans le plan), la différence entre l'ACP et l'AFD : l'ACP cherche le sous-espace (unidimensionnel ici) qui maximise la *variance* des projections alors que l'AFD cherche le sous-espace qui maximise la séparation entre les classes.

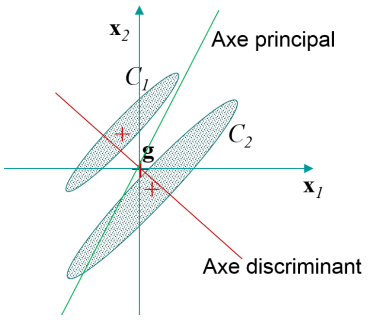


Fig. 2.10 – AFD versus ACP

Pour comprendre de quelle façon l'AFD procède il est nécessaire de s'intéresser aux calculs des covariances entre les variables initiales, selon qu'on considère les données dans leur totalité ou séparées en classes :

1. Covariances inter-classes : calculées en considérant que les seules observations sont les centres de gravité des q classes $\rightarrow$ matrice $\mathbf{E}$.
2. Covariances intra-classes : calculées sur les observations de départ, en centrant chaque classe sur son centre de gravité $\rightarrow$ matrice $\mathbf{D}$.
3. Covariances totales : calculées sur les observations de départ $\rightarrow$ matrice $\mathbf{S}$.

Ces covariances sont liées par la relation de Huygens $\mathbf{S} = \mathbf{E} + \mathbf{D}$.

Pour trouver le sous-espace de dimension k le plus discriminant on cherche à séparer au mieux les centres de gravité des classes, tout en tenant compte de la forme des classes. On peut montrer (voir par ex. *[CABB04]* (page 131), *[Sap11]* (page 131)) que le sous-espace recherché est généré par les k vecteurs propres $\mathbf{u}_\alpha$ associés aux k plus grandes valeurs propres λ_α de l'équation de valeurs et vecteurs propres *généralisée* $\mathbf{E}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{S}\mathbf{u}_\alpha$,

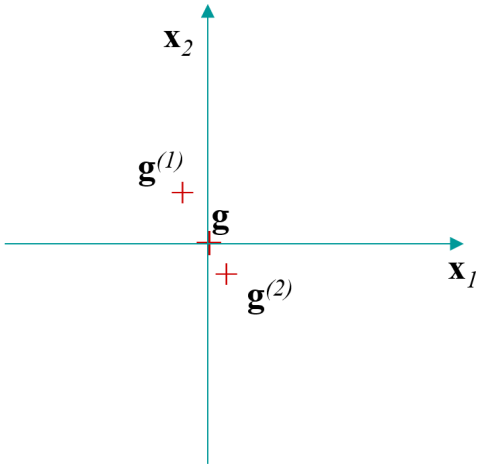


Fig. 2.11 – Inter-classes

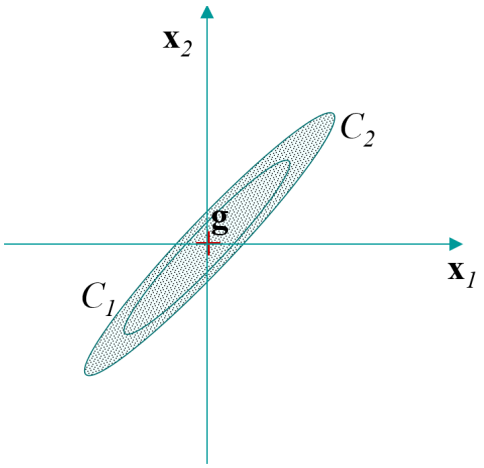


Fig. 2.12 – Intra-classes

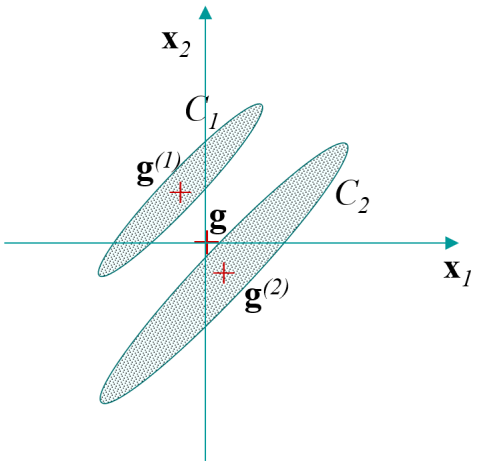


Fig. 2.13 – Totale

$\alpha \in \{1, \ldots, k\}$. Il est possible de résoudre plutôt $\mathbf{E}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{D}\mathbf{u}_\alpha$ si le rang de $\mathbf{D}$ n'est pas inférieur à celui de $\mathbf{S}$ (le rang de $\mathbf{D}$ ne peut pas être supérieur à celui de $\mathbf{S}$). Aussi, si $\mathbf{S}$ est inversible (et bien conditionnée) cela revient à résoudre $\mathbf{S}^{-1}\mathbf{E}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{u}_\alpha$.

Lorsque la matrice $\mathbf{S}$ est singulière, une approche fréquente est de réduire la dimension avec une ACP, pour que dans l'espace réduit $\mathbf{S}'$ soit de rang complet, ensuite résoudre ${\mathbf{S}'}^{-1}\mathbf{E}'\mathbf{u}_\alpha = \lambda_\alpha \mathbf{u}_\alpha$ dans cet espace réduit.

Remarque importante : si l'ACP est appliquée pour réduire la dimension au-delà de la nécessité de rendre $\mathbf{S}'$ de rang complet (par exemple, pour rendre $\mathbf{S}'$ bien conditionnée), il y a un risque non négligeable d'élimination de variables discriminantes ! Il faudrait préférer dans ce cas une approche de *régularisation*, par exemple remplacer $\mathbf{S}$ par $\mathbf{S} + r\mathbf{I}_m$. La constante de régularisation $r > 0$ doit être assez grande pour que la matrice $\mathbf{S} + r\mathbf{I}_m$ soit bien conditionnée (mais pas *trop* grande, pour éviter de dénaturer la solution). Le choix du nombre de composantes discriminantes à retenir (valeur de k) peut être réalisé à l'aide de tests statistiques s'il est possible de considérer que les classes sont issues de lois normales multidimensionnelles :

1. Test de Rao : test d'égalité à 0 de la i-ème valeur propre.
2. Test du lambda de Wilks : test de l'apport des axes au-delà du i-ème.
3. Test incrémental : test de l'apport du $i+1$-ème axe.

Si l'AFD est employée comme un prétraitement avant application de méthodes décisionnelles, il est également possible de considérer le nombre d'axes (de facteurs discriminants) comme un paramètre supplémentaire de la méthode décisionnelle et de se servir ensuite d'une technique de *sélection de modèle* décisionnel pour choisir le nombre d'axes.

Il est important de noter que pour l'AFD (méthode linéaire) le nombre k de facteurs discriminants est $k < q$, q étant le nombre de classes. En effet, avec q classes, le rang de $\mathbf{E}$ ne peut être supérieur à $q-1$ et donc l'équation $\mathbf{E}\mathbf{u}_\alpha = \lambda_\alpha \mathbf{S}\mathbf{u}_\alpha$ ne peut avoir plus de $q-1$ valeurs propres non nulles. Par exemple, pour un problème à 2 classes on ne peut trouver qu'un **seul** axe discriminant ($k = 1$).

Analyse factorielle des correspondances

L'analyse des correspondances est une méthode d'analyse exploratoire de données décrites par des variables *nominales* (à modalités). Nous examinerons brièvement ici l'analyse des correspondances multiples (ACM) qui considère N observations caractérisées par $m > 2$ variables nominales, représentées par un *tableau disjonctif complet* (TDC, voir la figure ci-dessous) ; chaque observation possède exactement une modalité pour chaque variable.

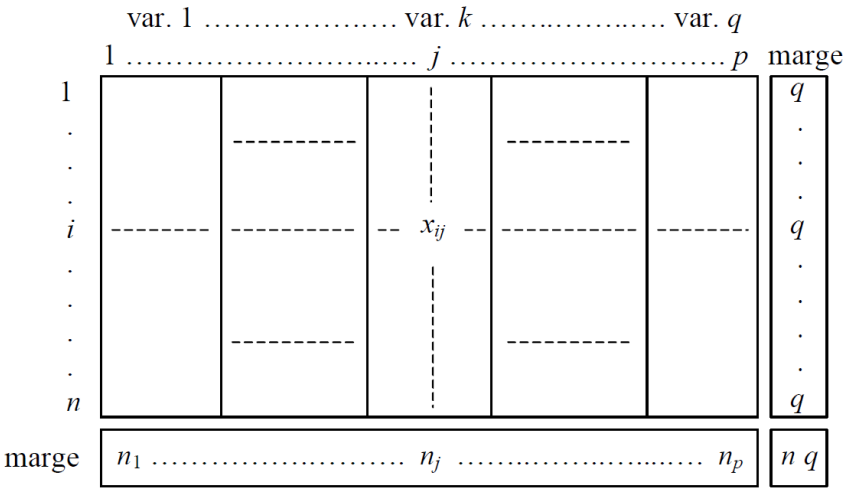


Fig. 2.14 – Tableau disjonctif complet (TDC)

Il est également possible d'utiliser pour l'ACM un *tableau de Burt* qui est la concaténation des tables de contingences par paires de variables nominales. L'utilisation du TDC permet d'analyser à la fois le nuage des observations et le nuage des modalités des variables, alors que le tableau de Burt permet d'étudier seulement le nuage des modalités (car les observations individuelles n'y sont pas décelables).

L'ACM cherche à mettre en évidence les relations dominantes entre des modalités des variables nominales initiales. Pour cela, k nouvelles variables quantitatives sont construites à partir des m variables nominales initiales, en conservant un maximum de variance.

L'ACM est utilisée traditionnellement dans le traitement d'enquêtes basées sur des questions fermées à choix multiples, afin de mettre en évidence des relations entre modalités ou éventuellement entre observations et modalités. Dans l'analyse de données massives, l'ACM peut servir à résumer un grand nombre (m) de variables qualitatives par un faible nombre (k) de variables quantitatives. Il est également possible d'inclure des variables quantitatives dans l'analyse, après leur transformation en variables nominales (voir plus loin).

Considérons un exemple issu de *[CABB04]* (page 131) et basé sur des données de [3] pour clarifier la nature des résultats obtenus par ACM (sur des données en faible volume). L'enquête « Les étudiants et la ville » [3] dont sont issues les données inclut, entre autres, les questions suivantes :

1. Habitez-vous : seul(e), en colocation, en couple, avec les parents.
2. Quel type d'habitation occupez-vous : cité U, studio, appartement, chambre chez l'habitant, autre.
3. Si vous vivez en dehors du foyer familial, depuis combien de temps : moins d'1 an, de 1 à 3 ans, plus de 3 ans, non applicable (NA).
4. A quelle distance de l'université vivez-vous : moins d'1 km, de 1 à 5 km, plus de 5 km.
5. Quelle est la surface habitable de votre logement : moins de $10m^2$, de $10m^2$ à $20m^2$, de $20m^2$ à $30m^2$, plus de $30m^2$.

Nous pouvons observer que trois des cinq variables sont issues de variables quantitatives discrétisées avant la réalisation du sondage.

La figure suivante montre les résultats de l'analyse des *modalités* des variables nominales initiales à travers leurs projections sur les deux premiers facteurs :

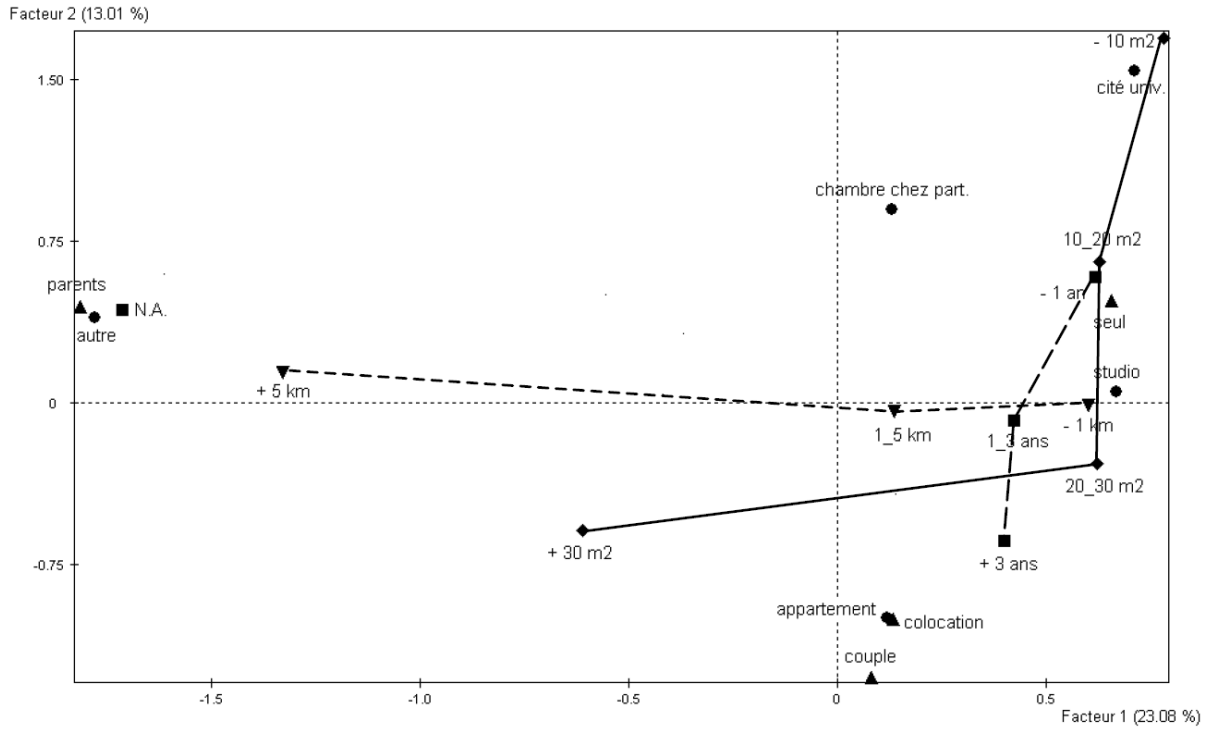


Fig. 2.15 – ACM : résultats sur l'exemple « Les étudiants et la ville »

Nous pouvons examiner les similarités et des oppositions entre projections de modalités de variables différentes (deux modalités sont similaires si elles concernent les mêmes populations d'observations) ou d'une même variable (et donc mutuellement exclusives ; deux telles modalités seront « similaires » si leurs populations sont similaires

3. Données issues de l'enquête « Les étudiants et la ville » réalisée en 2001 sous la direction de S. Denèfle à l'Université de Tours, voir *[CABB04]* (page 131).

par rapport aux autres variables nominales). Sur cet exemple nous pouvons constater, par exemple, des similarités fortes entre les modalités « parents » (variable « Habitez-vous »), « NA » (variable « depuis combien de temps ») et « autre » (variable « type d'habitation »), ainsi que des oppositions fortes entre « seul » et « parents » (même variable « Habitez-vous »), ou entre « +5 km » et « -1 km » (variable « distance »). Pour améliorer la lisibilité, les modalités « successives » des variables ordinales (issues ici de la discrétisation préalable de variables quantitatives) ont été reliées entre elles par des traits.

Pour réaliser l'ACM on cherche, comme pour l'ACP, le sous-espace de dimension k qui résume le mieux la variance (on parle parfois de « dispersion ») du nuage analysé. En revanche, contrairement à l'ACP où les données (observations ou variables) sont en général non pondérées, pour l'ACM on pondère chaque modalité (représentée par une colonne du TDC) par sa fréquence relative. Aussi, pour l'ACM on emploie la distance du χ^2 qui permet de pondérer dans le calcul de distance l'influence de chaque composante par l'inverse de son poids. Des développements plus détaillés peuvent être trouvés par ex. dans *[CABB04]* (page 131), *[Sap11]* (page 131). La distance du χ^2 présente une propriété intéressante d'équivalence distributionnelle : si deux colonnes (modalités) proportionnelles sont cumulées en une seule (fusion de deux modalités), les résultats de l'analyse ne changent pas.

Il est possible d'inclure des variables quantitatives dans l'ACM si les domaines de variation de ces variables sont découpés en intervalles et chaque intervalle est assimilé à une modalité de variable nominale. Cela permet de trouver des relations entre modalités de variables qualitatives et *intervalles* de valeurs prises par des variables quantitatives. Cela donne aussi la possibilité de mettre en évidence des relations *non linéaires* entre (intervalles de) valeurs prises par des variables quantitatives. Le découpage en intervalles du domaine de variation d'une variable quantitative peut être réalisé sur la base de connaissances *a priori* concernant les intervalles « pertinents » ou à partir de l'histogramme, comme dans la figure suivante (pour un découpage robuste, les frontières entre intervalles sont choisies dans les creux de l'histogramme) ; chaque intervalle sera une modalité de la nouvelle variable ordinale (les valeurs numériques sont ordonnées, les intervalles le seront donc aussi).

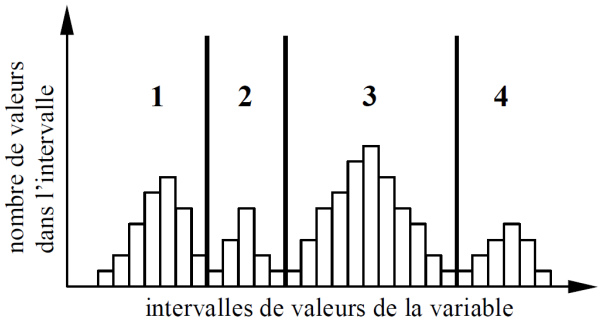


Fig. 2.16 – Découpage en intervalles du domaine de variation d'une variable quantitative afin d'obtenir des modalités

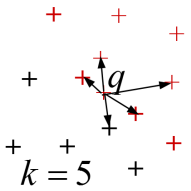
CHAPITRE 3

Cours - Réduction de l'ordre de complexité

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/3reductionComplexite.pdf)]

Une seconde famille de méthodes permettant de réduire le volume de calculs travaille sur toutes les données (et variables) mais en exploitant leurs caractéristiques de similarité afin de diminuer *l'ordre de complexité* des calculs à faire. En effet, les modèles décisionnels sont souvent « locaux ». La décision pour une nouvelle donnée dépend principalement (et parfois exclusivement) des données étiquetées proches.

Par exemple, pour une décision sur la base des k plus proches voisins, seules comptent les k données étiquetées les plus proches de la nouvelle donnée q à laquelle il faut affecter une étiquette de classe :



Dans la suite de ce chapitre, après une présentation rapide du principe de la réduction de l'ordre de complexité et un passage en revue des problèmes ciblés, nous examinerons de plus près le hachage sensible à la similarité (*Locality Sensitive Hashing, LSH*).

Principe des méthodes et typologie des besoins

Grâce à la construction préalable d'un *index* multidimensionnel ou métrique il est parfois possible de réduire l'ordre de complexité des calculs à faire, par exemple $O(N) \rightarrow O(\log(N))$ ou $O(1)$, $O(N^2) \rightarrow O(N \log(N))$, N étant le nombre total de données. Ces méthodes peuvent être exactes (fournir les mêmes résultats qu'un calcul exhaustif) ou, plus souvent, approximatives. En revanche, ces méthodes deviennent peu efficaces (réduction insuffisante de la complexité et même augmentation du coût) lorsque le nombre de variables décrivant les données est élevé. Aussi, leur mise en œuvre efficace sur une plate-forme distribuée peut poser quelques difficultés, ce qui explique certaines réserves dans leur implémentation.

Des combinaisons de méthodes de réduction de complexité peuvent également être intéressantes. Par exemple, appliquer une réduction de dimension avant la réduction de l'ordre de complexité permet de diminuer l'impact de la malédiction de la dimension sur la structure d'index employée ; nous y reviendrons ultérieurement.

Examinons d'abord brièvement les principaux types de problèmes pour lesquels une réduction de l'ordre de complexité est envisageable :

1. Recherche par similarité.
2. Décision : k plus proches voisins, machine à noyaux.
3. Estimation de densité (pour la description ou la décision).
4. Classification automatique, quantification.
5. Jointure par similarité.

6. Apprentissage supervisé, apprentissage semi-supervisé, apprentissage actif.

La recherche par similarité a plusieurs applications dans la fouille de données ou dans la prise de décision (par ex. sur la base des *k* plus proches voisins), ainsi que des applications directes comme la recherche de plagiat, l'authentification biométrique, les systèmes de recommandation, les systèmes d'information géographiques.

Plusieurs types de recherche sont habituellement mis en œuvre ($\mathbf{q}$ est la requête, c'est à dire la donnée à laquelle il faut trouver, dans une base $\mathcal{D}$, d'autres données similaires) :

1. La recherche par intervalle (*range query*) : $Range_r(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} | \forall i, |x_i - q_i| \leq r_i\}$
2. La recherche dans un rayon (*sphere query*) : $Sphere_\epsilon(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} | d(\mathbf{x}, \mathbf{q}) \leq \epsilon\}$
3. La recherche des *k* plus proches voisins (kppv, *kNN*) : $kNN(\mathbf{q}) = \{\mathbf{x} \in \mathcal{D} | \, |kNN(\mathbf{q})| = k \wedge \forall \mathbf{y} \in \mathcal{D} - kNN(\mathbf{q}), d(\mathbf{y}, \mathbf{q}) > d(\mathbf{x}, \mathbf{q})\}$

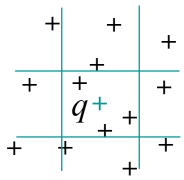


Fig. 3.1 – Recherche dans un intervalle

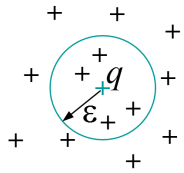


Fig. 3.2 – Recherche dans un rayon

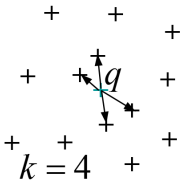


Fig. 3.3 – Recherche des kppv (*kNN*)

Si N est le nombre de données de $\mathcal{D}$, on souhaite disposer d'une méthode de recherche de complexité inférieure à la complexité $O(N)$ (*linéaire en* N) d'une recherche exhaustive.

La prise de décision sans modèle (par ex. sur la base des *k* plus proches voisins) ou avec un modèle peut avoir une complexité élevée si une méthode naive d'évaluation est adoptée. Considérons deux cas pour illustrer :

1. Pour la décision sur la base des *k* **plus proches voisins** il est d'abord nécessaire de trouver les kppv de la nouvelle donnée $\mathbf{x}$ à affecter à une classe, ensuite de faire voter ces kppv (vote à la majorité simple ou qualifiée, avec ou sans rejet d'ambiguïté). La recherche exhaustive des kppv a une complexité $O(N)$ (N étant le nombre de données).
2. La fonction de décision d'une **machine à noyaux** (par ex. machine à vecteurs support ou analyse discriminante à noyaux) est : $f^*(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^*$. Pour un problème de discrimination à deux classes, la nouvelle donnée $\mathbf{x}$ sera affectée à une des classes suivant le signe de $f^*(\mathbf{x})$. Si N est le nombre de données $\mathbf{x}_i$ (nombre de vecteurs support pour une SVM, ou nombre total de données étiquetées pour l'analyse discriminante à noyaux) alors un calcul exhaustif est de complexité $O(N)$. Or, si des noyaux locaux sont employés ($K(\mathbf{x}, \mathbf{x}_i) \approx 0$ pour $d(\mathbf{x}, \mathbf{x}_i) > \epsilon$, par ex. pour la loi normale multidimensionnelle), alors seuls comptent les $\mathbf{x}_i$ *dans un rayon* ϵ autour de $\mathbf{x}$, il est donc possible de réduire l'ordre de complexité par rapport au calcul exhaustif.

L'estimation de densité non paramétrique (par ex. par noyaux de Parzen) ou paramétrique (par ex. par mélange additif gaussien) peut être employée aussi bien dans l'analyse exploratoire que dans la modélisation décisionnelle. Pour examiner la complexité de l'estimation de densité, prenons deux exemples :

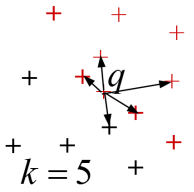


Fig. 3.4 – k plus proches voisins

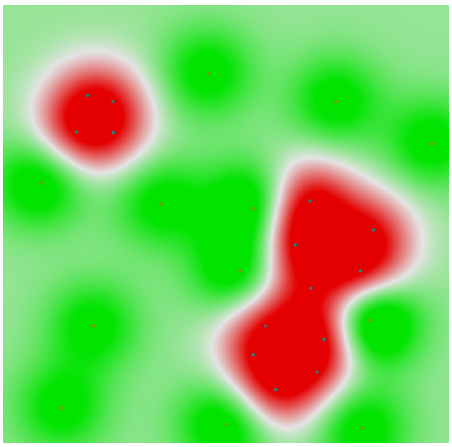


Fig. 3.5 – Machine à noyaux

1. Noyaux de Parzen : noyaux K identiques centrés sur les N observations initiales, la densité en $\mathbf{x}$ est $f(\mathbf{x}) \propto \sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)$.
2. Mélange additif gaussien : la densité en $\mathbf{x}$ est une somme pondérée de lois normales multidimensionnelles (les lois du mélange), on note par N le nombre de lois dans le mélange. Pour des données massives la valeur de N peut être élevée.

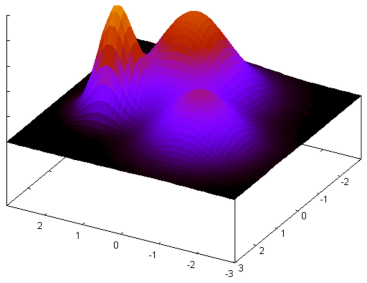


Fig. 3.6 – Estimation de densité par un mélange additif de lois normales bidimensionnelles

Dans les deux cas, un calcul exhaustif de la densité en $\mathbf{x}$ est de complexité $O(N)$. Ici encore, si des noyaux locaux (ou lois à décroissance rapide, comme la loi normale) sont employés ($K(\mathbf{x}, \mathbf{x}_i) \approx 0$ pour $d(\mathbf{x}, \mathbf{x}_i) > \epsilon$), alors seuls comptent les $\mathbf{x}_i$ *dans un rayon* ϵ autour du point $\mathbf{x}$ dans lequel on cherche à estimer la densité, il est donc possible de réduire l'ordre de complexité par rapport au calcul exhaustif.

La classification automatique est très utile pour mettre en évidence une structure simple (partitionnement ou hiérarchie de partitionnements) dans les données. Une méthode simple de classification automatique, qui peut néanmoins donner de bons résultats avec une initialisation convenable, est la méthode *k-means* (voir le chapitre sur la classification automatique). Chaque groupe est représenté par son centre de gravité. Pour N données on considère en général qu'il faut utiliser $O(N^{1/3})$ centres, la complexité de la classification sera donc $O(N^{4/3})$.

Si les données sont représentées dans un espace métrique non vectoriel, il est possible d'employer la méthode des *k-medoids* (un centre de gravité ne peut pas être calculé sur des données non vectorielles, un *medoid* est simplement l'élément le plus « central » d'un groupe). Dans ce cas, la complexité de la classification est plutôt $O(N^2)$ car l'identification d'un *medoid* est plus coûteuse que le calcul d'un centre de gravité.

Nous remarquerons aussi qu'en l'absence de groupes « naturels » dans les données, l'application d'une méthode comme *k-means* produit plutôt une **quantification des données** qui peut être également le but recherché dans certaines applications.

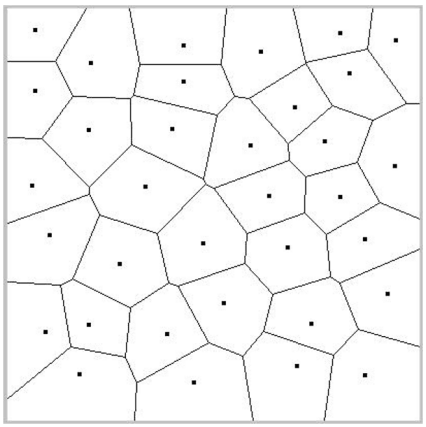


Fig. 3.7 – Quantification vectorielle

Si les calculs de distance entre une donnée et les centres (ou *medoids*) trop éloignés ont un faible impact sur les résultats, il est envisageable de gagner un facteur de $N^{1/3}$ dans la complexité de *k-means*.

Une **jointure par similarité** entre deux ensembles $\mathcal{D}_1$ (à N_1 éléments) et $\mathcal{D}_2$ (à N_2 éléments), avec un seuil de distance θ, permet de trouver les paires d'éléments des deux ensembles qui sont à une distance inférieure au seuil, c'est à dire $K_\theta = \{(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in \mathcal{D}_1, \mathbf{y} \in \mathcal{D}_2, d(\mathbf{x}, \mathbf{y}) \leq \theta\}$. On parle **d'auto-jointure** lorsque $\mathcal{D}_1 \equiv \mathcal{D}_2$ ($N_1 = N_2 = N$). Cette opération est utilisée pour identifier des quasi-copies de documents, pour exclure des analyses ultérieurs les données qui n'ont pas de voisins proches, etc.

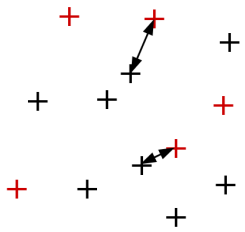


Fig. 3.8 – Jointure par similarité

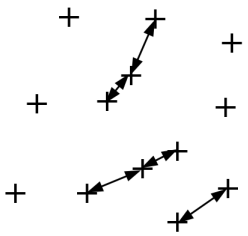


Fig. 3.9 – Auto-jointure par similarité

Si des comparaisons exhaustives sont employées, la complexité de la jointure par similarité est de $O(N_1 \times N_2)$ et celle de l'auto-jointure de $O(N^2)$. Une réduction de la complexité à $O(N \log N)$ ou même $O(N)$ est envisageable si les comparaisons entre données trop éloignées peuvent être évitées.

L'apprentissage supervisé consiste à construire un modèle décisionnel à partir de données étiquetées $\{\mathbf{x}_i, y_i\}_{1 \leq i \leq N}$. Par exemple, pour un ν-SVM, la détermination des vecteurs de support et de leurs coefficients

revient à résoudre le problème d'optimisation sous contraintes suivant :

$$\begin{array}{l} \min_\alpha \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ 1/N \geq \alpha_i \geq 0, 1 \leq i \leq N \\ \sum_{i=1}^N \alpha_i y_i = 0 \\ \sum_{i=1}^N \alpha_i \geq \nu \end{array} \tag{3.1}$$

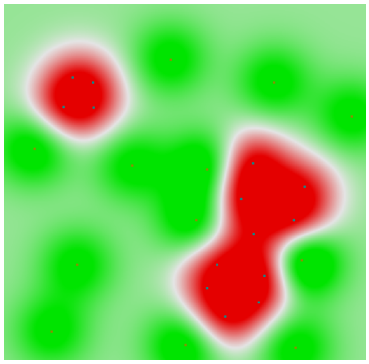


Fig. 3.10 – Modèle de discrimination à 2 classes appris à partir des données étiquetées (représentées par les points) ; la frontière de décision est représentées par les courbes blanches et l'intérieur des classes par la couleur verte et respectivement rouge

Avec N données d'apprentissage $\{\mathbf{x}_i, y_i\}$ la complexité est de $O(N^2)$. En revanche, si le noyau est local ($K(\mathbf{x}_i, \mathbf{x}_j) \approx 0$ pour $d(\mathbf{x}_i, \mathbf{x}_j) > \epsilon$) alors il n'est pas nécessaire de calculer $K(\mathbf{x}_i, \mathbf{x}_j)$ pour des données $\mathbf{x}_i$, $\mathbf{x}_j$ trop éloignées, la complexité peut éventuellement être réduite à $O(N \log N)$ ou $O(N)$. **L'apprentissage semi-supervisé** vise à utiliser, pour la construction d'un modèle décisionnel, non seulement les données étiquetées, en général relativement peu nombreuses, mais aussi les données non étiquetées disponibles, souvent bien plus nombreuses (N désignera ici le nombre de données non étiquetées). Par exemple, pour l'apprentissage transductif avec SVM on cherche à maximiser la marge non seulement par rapport aux données étiquetées mais aussi par rapport aux données non étiquetées en considérant, parmi tous leurs étiquetages possibles, le plus « favorable » (voir la figure de gauche ci-dessous). Il faudra s'intéresser pour cela aux seules données non étiquetées qui sont proches de la frontière de décision définie à partir des données étiquetées. Afin d'identifier ces données, une approche naïve consiste à évaluer la fonction de décision (obtenue sur les données étiquetées) pour chacune des N données non étiquetées. Si on dispose d'une méthode simple pour exclure les données non étiquetées trop éloignées de la frontière, alors la complexité peut être réduite à un ordre inférieur à $O(N)$. Dans **l'apprentissage actif** on considère que l'utilisateur peut donner des étiquettes à des données non encore étiquetées afin d'améliorer le modèle décisionnel existant. Il est nécessaire de faire le meilleur usage possible de ces interactions (coûteuses) entre le système et l'utilisateur. Une des solutions proposées consiste à demander à l'utilisateur d'étiqueter les données non encore étiquetées qui sont les plus « ambiguës » (et non redondantes, donc éloignées entre elles), c'est à dire les plus proches de la frontière de décision courante (voir la figure de droite ci-dessous pour un apprentissage actif avec SVM). Comme pour l'apprentissage transductif, la complexité de la recherche des données les plus proches de la frontière est de $O(N)$ mais cette complexité peut éventuellement être réduite si on dispose d'une méthode simple pour exclure les données non étiquetées trop éloignées.

Pour résumer, les **objectifs de réduction de l'ordre de complexité** sont

1. Recherche par l'exemple, décision, estimation de densité (la requête est de même nature que les données) : $O(N) \longrightarrow O(logN)$ ou $O(1)$.
2. Apprentissage semi-supervisé, apprentissage actif, « recherche par détecteur » (trouver les données les plus proches d'une frontière de décision) : $O(N) \longrightarrow O(logN)$ ou $O(1)$.
3. Jointure par similarité, apprentissage supervisé à noyaux, *N-body problems*, certaines méthodes de classification automatique ou de quantification : $O(N^2) \longrightarrow O(NlogN)$ ou $O(N)$.

Il est important de noter ici que la complexité ne peut être inférieure à la taille du résultat obtenu ! Ainsi, une recherche dans un rayon ϵ très large, qui couvre la totalité des N données, ne peut être de complexité inférieure à $O(N)$ car toutes les données doivent être retournées.

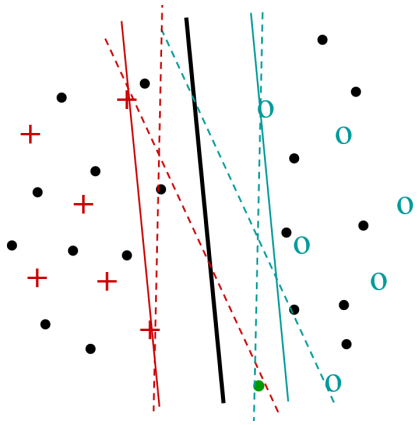


Fig. 3.11 – Transduction avec SVM

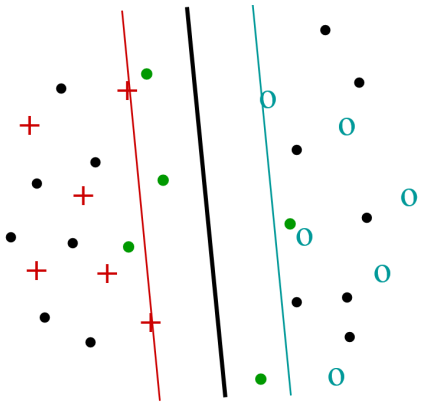


Fig. 3.12 – Apprentissage actif

Réduction de l'ordre de complexité : méthodes

Le **principe général** de la réduction de l'ordre de complexité grâce à la similarité est de regrouper les données par similarité et d'éviter les opérations impliquant des données issues de groupes éloignés. Des structures de données particulières (index multidimensionnels ou métriques) permettent de mettre en œuvre ces regroupements et de sélectionner efficacement, pour chaque donnée (déjà présente dans la base ou nouvelle) vue comme « requête », les données suffisamment proches pour intervenir dans les calculs.

Pour que cette approche générale soit valide, trois hypothèses importantes doivent être satisfaites :

1. Seules les données proches de (ou similaires à) la « requête » sont utiles. Naturellement, si la similarité n'est pas un bon critère de sélection, cette approche est sans intérêt.
2. Très peu de données sont proches de la requête. Comme nous l'avons noté plus haut, la complexité ne peut être inférieure à la taille du résultat obtenu. Si une partie importante des données doit se retrouver dans le résultat, la complexité ne peut pas être réduite.
3. Les données proches de la requêtes sont *bien plus proches* que les autres. L'approche est basée sur l'élimination, à un coût aussi faible que possible, d'un maximum de données éloignées de la requête. Cela sera d'autant plus facile que les différences de proximité entre les données sont fortes. Cette hypothèse conditionne donc *l'efficacité* de la méthode de réduction de complexité.

Les figures suivantes illustrent la construction et l'utilisation d'un type d'index particulier, un arbre de recherche. La figure de gauche montre un partitionnement hiérarchique des données. Au niveau le plus grossier nous trouvons trois grandes partitions, A, B et C (A et C ne sont pas représentées en entier). A l'intérieur de B nous trouvons trois partitions plus fines, D, E et F. Cela peut se poursuivre (par ex. G est une sous-partition de D). Une recherche par similarité avec la requête q, dans un rayon (cercle vert dans la figure), se déroule de façon hiérarchique, comme indiqué dans la figure de droite. Un calcul de distance entre q et le centre de A permet de constater que la sphère de recherche (cercle vert ici) ne peut pas intersecter la partition A, toutes les données de A sont donc exclues de la suite de la recherche. Idem pour la partition C. La recherche se poursuit alors dans B par des calculs de distance entre q et les centres des partitions D, E et F. Les données de F sont exclues car il ne peut y avoir d'intersection entre la sphère de recherche et F. La recherche se poursuivra donc dans D et E. Idéalement, l'arbre est traversé du nœud racine (correspondant aux partitions les plus larges) vers les feuilles, sans exploration de branches alternatives. Avec N données nous avons $O(N)$ feuilles, donc la longueur d'un chemin descendant (la hauteur de l'arbre) est $O(logN)$. La complexité de la *recherche* est alors réduite de N (avec comparaisons exhaustives) à $O(logN)$ avec cet arbre de recherche. La complexité de la *construction* de l'arbre est de $O(NlogN)$, l'utilisation de l'arbre reste intéressante si l'arbre est construit une fois (hors ligne, sans contraintes de temps) et employé ensuite de très nombreuses fois pour répondre à des requêtes sous contraintes de temps.

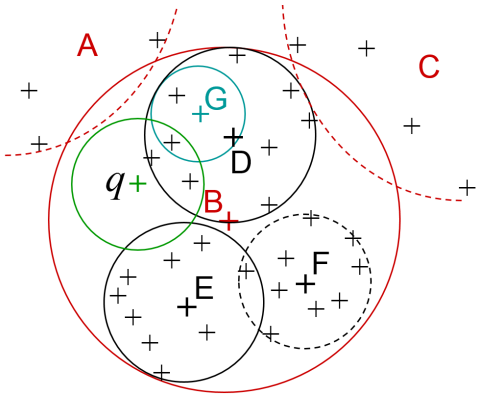


Fig. 3.13 – Partitionnement hiérarchique des données

De nombreuses structures d'index ont été proposées pour réduire la complexité de la recherche par similarité, qui à son tour peut servir à diminuer la complexité des autres familles de méthodes listées plus haut (décision, estimation de densité, apprentissage, etc.). Pour faire une typologie sommaire de ces propositions on s'intéresse d'abord à la méthode de réduction de l'espace de recherche et on trouve notamment :

1. Partitionnement de l'espace de description (utile lorsque les données « couvrent » bien cet espace) : k-d-B-tree, LSD-tree, etc.

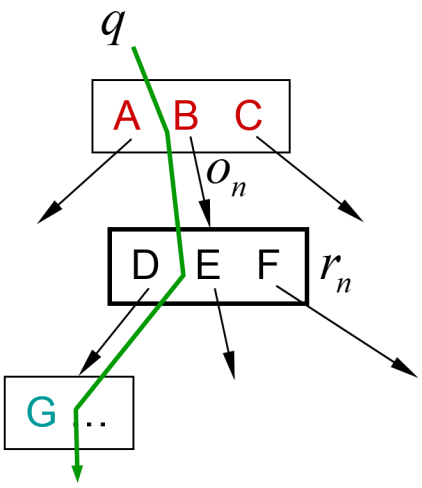


Fig. 3.14 – Recherche dans une hiérarchie de partitions

2. Partitionnement des données (utile lorsque les données sont présentes seulement dans certaines parties de l'espace) : R-tree, SR-tree, M-tree, etc.
3. Filtrage des données (utile lorsque la *malédiction de la dimension*, que nous examinerons dans le prochain chapitre, rend inefficaces les deux approches précédentes) : VA-file, LPC-file, etc.

La plupart de ces structures d'index ont été conçues pour la recherche « exacte » : toutes les données qui satisfont la condition de similarité sont retournées dans les résultats et tous les résultats satisfont la condition de similarité. Certaines méthodes peuvent être modifiées pour faire plutôt une recherche « approximative », plus efficace que la recherche exacte et satisfaisante dans de nombreux cas. D'autres méthodes ont été conçues dès le départ pour la recherche approximative et n'ont pas de variante « exacte ». Nous étudierons de plus près, dans la suite, une de ces méthodes approximatives, la hachage sensible à la similarité (*Locality Sensitive Hashing, LSH*). Une présentation détaillée de bon nombre d'index multidimensionnels et métriques peut être trouvée dans la monographie *[Sam05]* (page 131).

Il faut noter ici que la façon dont les données sont stockées peut avoir un impact majeur sur l'intérêt de l'utilisation d'un index par rapport à une recherche exhaustive. En effet, si les données se trouvent sur un stockage de masse (disque classique ou *flash*) alors une lecture séquentielle est de plusieurs ordres de grandeur plus lente qu'une lecture « aléatoire » (voir cette figure (http ://cedric.cnam.fr/vertigo/Cours/RCP216/figures/storage-hierarchy.png)), or l'utilisation d'un index multidimensionnel ou métrique implique le plus souvent une lecture aléatoire. Par exemple, si l'emploi d'un index permet une division par 100 du nombre d'opérations à réaliser (et de données à lire) mais la lecture aléatoire est 1000 fois plus lente que la lecture séquentielle, la recherche exhaustive sera env. 10 fois plus rapide que celle utilisant l'index.

Enfin, suivant l'utilisation faite de l'index, il est **important de tenir compte de la complexité de la construction de cet index**. Par exemple, pour un problème de recherche par similarité, si des recherches doivent être faites à faible coût pour de très nombreuses nouvelles requêtes, il sera utile d'employer un index pour réduire la complexité de chaque recherche de $O(N)$ à $O(\log N)$ même si la construction de l'index (réalisée une fois, en amont, hors ligne) est $O(N \log N)$. En revanche, construire un tel index pour réduire la complexité d'une seule classification automatique de $O(N^{4/3})$ à $O(N)$ ne sera pas intéressant.

Le hachage

Supposons que nous souhaitons associer à chaque donnée d'un ensemble $\mathcal{D}$ (appelé en général ensemble des *clés*) une donnée d'un autre ensemble $\mathcal{C}$. Si les données de $\mathcal{D}$ sont simples (par exemple, chaque donnée est un entier) il est possible de s'en servir comme indice dans un tableau qui stocke les valeurs correspondantes de $\mathcal{C}$.

En revanche, si chaque donnée de $\mathcal{D}$ est relativement complexe (par exemple, chaîne de caractères de longueur variable, etc.) on peut employer une table d'association avec une colonne pour $\mathcal{D}$ et une autre pour $\mathcal{C}$; afin de trouver la donnée de $\mathcal{C}$ correspondant à $v \in \mathcal{D}$ il suffit de comparer v successivement aux différentes entrées de la colonne pour $\mathcal{D}$ et, lorsqu'il y a identité, de lire la donnée associée dans la colonne pour $\mathcal{C}$. La complexité de cette

opération est $O(N)$ (avec N la cardinalité de $\mathcal{D}$), ce qui est coûteux si N est élevé. Ce serait intéressant de pouvoir calculer *directement* (complexité $O(1)$), à partir de $v \in \mathcal{D}$, l'indice dans le tableau où la valeur correspondante de $\mathcal{C}$ se trouve. C'est ce que font les *fonctions de hachage*.

Une fonction de hachage h associe à chaque donnée v d'un domaine $\mathcal{D}$ une valeur, souvent de $\mathbb{Z}^+$ (ensemble des entiers positifs). Cette valeur, appelée *hash* (ou empreinte, ou condensé) permet d'identifier une case (ou un bloc) mémoire où est stockée la donnée de $\mathcal{C}$ à associer à v. On parle de *collision* lorsque la fonction de hachage associe une *même* valeur de *hash* à deux valeurs différentes $v_1, v_2 \in \mathcal{D}$.

Dans les bases de données relationnelles, le hachage est une solution intéressante pour réduire la complexité de la *recherche par identité* illustrée par l'exemple suivant.

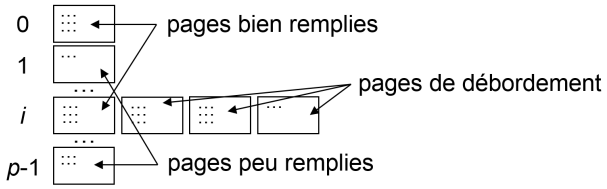
Exemple :

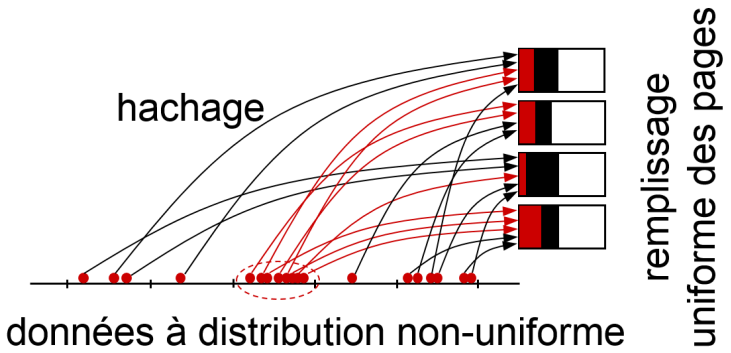
Considérons la table (la relation) `Film(titre, realisateur, annee)`, dans laquelle les valeurs de l'attribut `titre` sont des chaînes de caractères de longueur variable. Nous souhaitons pouvoir retrouver rapidement le nom du réalisateur et l'année de production à partir du titre d'un film. Supposons que p pages disque sont nécessaires pour stocker les $O(N)$ lignes (ou n-uplets, ou enregistrements) de la table `Film`. Pour cela, un hachage sur l'attribut `titre` (qui est la clé de hachage) avec la fonction de hachage $h : \mathcal{D} \rightarrow \{0, 1, 2, \ldots, p-1\}$ suivante est employé (où p est un nombre premier) :

$h(\texttt{titre}) = \left(\sum_{c_i \text{ caractère}} ascii(c_i(\texttt{titre}))\right) \mod p$

A chaque insertion d'un n-uplet dans la table `Film`, la clé (la valeur de l'attribut `titre`) est hachée avec h et le n-uplet est stocké sur la page ainsi identifiée. Si cette page est pleine, des pages de débordement sont utilisées, comme illustré dans la figure suivante (gauche).

Lors d'une recherche, le titre du film est haché avec h, la page identifiée par le *hash* obtenu est lue, ainsi que ses éventuelles pages de débordement, l'information recherchée doit s'y trouver (si elle est présente dans la table). La complexité de la recherche est donc $O(1)$ si une seule page disque est lue.





Les données de $\mathcal{D}$ ont en général une distribution très peu uniforme. Comme le montre cet exemple, les fonctions de hachage utilisées pour ce type de recherche doivent « disperser » le plus uniformément possible les *hash* associées afin d'optimiser le remplissage des pages : les pages trop peu remplies gaspillent l'espace de stockage, les pages de débordement allongent le temps de recherche. Pour cette raison, la similarité entre les clés (données de $\mathcal{D}$) n'est généralement pas conservée entre leurs valeurs de *hash* (figure précédente, droite). Ces fonctions de hachage sont donc en général inadaptées à la recherche par intervalle ou par similarité.

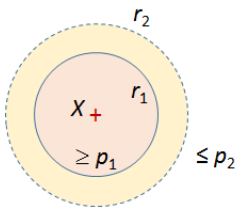
Hachage sensible à la similarité (LSH)

Le hachage sensible à la similarité (*Locality-Sensitive Hashing, LSH*) est une adaptation du hachage à la recherche par similarité. Considérons les données d'un domaine $\mathcal{D}$ doté d'une métrique $d_{\mathcal{H}} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}^+$, ainsi qu'un ensemble de *hash* $\mathcal{Q}$.

Définition :
$\mathcal{H} = \{h : \mathcal{D} \to \mathcal{Q}\}$ est un **ensemble de fonctions de hachage** (r_1, r_2, p_1, p_2)-**sensibles** (avec $r_2 > r_1 > 0$, $p_1 > p_2 > 0$) si (voir *[GIM99]* (page 131)) :

$$\forall x, y \in \mathcal{D}, \quad \begin{array}{l} d_{\mathcal{H}}(x,y) \leq r_1 \Rightarrow P_{h\in\mathcal{H}}(h(x) = h(y)) \geq p_1 \\ d_{\mathcal{H}}(x,y) > r_2 \Rightarrow P_{h\in\mathcal{H}}(h(x) = h(y)) \leq p_2 \end{array} \tag{3.2}$$

La définition exige que la probabilité de collision soit plus forte ($\geq p_1$, avec $p_1 > p_2$) entre données proches ($d_{\mathcal{H}}(x,y) \leq r_1$) qu'entre données plus éloignées ($\leq p_2$ si $d_{\mathcal{H}}(x,y) > r_2$, avec $r_2 > r_1$). Cela est illustré dans la figure suivante pour des données de $\mathcal{D} \subset \mathbb{R}^2$ et la distance euclidienne classique.



LSH peut être employé pour la **recherche par similarité** de la façon suivante :

1. En amont (avant toute requête) il est nécessaire de :
   (a) Calculer la valeur de *hash* pour chacune des données de la base.
   (b) Stocker les données de même valeur de *hash* dans une même page (appelée aussi *bucket*) ; utiliser des pages de débordement si nécessaire.
2. Ensuite, pour chaque donnée-requête :
   (a) Hachage de la donnée-requête pour obtenir son *hash*.
   (b) Lecture de la page (avec les éventuelles pages de débordement) associée à ce *hash*.
   (c) Retour de toutes les données de cette page, éventuellement après filtrage de ces données par calcul des distances entre la requête et chacune de ces données.

La complexité de la recherche est $O(1)$ pour chaque requête.

La figure suivante illustre cette méthode. Les points représentés constituent l'ensemble $\mathcal{D} \subset \mathbb{R}^2$. La grille oblique (en bleu) correspond au partitionnement de $\mathcal{D}$ en pages (ou *buckets*) par une fonction de hachage adaptée à la distance euclidienne classique dans $\mathbb{R}^2$. Les points situés dans un même parallélogramme sont dans une même page (ont une même valeur de *hash*). Lors de la recherche, les données de même *hash* que la requête (représentée par le point rouge) sont retournées.

On peut constater que, pour une recherche dans un rayon (cercle de recherche en rouge), certaines données retournées ne sont pas dans le rayon de recherche (constituent les « faux positifs » si le filtrage par calcul de distances n'est pas réalisé) alors que certaines données qui sont dans le rayon de recherche ne se trouvent pas sur la même page et ne sont donc pas retournées (ce sont les « faux négatifs »). Les résultats de la recherche sont donc une *approximation* de l'ensemble correspondant à la recherche exacte dans le rayon indiqué.

LSH pour métriques courantes

Nous passerons en revue dans la suite quelques familles de fonctions LSH adaptées à différents types de données et métriques associées.

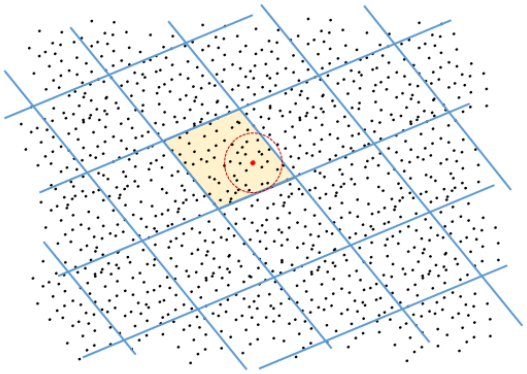


Fig. 3.15 – Données bidimensionnelles, fonction de hachage et requête

LSH pour la métrique euclidienne. Considérons des données décrites par m variables quantitatives, pour lesquelles on emploie la métrique euclidienne classique : $\mathcal{D} \subset \mathbb{R}^m$, $d_{\mathcal{H}}$ est la métrique L_2. Une famille de fonctions LSH (r_1, r_2, p_1, p_2)-sensibles est $d_{\mathcal{H}} = \{h\}$, $h : \mathbb{R}^m \to \mathbb{Z}$, $h_{\mathbf{a},b,w}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a}^T \cdot \mathbf{x} + b}{w} \right\rfloor$ avec $\mathbf{a} \in \mathbb{R}^m$ de composantes tirées indépendamment suivant $\mathcal{N}(0,1)$ et $b \in \mathbb{R}$ tiré suivant la loi uniforme dans $[0,1)$, $w \in \mathbb{R}$.

Dans $\mathbb{R}^2$, une telle fonction de hachage élémentaire correspond à une famille de droites parallèles, comme le montre la figure suivante (gauche). Chaque page de hachage correspond à un domaine borné dans une seule dimension ; sur une même page de hachage que la requête on peut trouver des données éloignées de la requête (« faux positifs »). Aussi, des données similaires à la requête peuvent se trouver sur des pages *voisines* (et sont donc des « faux négatifs »), surtout lorsque la requête est proche de la droite qui sépare deux pages.

On construit une *table de hachage* (ou une fonction de hachage composée) en faisant l'intersection entre les pages (*buckets*) de n fonctions élémentaires *indépendantes* ; le *hash* résultant est un n-uplet d'entiers. La figure suivante (droite) illustre cela dans $\mathbb{R}^2$, avec 3 fonctions élémentaires. Par rapport à l'utilisation d'une seule fonction élémentaire, l'emploi d'une table de hachage permet de réduire le nombre de faux positifs sans augmenter trop fortement le nombre de faux négatifs.

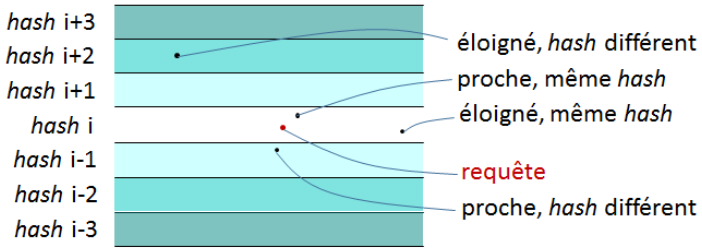


Fig. 3.16 – Fonction LSH élémentaire dans $\mathbb{R}^2$

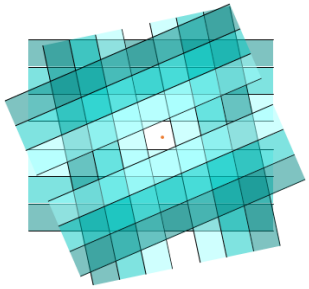


Fig. 3.17 – Composition de 3 fonctions élémentaires

En effet, une fonction élémentaire n'est pas assez sélective car le domaine correspondant à une page est borné dans une seule direction. Pour retenir seulement les données correspondant à la requête (situées à l'intérieur du cercle en rouge sur la figure suivante à gauche) il faut calculer la distance entre la requête et chacune des données de la même page que la requête, ce qui est coûteux. Si ce calcul n'est pas fait, nous obtenons de nombreux « faux positifs » et donc la *précision* des résultats (le rapport entre le nombre de *bons* résultats retournés et le nombre total de résultats retournés) sera faible.

Employer une grille plus fine (augmenter w dans la définition de la fonction, figure suivante au centre) réduira le nombre de « faux positifs » mais augmentera fortement celui de « faux négatifs » car plus de données situées à l'intérieur du cercle ne seront pas retournées. La précision augmentera mais le rappel (le rapport entre le nombre de bons résultats retournés et le nombre total de données de la base satisfaisant la condition de similarité à la requête) diminuera sensiblement.

Construire une table de hachage en composant plusieurs fonctions élémentaires indépendantes permettra d'exclure de nombreux « faux positifs », car les domaines correspondant aux pages de la table seront bornés dans plusieurs directions différentes, tout en augmentant un peu le nombre de « faux négatifs » (figure suivante à droite).

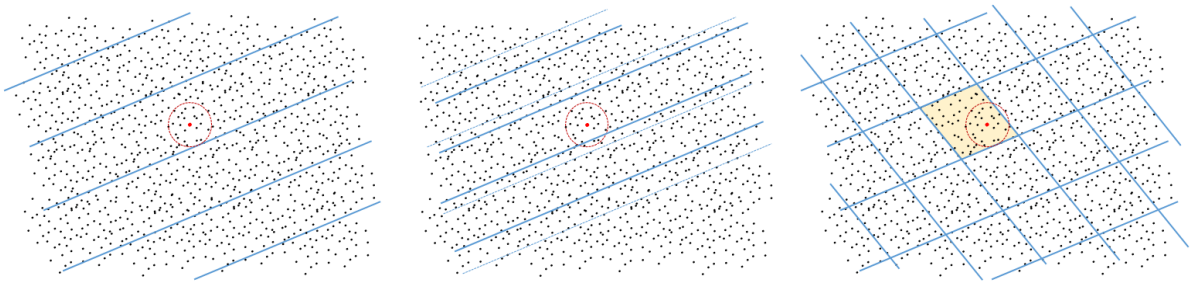


Fig. 3.18 – Composer des fonctions de hachage pour augmenter la précision

Il est important de noter que, dans la pratique, la dimension m de l'espace est très élevée et le nombre de fonctions de hachage composées dans une même table de hachage est $n \ll m$. En conséquence, contrairement au cas illustré dans les figures précédentes, les domaines correspondant aux pages de la table ne seront généralement pas bornés dans toutes les directions. Sans filtrage des résultats par calcul des distances, la précision sera en général inférieure à 1.

Si la requête est proche d'une frontière entre pages, des données proches de la requête sont exclues des résultats car situées sur une page différente de celle de la requête. Dans une table de hachage constituée par la composition de n fonctions élémentaires, les « faux négatifs » sont d'autant plus nombreux que la requête est proche de l'intersection entre plusieurs frontières.

Une solution pour réduire le nombre de « faux négatifs » (et augmenter donc le rappel) est d'utiliser $t > 1$ tables de hachage *indépendantes* (constituées de fonctions élémentaires indépendantes), de retourner à partir de chaque table l'ensemble des données de même *hash* que la requête et ensuite de faire la *réunion* de ces t ensembles. Les tables étant indépendantes, il est probable que les données proches de la requête et qui exclues par une des tables soient retournées grâce à une autre.

La figure suivante illustre l'emploi de 4 tables, chacune constituées de 3 fonctions de hachage élémentaires indépendantes. La requête est le point rouge, proche de certaines frontières dans une table et d'autres frontières dans les autres tables. La réunion des pages couvre relativement bien le voisinage de la requête.

La composition de plusieurs fonctions de hachage dans une même table de hachage afin d'augmenter la précision et l'utilisation conjointe de plusieurs tables de hachage pour augmenter le rappel seront examinés de façon plus précise dans la section Amplification de fonctions de hachage plus loin.

Multiplier les tables multiplie d'autant le temps nécessaire pour traiter la requête *et* l'espace nécessaire, car chaque table doit être stockée séparément. Afin de réduire le nombre de tables, *Multi-probe LSH* (voir *[LJW07]* (page 131)) propose de retourner, pour chaque table, non seulement les données de la même page que la requête mais aussi un échantillon des données des pages voisines. Le taux d'échantillonnage est plus élevé pour les pages les plus proches. La figure suivante montre quelles pages sont considérées pour une des tables de la figure précédente. A rappel constant, le nombre de tables peut ainsi être réduit d'un ordre de grandeur.

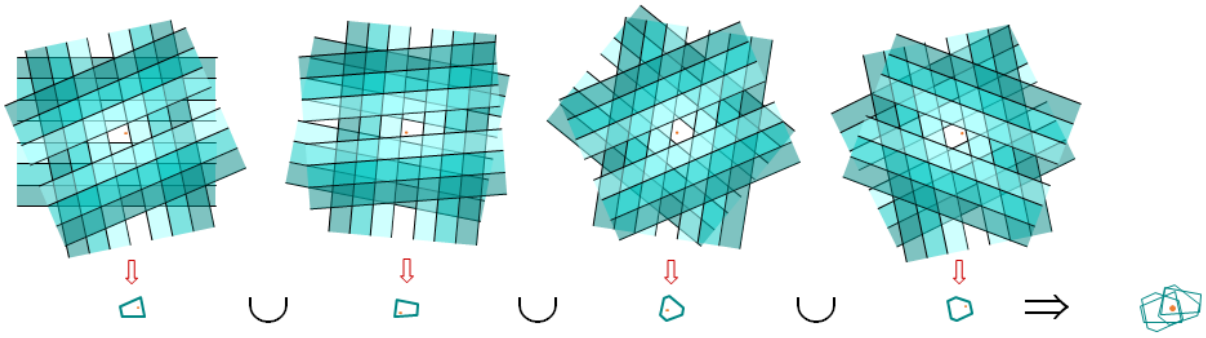


Fig. 3.19 – Composer des tables de hachage pour augmenter le rappel

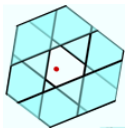


Fig. 3.20 – *Multi-probe* LSH

LSH pour la distance de Hamming. Dans de nombreux cas, les données (observations) sont décrites par un nombre m très élevé de variables binaires (vrai/faux, achat/non achat, consultation/non consultation, etc.). Chaque observation est ainsi représentée par un motif binaire de longueur m. Le domaine est alors $\mathcal{D} \subset \{0,1\}^m$ et la distance entre deux motifs binaires $\mathbf{x}, \mathbf{y} \in \mathcal{D}$ est la distance de Hamming $d_H(\mathbf{x}, \mathbf{y})$, c'est à dire le nombre de positions sur lesquelles $\mathbf{x}$ et $\mathbf{y}$ diffèrent.

Les fonctions de hachage élémentaires $h_i : \mathcal{D} \to \{0,1\}$, $h_i(\mathbf{x}) = x_i$ (i-ème bit de $\mathbf{x}$, voir la figure suivante) forment une famille de fonctions (r_1, r_2, p_1, p_2)-sensibles. Ces fonctions sont employées pour m très élevé et un nombre de fonctions de hachage n comparativement faible dans chaque table de hachage.

$$\begin{array}{c} \downarrow \text{position } i \\ 110\ldots0\,1\,1\ldots010 \\ \underbrace{100\ldots0\,1\,0\ldots100}_{m \text{ bits}} \end{array} \tag{3.3}$$

Similarité entre ensembles finis. Dans de nombreux cas, les données (observations) sont des sous-ensembles d'un grand ensemble fini. Par exemple, un texte est un sous-ensemble de l'ensemble des mots d'une langue. Le profil d'achat d'un client est un sous-ensemble de l'ensemble des articles disponibles (dans le présent et le passé). Parfois, les ensembles sont représentés à travers leurs fonctions caractéristiques et comparés grâce à la distance de Hamming (voir ci-dessus). Dans d'autres cas, les occurrences des éléments sont *pondérées* et les comparaisons font appel à la distance cosinus (nous y reviendrons dans le chapitre suivant). Souvent, les (sous-)ensembles sont comparés directement grâce à *l'indice* (ou la similarité) *de Jaccard*.

Considérons un ensemble total $\mathcal{E}$, le domaine qui nous intéresse est $\mathcal{D} = \mathcal{P}(\mathcal{E})$, l'ensemble des *parties* de $\mathcal{E}$. L'indice de Jaccard entre deux sous-ensembles quelconques $\mathcal{A}, \mathcal{B} \in \mathcal{P}(\mathcal{E})$ est

$$s_J(\mathcal{A}, \mathcal{B}) = \frac{|\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A} \cup \mathcal{B}|} \tag{3.4}$$

c'est à dire le rapport entre le nombre d'éléments communs entre $\mathcal{A}$ et $\mathcal{B}$ et le nombre d'éléments de leur réunion. Cet indice varie entre 0 (aucun élément commun) et 1 ($\mathcal{A} = \mathcal{B}$). On note que $d_J(\mathcal{A}, \mathcal{B}) = 1 - s_J(\mathcal{A}, \mathcal{B})$ est une *métrique* sur $\mathcal{P}(\mathcal{E})$. Des fonctions LSH adaptées aux ensembles peuvent être définies sur la base de cet indice de Jaccard.

On fixe un ordre des éléments de $\mathcal{E}$ et on note par π une permutation des éléments de $\mathcal{E}$. La figure suivante montre un exemple de permutation pour un ensemble dont les éléments sont notés par $a, b, c, d \ldots$:

$$\pi = \begin{pmatrix} a & b & c & d & \ldots \\ c & d & a & b & \ldots \end{pmatrix} \tag{3.5}$$

Les fonctions de hachage élémentaires $h_\pi : \mathcal{P}(\mathcal{E}) \to \mathcal{E}$, $h_\pi(\mathcal{A}) = \min \pi(\mathcal{A})$, forment une famille de fonctions (r_1, r_2, p_1, p_2)-sensibles. Nous avons noté par $\min \pi(\mathcal{A})$ l'élément de $\mathcal{A}$ qui se retrouve premier après cette permutation.

Le tableau suivant montre le résultat de la permutation considérée ci-dessus sur trois ensembles $\mathcal{A}, \mathcal{B}, \mathcal{C}$. Ainsi, $\min \pi(\mathcal{A}) = c$, $\min \pi(\mathcal{B}) = c$ et $\min \pi(\mathcal{C}) = d$.

$\mathcal{E}$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\pi(\mathcal{E})$	$\pi(\mathcal{A})$	$\pi(\mathcal{B})$	$\pi(\mathcal{C})$
a	1	0	0	c	1	1	0
b	0	0	1	d	0	0	0
c	1	1	0	a	1	0	0
d	0	0	0	b	0	0	1
...				...			

Examinons de plus près la probabilité de collision. Soit x le nombre d'éléments *communs* entre $\mathcal{A}$ et $\mathcal{B}$, c'est à dire $|\mathcal{A} \cap \mathcal{B}|$. Soit y le nombre d'éléments *spécifiques* à $\mathcal{A}$ ou à $\mathcal{B}$, c'est à dire $|(\mathcal{A} - \mathcal{B}) \cup (\mathcal{B} - \mathcal{A})|$. Alors, l'indice de Jaccard entre $\mathcal{A}$ et $\mathcal{B}$ est $s_J(\mathcal{A}, \mathcal{B}) = \frac{x}{x+y}$. Aussi, pour toute fonction h_π, la probabilité de trouver en premier après la permutation π un élément commun plutôt qu'un élément spécifique est $\frac{x}{x+y}$. Donc, $p(h_\pi(\mathcal{A}) = h_\pi(\mathcal{B})) = \frac{x}{x+y}$. Ainsi, pour toute fonction h_π et quels que soient les ensembles $\mathcal{A}, \mathcal{B} \in \mathcal{P}(\mathcal{E})$, **la probabilité de collision est égale à l'indice de Jaccard** entre les deux ensembles.

Amplification de fonctions *LSH*

Pour la comparaison des ensembles, examinons l'impact du regroupement de n fonctions de hachage choisies aléatoirement de façon indépendante ; le *hash* résultant est la concaténation des *hash* des fonctions regroupées. Considérons deux ensembles quelconques $\mathcal{A}, \mathcal{B}$, on note par s l'indice de Jaccard entre les deux ensembles, $s = s_J(\mathcal{A}, \mathcal{B})$. Deux ensembles sont en collision dans la table (ont un même *hash* résultant) s'ils sont en collision par rapport la première fonction ET par rapport à la deuxième ET ... ET par rapport à la dernière, donc par rapport à chacune des n fonctions indépendantes. La probabilité de collision dans la table est donc s^n ; comme $0 \leq s \leq 1$, $s^n \leq s$.

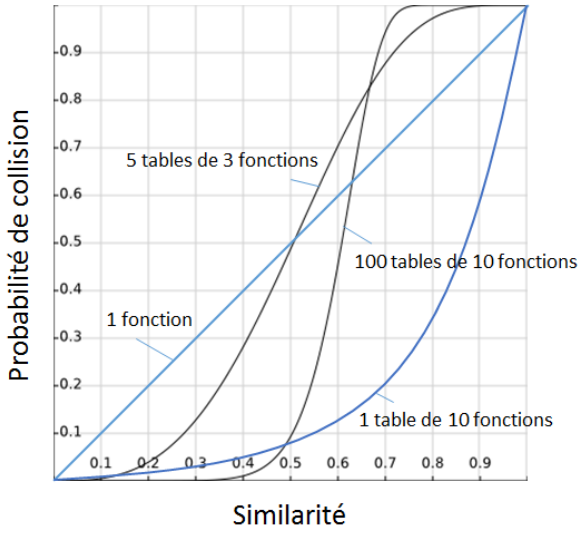
Intéressons-nous maintenant à l'utilisation de t tables, obtenues à partir de fonctions élémentaires indépendantes (chaque table à partir de n fonctions). Deux ensembles $\mathcal{A}, \mathcal{B}$ (d'indice de Jaccard s) sont en collision par rapport à cet ensemble de tables s'ils sont en collision dans la première OU dans la deuxième OU ... OU dans la dernière table, donc dans *au moins une* des t tables. Dans chaque table, la probabilité de collision est s^n (comme montré plus haut). La probabilité de *non collision* dans une table est alors $1 - s^n$. Les tables étant indépendantes, la probabilité de *non collision* dans aucune des t tables est $(1 - s^n)^t$. En conséquence, la probabilité de *collision* dans au moins une table est $1 - (1 - s^n)^t$.

Le graphique suivant montre la probabilité de collision ($\in (0, 1)$) en fonction de $s_J(\mathcal{A}, \mathcal{B})$ ($\in (0, 1)$) pour l'utilisation de :

1. Une seule fonction élémentaire : $P_c = s$.
2. Une table constituée de 10 fonctions élémentaires indépendantes : $P_c = s^{10}$.
3. Cinq tables, chacune constituée de 3 fonctions élémentaires indépendantes : $P_c = 1 - (1 - s^3)^5$.
4. Cent tables, chacune constituée de 10 fonctions élémentaires indépendantes : $P_c = 1 - (1 - s^{10})^{100}$.

Toutes ces solutions sont (r_1, r_2, p_1, p_2)-sensibles, pour le voir il suffit de fixer $r_1 < r_2$ et de lire $p_1 > p_2$ sur la courbe correspondante (noter que r désigne ici une distance, or si s est l'indice de Jaccard alors $r = 1 - s$). En revanche, plus le nombre de fonctions par table (n) et le nombre de tables (t) augmente, plus la courbe s'applatit pour s faible et pour s élevé, c'est à dire plus la probabilité de collision approche 0 pour les ensembles dissimilaires et 1 pour les ensembles similaires. On tend vers un **effet de seuil** sur la similarité. Cela a un impact positif à la fois sur

1. la précision des résultats : la probabilité de collision entre données dissimilaires étant proche de 0, les résultats contiendront peu de « faux positifs » (données non similaires à la requête, retournées) et
2. le rappel : la probabilité de collision entre données similaires étant proche de 1, il y a peu de « faux négatifs » (données similaires à la requête, non retournées).



Cette méthode d'**« amplification » de fonctions LSH** est générale. Considérons une famille de fonctions de hachage (r_1, r_2, p_1, p_2)-sensibles (avec $r_2 > r_1 > 0$, $p_1 > p_2 > 0$), voir la définition dans les équations (**??**). L'objectif de l'amplification est de rapprocher p_2 (la probabilité de collision entre données dissimilaires) de 0 et p_1 (la probabilité de collision entre données similaires) de 1.

Le regroupement de n fonctions dans de nouvelles fonctions h_{AND} (appelées jusqu'ici tables de hachage) revient à faire un ET logique entre les collisions par rapport aux n fonctions : $h_{\text{AND}}(x) = h_{\text{AND}}(y)$ si et seulement si $h_i(x) = h_i(y)$ pour tout i, $1 \leq i \leq n$. On constate que ces nouvelles fonctions h_{AND} sont (r_1, r_2, p_1^n, p_2^n)-sensibles.

Faire la réunion des résultats de t fonctions revient à un OU logique entre les t fonctions : $h_{\text{OR}}(x) = h_{\text{OR}}(y)$ si et seulement si $h_i(x) = h_i(y)$ pour au moins une valeur de i, $1 \leq i \leq n$. Les nouvelles fonctions h_{OR} sont $(r_1, r_2, 1-(1-p_1)^t, 1-(1-p_2)^t)$-sensibles.

Enfin, faire la réunion des résultats de t fonctions h_{AND} (c'est à dire la réunion des résultats de plusieurs tables de hachage) engendre de nouvelles fonctions $h_{\text{OR, AND}}$ qui sont $(r_1, r_2, 1-(1-p_1^n)^t, 1-(1-p_2^n)^t)$-sensibles.

CHAPITRE 4

Cours - Recherche par similarité. Application aux systèmes de recommandation

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/4similariteRecommandation.pdf)]

Dans la première partie de ce chapitre nous examinerons de plus près la recherche et la jointure par similarité, en poursuivant d'abord l'étude de LSH par des fonctions adaptées à la distance cosinus et en considérant ensuite les phénomènes liés à la « malédiction de la dimension » (*curse of dimensionality*), ainsi que leur impact sur l'utilisation de la similarité.

Dans la seconde partie nous nous intéresserons aux systèmes de recommandation, qui sont une application importante de l'analyse et modélisation de données massives. Après une étude de l'utilisation dans ce contexte de méthodes basées sur la similarité, nous présenterons des méthodes plus élaborées qui emploient des factorisations matricielles.

Recherche et jointure par similarité

La similarité entre données a de nombreuses applications directes et intervient également dans des opérations de modélisation à partir de données ou de prise de décision. Quelques exemples d'applications directes :

1. Un outil de recommandation de livres peut employer deux stratégies pour faire des propositions à un utilisateur. Une première stratégie consiste à identifier des utilisateurs dont le « profil » (issu par ex. de notes données par l'utilisateur à des ouvrages classiques, de notes données à des lectures proposées par l'outil, de caractéristiques socio-professionnelles, de liens d'amitié, etc.) est similaire au profil de l'utilisateur ciblé ; les lectures appréciées par ces utilisateurs similaires seront proposées à l'utilisateur ciblé. Une seconde stratégie consiste à proposer à l'utilisateur des ouvrages similaires (présentant des similarités de genre littéraire ou de style, ou lus et appréciés par les mêmes utilisateurs) aux ouvrages qu'il a appréciés. Nous reviendrons sur les systèmes de recommandation dans la seconde partie de ce chapitre.
2. Un outil de recherche de plagiat doit identifier, dans une base locale ou sur le web, la ou les sources exploitées de façon directe dans la réalisation d'un nouveau contenu (texte, image, vidéo, audio). Le contenu est segmenté et, pour chaque segment, on cherche des sources potentielles, c'est à dire des contenus antérieurs très similaires.
3. Un outil d'agrégation de contenus doit détecter les *variantes* d'un même contenu issues de sources différentes afin d'en faire une synthèse (plutôt que de les présenter comme des contenus indépendants). Pour déterminer si un contenu est une variante d'un autre contenu on définit une mesure de similarité adaptée et une valeur seuil (éventuellement dépendant du contexte) sur cette mesure.

Aussi, comme nous l'avons déjà remarqué au début du chapitre précédent, la similarité entre données peut être utilisée à la fois dans la construction d'un modèle décisionnel et dans la prise de décision (avec ou sans modèle).

Deux opérations de base exploitent la similarité directement : la recherche par similarité et la jointure par similarité. La recherche par similarité doit retourner les données similaires à une donnée « requête ». Dans les exemples qui précèdent, cette opération est employée par l'outil de recommandation (lorsqu'un utilisateur se connecte, pour identifier les utilisateurs similaires ou les ouvrages similaires) et par l'outil de recherche de plagiat (afin d'identifier les éventuelles sources pour les différents segments de contenu).

La jointure par similarité doit retourner les *paires* de données très similaires (il n'y a pas de donnée « requête »). Dans les exemples qui précèdent, la jointure par similarité est employée par l'outil d'agrégation pour identifier, dans une base qui regroupe les sources disponibles, les contenus redondants, c'est à dire les contenus pour lesquels des variantes sont également présentes dans la base. La jointure peut également être employée par l'outil de recommandation pour faire des propositions de façon *proactive* à l'ensemble des utilisateurs (connectés ou non).

Les méthodes du hachage sensible à la similarité (LSH) peuvent être employées (comme d'autres méthodes d'indexation multidimensionnelle ou métrique) pour réduire l'ordre de complexité de le recherche et de la jointure par similarité. Avant d'examiner les difficultés rencontrées par les index lorsque la dimension des données est très élevée, poursuivons l'étude de LSH en nous intéressant aux fonctions adaptées à la distance cosinus.

LSH pour la distance cosinus

Les données peuvent être décrites par un nombre élevé de variables de nature différente :

1. Variables « ensemble » : chaque valeur d'une telle variable est un sous-ensemble d'un ensemble plus large. Par exemple, une variable peut correspondre à la description textuelle des articles en vente et une valeur de cette variable est le texte descriptif d'un article, donc un ensemble de mots de la langue. D'autres exemples de telles variables sont l'historique d'achat des clients (vu comme un simple ensemble d'articles acquis), ou la description de chaque ouvrage en vente par un ensemble d'étiquettes fournies par les utilisateurs (*tags*). La valeur que prend une telle variable pour une donnée particulière peut correspondre à un ensemble très grand (par ex., les mots d'un texte très long) ou très petit (par ex., les *tags* associés à un nouvel ouvrage). Les valeurs d'une variable étant des sous-ensembles d'un ensemble plus grand, il est possible de les comparer grâce, par exemple, à l'indice de Jaccard ou à la distance qui en est issue (voir le chapitre précédent). Une valeur (qui est un sous-ensemble) peut être représentée par la liste des éléments ou par la fonction caractéristique du sous-ensemble.
2. Variables nominales (à modalités) : une telle variable prend des valeurs dans un ensemble fini (et souvent d'assez faible cardinalité), des valeurs de la variable peuvent être identiques ou différentes mais nous ne pouvons pas les comparer d'une autre façon. Par exemple, une œuvre littéraire peut être caractérisée par une variable « genre littéraire » (classique, policier, SF, etc.), par une variable « pays de publication », un utilisateur par le navigateur web utilisé, par le mode de paiement, etc. Une valeur est en général représentée par un vecteur dont la dimension est égale au nombre de modalités de la variable et dont toutes les composantes sont 0 sauf celle correspondant à la modalité présente, qui est 1.
3. Variables quantitatives : les valeurs d'une telle variable sont des éléments d'un ensemble (souvent $\mathbb{R}$ ou $\mathbb{N}$) sur lequel on peut définir différentes métriques, un ordre total, etc., ces valeurs peuvent donc être comparées de multiples façons. Par exemple, le nombre de pages d'un ouvrage, le montant total déjà dépensé par un client, etc. Une valeur d'une variable quantitative est en général représentée telle quelle.

Pour comparer des données (observations) décrites par plusieurs types de variables il faut disposer d'une métrique qui intègre toutes les contributions dans une mesure globale et qui présente une certaine flexibilité. La distance issue de l'indice de Jaccard est adaptée aux variables « ensemble » et aux variables nominales mais pas aux variables quantitatives. Par ailleurs, cette distance ne peut pas prendre en compte d'éventuelles différences de pondération entre des éléments des ensembles. La distance euclidienne classique est souvent utilisée pour des variables quantitatives mais ne permet pas de comparer directement de grands ensembles avec de petits ensembles (représentés par leurs fonctions caractéristiques) comme valeurs de variables « ensemble », une normalisation préalable est nécessaire. La distance cosinus est souvent employée pour cumuler les contributions des différents types de variables décrivant les données. Pour chaque donnée (observation), les représentations des différentes variables sont directement concaténées dans un vecteur de dimension élevée.

Pour $\mathbf{x}, \mathbf{y} \in \mathcal{D} \subset \mathbb{R}^m$, la **distance cosinus** est $d_{\cos}(\mathbf{x}, \mathbf{y}) = \arccos \frac{\mathbf{x}^T \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$, c'est à dire *l'angle* (mesuré en degrés ou en radians) entre les deux vecteurs $\mathbf{x}, \mathbf{y}$. Cette distance permet d'intégrer facilement des pondérations pour les différentes composantes des vecteurs (par exemple, pour les différents mots employés dans un texte, comme nous le verrons de plus près dans le chapitre suivant).

On peut remarquer que la normalisation est intégrée au calcul de la distance cosinus, mais cela peut parfois s'avérer nocif. Si pour une variable « ensemble » certaines données ont des valeurs de cardinalité très élevée et d'autres des valeurs de cardinalité très faible, cette normalisation globale peut nuire à la prise en compte des autres variables. Dans un tel cas il est nécessaire de normaliser séparément les composantes correspondant aux variables « ensemble ».

Aussi, suivant la méthode d'analyse ou de modélisation décisionnelle employée, il peut être nécessaire de choisir des pondérations relatives pour les différentes variables (ou types de variables).

LSH pour la distance cosinus. Considérons les fonctions de hachage élémentaires $h \in \mathcal{H}_{\cos}$, $h : \mathbb{R}^m \to \{0, 1\}$ définies par

$$h_{\mathbf{v}}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x}^T \cdot \mathbf{v} \geq 0 \\ 0 & \mathbf{x}^T \cdot \mathbf{v} < 0 \end{cases} \tag{4.1}$$

avec $\mathbf{v} \in \mathbb{R}^m$ tiré suivant la loi uniforme sur l'hypersphère unité ($\|\mathbf{v}\| = 1$). L'ensemble $\mathcal{H}_{\cos}$ est un ensemble de fonctions de hachage $(r_1, r_2, 1 - \frac{r_1}{180}, 1 - \frac{r_2}{180})$-sensibles (les angles et les distances cosinus entre vecteurs, donc r_1, r_2 également, sont ici mesurés en degrés). Une telle fonction élémentaire associe une valeur de *hash* égale à 0 aux données situées d'un côté de l'hyperplan de vecteur normal $\mathbf{v}$ (et passant par l'origine des axes) et à 1 aux données situées de l'autre côté (voir la figure suivante).

Pour justifier le résultat concernant l'ensemble $\mathcal{H}_{\cos}$, nous remarquerons que $h_{\mathbf{v}}(\mathbf{x}) \neq h_{\mathbf{v}}(\mathbf{y})$ si et seulement si l'hyperplan de vecteur orthogonal $\mathbf{v}$ passe entre $\mathbf{x}$ et $\mathbf{y}$. La probabilité pour que cela arrive est $\frac{d_{\cos}(\mathbf{x},\mathbf{y})}{180}$, donc la probabilité de collision ($h_{\mathbf{v}}(\mathbf{x}) = h_{\mathbf{v}}(\mathbf{y})$) est $1 - \frac{d_{\cos}(\mathbf{x},\mathbf{y})}{180}$. Si $d_{\cos}(\mathbf{x}, \mathbf{y}) < r_1$ alors la probabilité de collision est supérieure à $p_1 = 1 - \frac{r_1}{180}$, si $d_{\cos}(\mathbf{x}, \mathbf{y}) \geq r_2$ alors la probabilité de collision est inférieure ou égale à $p_2 = 1 - \frac{r_2}{180}$.

L'exemple suivant illustre la composition de 4 fonctions élémentaires de $\mathcal{H}_{\cos}$ dans une même table de hachage (ou fonction composée $h_{\texttt{AND}}$) pour des données de $\mathbb{R}^2$:

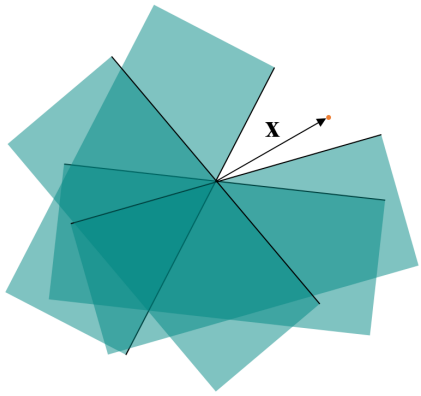


Fig. 4.1 – Composition de 4 fonctions LSH pour la distance cosinus

Nous avons vu dans le chapitre précédent comment LSH était employé pour réduire l'ordre de complexité de la recherche par similarité (et, par conséquent, d'autres opérations qui peuvent faire appel à la recherche par similarité). Regardons maintenant comment est réalisée la jointure par similarité avec LSH.

Jointure par similarité avec LSH

Examinons d'abord l'auto-jointure par similarité, la solution sera ensuite généralisée à la jointure. Étant donné un ensemble $\mathcal{D}$ à N éléments et un seuil de distance θ, l'auto-jointure par similarité consiste à trouver les paires d'éléments de $\mathcal{D}$ qui sont à une distance inférieure au seuil, c'est à dire $K_\theta = \{(\mathbf{x}, \mathbf{y}) | \mathbf{x}, \mathbf{y} \in \mathcal{D}, d(\mathbf{x}, \mathbf{y}) \leq \theta\}$.

La figure suivante présente un exemple avec $\mathcal{D} \subset \mathbb{R}^2$; les traits rouges identifient les paires de points $\mathbf{x}, \mathbf{y}$ pour lesquels $d(\mathbf{x}, \mathbf{y}) \leq \theta\}$.

LSH permet de réduire l'ordre de complexité de l'auto-jointure en évitant de calculer la distance entre des données qui ne sont pas suffisamment similaires. Plus précisément, une auto-jointure par similarité utilisant *LSH* se déroule suivant trois étapes :

1. Construction des fonctions de hachage. Une famille de fonctions *LSH* élémentaires est choisie suivant la nature des données et la distance employée. Ensuite, l'« amplification » est réglée en fonction du seuil de distance θ, ainsi que du coût de calcul et de stockage qui augmentent avec n (nombre de fonctions par table de hachage) et avec t (nombre de tables). On obtient la fonction $h_{\texttt{OR}, \texttt{AND}}$ à utiliser.

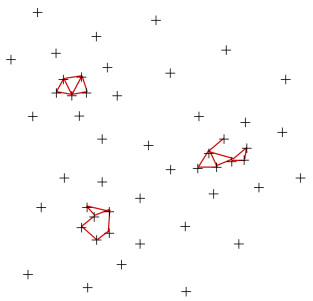


Fig. 4.2 – Auto-jointure par similarité

2. Application de la fonction de hachage à toutes les données. Pour chaque valeur de $h_{\texttt{OR},\texttt{AND}}$, les données pour lesquelles le *hash* a cette valeur forment un *bucket* (une page).
3. Auto-jointure par similarité *à l'intérieur* de chaque *bucket* non vide : calcul des distances $d(\mathbf{x}, \mathbf{y})$ pour tout $\mathbf{x} \neq \mathbf{y}$, $\mathbf{x}, \mathbf{y} \in$ *bucket*.

Avec un calcul exhaustif des distances, la complexité de l'auto-jointure par similarité est $\frac{N(N-1)}{2}$. Avec *LSH*, la complexité est $O(N)$ pour l'étape (2), alors que pour l'étape (3) la complexité dépend de la cardinalité du résultat (souvent $O(N)$ également). Rappelons que l'utilisation de *LSH* produit en général une solution *approximative*. Plus précisément, l'étape (3) éliminant les éventuels faux positifs (données éloignées mais trouvées quand même en collision avec la fonction *LSH* considérée), restera le problème des faux négatifs (données très similaires mais qui ne sont pas en collision par *LSH*). La précision sera donc égale à 1 mais le rappel en général inférieur à 1.

Pour illustrer l'auto-jointure par similarité avec *LSH*, nous présentons brièvement un exemple issu de *[PCB08]* (page 131). L'objectif du travail était de trouver, dans une base de vidéos, des séquences *dupliquées avec transformations*. La figure suivante montre une trame vidéo originale (à gauche) et une trame vidéo qui est une copie transformée de la première (le personnage de la première a été incrusté dans une autre vidéo).

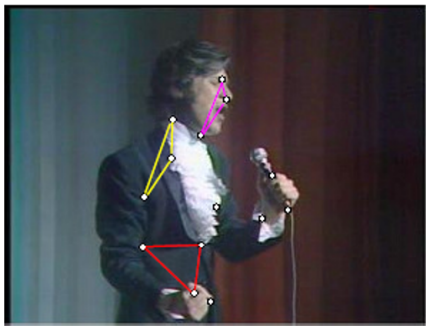


Fig. 4.3 – Image originale

Fig. 4.4 – Image transformée (incrustation)

Les données sont les trames des vidéos de la base (sélectionner env. une trame par seconde suffit), décrites par un ensemble de descripteurs photométriques locaux de « points d'intérêt » (correspondant globalement à des « coins » dans l'image) détectés automatiquement. La métrique utilisée compte les triplets similaires de points d'intérêt

entre les images (trames vidéo) comparées. Une famille de fonctions *LSH* spécifiques a été mise au point pour ce problème et est basée sur l'indice de Jaccard appliqué aux ensembles de triplets de points, avec la prise en compte d'une caractéristique géométrique simple du triplet. La figure suivante montre les résultats obtenus sur les 63 heures de vidéo qui cumulent les 1000 premières réponses de YouTube à une requête avec le mot clé « Madonna ». Chaque composante connexe correspond à des séquences similaires deux à deux. En haut à gauche le graphe est directement issu des séquences vidéo similaires, alors qu'en bas à droite le graphe est issu de vidéos entières entre lesquelles on trouve des séquences similaires ; le point rouge correspond à une compilation vidéo.

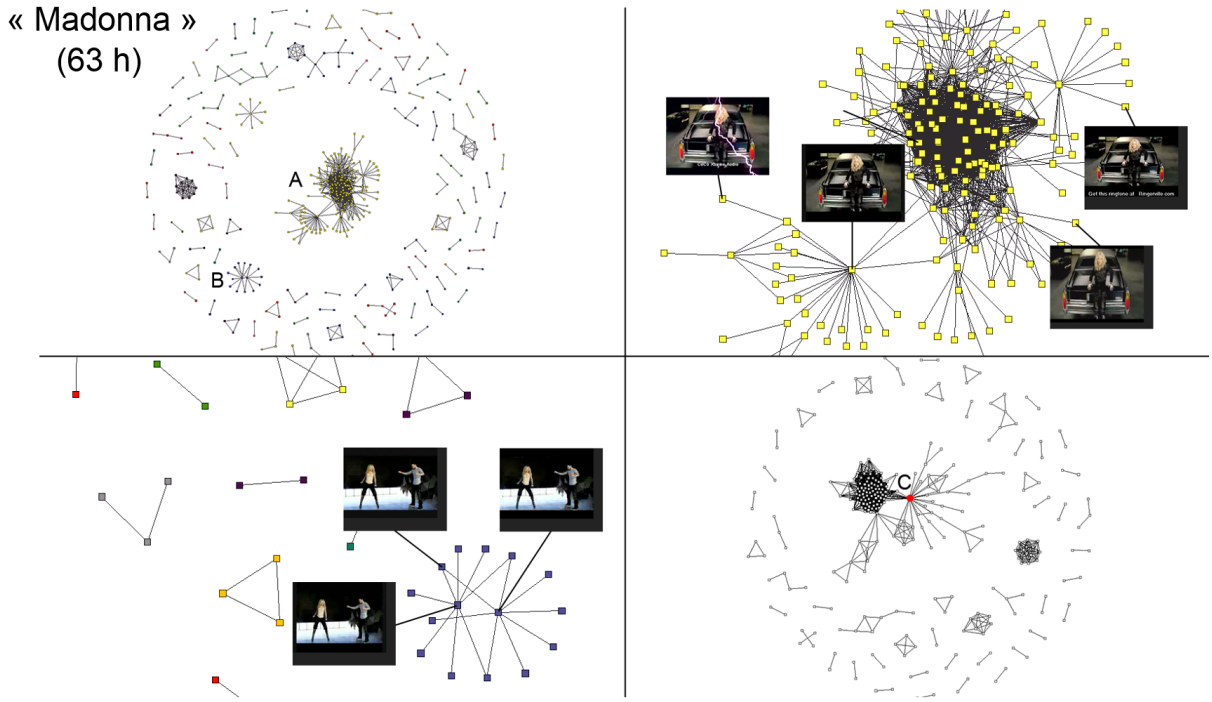


Fig. 4.5 – Auto-jointure par similarité avec LSH pour l'extraction de quasi-duplicats dans une base de vidéos

L'utilisation de *LSH* peut être facilement généralisée au traitement des *jointures* par similarité. Une jointure entre $\mathcal{D}_1$ (de cardinalité N_1) et $\mathcal{D}_2$ (de cardinalité N_2), avec un seuil de distance θ, permet de trouver les paires d'éléments des deux ensembles qui sont à une distance inférieure au seuil $K_\theta = \{(\mathbf{x}, \mathbf{y}) | \mathbf{x} \in \mathcal{D}_1, \mathbf{y} \in \mathcal{D}_2, d(\mathbf{x}, \mathbf{y}) \leq \theta\}$. La jointure avec *LSH* suit les mêmes trois étapes indiquées pour l'auto-jointure :

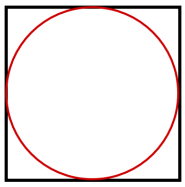
1. Construction des fonctions de hachage. Pour qu'une distance soit calculée entre $\mathbf{x} \in \mathcal{D}_1$ et $\mathbf{y} \in \mathcal{D}_2$, les deux ensembles doivent être des sous-ensembles d'un même espace métrique. Une même fonction $h_{\text{OR, AND}}$ sera employée pour les deux espaces.
2. Application de la fonction de hachage à toutes les données de $\mathcal{D}_1$ et $\mathcal{D}_2$.
3. Jointure par similarité *à l'intérieur* de chaque *bucket* (non vide) : calcul des distances $d(\mathbf{x}, \mathbf{y})$ pour tout $\mathbf{x} \in \mathcal{D}_1, \mathbf{y} \in \mathcal{D}_2$, avec $\mathbf{x}, \mathbf{y} \in$ *bucket*.

Malédiction de la dimension

Les données massives sont souvent caractérisées par un nombre élevé de variables, dont certaines peuvent exiger un nombre élevé de dimensions pour être représentées. C'est le cas, par exemple, pour une variable nominale qui a de nombreuses modalités ou pour une variable « ensemble » pour laquelle l'ensemble total est de grande cardinalité.

La grande dimension de la représentation des données engendre un certain nombre de difficultés regroupées sous le nom de « malédiction de la dimension » (ou « fléau de la dimension », *curse of dimensionality*). Ces difficultés peuvent être majeures et concernent aussi bien la modélisation statistique des données que l'efficacité des algorithmes d'organisation ou de recherche dans ces données. Nous résumons ici quelques-unes de ces difficultés :

1. A nombre de données fixé, la densité diminue exponentiellement avec la dimension. Les données deviennent « rares » ou « isolées » dans l'espace. Il n'est plus possible d'estimer la densité de façon fiable et différents tests statistiques deviennent inexploitables. Le nombre de données devrait augmenter de façon exponentielle avec la dimension pour conserver les capacités de modélisation.
2. Les données uniformément distribuées dans des volumes en dimension d sont proches des hypersurfaces externes (de dimension $d-1$). Par exemple, si on considère une (hyper-)sphère inscrite dans un (hyper-)cube, comme dans la figure suivante (illustration en 2D), le rapport entre le volume de l'hypersphère et le volume de l'hypercube diminue rapidement avec l'augmentation de la dimension, comme indiqué dans le tableau qui suit. Cela implique que la plupart des données de l'hypercube *ne sont pas* dans l'hypersphère mais plutôt dans les « coins » de l'hypercube. Supposons qu'on utilise *LSH* pour la distance euclidienne afin de retourner les données situées dans un rayon autour d'une requête qui serait le centre de l'hypersphère. On constate que la plupart des données du même *bucket* que la requête (*bucket* qui sera, dans le meilleur des cas, un hyper-parallélépipède) ne seront pas dans l'hypersphère et devront être filtrées par des calculs de distance pour conserver une bonne précision (éliminer les faux positifs). Quand la dimension augmente, le nombre de données à filtrer sera beaucoup plus grand que celui de données utiles.



Dimension	Volume sphère / volume cube englobant
1	1
2	$0,78732$
4	$0,329707$
6	$0,141367$
8	$0,0196735$
10	$0,00399038$

3. La variance de la distribution des distances entre données diminue avec l'augmentation de la dimension (voir la figure suivante). Cette difficulté, qui est une des manifestations de la « concentration des mesures », peut rendre inexploitable la décision sur la base des *k* plus proches voisins car la représentativité de ces voisins pour une donnée devient comparable à la représentativité des autres données. Aussi, il devient difficile de trouver des regroupements dans les données (les données à l'intérieur d'un groupe ne sont pas tellement plus proches entre elles que des données d'autres groupes), donc l'intérêt de la classification automatique diminue. Enfin, la troisième condition que nous avons mentionnée dans le chapitre précédent (« les données proches de la requête sont bien plus proches que les autres ») devient fausse et les index perdent leur efficacité dans la réduction de la complexité de la recherche.

Il est important de mentionner ici deux facteurs modérateurs de la malédiction de la dimension :

1. La distribution des données a une grande importance dans la gravité de la malédiction de la dimension. Plus la distribution des données est non uniforme, plus élevée est la dimension à partir de laquelle les capacités de modélisation et l'efficacité des index diminuent de façon sensible.
2. La dimension des données peut éventuellement être réduite par différentes méthodes, comme nous l'avons vu dans un cours précédent. La dimension qui compte sera la « dimension intrinsèque » des données, qui peut être bien plus faible que la dimension apparente. Plusieurs définitions formelles existent pour la notion de « dimension intrinsèque », certaines proposent des estimateurs opérationnels.

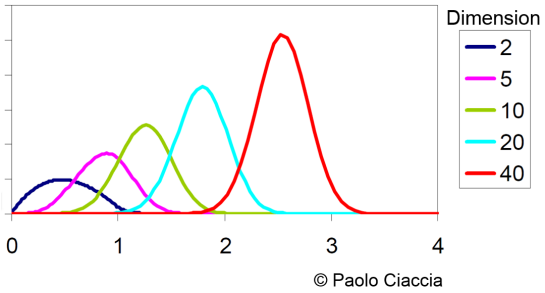


Fig. 4.6 – Concentration des mesures pour des données qui suivent une distribution uniforme

Systèmes de recommandation

Les systèmes de recommandation (SRec) visent à proposer des « articles » (*items*) à des « utilisateurs » (*users*). Employés au départ dans le commerce en ligne, leur domaine d'application ne cesse de s'élargir : musique, films, livres, sites web, blogs, destinations de voyages, applications pour mobiles, publications de recherche, etc. Les SRec intègrent des informations de différents types, issues de plusieurs sources, explicites ou implicites : caractéristiques des utilisateurs et des articles, filtrage collaboratif, liens sociaux entre utilisateurs, données issues des capteurs (par ex. GPS), etc.

Nous examinerons dans la suite de ce chapitre la problématique des SRec et passerons en revue les principales familles de méthodes utilisées pour obtenir des recommandations. Une synthèse assez récente des approches employées pour les SRec en général peut être trouvée dans *[BOH13]* (page 131). Il est également intéressant de lire *[BGL15]* (page 131), une synthèse sur les SRec de publications de recherche.

Le problème et les approches

Dans un SRec nous sommes en présence de deux types d'entités, les utilisateurs et les articles. Chaque utilisateur peut choisir et/ou noter un ou plusieurs articles. Les données disponibles concernant les choix passés (ou les notes) sont représentées sous la forme d'une matrice d'« utilités », chaque utilisateur étant associé à une ligne et chaque article à une colonne, comme dans la figure suivante :

	$A1$	$A2$	$A3$	$A4$	$A5$	$A6$	$A7$	...
$U1$	9						2	
$U2$	2		8				7	
$U3$					4			
...								

Lorsque les données disponibles correspondent à des informations explicites, par ex. issues d'achats effectués ou de notes données, cette matrice est **très** creuse. Si des informations implicites sont également présentes, par ex. issues des durées de visualisation des pages décrivant des articles (un utilisateur a tendance à regarder plus longtemps la description d'un article qui l'intéresse et éventuellement à revenir sur cette description lors de sessions successives), alors la matrice sera (un peu) moins creuse.

L'objectif général d'une méthode de recommandation est de *prédire les valeurs manquantes* de la matrice, c'est à dire les choix que ferait chaque utilisateur s'il devait se prononcer sur chaque article ou les notes qu'il donnerait s'il devait évaluer tous les articles. Naturellement, lorsque la matrice contient des notes, ce sont en général les *valeurs élevées* prédites qui intéressent car elles correspondent (si la prédiction est fiable) à des articles que l'utilisateur apprécierait et que le SRec pourrait donc utilement lui proposer.

Les SRec emploient une des approches suivantes :

1. Recommandation par **similarité de contenu** (*content-based filtering*). Pour ces méthodes, un accès à des *descriptions* des articles est indispensable. Le principe est le suivant : à partir des descriptions des articles choisis ou notés par un utilisateur (ainsi que d'éventuelles informations concernant l'utilisateur directement), un *profil* de l'utilisateur est construit ; sont ensuite proposés à l'utilisateur des articles dont les caractéristiques sont similaires à son profil. Plus le nombre d'article choisis ou notés par un utilisateur est

faible, moins le profil ainsi obtenu et donc les recommandations faites sur cette base seront fiables. La recommandation étant réalisée à partir des *caractéristiques* des articles, il est facile d'identifier des articles de substitution lorsque l'article envisagé n'est plus disponible. En revanche, il est clairement difficile d'extrapoler d'un domaine à un autre. Par exemple, connaître les lectures préférées d'un utilisateur donne peu d'indications sur ses goûts musicaux.

2. Recommandation par **filtrage collaboratif** (*collaborative filtering*). Ces méthodes se basent uniquement sur la *matrice d'utilités*, aucune connaissance intrinsèque des articles n'est nécessaire. La matrice d'utilités permet de définir des similarités entre utilisateurs à partir des articles choisis, ainsi que des similarités entre articles à partir des utilisateurs qui les ont choisis. Le principe est alors de proposer à un utilisateur des articles similaires à ceux qu'il a déjà choisi (ou bien noté), ou alors des articles choisis (ou bien notés) par les utilisateurs similaires. Les similarités n'étant pas liées aux caractéristiques intrinsèques des articles, les prédictions dans un domaine peuvent être faites à partir de données concernant d'autres domaines (l'hypothèse d'une cohérence relative inter-domaines est sous-jacente). En revanche, l'ignorance des caractéristiques des articles rend problématique la substitution d'articles manquants.
3. Recommandation **hybride**. Bien entendu, lorsque l'on possède à la fois des descriptions des articles et une matrice d'utilités il est souhaitable de tirer profit de ces deux sources d'information pour améliorer les prédictions. Les SRec actuels emploient en général des combinaisons de méthodes suivant les deux approches précédentes.

Recommandation par similarité de contenu

Suivant la nature des articles, leur **description** peut inclure des variables diverses. On rencontre souvent des variables ensemble, par ex. pour des films de cinéma les acteurs, les prix obtenus ou l'ensemble de mots d'une description textuelle. Les variables nominales, dont les modalités correspondent souvent aux sous-catégories d'une catégorie, sont également présentes (par ex. le *genre* d'un film). Enfin, des variables quantitatives peuvent avoir leur importance, par ex. pour des films de cinéma le budget de réalisation ou les recettes lors de la sortie en salles.

Les variables qui interviennent dans les descriptions peuvent être pondérées par des méthodes simples, par ex. *term frequency* x *inverse document frequency* (degré de présence x « potentiel discriminant ») pour les éléments d'ensembles, voir le chapitre suivant. D'autres méthodes de pondération tiennent compte d'un « potentiel explicatif » défini par exemple à partir de l'homogénéité des notes données par des utilisateurs similaires. Enfin, des méthodes de pondération plus complexes peuvent être utilisées lorsque, plutôt qu'une simple recherche par similarité, des modèles décisionnels sont mis en œuvre.

Des méthodes de réduction de dimension sont souvent appliquées en présence de variables de type ensemble avec un grand nombre d'éléments (par ex. description textuelle). Cela permet de résumer les variables initiales par un plus petit nombre de variables (révéler des « facteurs »), ce qui diminue en général le « bruit » présent dans les descriptions et réduit la gravité de la malédiction de la dimension.

La recommandation par similarité de contenu peut employer

1. La recherche par similarité : sont proposés à l'utilisateur des articles dont les descriptions sont les plus similaires au profil de l'utilisateur. Dans ce cas, le **profil d'un utilisateur** possède les mêmes variables que les descriptions des articles afin de permettre une comparaison directe avec ces descriptions. La représentation d'un profil est un vecteur qui a la même structure que les représentations des descriptions des articles. Le profil d'un utilisateur est en général obtenu comme la moyenne des descriptions des articles choisis et/ou notés par cet utilisateur. Dans le calcul de la moyenne, les descriptions des articles notés sont pondérées par les notes accordées par l'utilisateur. Des variantes de calcul consistent à approcher le profil des articles bien notés par l'utilisateur tout en l'éloignant des articles qu'il a mal notés. L'utilisation typique est la suivante : lorsqu'un utilisateur se connecte, son profil est employé comme une requête dans la base des articles disponibles et les articles les plus similaires sont affichés. Il est toutefois possible d'utiliser de façon différente les vecteurs qui représentent les descriptions des articles et les profils des utilisateurs. Par exemple, une jointure par similarité entre l'ensemble des descriptions d'articles et l'ensemble des profils d'utilisateurs peut retourner un ensemble de paires <article, utilisateur> très pertinentes qui peuvent faire l'objet d'une campagne promotionnelle passant par l'envoi de courriels (*push*) alors que les utilisateurs ne sont pas connectés.
2. Des modèles décisionnels : sont proposés à l'utilisateur les articles pour lesquels le modèle décisionnel *spécifique à l'utilisateur* prédit les meilleures notes. Dans ce cas, le profil d'un utilisateur est un modèle décisionnel construit à partir des articles choisis et/ou notés par cet utilisateur. Ce modèle peut employer

aussi des variables prédictives spécifiques aux utilisateurs (par ex. catégorie socio-professionnelle, plateforme matérielle et logicielle employée pour se connecter au site, âge, localisation géographique, etc.) et non simplement leurs choix d'articles. L'emploi de modèles décisionnels permet d'obtenir des fonctions de décision (de recommandation) plus complexes que la simple similarité vectorielle. En revanche, la construction d'un modèle décisionnel fiable exige un volume de données plus important *par utilisateur* qu'une simple recherche par similarité. Pour disposer de plus de données, il est préférable de développer un modèle par *groupe d'utilisateurs très similaires* plutôt que par utilisateur individuel.

La disponibilité de descriptions des articles permet de définir des critères *pragmatiques* complémentaires pour améliorer les recommandations, par ex. réduire la redondance des propositions (éviter de proposer plusieurs articles très similaires entre eux) ou augmenter la diversité (proposer des articles appartenant à des « familles » différentes plutôt que plusieurs articles d'une même famille).

Recommandation par filtrage collaboratif

Ces méthodes considèrent que les utilisateurs, comme les articles, sont décrits essentiellement (ou exclusivement) par le contenu de la matrice de données $\mathbf{X}$ (issue des notes ou des choix passés) :

	$A1$	$A2$	$A3$	$A4$	$A5$	$A6$	$A7$	$\ldots$
$U1$	9						2	
$U2$	2		8				7	
$U3$					4			

Chaque utilisateur est associé à une ligne et chaque article à une colonne. On note par n_u le nombre d'utilisateurs et par n_a le nombre d'articles. $\mathbf{X}$ est donc une matrice $n_u \times n_a$. La matrice de données peut subir diverses transformations (par exemple des normalisations) avant d'être utilisée pour obtenir les recommandations.

Nous pouvons mentionner plusieurs familles de méthodes de filtrage collaboratif :

1. Méthodes basées sur la similarité (*memory-based*) : la similarité entre utilisateurs ou entre articles est employée pour obtenir des recommandations. Nous remarquerons que les utilisateurs (lignes de la matrice de données) et les articles (colonnes de la matrice de données) ne sont pas directement comparables.

   La matrice de données est « normalisée » : la moyenne des notes *présentes* devrait être nulle pour chaque utilisateur (ligne de la matrice) afin d'équilibrer les « niveaux d'exigence » des utilisateurs. Les moyennes des notes présentes restent en général différentes entre articles (colonnes de la matrice), ces écarts reflètent des différences de (perception de la) « qualité intrinsèque » entre articles. On parle de « profils » lignes (ou « profils » utilisateurs) et de « profils » colonnes (ou « profils » articles). Pour comparer les lignes ou les colonnes sont employées en général la distance cosinus ou la corrélation linéaire (qui est une mesure de similarité et non une distance). On peut distinguer deux types de méthodes :

1. *User-based* (la similarité est calculée entre utilisateurs) : (i) trouver les utilisateurs les plus « représentatifs » (par ex. les *k* les plus similaires) pour l'utilisateur cible u, ensuite (ii) agréger leurs choix pour faire des propositions à u (par ex. les articles les plus choisis ou les mieux notés par ces k « voisins »). Une difficulté importante de cette approche est la présence d'une grande diversité parmi les utilisateurs : par ex. x apprécie la musique classique et le jazz, alors que u apprécie la musique classique mais pas le jazz ; si la matrice de données contient pour u uniquement des données concernant la musique classique, u recevrait des propositions de jazz (faites à partir de x qui est très similaire à u) qui seraient erronées.
2. *Item-based* (la similarité est calculée entre articles) : (i) répertorier les articles a_i choisis (ou bien notés) par l'utilisateur u, ensuite (ii) proposer à u les articles les plus similaires aux a_i (c'est à dire choisis ou bien notés ensemble par d'autres utilisateurs). Comme le nombre d'utilisateurs est souvent supérieur au nombre d'articles, la dimension des « profils » articles est supérieure à la dimension des « profils » utilisateurs, la malédiction de la dimension peut donc se manifester avec plus de force pour les « profils » articles.

2. Méthodes basées sur un modèle (*model-based*) : un modèle (par ex. catégorisation des utilisateurs et des articles, facteurs explicatifs latents) est obtenu à partir de la matrice de données et sert à la prise de décisions de recommandation. Une telle méthode est présentée avec plus de détails dans la suite.

3. Méthodes hybrides. Ces méthodes sont des combinaisons de méthodes des deux familles précédentes.

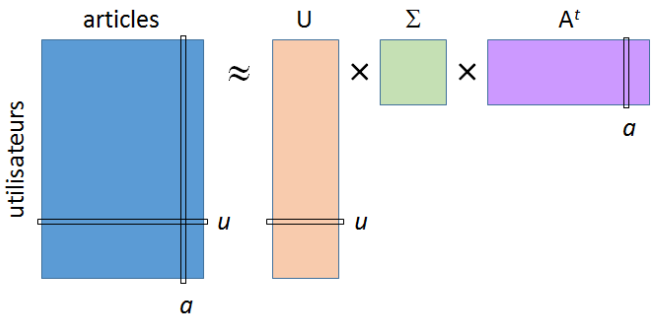
Filtrage collaboratif basé sur la factorisation matricielle

Le principe de ces méthodes est de chercher des *facteurs latents*, en nombre relativement faible (souvent de l'ordre de 10^2), qui « expliquent » le contenu de la matrice de données. Dans ce cadre, les articles comme les utilisateurs sont décrits par des vecteurs de même dimension, cette dimension étant donnée par le nombre de facteurs latents considérés. Dans la description d'un article, les composantes correspondent aux valeurs prises par les facteurs latents respectifs pour cet article. Dans la description d'un utilisateur, les composantes correspondent aux contributions des facteurs latents respectifs à la note que l'utilisateur donnerait à un article.

Une des premières méthodes de factorisation explorées a été la décomposition en valeurs singulières (*Singular Value Decomposition*, SVD) de la matrice de données (ou utilisateurs-articles) $\mathbf{X}$, avec approximation de rang réduit m :

$$\mathbf{X} \approx \mathbf{U} \cdot \Sigma \cdot \mathbf{A}^t \tag{4.2}$$

Ici Σ est une matrice diagonale $m \times m$ qui donne sur sa diagonale principale les poids des m facteurs. Chacune des n_a colonnes de $\mathbf{A}^t$ est la représentation « réduite » (de dimension m, égale au nombre de facteurs) d'un article. Chacune des n_u lignes de $\mathbf{U}$ (de dimension m) est la représentation « réduite » d'un utilisateur. La figure suivante illustre cette factorisation :



Cette méthode souffre d'un problème majeur : la décomposition exige une matrice complète, or la matrice utilisateurs-articles ($\mathbf{X}$) est très creuse, les valeurs absentes sont **manquantes** et non équivalentes à des 0. Une décomposition qui assimilerait les valeurs absentes à 0 donnerait une solution non pertinente.

Il est donc nécessaire de tenir compte exclusivement des données **présentes** dans la matrice. Avec une matrice utilisateurs-articles très creuse le problème est sous-déterminé, une solution de régularisation est nécessaire.

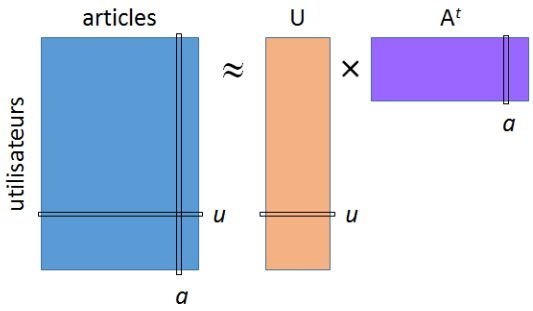
La *factorisation régularisée* (voir par ex. *[Pat07]* (page 131), *[KBV09]* (page 131)) correspond à une famille de méthodes qui cherchent une approximation de rang m réduit tenant compte seulement des valeurs **présentes** dans la matrice $\mathbf{X}$ et incluant une technique de régularisation. Le problème d'optimisation correspondant fréquemment employé est le suivant :

$$\min_{\mathbf{u}_i, \mathbf{a}_j} \sum_{\text{Present}(i,j)} (x_{ij} - \mathbf{u}_i^T \cdot \mathbf{a}_j)^2 + \lambda(\sum_i \|\mathbf{u}_i\|^2 + \sum_j \|\mathbf{a}_j\|^2) \tag{4.3}$$

Cette factorisation est illustrée dans la figure suivante :

Ici $\mathbf{u}_i$ est la représentation « réduite » (de dimension m, égale au nombre de facteurs) d'un utilisateur et $\mathbf{a}_j$ est la représentation « réduite » (de dimension m) d'un article. La constante λ contrôle la régularisation.

Le problème de minimisation est résolu par des algorithmes itératifs comme les moindres carrés alternés (dans Spark, par exemple) ou la descente de gradient stochastique.



Après avoir trouvé les $\mathbf{u}_i$ et $\mathbf{a}_j$ pour tous les utilisateurs et respectivement tous les articles, la prédiction de la note que devrait donner l'utilisateur k à l'article l (note inconnue) est :

$$x_{kl} = \mathbf{u}_k^T \cdot \mathbf{a}_l \tag{4.4}$$

La factorisation régularisée (**??**) et l'algorithme de résolution itérative associé permettent d'intégrer dans le modèle d'autres aspects (voir par ex. *[KBV09]* (page 131)) comme :

1. La modélisation d'un *biais par utilisateur* et d'un *biais par article*. Plutôt que d'employer la solution simple, mentionnée plus haut, de « normalisation » de la matrice de données (imposer, pour chaque ligne, une moyenne nulle pour les notes *présentes* afin d'équilibrer les niveaux d'exigence des utilisateurs), il est possible d'estimer les biais de notation par utilisateur et par article. Le problème d'optimisation correspondant est
   $\min_{\mathbf{u}_i,\mathbf{a}_j,b_\mathbf{u},b_\mathbf{a}} \sum_{\text{Present}(i,j)} (x_{ij} - \mu - b_{\mathbf{u}_i} - b_{\mathbf{a}_j} - \mathbf{u}_i^T \cdot \mathbf{a}_j)^2 + \lambda \left[\sum_i (\|\mathbf{u}_i\|^2 + b_{\mathbf{u}_i}^2) + \sum_j (\|\mathbf{a}_j\|^2 + b_{\mathbf{a}_j}^2)\right]$
   où μ est la moyenne globale des notes présentes, $b_{\mathbf{u}_i}$ le biais pour l'utilisateur i et $b_{\mathbf{a}_j}$ le biais pour l'article j.
2. La modélisation de *niveaux de confiance* dans les notes présentes dans la matrice de données. En effet, il peut être nécessaire de moduler l'impact des différentes notes à partir de connaissances concernant les utilisateurs, le mode d'acquisition de l'information, etc. Le problème d'optimisation est dans ce cas
   $\min_{\mathbf{u}_i,\mathbf{a}_j,b_\mathbf{u},b_\mathbf{a}} \sum_{\text{Present}(i,j)} c_{ij} (x_{ij} - \mu - b_{\mathbf{u}_i} - b_{\mathbf{a}_j} - \mathbf{u}_i^T \cdot \mathbf{a}_j)^2 + \lambda \left[\sum_i (\|\mathbf{u}_i\|^2 + b_{\mathbf{u}_i}^2) + \sum_j (\|\mathbf{a}_j\|^2 + b_{\mathbf{a}_j}^2)\right]$
   où c_{ij} est la confiance dans la note donnée par l'utilisateur i à l'article j ou, plus généralement, une *pondération* (dont la signification n'est pas nécessairement celle de degré de confiance) de cette note.

Filtrage collaboratif avec Spark

La méthode mise en œuvre dans Spark est la factorisation régularisée (**??**) avec une modification de la régularisation suivant *[ZWS08]* (page 131) :

$$\min_{\mathbf{u}_i,\mathbf{a}_j} \sum_{\text{Present}(i,j)} (x_{ij} - \mathbf{u}_i^T \cdot \mathbf{a}_j)^2 + \lambda (\sum_i n_i \|\mathbf{u}_i\|^2 + \sum_j n_j \|\mathbf{a}_j\|^2)$$

où n_i est le nombre total de notes données par l'utilisateur i et n_j est le nombre total de notes reçues par l'article j.

Ce problème de minimisation est résolu par l'algorithme itératif *Alternating Least Squares* (ALS). A chaque itération de l'algorithme on alterne deux phases :

— avec $\mathbf{u}_i$ fixés, $1 \leq i \leq n_u$, on obtient les $\mathbf{a}_j$, $1 \leq j \leq n_a$, comme solution d'un système linéaire ;
— avec $\mathbf{a}_j$ fixés, $1 \leq j \leq n_a$, on obtient les $\mathbf{u}_i$, $1 \leq i \leq n_u$, comme solution d'un système linéaire.

Il est très utile de suivre cet exemple de recommandation de films avec MLlib (présenté au *Spark Summit* en 2014) : https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

CHAPITRE 5

Cours - Classification Automatique

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/5classificationAutomatique.pdf)]

L'identification *automatique* de *groupes* de données similaires dans un (grand) ensemble de données est une composante importante de la fouille de données. La classification automatique (*cluster analysis* ou *clustering* en anglais) cherche à regrouper les données de façon à obtenir des groupes tels que les données sont plus similaires entre elles à l'intérieur d'un même groupe (*cluster* en anglais) qu'entre groupes. Dans la mesure où les notions de similarité et de groupe peuvent être explicitées de multiples façons, de nombreuses méthodes de classification automatique ont été proposées depuis les années 1930. Des synthèses partielles peuvent être trouvées, par exemple, dans *[JMF99]* (page 132) ou *[BBS14]* (page 132).

Après une brève typologie des méthodes de classification automatique, nous nous intéresserons à la mise en œuvre d'algorithmes de classification dans un contexte de données massives.

Une première distinction entre méthodes de classification automatique peut être faite suivant leur objectif. La plupart des méthodes visent à obtenir un *partitionnement* des données, comme dans l'exemple suivant, plus éventuellement de trouver une donnée « représentative » (« prototype ») par groupe :

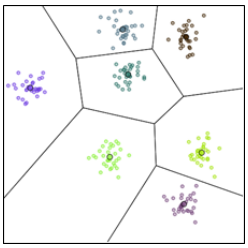


Fig. 5.1 – Exemple de partitionnement de données bidimensionnelles

D'autres méthodes cherchent plutôt à obtenir une hiérarchie de regroupements, qui fournit une information plus riche concernant la structure de similarité des données. Noter qu'à partir d'une telle hiérarchie il est facile d'extraire plusieurs partitionnements, à des niveaux de « granularité » différents, comme dans l'exemple suivant :

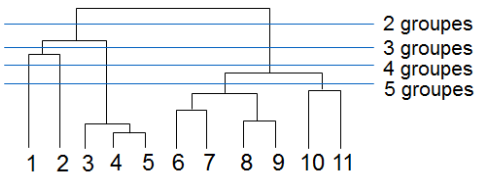


Fig. 5.2 – Exemple de regroupement hiérarchique qui peut servir à obtenir plusieurs partitionnements

Une seconde distinction peut être faite suivant la nature des données à regrouper : données numériques, données catégorielles ou données mixtes. Si la plupart des méthodes visent les données numériques, certaines peuvent

traiter directement des données catégorielles. Toutefois, les données catégorielles sont fréquemment transformées en données numériques (par ex. par une analyse factorielle des correspondances multiples) avant d'être traitées par des méthodes de classification adaptées aux données numériques.

Une autre distinction correspond à la représentation des données. De nombreuses méthodes de classification automatique travaillent sur des représentations vectorielles qui, en plus de différentes métriques, permettent de définir des centres de gravité de groupes, des densités de probabilité, des intervalles, des sous-espaces, etc. Certaines méthodes se satisfont, en revanche, d'une simple structure métrique sur l'espace des données ; cela leur confère une grande généralité.

Une distinction importante concerne la nature des groupes recherchés : sont-il mutuellement exclusifs ou non ? Sont-ils « nets » (*crisp*) ou « flous » (*fuzzy*) ? Si la plupart des méthodes de classification automatique s'intéressent aux partitionnement exclusifs, certaines permettent d'obtenir des groupes non exclusifs (par ex., dans un mélange gaussien, deux lois peuvent présenter une « intersection » significative). Aussi, des extensions floues existent pour des méthodes de classification automatiques bien connues (par ex. *fuzzy c-means* pour *K-means*, voir *[JMF99]* (page 132)). Les extensions floues peuvent être plus robustes que les méthodes « nettes » de départ.

Enfin, une autre distinction importante peut être faite suivant le critère de regroupement. Certaines méthodes cherchent des groupes « compacts » et relativement éloignés entre eux, comme dans cet *exemple de partitionnement* (page 53). D'autres méthodes s'intéressent à des groupes denses (et non nécessairement « compacts ») séparés par des régions moins denses, comme dans l'exemple suivant :

Fig. 5.3 – Exemple de groupes denses séparés par des régions moins denses

Il faut noter que ce critère de regroupement n'est pas toujours explicite, pourtant son impact sur les résultats obtenus est majeur.

Enfin, revenons sur la distinction entre classification automatique et auto-jointure par similarité. Pour un ensemble de données $\mathcal{D}$ et un seuil de distance θ, l'auto-jointure par similarité doit retourner les paires d'éléments de $\mathcal{D}$ qui sont à une distance inférieure au seuil, c'est à dire $K_\theta = \{(\mathbf{x}, \mathbf{y}) | \mathbf{x}, \mathbf{y} \in \mathcal{D}, d(\mathbf{x}, \mathbf{y}) \leq \theta\}$, voir par exemple la figure suivante :

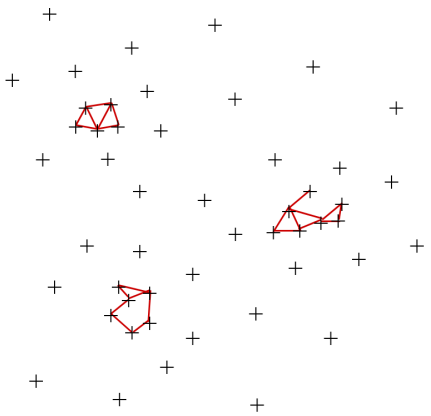


Fig. 5.4 – Exemple d'auto-jointure par similarité ; les traits correspondent aux liens de forte similarité

A partir du graphe résultant il est ensuite possible d'extraire des cliques, des composantes connexes, etc. L'intérêt de cette opération est d'identifier des données hautement similaires, cette forte similarité ayant une signification particulière (par ex. « variantes » d'une même donnée). Les données « isolées », qui n'ont pas de voisin suffisamment proche (au vu de θ), sont simplement ignorées dans les résultats.

La classification automatique, en revanche, vise un regroupement des données par similarité. En général, chaque donnée appartiendra à un groupe, même si elle n'a pas de voisins proches, comme dans la figure suivante. Il faut toutefois noter que certaines méthodes de classification automatique retournent séparément (hors groupes) des données considérées « isolées » dans le sens « mal expliquées » par tous les groupes trouvés.

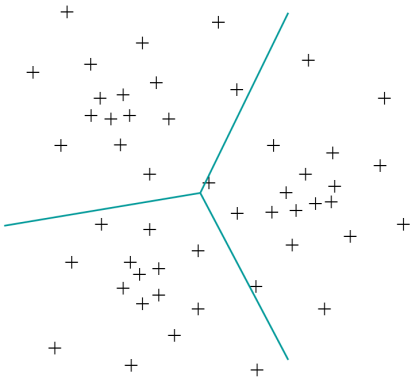


Fig. 5.5 – Exemple de classification automatique ; les traits correspondent aux frontières de séparation entre les trois groupes

K-means

Considérons un ensemble $\mathcal{E}$ de N données décrites par p variables à valeurs dans $\mathbb{R}$ et d une distance sur $\mathbb{R}^p$. Pour regrouper ces données en k groupes disjoints $\mathcal{E}_1, \ldots, \mathcal{E}_k$, inconnus *a priori*, on s'intéresse souvent à un critère qui correspond à la somme des inerties intra-classe des groupes :

$$\phi_{\mathcal{E}}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{E}_j} d^2(\mathbf{x}_i, \mathbf{m}_j) \tag{5.1}$$

avec $\mathcal{C} = \{\mathbf{m}_j, 1 \leq j \leq k\}$ l'ensemble des centres de gravité des k groupes. Pour $\mathcal{E}$ et k donnés, plus la valeur de $\phi_{\mathcal{E}}(\mathcal{C})$ est faible, plus les groupes sont « compacts » autour de leurs centres et donc meilleure est la qualité du partitionnement obtenu.

Trouver le minimum global de la fonction (**??**) est un problème NP-difficile, mais on dispose d'algorithmes de complexité polynomiale dans le nombre de données N qui produisent une solution en général sous-optimale.

Un tel algorithme est l'algorithme des **centres mobiles** décrit ci-dessous :

`Entrées` : ensemble $\mathcal{E}$ de N données de $\mathbb{R}^p$;
`Sorties` : k groupes (*clusters*) disjoints $\mathcal{E}_1, \ldots, \mathcal{E}_k$ et ensemble $\mathcal{C}$ de leurs centres ;

1. Initialization aléatoire des centres $\mathbf{m}_j$, $1 \leq j \leq k$;
2. `tant que` (centres non stabilisés) `faire`
   (a) Affectation de chaque donnée au groupe du centre le plus proche ;
   (b) Remplacement des centres par les centres de gravité des groupes ;

`fin tant que`

On peut montrer que la valeur de $\phi_{\mathcal{E}}(\mathcal{C})$ diminue lors de chacune des deux étapes du processus itératif (affectation de chaque donnée à un groupe, recalcul des centres). Comme $\phi_{\mathcal{E}}(\mathcal{C}) \geq 0$, le processus itératif doit converger. La solution obtenue sera néanmoins un minimum *local*, dépendant de l'initialisation, souvent de valeur $\phi_{\mathcal{E}}(\mathcal{C})$ bien plus élevée que celle correspondant au minimum global (inconnu).

Parfois on appelle k-moyennes (*K-means*) une version stochastique de la méthode des centres mobiles décrite ci-dessous (les entrées et les sorties sont les mêmes que pour l'algorithme précédent) :

1. Initialization aléatoire des centres $\mathbf{m}_j$, $1 \leq j \leq k$;
2. `tant que` (centres non stabilisés) `faire`
   (a) Choix aléatoire d'une des données ;
   (b) Affectation de la donnée au groupe du centre le plus proche ;
   (c) Recalcul des centres pour le groupe rejoint par la donnée et le groupe quitté par la donnée ;

`fin tant que`

Dans cette perspective, l'algorithme des centres mobiles est une variante « par lots » (*batch*) de *K-means*. Toutefois, *K-means* est souvent utilisé comme parfait synonyme de l'algorithme des centres mobiles ; c'est ainsi que ce terme sera employé dans la suite du cours.

Des étapes successives dans l'application de *K-means* à des données issues de 7 lois normales bidimensionnelles peuvent être visualisées dans la figure suivante. La distance euclidienne classique est employée et la classification est réalisée avec avec $k = 7$ centres, dont la position initiale (choisie aléatoirement) est indiquée dans la première image.

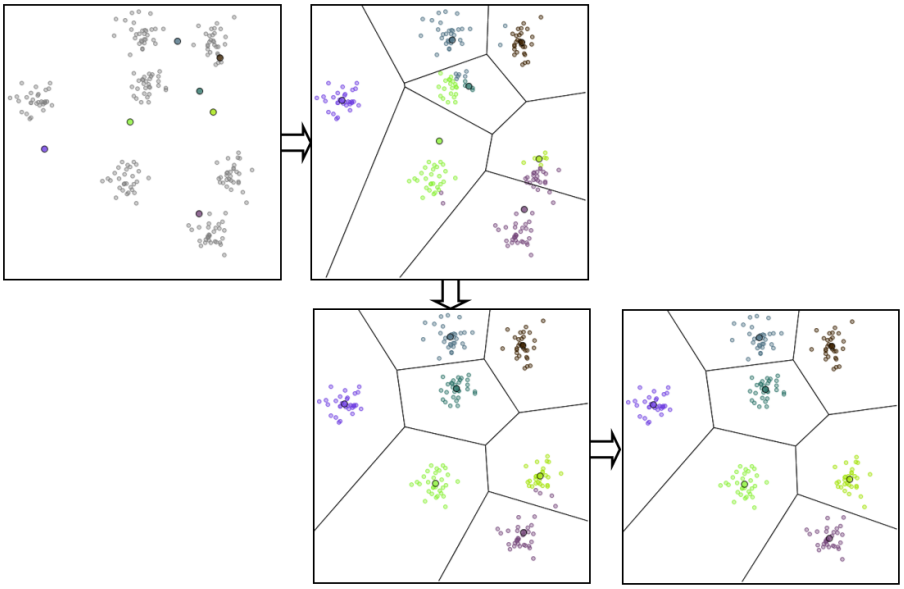


Fig. 5.6 – Etapes successives dans l'application de *K-means*

Dans cet exemple, la solution obtenue est optimale et est atteinte après seulement 2 itérations.

K-means : une implémentation simple MapReduce

L'algorithme *K-means* est facile à exécuter sur une plate-forme distribuée en utilisant le mécanisme MapReduce. Une implémentation simple consiste à assigner aux tâches *Map* l'affectation des données aux groupes et aux tâches *Reduce* le recalcul des centres. Cette implémentation est détaillée dans la suite :

`Entrées` : ensemble $\mathcal{E}$ de N données de $\mathbb{R}^p$;
`Sorties` : k groupes (*clusters*) disjoints $\mathcal{E}_1, \ldots, \mathcal{E}_k$ et ensemble $\mathcal{C}$ de leurs centres ;

1. L'ensemble $\mathcal{E}$ de N données est découpé en fragments distribués aux nœuds de calcul ; un fragment doit tenir dans la mémoire d'un nœud ;
2. Un nœud initialise les k centres ;
3. `tant que` (centres non stabilisés) `faire`
   (a) Transmettre l'ensemble $\mathcal{C}$ des centres à tous les nœuds de calcul ;

(b) Chaque tâche *Map(t)*, pour chaque élément $\mathbf{x}_i$ de son fragment t, trouve le centre le plus proche j ; ensuite, pour chaque centre j ainsi trouvé, génère $(j, (n_{jt}, \tilde{\mathbf{m}}_{jt} = \sum_t \mathbf{x}_i))$, où n_{jt} est le nombre de données du fragment t qui ont comme centre le plus proche le centre j et la somme est faite sur les $\mathbf{x}_i$ du fragment t plus proches du centre j ;

(c) Les paires $(j, (n_{jt}, \tilde{\mathbf{m}}_{jt}))$ sont groupées par j pour les tâches *Reduce* ;

(d) Chaque tâche *Reduce* reçoit toutes les paires correspondant à une valeur de j, calcule $\mathbf{m}_j = \frac{\sum_t \tilde{\mathbf{m}}_{jt}}{\sum_t n_{jt}}$ et stocke le $\mathbf{m}_j$ résultant ;

`fin tant que`

Une variante stochastique peut être facilement obtenue en utilisant des échantillons des données (plutôt que la totalité des données de chaque fragment) dans les tâches *Map*.

Initialisation de *K-means* par *K-means++*

L'initialisation des centres est une étape critique de *K-means*. La figure suivante illustre les résultats obtenus avec *K-means* à partir de trois autres initialisations aléatoires des centres :

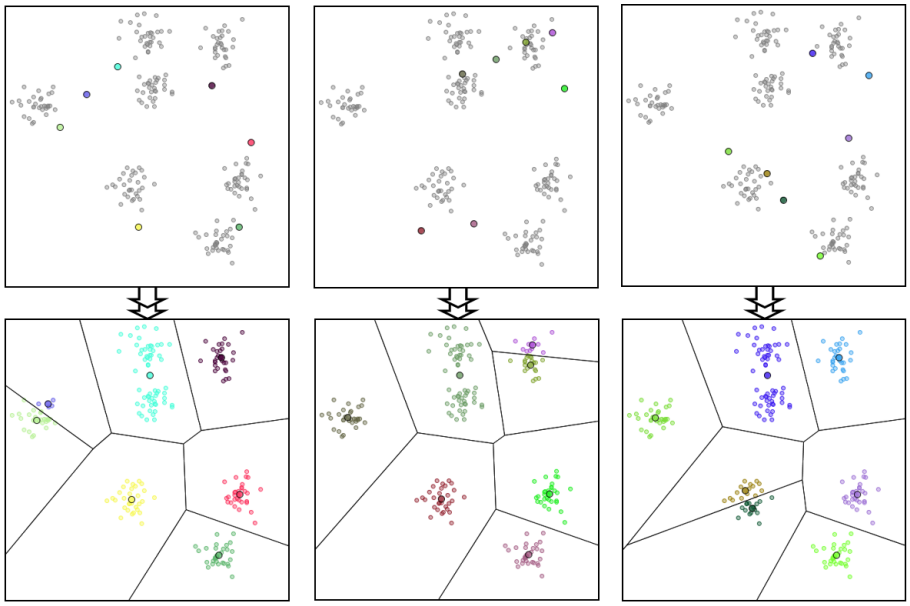


Fig. 5.7 – Résultats différents avec initialisations aléatoires différentes de *K-means*

On constate que ces trois solutions sont sous-optimales : pour chacune, un des groupes « naturels » (on peut en juger visuellement car les données sont bidimensionnelles et peu nombreuses, ce ne sera pas le cas en général) est divisé en deux, et deux des groupes « naturels » sont regroupés. Les valeurs de $\phi_{\mathcal{E}}(\mathcal{C})$ sont plus élevées pour ces trois résultats que pour le résultat optimal illustré plus haut. Exécuter *K-means* plusieurs fois, à partir d'initialisations aléatoires différentes, ne garantit pas de trouver la solution optimale (ou, dans le cas général, une solution proche de la solution optimale).

Une bonne initialisation de l'algorithme *K-means* doit permettre d'obtenir une solution de meilleure qualité mais aussi une convergence plus rapide (c'est à dire avec moins d'itérations) vers cette solution.

Parmi les nombreux algorithmes d'initialisation de *K-means* nous considérerons ici *K-means++* proposé dans *[AV07]* (page 131). L'idée de départ est assez simple : choisir les centres l'un après l'autre, suivant une loi *non uniforme* qui évolue après le choix de chaque centre et qui privilégie les candidats éloignés des centres déjà sélectionnés. L'algorithme d'*initialisation K-means++* est :

`Entrées` : ensemble $\mathcal{E}$ de N données de $\mathbb{R}^p$ et nombre souhaité de centres k ;

`Sorties` : $\mathcal{C} = \{\mathbf{c}_j, 1 \leq j \leq k\}$, à utiliser après comme centres de départ dans *K-means* ;

1. $\mathcal{C} \leftarrow$ un $\mathbf{x}$ de $\mathcal{E}$ choisi au hasard ;

2. `tant que` ($\|\mathcal{C}\| \leq k$) `faire`
   (a) Sélectionner $\mathbf{x} \in \mathcal{E}$ avec la probabilité $\frac{d^2(\mathbf{x},\mathcal{C})}{\phi_{\mathcal{E}}(\mathcal{C})}$;
   (b) $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{x}\}$;

`fin tant que`

Dans cet algorithme, nous avons noté $d^2(\mathbf{x},\mathcal{C}) = \min_{j=1,\dots,t} d^2(\mathbf{x},\mathbf{c}_j)$, la fonction (**??**) peut donc être écrite $\phi_{\mathcal{E}}(\mathcal{C}) = \sum_{\mathbf{x}\in\mathcal{E}} d^2(\mathbf{x},\mathcal{C})$.

Les figures suivantes illustrent l'évolution des probabilités lors du déroulement de l'algorithme *K-means++*. La probabilité de sélection d'une donnée dans l'étape suivante (proportionnelle à $d^2(\mathbf{x},\mathcal{C})$) est d'autant plus élevée que la couleur est proche du rouge. La première coordonnée correspond à l'axe horizontal, la seconde à l'axe vertical.

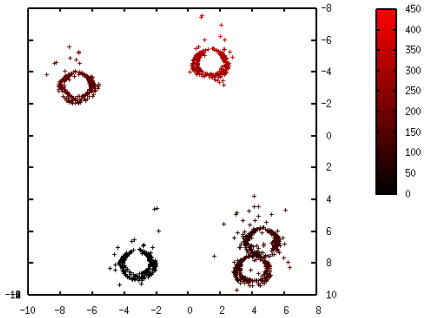


Fig. 5.8 – Probabilités après la sélection du premier centre (en bas à gauche)

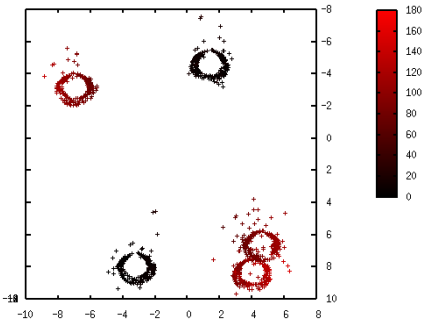


Fig. 5.9 – Probabilités après la sélection du deuxième centre (en haut à droite)

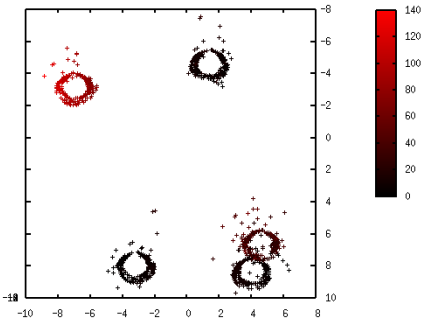


Fig. 5.10 – Probabilités après la sélection du troisième centre (en bas à droite)

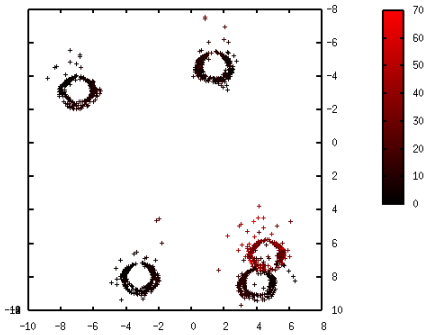


Fig. 5.11 – Probabilités après la sélection du quatrième centre (en haut à gauche)

La sélection séquentielle des centres opérée par *K-means++* permet de bien répartir les centres dans les données. Cela diminue le risque d'avoir, d'une part, des groupes de données sans aucun centre à proximité au démarrage de *K-means* et, d'autre part, plusieurs centres très proches entre eux. Ce sont ces insuffisances d'une initialisation aléatoire uniforme qui avaient provoqué la convergence vers des solutions sous-optimales dans *les cas simples illustrés plus tôt* (page 57).

Malheureusement, *K-means++* n'est pas directement parallélisable ; dans le cas de données massives, le nombre N de données et, implicitement, le nombre k de centres nécessaires peuvent être très élevés. Employer *K-means++* pour générer de l'ordre de 10^5 centres ou plus ralentirait significativement l'opération de classification automatique. Comment conserver les bonnes propriétés de l'initialisation par *K-means++* tout en permettant de l'accélérer grâce à une exécution distribuée ?

Initialisation *K-means* parallélisable : *K-means||*

L'algorithme *K-means||* a été proposé dans *[BMV12]* (page 131) comme variante parallélisable de *K-means++*. Son principe est de choisir bien plus qu'un seul centre à chaque itération d'initialisation, suivant également une loi non uniforme. On obtient beaucoup plus de centres que nécessaire (taux de sur-échantillonnage $l \in \Omega(k)$), ensuite on applique *K-means* pour regrouper ces centres en k groupes dont les centres seront employés dans l'application de *K-means* à la totalité des données.

K-means|| est l'algorithme suivant :

`Entrées` : ensemble $\mathcal{E}$ de N données de $\mathbb{R}^p$, nombre souhaité de centres k et taux de sur-échantillonnage $l \in \Omega(k)$;

`Sorties` : $\mathcal{C} = \{\mathbf{c}_j, 1 \leq j \leq k\}$, à utiliser après comme centres de départ dans *K-means* ;

1. $\mathcal{C} \leftarrow$ un $\mathbf{x}$ de $\mathcal{E}$ choisi au hasard ;
2. $\psi \leftarrow \phi_{\mathcal{E}}(\mathcal{C})$;
3. `pour` $(O(\log \psi)$ `fois)` `faire`
   (a) $\mathcal{C}' \leftarrow$ sélectionner chaque $\mathbf{x} \in \mathcal{E}$ indépendamment avec la probabilité $\frac{l \cdot d^2(\mathbf{x},\mathcal{C})}{\psi}$;
   (b) $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$; $\psi \leftarrow \phi_{\mathcal{E}}(\mathcal{C})$;

   `fin pour`
4. `pour` $(\mathbf{m}_j \in \mathcal{C})$ `faire`
   $w_m \leftarrow$ nombre de données de $\mathcal{E}$ plus proches de $\mathbf{m}_j$ que de tout autre point de $\mathcal{C}$;

   `fin pour`
5. Classification en k groupes des données $\mathcal{C}$ *pondérées par leurs poids* w_m. Les centres de ces groupes, en nombre de k, formeront l'ensemble final de centres $\mathcal{C} = \{\mathbf{c}_j, 1 \leq j \leq k\}$ à utiliser ensuite comme initialisation de *K-means*.

L'algorithme démarre par le choix aléatoire du premier centre. Ensuite, lors de chaque itération de (3.1)-(3.2), plusieurs données sont sélectionnées comme centres potentiels, suivant une loi qui rend plus probable le choix de candidats éloignés des centres déjà présents, et ensuite ajoutées à l'ensemble des candidats $\mathcal{C}$. Lors de l'étape (3.1), $\sum_{\mathbf{x} \in \mathcal{E}} \frac{l \cdot d^2(\mathbf{x},\mathcal{C})}{\psi} = l$, donc à chaque itération env. l nouvelles données sont choisies.

Dans l'étape (4), les candidats centres de $\mathcal{C}$ sont pondérés. Le poids de chaque candidat est le nombre de données de l'ensemble global $\mathcal{E}$ qui sont plus proches de ce candidat que de tout autre candidat. Ce poids est donc une mesure de l'importance du candidat. Si de nombreuses données de $\mathcal{E}$ sont « représentées » par un même candidat, il faudrait avoir au moins un centre à proximité de ce candidat. En revanche, si un candidat « représente » peu de données alors soit il est dans une région isolée dans l'espace qui ne « mérite » pas d'avoir un centre à proximité, soit il est trop proche d'autres candidats et devrait donc partager un même centre avec eux.

L'étape finale (5) consiste à faire une classification automatique des candidats de $\mathcal{C}$ en k groupes. Les données de $\mathcal{C}$ interviennent pondérées : pour chaque itération de l'algorithme de classification, les données affectées au groupe sont pondérées lors du calcul du centre de gravité du groupe. L'étape (5) traite un nombre relativement réduit de données, $O(l \cdot \log \psi)$, elle peut donc se dérouler sur un seul nœud de calcul et employer un algorithme séquentiel comme *K-means++* pour choisir les centres.

Comme *K-means++*, *K-means||* donne des garanties de qualité de la solution obtenue (voir *[BMV12]* (page 131)) : $E[\phi_{\mathcal{E}}(\mathcal{C})] \leq O(\log k) \cdot \phi_{\mathcal{E}}(\mathcal{C}^*)$, E étant l'espérance et $\mathcal{C}^*$ la solution optimale. Noter que les groupes sont définis sans équivoque par l'ensemble des centres, le partitionnement optimal est celui défini par l'ensemble optimal de centres $\mathcal{C}^*$.

D'après *[BMV12]* (page 131), cinq itérations dans l'étape (3) suffisent en général pour atteindre une très bonne solution, il n'est pas indispensable d'effectuer $O(\log \psi)$ itérations.

L'**implémentation MapReduce** de l'algorithme d'initialisation *K-means||* est facile. Chaque itération de (3) est exécutée de façon distribuée ; les tâches *Map* effectuent les opérations de (3.1), c'est à dire le calcul des $d^2(\mathbf{x}, \mathcal{C})$ et les tirages, ainsi qu'une partie du calcul de ψ sur les données du fragment local ; les tâches *Reduce* finalisent le calcul de ψ (comme dans l'implémentation MapReduce de *K-means*). Comme pour l'implémentation de *K-means*, après sa mise à jour dans l'étape (3.2), l'ensemble $\mathcal{C}$ des candidats centres est transmis à tous les nœuds de calcul pour l'étape (3.1).

Dans l'étape (4), le calcul des poids w_m est réalisé comme le calcul des n_j dans l'implémentation MapReduce de *K-means* : les tâches *Map* trouvent l'élément de $\mathcal{C}$ le plus proche de chaque donnée de leur fragment, puis calculent des valeurs partielles des w_m sur leur fragment. Les cumuls entre fragments sont ensuite réalisés par les tâches *Reduce*.

La classification des données (candidats centres) de $\mathcal{C}$ en k groupes dans l'étape (5) est réalisée comme pour l'implémentation MapReduce de *K-means*. Les centres résultants, $\mathcal{C} = \{\mathbf{c}_j, 1 \leq j \leq k\}$, sont conservés et transmis comme centres initiaux à l'algorithme *K-means*.

Classification ascendante hiérarchique

Parmi les méthodes qui visent à obtenir une hiérarchie de regroupements, nous examinons brièvement dans la suite la classification ascendante hiérarchique (CAH). Rappelons que, par rapport à un simple partitionnement des données, une hiérarchie de regroupements fournit une information plus riche concernant la structure de similarité des données. La classification *ascendante* procède par *agrégations successives* de groupes. A partir de la hiérarchie de groupes résultante il est possible d'observer l'ordre des agrégations de groupes, d'examiner les rapports des similarités entre groupes, ainsi que d'obtenir plusieurs partitionnements à des niveaux de granularité différents.

L'exemple suivant montre la hiérarchie de groupes (« dendrogramme ») obtenue par agrégations successives à partir d'un petit ensemble de données bidimensionnelles :

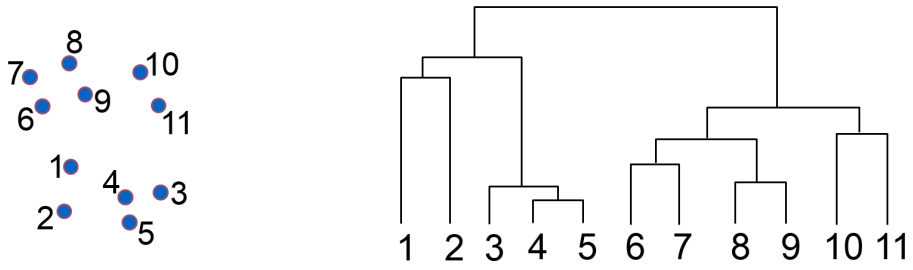
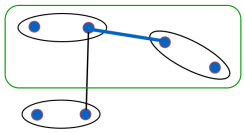


Fig. 5.12 – Exemple illustratif de classification ascendante hiérarchique

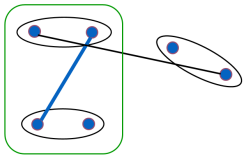
Le principe de la classification ascendante est de regrouper (ou d'agréger), à chaque itération, les données et/ou les groupes les plus proches qui n'ont pas encore été regroupé(e)s.

Considérons un ensemble de données $\mathcal{E} \subset \mathcal{X}$ et une distance $d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$. Les données individuelles peuvent être comparées grâce à cette distance. Mais, une fois deux données agrégées dans un groupe (noté h_p, par ex.), comment mesurer la similarité de ce groupe à une autre donnée ou à un autre groupe (h_q, par ex.) ? Il est nécessaire de définir une nouvelle mesure pour la dissimilarité entre groupes (ou entre une donnée et un groupe). Cette mesure est en général appelée « indice d'agrégation » et plusieurs choix fréquents sont :

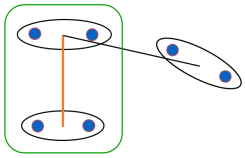
1. Le lien minimum (*single linkage*) : l'indice d'agrégation entre deux groupes h_p, h_q est la valeur la plus *faible* des distances entre une donnée du premier groupe et une donnée du second. Formellement, $\delta_s(h_p, h_q) = \min_{x_i \in h_p, x_j \in h_q} d_{\mathcal{X}}(x_i, x_j)$. Il *suffit* donc que deux groupes contiennent *deux* éléments proches pour que la valeur de l'indice soit faible :



2. Le lien maximum (*complete linkage*) : l'indice d'agrégation entre deux groupes h_p, h_q est la valeur la plus *élevée* des distances entre une donnée du premier groupe et une donnée du second (appelée parfois « diamètre » de l'agrégat). Formellement, $\delta_s(h_p, h_q) = \max_{x_i \in h_p, x_j \in h_q} d_{\mathcal{X}}(x_i, x_j)$. Il est donc nécessaire que *tous* les éléments d'un des groupes soient proches de *tous* les éléments de l'autre groupe pour que la valeur de l'indice soit faible :



3. Le lien moyen (*average linkage*) : l'indice d'agrégation entre deux groupes h_p, h_q est la moyenne des distances entre une donnée du premier groupe et une donnée du second. Formellement, $\delta_s(h_p, h_q) = \frac{1}{\|h_p\| \cdot \|h_q\|} \sum_{x_i \in h_p, x_j \in h_q} d_{\mathcal{X}}(x_i, x_j)$:



Les illustrations précédentes représentent des données de $\mathbb{R}^2$, mais les définitions des trois indices montrent bien que la seule exigence est l'existence d'une distance entre données individuelles et non d'une structure d'espace vectoriel.

4. L'indice de Ward est défini uniquement pour des données vectorielles. La valeur de l'indice correspond à l'augmentation de l'inertie intra-groupe lors de l'agrégation de deux groupes. Elle est la différence entre l'inertie intra-groupe de l'agrégat et la somme des inerties intra-groupe des deux groupes agrégés. Formellement, $\delta_s(h_p, h_q) = \frac{\|h_p\| \cdot \|h_q\|}{\|h_p\| + \|h_q\|} d^2_{\mathcal{X}}(\mathbf{m}_p, \mathbf{m}_q)$.

Contrairement à l'algorithme *K-means*, qui exige des données vectorielles, la CAH se satisfait, pour de nombreux indices d'agrégation (par ex. lien minimum, lien maximum, lien moyen) d'une simple distance définie sur l'espace des données. Il est donc possible de l'employer directement pour des données qui n'ont pas de représentation vectorielle naturelle.

Le choix de l'indice d'agrégation a un impact parfois significatif sur les résultats obtenus. Au-delà des contraintes liées à la nature des données (éléments d'un espace vectoriel ou simplement métrique), une bonne compréhension du problème doit aider ce choix. Par exemple, l'utilisation du lien minimum mène à un résultat qui se rapproche du regroupement par densité, alors que l'emploi du lien maximum (ou de l'indice de Ward) produit plutôt des groupes « compacts » séparés entre eux.

L'algorithme de CAH est très simple : on agrège, lors d'itérations successives, les groupes entre lesquels l'indice d'agrégation choisi a la valeur minimale.

`Entrées` : $\mathcal{E}$ de N données de $\mathcal{X}$ muni de la distance $d_{\mathcal{X}}$, indice d'agrégation δ_s ;

`Sorties` : Hiérarchie de groupes (dendrogramme) ;

1. Chaque donnée définit un groupe ;
2. `tant que` (nombre de groupes > 1) `faire`
   (a) Calcul des valeurs de l'indice d'agrégation entre tous les groupes issus de l'itération précédente ;
   (b) Regroupement des 2 groupes ayant la plus petite valeur de l'indice d'agrégation ;

`fin tant que`

Dans la hiérarchie de regroupements (dendrogramme) *illustrée plus haut* (page 60), la hauteur de chaque palier (qui représente une agrégation) est égale à la valeur de l'indice d'agrégation entre les groupes agrégés.

La complexité algorithmique pour un choix non contraint de l'indice d'agrégation est $O(N^2 \log N)$. En revanche, pour l'indice du lien minimum et l'indice du lien maximum existent des algorithmes de complexité $O(N^2)$, complexité qui reste néanmoins élevée. Une pratique assez fréquente est d'appliquer d'abord l'algorithme *K-means* avec une valeur élevée de k (mais néanmoins $k \ll N$), ensuite d'utiliser la classification ascendante hiérarchique pour regrouper les k groupes issus de *K-means*.

Si l'indice du lien minimum est choisi, la classification ascendante hiérarchique est équivalente à la recherche de l'arbre couvrant de poids minimal. Cet algorithme est parallélisable par découpage simple de l'ensemble $\mathcal{E}$ de données, calcul dans chaque partition (fragment de données) et ensuite fusion des résultats par partition. Cette solution a une implémentation Spark (http ://scholar.google.com/scholar ?oi=bibs&cluster=12245802843155845 06&btnI=1).

Classification automatique dans Spark

Spark propose des implémentations de plusieurs algorithmes de classification automatique :

1. *K-means*, avec une initialisation par *K-means*‖ (*[BMV12]* (page 131)) que nous avons étudié plus haut.
2. Estimation de mélanges gaussiens en utilisant l'algorithme espérance-maximisation (*Expectation-Maximization*, EM). Le mélange gaussien résultant est une estimation de densités qui peut également servir à la classification automatique : chaque donnée sera « affectée » au groupe défini par la composante du mélange qui « explique » le mieux cette donnée. EM est un algorithme itératif, chaque itération comporte une étape de calcul de l'espérance de la vraisemblance et une étape de calcul des paramètres (paramètres des lois normales du mélange et coefficients de mélange) qui maximisent cette vraisemblance. Les équations de mise à jour sont facilement parallélisables.
3. *Power Iteration Clustering* (PIC, voir *[LC10]* (page 132)) est une simplification de la *classification spectrale* qui travaille sur la matrice des similarités entre données.
4. Pour *Latent Dirichlet Allocation* (LDA, voir *[BNJ03]* (page 132)), l'objectif initial était d'identifier des « thèmes » (*topics*) dans un ensemble de documents textuels et ensuite d'« expliquer » chaque document par un ou plusieurs de ces thèmes. La méthode a toutefois une applicabilité plus large.

CHAPITRE 6

Cours - Fouille de données textuelles

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/6fouilleTextes.pdf)]

Les données textuelles contiennent des informations potentiellement très utiles pour la fouille. Ces données sont présentes sous des formes très diverses, allant de textes élaborés, avec une bonne conformité grammaticale, à de simples « mots-étiquettes » (*tags*, souvent parties de mots ou mots issus d'un lexique de groupe), en passant par des phrases incomplètes en langage SMS, présentant un lexique particulier, de nombreuses fautes d'orthographe et une syntaxe très simplifiée. Ces données sont destinées à être lues et comprises par des humains, parfois appartenant à des groupes restreints.

Si les opérations de fouille de données s'appliquent à une population de m observations, nous considérons ici que chaque observation est caractérisée par un ensemble de variables quantitatives et nominales mais aussi par un texte (ou liste de mots-clés ou de *tags*). L'ensemble de ces textes sera noté par $\mathcal{T}$, avec $card(\mathcal{T}) = m$.

Au-delà de difficultés relativement superficielles, comme la non conformité lexicale ou syntaxique, le problème principal dans la fouille de données textuelles est le « fossé sémantique », c'est à dire l'écart entre l'interprétation qu'un ordinateur peut obtenir automatiquement à partir d'un texte et la signification de ce même texte pour un humain (de la catégorie ciblée par le texte). Des difficultés de même nature se manifestent pour la fouille d'autres types de données, comme les images ou les vidéos.

Même si les méthodes d'analyse de données textuelles ne sont pas encore capables de combler ce fossé sémantique, il est néanmoins souvent possible d'extraire de façon automatique des informations utiles à partir des données textuelles. Le volume de données aide parfois ce processus d'extraction d'informations. Comme les données textuelles ne sont pas directement exploitables par les méthodes classiques de fouille de données, des traitements préalables sont nécessaires suivant l'objectif visé.

Objectifs et applications

Parmi les principaux objectifs de la fouille de données textuelles, nous pouvons mentionner :

1. L'identification de thèmes, qui vise à regrouper des textes (ou parties de textes, ou ensembles de *tags*, etc.) en thèmes inconnus *a priori*. Cela implique en général l'utilisation de méthodes de classification automatique sur des données textuelles dont la représentation a été adaptée, par exemple par le passage à des représentations vectorielles (que nous examinerons plus loin). Il est parfois utile de trouver les occurrences de parties spécifiques des textes, comme les entités nommées, afin de les représenter séparément.
2. Le classement de textes, qui vise à affecter des textes (ou parties de textes, ou ensembles de *tags*, etc.) à des catégories (classes) prédéfinies. Cela passe en général par l'application de modèles décisionnels, obtenus par apprentissage supervisé, à des représentations vectorielles de textes (ou d'ensembles de *tags*). La détection d'entités nommées, à représenter séparément, peut être utile.
3. L'extraction d'informations, qui vise à mettre en correspondance des textes avec des « schémas » d'interprétation prédéfinis. Un schéma d'interprétation regroupe plusieurs variables qui reçoivent des valeurs à partir du texte traité. Ces variables sont ensuite utilisées pour la fouille, conjointement avec d'autres variables (quantitatives, nominales) décrivant la même population. Dans certains cas il est possible d'identifier de telles « variables » à partir d'un ensemble de textes, sans avoir à définir un ou plusieurs schémas *a priori*.

Rappelons ici que la fouille de textes protégés par le droit d'auteur, même en accès ouvert sur le web, est **interdite** (sauf accord explicite des auteurs) dans la plupart des pays européens. Le Royaume-Uni est une exception notable à cette règle.

Comme l'indiquent les dernières enquêtes en date sur le site de KDnuggets (http ://www.kdnuggets.com/polls/index.html) (à prendre avec des précautions car rien ne permet d'évaluer leur représentativité), la fouille de données s'intéresse beaucoup aux données textuelles. Dans ces enquêtes on trouve des données textuelles aussi bien dans le type *text* que dans les sources *Twitter*, *web content*, *email* ou *social network* (en partie).

Avant de regarder de plus près les opérations de traitement nécessaires, il est utile de passer en revue quelques applications afin de mieux comprendre leurs exigences pour la fouille de textes.

1. La gestion de la relation client regroupe plusieurs applications potentielles de la fouille de textes. Il est possible de déterminer des catégories de clients à partir des termes employés dans leurs échanges avec le service client. Une classification thématique des textes des courriels adressés au service client peut aider à rediriger les courriels mal adressés au départ. Les causes de retours négatifs fréquents peuvent être identifiées par la détection d'entités nommées dans les messages ou éventuellement la mise en correspondance avec des schémas d'interprétation. L'image d'une famille de produits peut être caractérisée par la nature des retours (au service client, sur des forums web) et les termes employés dans ces retours. La mise en correspondance avec des schémas d'interprétation peut permettre de déterminer, dans des suggestions faites au service client, des attentes majeures pour l'évolution des produits.
2. L'identification de tendances à partir de messages postés sur des médias sociaux. Le classement thématique de textes ou d'intitulés de messages permet d'identifier les fils de discussion d'intérêt. L'association fréquente d'entités nommées correspondant à des produits ou familles de produits ou marques et de mots (répertoriés au préalable ou non) fournit des indications sur l'appréciation des produits ou marques, sur l'impact affectif d'événements liés à des marques ou sur des caractéristiques recherchées pour ces produits.
3. L'identification de supports adaptés pour la publicité en ligne. Le classement thématique de textes ou d'intitulés de messages, ainsi que l'identification d'entités nommées correspondant à des produits ou familles de produits ou marques, permettent de catégoriser et donc de cibler des blogs ou des fils de discussion sur des forums.
4. L'identification de risques sécuritaires par la surveillance d'échanges sur des réseaux sociaux. Cela passe en général par le classement thématique de messages et de fils de discussion, avec suivi de l'évolution thématique, ainsi que par la quantification de l'impact affectif d'événements. Le classement thématique fait appel à des listes de mots ou de locutions (groupes de mots) qui évoluent en général dans le temps. La quantification de l'impact affectif d'événements nécessite un classement sur la base de locutions faisant référence à des événements, ainsi qu'une extraction de mots et locutions caractérisant ce type d'impact.
5. L'identification de concepts ou de liens entre concepts. Il arrive fréquemment que plusieurs communautés s'intéressent à un même concept mais avec des perspectives différentes ou des outils méthodologiques distincts. Une communauté suit rarement les travaux qui se déroulent dans une autre communauté. Pour profiter de résultats obtenus dans d'autres communautés il est nécessaire de pouvoir déterminer les concepts communs ou de trouver des liens entre les concepts. La fouille de textes cherchera dans une telle situation à « cartographier » progressivement les concepts de chaque communauté (quels sont les concepts et les liens entre ces concepts) et à associer ensuite des parties de cartes entre différentes communautés. Les concepts sont désignés par des entités nommées (pas nécessairement les mêmes entre communautés différentes), des outils de recherche d'entités nommées sont donc nécessaires. Dans une même communauté, les liens entre concepts seront issus souvent de proximités dans le texte entre les entités nommées correspondantes. *Entre* communautés, les liens sont établis à partir de concepts très vraisemblablement communs (entités nommées très similaires) et sont ensuite étendus à d'autres concepts grâce à leurs contextes sur les cartes intra-communautés.

Approche générale

La fouille de données textuelles passe par plusieurs étapes brièvement décrites dans la suite. Toutes ces étapes ne sont pas systématiquement présentes, le choix des étapes nécessaires dépend de l'objectif et du contexte de la fouille. Certaines étapes, comme la collecte de données ou le pré-traitement, font partie de tout projet de fouille de données et présentent des particularités qui dépendent de la nature des données.

— La **collecte des données textuelles**. On trouve en général dans cette étape l'identification des sources, la récupération des contenus à partir de ces sources et l'extraction des textes à partir des contenus obtenus. Par exemple, une page web contient aussi d'autres données comme le balisage HTML, des menus, parfois des scripts Javascript, etc., qui doivent en général être séparés des données textuelles à traiter.
— Le **pré-traitement des données textuelles**. Les données textuelles collectées peuvent présenter une certaine diversité ou d'autres caractéristiques qui entravent les opérations d'analyse, des pré-traitements sont donc souvent nécessaires. On inclut ici l'uniformisation du codage (d'autant plus importante pour des caractères à accents), l'élimination éventuelle de certains caractères spéciaux (symboles, etc., suivant l'objectif), la « traduction » éventuelle d'éléments de langage SMS, etc.
— L'**extraction d'entités primaires**. Le texte est découpé en mots et signes de ponctuation. Des locutions (groupes de mots qui forment une même entité) nominales (« chemin de fer »), verbales (« arrondir les angles »), etc., peuvent éventuellement être extraites si elles sont présentes dans le lexique employé.
— L'**étiquetage grammatical** ou étiquetage morpho-syntaxique (*part-of-speech tagging* ou *POS tagging*). Chaque entité primaire extraite du texte (mot, très rarement locution) est caractérisée par une catégorie (et éventuellement une sous-catégorie) lexicale comme nom, verbe, adverbe, etc., des informations concernant le genre, le nombre, la conjugaison, etc. Est parfois proposée une fonction dans la phrase, comme sujet, complément d'objet direct, etc., mais cela exige en général une analyse syntaxique.
— L'**extraction d'entités nommées**. Les entités nommées peuvent être des noms de personnes (« Louis XIII »), de lieux (« Mont Blanc »), d'organisations (« Nations Unies »), des dates, etc., et jouent souvent un rôle important dans les opérations de fouille.
— La **résolution référentielle** cherche à identifier l'entité, explicitement présente ailleurs dans le texte, à laquelle se réfère une partie d'une phrase ; par ex., à qui fait référence *Il* dans « Barack Obama est le 44ème président des États-Unis. *Il* est né le 4 août 1961 à Honolulu ». Cela est d'autant plus important qu'une règle stylistique souvent respectée conseille d'éviter la répétition.
— L'**analyse syntaxique** cherche à mettre en évidence la structure hiérarchique des phrases d'un texte et joue un rôle important dans la compréhension du texte. Une analyse complète rencontre de nombreuses difficultés et n'est souvent pas indispensable par rapport à l'objectif poursuivi, une analyse partielle est ainsi le plus souvent pratiquée. On peut mentionner, par exemple, le traitement de la négation (pour permettre la distinction entre affirmation et négation) ou la « quantification » d'une variable à partir des adverbes employés (par ex. « *très* abouti / *plus ou moins* abouti / *peu* abouti »).
— L'**extraction d'informations** cherche à mettre des textes en relation avec des schémas d'interprétation prédéfinis et fait appel pour cela à plusieurs outils dont l'extraction d'entités nommées, l'analyse syntaxique superficielle, l'analyse pragmatique, etc.
— La **lemmatisation** consiste à remplacer chaque mot (par ex. « pensons ») par sa forme canonique (« penser »), alors que la **racinisation** le remplace par sa racine (« pense », dans l'exemple). Cette étape est utile pour la classification thématique de textes (en passant ou non par des représentations vectorielles) car elle permet de traiter comme un mot unique les différentes variantes issues d'une même forme canonique ou racine. Noter que la racinisation notamment peut engendrer des confusions, par ex. « organ » est à la fois la racine de « organe » et de « organisation ».
— La **représentation vectorielle des textes** vise à faciliter l'application aux textes de méthodes de fouille conçues pour des données vectorielles. Plusieurs techniques différentes existent, à commencer par le modèle classique de Salton *[SMG86]* (page 132), avec différentes pondérations, en passant par l'applications de méthodes de réduction de dimension comme dans l'analyse sémantique latente (*LSA*, *[DDL90]* (page 132)) et en allant jusqu'aux représentations compactes de type Word2Vec (*[MCC13]* (page 132), *[MSC13]* (page 132)).
— Le développement de modèles sur la base du contenu textuel seul ou en ajoutant des variables quantitatives et nominales.
— L'utilisation des modèles développés et l'évaluation des résultats.

Principales opérations

Nous examinons maintenant les principales opérations réalisées lors des différentes étapes spécifiques à la fouille de données textuelles.

Collecte et pré-traitement des données textuelles

Les données textuelles potentiellement utiles dans un projet de fouille de données massives peuvent provenir de sources variées et se trouver dans des formats différents. Les opérations suivantes sont en général nécessaires :

1. Identification des sources. Il est parfois nécessaire de regrouper des données issues de plusieurs sources, données qui peuvent avoir des caractéristiques très différentes :
   — Les textes sont des séquences parfois longues de phrases structurées et sémantiquement liées entre elles, mais peuvent présenter une *conformité lexicale et grammaticale* variable. Même si la conformité est moyenne, la présence d'une structuration grammaticale de la séquence de mots aide le traitement de ces données.
   — Les mots-clés, les « mots-étiquettes » (*tags*, souvent parties de mots ou mots issus d'un lexique de groupe) se trouvent sous la forme de simples listes, en général courtes (2 à 5 éléments), pour chaque unité d'analyse. Dans ces listes il y a souvent beaucoup de « bruit », dû soit aux fautes de frappe, soit à la présence d'éléments non pertinents (*tag spam*). L'absence de structure grammaticale qui relierait les mots-clés successifs et, parfois, l'absence de lexique spécifique pour les *tags*, rendent difficile l'exploitation de ces données. Par ailleurs, le fait que la liste correspondant à chaque unité d'analyse soit courte a un impact négatif sur les résultats de traitements statistiques qui leur sont appliqués.
   — Les messages de type SMS (qui se retrouvent aussi dans des sources web et non seulement dans les échanges directs entre téléphones portables) sont très souvent constitués de mots issus d'un lexique spécifique, parfois codé, d'expressions *ad hoc*, sont truffés de fautes de frappe (ou de sélection) et d'émoticônes (*smiley*). De nombreux traitements spécifiques sont nécessaires, avec des résultats de qualité très variable.
   — La transcription de la parole présente dans les vidéos téléchargées sur des réseaux sociaux, dans des programmes audiovisuels ou dans des échanges téléphoniques est de plus en plus utilisée comme source de texte. Cette transcription est toutefois difficile et les résultats comportent de nombreuses erreurs de différents types. Cela explique pourquoi sur de telles données on se contente en général d'appliquer des méthodes de recherche de mots-clés (*word spotting*) puis éventuellement d'extraction d'informations suivant des schémas simples et basés sur les mots-clés.

   Au vu de la diversité des caractéristiques des différents types de contenus textuels, il est important d'examiner (avant de démarrer le coûteux travail de collecte des données) dans quelle mesure les sources choisies sont compatibles avec l'objectif de la fouille de données. Par exemple, la résolution référentielle et l'analyse syntaxique exigent en général une assez bonne conformité grammaticale. Aussi, pour employer avec de bons résultats des représentations vectorielles il est en général nécessaire que le volume de données textuelles correspondant à chaque unité d'analyse soit suffisant.
2. Récupération des contenus à partir des sources. Pour chaque type de source, différents outils sont disponibles. Lorsque les données se trouvent sur des sites web, des outils simples de type `wget` permettent d'« aspirer » le contenu. Il faut veiller à respecter les règles de chaque site : sous-répertoires autorisés, débit des requêtes, etc. Ne pas oublier que la fouille de textes protégés par le droit d'auteur, même en accès ouvert sur le web, n'est possible qu'avec accord explicite des auteurs. Parfois, les données se trouvent déjà dans un *data warehouse* géré par exemple par Apache Hive, dans ce cas le langage d'interrogation associé (par ex. HiveQL) devrait permettre de délimiter le domaine des données à extraire.
3. Extraction des données textuelles. Les contenus récupérés peuvent contenir aussi d'autres données en plus des textes d'intérêt. Par exemple, dans une page web on trouve également le balisage HTML, des menus, parfois des scripts Javascript, etc. Ces données peuvent néanmoins intervenir dans la fouille, par ex. le balisage peut contribuer à pondérer des éléments textuels dans une analyse ultérieure (l'occurrence d'un terme dans un titre peut recevoir un « poids » supérieur à une occurrence dans le texte normal), les liens HTML peuvent servir à construire un graphe utile entre textes ou à identifier des références externes importantes, etc. Pour appliquer des traitements spécifiques aux données textuelles d'intérêt, il est néanmoins nécessaire de les séparer des autres données. Des outils configurables existent, ainsi que des librairies adaptées par ex. aux données XML dans différents langages comme Python, Scala ou Java, directement utilisables à partir de Spark.
4. Pré-traitements. L'objectif des pré-traitements est d'obtenir pour les données textuelles une représentation uniforme et compatible avec les outils d'analyse ultérieurs. On peut mentionner l'uniformisation du codage des caractères (d'autant plus importante lorsque des caractères accentués sont employés), l'élimination de certains caractères spéciaux (symboles, etc., suivant l'objectif), la « traduction » éventuelle d'éléments de langage SMS, etc. Des librairies adaptées en Python, Scala ou Java sont également disponibles mais une configuration adaptée aux caractéristiques des données est nécessaire.

Les opérations 3, 4 et éventuellement 2 (si les sources sont nombreuses ou alors un débit d'interrogation élevé est autorisé) sont facilement parallélisables sur une plate-forme distribuée car le même traitement doit être appliquée aux différents fragments de données.

Extraction d'entités primaires

L'analyse d'un texte démarre en général par une étape d'extraction d'entités « primaires », employées ensuite pour construire des structures plus complexes ou des représentations vectorielles. Le texte est découpé en *lemmes* et signes de ponctuation grâce à un outil de segmentation (*tokenizer*) qui utilise des règles dépendantes de la langue et un *lexique*. Le lexique contient l'ensemble des lemmes (unités autonomes) d'une langue et/ou d'un domaine particulier (par ex. la biochimie), ainsi que des informations additionnelles, morphologiques (formes possibles, avec racine et suffixes, préfixes) ou parfois syntaxiques. Un lexique peut souvent être enrichi par des lemmes spécifiques

Un lemme peut être constitué d'un ou plusieurs mots. Parmi les lemmes qui comportent plusieurs mots on peut distinguer les mots composés (par ex. « chauve-souris ») et les locutions (groupes de mots qui forment une même entité). On trouve des locutions nominales (« chemin de fer »), verbales (« arrondir les angles »), etc. Si les mots composés sont en général présents dans les lexiques, la situation est plus nuancée pour les locutions. Par ailleurs, la détection de locutions exige parfois une analyse plus approfondie et non seulement une recherche dans un lexique. Par exemple « arrondir les angles » est vraisemblablement une locution dans un texte qui traite de négociations mais n'est probablement pas une locution dans des instructions de débavurage.

Il faut noter ici que l'association des lemmes extraits à des *concepts*, nécessaire pour l'interprétation des textes, est un problème difficile et incomplètement résolu. Si les cas d'*homonymie* (un même lemme peut être associé à plusieurs concepts différents, par ex. le lemme « avocat ») ne sont pas si fréquents, une certaine ambiguïté d'interprétation est souvent présente. Les concepts et leurs relations (par ex. `sorte_de`, `partie_de`) sont décrits dans des *ontologies* (générales ou spécifiques à un domaine), qui incluent aussi des informations sur l'expression des concepts par différents lemmes dans une langue particulière. Dans le cadre d'un traitement automatique dont l'objectif est la fouille de données, l'identification des concepts associés aux lemmes extraits sera en général réalisée pour un nombre limité de concepts d'intérêt majeur par rapport au problème.

Étiquetage grammatical

Lors de l'étape d'étiquetage grammatical ou morpho-syntaxique, chaque lemme extrait est caractérisé par une catégorie lexicale (nom, verbe, adverbe, etc.) et, lorsque cela est pertinent, des informations concernant le genre, le nombre, le mode, le temps, etc. Cette opération n'est pas triviale car de nombreux lemmes peuvent appartenir à plusieurs catégories lexicales. Par exemple, « bien » peut être aussi bien un adverbe (« c'est *bien* fait »), un nom (« le *bien* et le mal »), un adjectif (« des gens *bien* ») ou une interjection (« *Bien* ! »). Une analyse du contexte est nécessaire pour enlever cette ambiguïté. Cette analyse est souvent superficielle, basée sur sur le voisinage local du lemme dans la phrase. Des erreurs d'étiquetage sont possibles, surtout lorsque le voisinage des lemmes est atypique, en raison par exemple d'une faible conformité grammaticale du texte.

Un exemple d'outil efficace et facile d'emploi est TreeTagger (voir http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/). L'étiquetage est fait grâce à des règles qui peuvent être fournies directement ou *apprises à partir d'exemples*.

Considérons une partie du premier paragraphe du chapitre introductif de ce cours, voici les résultats fournis par TreeTagger :

```
La      DET:ART le
fouille NOM     fouille
de      PRP     de
données NOM     donnée
peut    VER:pres        pouvoir
être    VER:infi        être
définie VER:pper        définir
comme   ADV     comme
le      DET:ART le
"       PUN:cit "
```

```
Processus       NOM     processus
d'      PRP     de
extraction      NOM     extraction
non     ADV     non
triviale        ADJ     trivial
d'      PRP     de
informations    NOM     information
implicites      ADJ     implicite
,       PUN     ,
inconnues       ADJ     inconnu
auparavant      ADV     auparavant
...
.       SENT    .
```

NOM, VER, ADV et ADJ signifient respectivement nom, verbe, adverbe et adjectif, DET:ART signifie « article » , PRP signifie « préposition », PUN signifie « ponctuation » et SENT indique la fin de la phrase. Observons que pour chaque lemme TreeTagger fournit sa forme canonique (« pouvoir » pour « peut », « le » pour « la »). L'outil peut être testé en ligne sur ce site de l'UCL (http ://cental.fltr.ucl.ac.be/treetagger/).

Extraction d'entités nommées et résolution référentielle

Une entité nommée est un élément du langage qui fait référence à une entité unique du domaine du discours. Les entités nommées peuvent être de différents types : noms de personnes (« Barack Obama Jr. »), de lieux (« Mont Blanc »), d'organisations (« Mouvement international de la Croix-Rouge et du Croissant-Rouge »), de produits (« iPhone 6s », « Galaxy S6 »), dates (« 5 mai 1789 », « 18 brumaire 1799 »), etc. L'extraction des entités nommées est très utiles dans la fouille de données textuelles car ces entités donnent des indications fortes sur le contenu d'un texte.

Les premières approches d'extraction étaient basées sur l'utilisation de règles définies explicitement et impliquaient donc une étape préparatoire laborieuse. Les approches plus récentes font appel à des méthodes d'apprentissage à partir d'un *corpus* annoté, avec éventuellement la prise en compte d'un nombre limité de règles explicitement définies. Les règles d'extraction doivent prendre en compte le contexte en raison de difficultés liées à la polysémie (par ex. l'entité « Washington » fait référence à la ville, à l'état ou à la personne) ou à la métonymie (par ex. « l'Elysée » fait référence à la présidence de la République Française ou simplement au palais).

Il est très utile d'examiner l'approche adoptée par Nuxeo (voir http://www.nuxeo.com/blog/mining-wikipedia-with-hadoop-and-pig-for-natural-language-processing/) pour définir grâce à DBpedia des règles d'extraction d'entités nommées en français.

On peut considérer que les meilleurs outils permettent l'extraction correcte de 90 à 95% des entités nommées génériques et s'approchent ainsi des performances d'un humain. Si les entités sont spécifiques à un domaine, un travail important est néanmoins nécessaire pour définir des règles adéquates ou annoter un corpus suffisant.

La résolution référentielle (correspondant globalement à *entity linking*) cherche à résoudre les problèmes de coréférence, c'est à dire à identifier l'entité, explicitement présente ailleurs dans le texte, à laquelle se réfère une partie d'une phrase. Par exemple, à qui fait référence le pronom *Il* dans « Barack Obama est le 44ème président des États-Unis. *Il* est né le 4 août 1961 à Honolulu ». Cela est d'autant plus important qu'une règle stylistique souvent respectée dans les textes rédigés conseille d'éviter la répétition. L'emploi de plusieurs mots ou groupes de mots différents pour désigner une même entité est aussi un problème de coréférence, comme dans « Google [...]. *La société de Mountain View* [...] » (le siège de Google est à Mountain View et Google est la société la plus connue ayant son siège à Mountain View).

Plusieurs types de méthodes contribuent au traitement de la coréférence car tous les problèmes ne sont pas de même nature. Si la résolution pronominale (exemple avec *Il* ci-dessus) fait plutôt appel à l'analyse syntaxique, pour relier entre elles les différentes façons de désigner une même entité il est en général nécessaire de disposer d'une base de données et éventuellement de quelques règles spécifiques. La qualité des résultats dépend fortement de la complétude de la base de données.

Analyse syntaxique

L'analyse syntaxique cherche à mettre en évidence la structure hiérarchique des phrases d'un texte et joue un rôle important dans la compréhension du texte. Bien entendu, l'analyse syntaxique n'a de sens que si les données textuelles sont sous forme de phrases et non simplement de mots-clés. Aussi, la bonne conformité grammaticale est critique pour ce type d'analyse. Une analyse complète doit produire un ensemble d'arbres syntaxiques décrivant chaque phrase du texte et permettant ultérieurement son « interprétation ».

Dans la pratique, l'analyse syntaxique rencontre de nombreuses difficulté même sur des textes qui présentent une assez bonne conformité grammaticale. Parmi les causes de ces difficultés nous pouvons mentionner la présence de mots hors lexique, de structures non répertoriées par la grammaire, ainsi que de diverses formes d'ambiguïté. La conséquence de ces difficultés est en général la présence d'un nombre élevé d'arbres d'analyse pour chaque phrase, qui mènent à une multitude d'interprétations entre lesquelles un choix automatique est souvent impossible. Les travaux actuels visent des méthodes d'analyse « robuste », qui proposent une analyse pour chaque élément avec peu d'analyses alternatives.

Heureusement, une analyse syntaxique complète n'est en général pas indispensable par rapport à l'objectif de fouille de texte. Il est possible de pratiquer une analyse « de surface » qui ne met pas en évidence la structure complète des phrases et ne cherche pas à enlever toute ambiguïté, mais se contente d'extraire des parties (*chunks*) comme les groupes nominaux et les groupes verbaux, ainsi que des relations simples entre eux. Parfois c'est une analyse partielle qui est employée, analyse qui se concentre sur des fragments spécifiques importants qui présentent des constructions simples et avec peu de variabilité. On peut mentionner, par exemple, le traitement de la négation (pour permettre la distinction entre affirmation et négation) ou la « quantification » d'une variable à partir des adverbes employés (par ex. « *très* abouti / *plus ou moins* abouti / *peu* abouti »). L'analyse peut aussi être limitée au voisinage de certains lemmes jugés importants pour l'interprétation, comme des entités nommées.

Extraction d'informations

L'extraction d'informations à partir de textes est un des principaux objectifs de la fouille de textes. Cette étape vise à mettre en correspondance des textes avec des « schémas » d'interprétation prédéfinis qui ont un rapport direct avec l'application de fouille. Un schéma d'interprétation (ou patron sémantique) regroupe plusieurs variables qui ont un rapport direct avec l'application de fouille, par exemple :

> [Fait : ?]{[Où : ?][Quand : ?][Qui : ?][Nature : ?]...}

Ces variables reçoivent des valeurs à partir de l'analyse du texte traité qui sera ainsi décrit à travers un ou plusieurs schémas. Les variables des schémas sont ensuite utilisées pour la fouille, conjointement avec d'autres variables (quantitatives, nominales) décrivant la même population.

Considérons par exemple le texte « Tôt le 25 avril 1974, au Portugal, des capitaines en rupture avec le système de Salazar se révoltent et prennent le pouvoir ». L'analyse de ce texte doit permettre de donner les valeurs suivantes au schéma défini ci-dessus :

> [Fait : *événement politique*]{[Où : *Portugal*][Quand : *25/04/1974*][Qui : *forces armées*][Nature : *prise de pouvoir*]...}

Si on considère un autre schéma, mieux adapté à la représentation du contenu d'interventions sur des médias sociaux, par ex.

> [Fait :]{[Objet :][Quand :][Qui :][Nature :][Degré :][Arguments :]...}

alors le texte « J'ai été terriblement déçu par ce roman. Je trouvais l'écriture fade et le style pseudo-éthérique très agaçant », devrait permettre d'obtenir

> [Fait : *avis*]{[Objet : *roman X*][Quand : *21/01/2016*][Qui : *utilisateur Y*][Nature : *négatif*][Degré : *fort*][Arguments : *écriture*, *style*]}

L'extraction d'informations demande en général de définir le lexique et les entités nommées d'intérêt, les schémas d'interprétation pertinents et des règles d'inférence spécifiques (permettant d'identifier dans le texte les valeurs à affecter aux différentes variables du schéma). Cela implique en général un travail spécifique conséquent.

Suivant la complexité des schémas et la nature des textes traités, différentes opérations peuvent être nécessaires pour identifier le(s) schéma(s) adapté(s) à un texte est extraire de ce texte les valeurs des variables correspondantes : l'étiquetage grammatical, l'extraction d'entités nommées, la résolution référentielle, l'analyse syntaxique locale, des règles d'inférence issues d'une ontologie ou définies spécifiquement, des règles pragmatiques (par ex. concernant la mise en forme du texte), etc.

Lemmatisation ou racinisation

Dans des textes et parfois dans des listes de mots-clés, un même lemme du lexique peut prendre des formes variables. Par exemple, en français la forme d'un verbe varie suivant le mode, le temps, la personne et le nombre. Ces différences, nécessaires pour certaines opérations comme l'étiquetage grammatical et l'analyse syntaxique, peuvent nuire à d'autres opérations. Par exemple, pour la classification thématique de textes (en passant ou non par des représentations vectorielles) il est préférable de traiter comme un lemme unique les différentes variantes issues d'une même forme canonique (par ex. « penser » plutôt que « pensons », « pense », « penserons »). Cela permet d'ignorer une partie de la variance (stylistique, pragmatique, etc.) pour se concentrer sur le fond thématique. Également, diverses ressources linguistiques comme les ontologies contiennent des règles dans lesquelles seules les formes canoniques sont présentes ; leur utilisation nécessite donc une étape préalable de mise sous forme canonique.

Deux opérations différentes peuvent être employées pour réduire cette variabilité de formes. La *lemmatisation* consiste à remplacer chaque mot (par ex. « pensons ») par sa forme canonique (« penser »). La *racinisation* consiste à remplacer chaque mot (par ex. « pensons ») par sa racine (« pense »). Notons que la racine n'est pas nécessairement un mot de la langue, par ex. la racine de « cheval » et de « chevaux » est « cheva ». La lemmatisation fait appel à l'analyse lexicale avec étiquetage grammatical. La racinisation emploie plutôt des règles simples de construction des mots dans une langue spécifique.

La racinisation engendre assez souvent des confusions entre lemmes différents qui ont une même racine, par ex. « organ » est aussi bien la racine de « organe » que de « organisation ». L'élimination du préfixe peut également avoir un impact négatif sur les étapes de traitement ultérieures, par ex. « mange » est aussi bien la racine de « mangeable » que de « immangeable ». Ces confusions étant plus fréquentes en français, la lemmatisation est en général préférée pour le français alors qu'en anglais la racinisation est souvent employée.

Représentation vectorielle des textes

L'analyse et la fouille de données font souvent appel à des méthodes qui travaillent sur des représentations vectorielles des données. Pour appliquer de telles méthodes à des données textuelles (ou à des données qui combinent textes, variables quantitatives et variables nominales), il est nécessaire de pouvoir représenter des textes (ou des ensembles de mots-clés ou de *tags*) sous une forme vectorielle. Le modèle classique de Salton *[SMG86]* (page 132) a été employé (avec diverses variations et améliorations) pour la recherche d'informations et la classification thématique de textes durant de nombreuses années. Plus récemment, la disponibilité de corpus très vastes de textes a permis le développement de méthodes nouvelles, comme l'analyse sémantique explicite (*ESA [GM07]* (page 132)) ou Word2Vec *[MCC13]* (page 132), *[MSC13]* (page 132).

Pour construire la représentation vectorielle de textes, plusieurs étapes préalables sont en général nécessaires : l'extraction d'entités primaires, l'étiquetage grammatical, la lemmatisation (ou la racinisation, qui permet d'éviter l'étape d'étiquetage grammatical) et la suppression des « mots à ignorer » (*stop words*). Cette dernière étape vise à éliminer du texte les mots très fréquents mais peu discriminants car nécessairement présents dans tous les textes : prépositions, conjonctions, articles, verbes auxiliaires, etc. Des dictionnaires plus ou moins larges de tels mots sont disponibles pour chaque langue. Si l'identification de locutions ou d'entités nommées (et résolution référentielle) est nécessaire par rapport à l'objectif de l'analyse, alors ces étapes doivent être réalisées avant la suppression des *stop words*. Après ces opérations, un texte devient une succession de lemmes, éventuellement locutions et entités nommées, qui seront toutes appelées « termes » dans la suite.

Considérons l'ensemble $\mathcal{T}$ de documents (textes ou de listes de mots-clés ou de *tags*), chaque texte étant associé à une observation de la population à analyser. La méthode de base de représentation vectorielle de textes, proposée dans *[SMG86]* (page 132), consiste à répertorier la totalité des termes présents dans $\mathcal{T}$ (nous noterons leur nombre par n), après lemmatisation et suppression des *stop words*, et à affecter une dimension (un axe) de l'espace vectoriel à chaque terme. Chaque texte est alors représenté par un vecteur : la composante associée à un terme est 0 si le terme est absent du texte et différent de 0 (plusieurs techniques de pondération peuvent être employées ici) si le terme est présent dans le texte. Ce vecteur est en général de très grande dimension, car le nombre de termes différents qui apparaissent dans un grand ensemble de textes $\mathcal{T}$ est très élevé. Un tel vecteur est aussi le plus souvent très creux, car chaque texte ne contient qu'une faible part des termes de $\mathcal{T}$. La figure suivante illustre ce type de représentation vectorielle de texte (de dimension 10...4, par ex. 100004) :

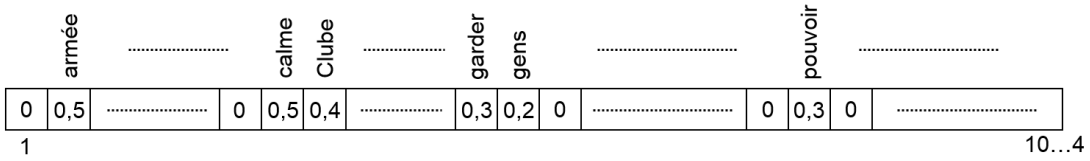


Fig. 6.1 – Exemple de représentation vectorielle de texte

Plus un texte est court, plus le vecteur associé sera en général creux et donc sa norme euclidienne faible. Pour utiliser la distance euclidienne il est donc nécessaire de normaliser les vecteurs associés aux différents textes de $\mathcal{T}$. Une solution souvent préférée est d'employer plutôt la distance cosinus.

Cette représentation vectorielle de base a connu de nombreuses évolutions, parmi lesquelles nous pouvons mentionner l'utilisation de pondérations de type *TF-IDF* (*term frequency* x *inverse document frequency*), la sélection de mots (par ex. en utilisant le test du χ^2), la réduction de dimension par « analyse sémantique latente » (*LSA [DDL90]* (page 132)), etc.

Pondération *TF-IDF*

Dans la comparaison des vecteurs qui représentent des textes, les différents termes n'ont pas tous la même importance. D'abord, un terme qui apparaît dans une majeure partie des documents textuels de $\mathcal{T}$ contribue moins aux distinctions entre textes qu'un autre terme qui est présent dans un faible nombre de textes. Ensuite, un terme qui a de multiples occurrences dans un même texte est plus significatif pour ce texte qu'un autre terme qui n'y figure qu'une seule fois. Ces deux critères sont les plus fréquemment employés pour obtenir des pondérations des termes et peuvent être exprimés de différentes façons. Les pondérations des termes sont systématiquement employées pour la recherche d'informations, mais on les retrouve également dans la classification thématique de textes.

Le premier critère se traduit en général par la pondération de chaque terme par un coefficient inversement proportionnel au nombre de documents (de l'ensemble $\mathcal{T}$) dans lesquels il apparaît. En anglais on parle de *inverse document frequency* (IDF). L'expression fréquemment utilisée pour ce coefficient est $idf_i = \log\left(\frac{m}{m_i}\right)$, m étant le nombre total de documents de $\mathcal{T}$ et m_i le nombre de documents contenant le terme i. Le logarithme permet ici d'éviter une sur-pondération des termes très rares. D'ailleurs, les termes dont le nombre d'occurrences dans la collection de documents est inférieur à un seuil sont en général exclus dès le départ (sans calcul de leur idf). De nombreuses variations existent pour cette composante IDF. Nous noterons que la pondération IDF dépend de l'*ensemble* des documents.

Le second critère mène à une pondération proportionnelle au nombre nombre d'occurrences du terme dans le document. En anglais on parle de *term frequency* (TF). L'expression employée en général est $tf_{ij} = \frac{n_{ij}}{\|d_j\|}$, n_{ij} étant le nombre d'occurrences du terme i dans le document j et $\|d_j\|$ la longueur du document d_j. Nous remarquerons que la pondération TF dépend uniquement du document j.

En tenant compte de ces deux critères, la pondération du terme i dans le document j est $tf_{ij} \cdot idf_i$. Si le terme i est absent du document, alors la composante correspondante du vecteur qui représente le document j est 0. Si le terme i est présent, la composante qui lui correspond est $tf_{ij} \cdot idf_i$.

Nous remarquerons que d'autres considérations peuvent intervenir dans le calcul de la pondération, par ex. surpondération des entités nommées par rapport aux lemmes simples, surpondération des occurrences dans des titres (éventuellement suivant le niveau du titre), etc.

Analyse sémantique latente

L'ensemble de documents textuels $\mathcal{T}$ peut être représenté par une **matrice documents-termes** $\mathbf{M}$ dans laquelle chacune des m lignes correspond à un document et chacune des n colonnes à un terme, avec des pondérations de type TF-IDF. Dans cette représentation, aussi bien les lignes que les colonnes sont très creuses, un document contenant une faible part de l'ensemble total des termes et un terme étant en général présent dans peu de documents. Toute similarité entre documents est expliquée par la présence de nombreux termes communs entre les documents, toute similarité entre termes est expliquée par leur présence commune dans un grand nombre de documents. La « lisibilité » de ces similarités est limitée par les grands nombres de termes et respectivement de documents. Aussi,

cette représentation présente également d'autres problèmes liés aux termes employés. Par exemple, deux synonymes (termes différents mais ayant une même signification, comme « voiture » et « automobile ») sont associés à deux dimensions (ou composantes) différentes des vecteurs qui représentent les documents ; si deux documents emploient chacun un de ces termes, la comparaison de leurs vecteurs ne mettra en évidence aucune similarité due à la signification commune des synonymes. Un autre exemple : un terme rare, qui correspond à une particularité stylistique de l'écriture et a peu de rapports avec la signification des documents, aura un poids IDF (et donc TF-IDF) élevé alors que pour la comparaison des documents (ou leur classification thématique) il peut être considéré comme du « bruit ». L'utilisation dans quelques documents d'un terme avec un sens alors que dans de nombreux autres documents le même terme est employé avec un autre sens (cas d'homonymie) produit également du « bruit ».

L'analyse (ou indexation) sémantique latente (*Latent Semantic Analysis*, LSA *[DDL90]* (page 132)) a été proposée comme une réponse à ces insuffisances de la représentation vectorielle classique. La LSA cherche à identifier des « concepts latents » (un tel concept correspond à des corrélations entre les occurrences de plusieurs termes, non à un concept bien identifié dans une ontologie) pour représenter les documents d'une collection. Par exemple, dans l'ensemble $\mathcal{T}$ on peut avoir de multiples documents qui contiennent les termes « juge », « prévenu », « procès », « avocat », « audience », « peine », et d'autres qui contiennent « four », « citron », « avocat », « assiette », « couteau », « mayonnaise ». Dans le premier cas, les corrélations indiqueront le concept (au sens de LSA) « justice » alors que dans le second le concept « cuisine ». Dans la plupart des cas, le contenu d'un document sera caractérisé par la présence de certains concepts (issus d'un ensemble relativement réduit de concepts). Un terme sera également caractérisé par sa cintribution à un nombre réduit de tels concepts. Aussi, le « bruit » dû à l'emploi d'un même terme « avocat » dans les deux contextes avec deux sens différents pourra être supprimé (ou réduit) car l'impact des corrélations multiples signalées sera dominant. Les concepts correspondant à des corrélations entre de multiples termes ne sont pas nommés, d'ailleurs ils ne sont pas toujours facilement interprétables par un humain (contrairement aux deux cas mentionnés ci-dessus).

L'analyse sémantique latente consiste à appliquer une décomposition en valeurs singulières (*Singular Value Decomposition*, SVD) à la matrice documents-termes, suivie par une réduction de rang, pour obtenir une approximation de faible rang (avec un faible nombre de « concepts ») de cette matrice. Plus précisément, la matrice documents-termes sera décomposée de la façon suivante :

$$\mathbf{M} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$$

avec $\mathbf{U}$ et $\mathbf{V}$ des matrices orthogonales (une matrice est orthogonale si par le produit avec sa transposée on obtient la matrice identité) et $\mathbf{S}$ une matrice diagonale.

La décomposition en valeurs singulières est liée à l'analyse en composantes principales : en calculant $\mathbf{M}\mathbf{M}^T$ et $\mathbf{M}^T\mathbf{M}$ on constate que $\mathbf{S}$ $(n \times n)$ contient sur sa diagonale principale les racines carrées des valeurs propres de $\mathbf{M}\mathbf{M}^T$ (ou de $\mathbf{M}^T\mathbf{M}$, les valeurs propres non nulles des deux matrices étant les mêmes), $\mathbf{U}$ $(m \times n)$ a pour colonnes les vecteurs propres de $\mathbf{M}\mathbf{M}^T$ et $\mathbf{V}$ $(n \times n)$ a pour colonnes les vecteurs propres de $\mathbf{M}^T\mathbf{M}$. Nous avons supposé ici que $m \geq n$ et que le rang de $\mathbf{M}$ était n.

Si on conserve uniquement les k plus grandes valeurs propres $(k < m, k < n)$ et les vecteurs propres associés, alors on obtient la meilleure approximation de rang k de $\mathbf{M}$ (de même dimension $m \times n$ que $\mathbf{M}$) :

$$\mathbf{M}_k = \mathbf{U}_k \cdot \mathbf{S}_k \cdot \mathbf{V}_k^T$$

Les valeurs sur la diagonale principale de $\mathbf{S}_k$ indiquent l'« importance » de chacun des k « concepts latents » (ou facteurs) identifiés. Pour chacun des m documents, la ligne correspondante de $\mathbf{U}_k$ permet de voir quels concepts y sont présents et avec quels poids. Pour chacun des k concepts, la colonne associée de $\mathbf{V}$ indique quels termes forment le concept (et avec quelles pondérations).

En général, les développements concernant LSA sont présentés à partir de la matrice termes-documents, qui est la transposée de la matrice documents-termes traitée ici (et dans la séance associée de travaux pratiques). Lorsque LSA a été proposée, le nombre de documents était le plus souvent inférieur au nombre de termes. Pour des données massives, le nombre de documents est en général bien supérieur au nombre de termes et l'utilisation dans Spark d'une `RowMatrix` pour contenir ces données impose d'employer la matrice documents-termes.

Cette idée de la recherche d'un nombre réduit de concepts latents qui s'expriment à travers des ensembles de termes et sont présents (à des degrés divers) dans des documents textuels a été développée ultérieurement dans l'analyse sémantique latente probabiliste (*Probabilistic Latent Semantic Analysis*, PLSA *[Hof99]* (page 132)) et ensuite dans l'allocation de Dirichlet latente (*Latent Dirichlet Allocation*, LDA *[BNJ03]* (page 132)).

Analyse sémantique explicite

Les relations entre termes et entre documents issues (directement, ou après LSA, PLSA, LDA, etc.) de la matrice $\mathbf{M}$ représentant un ensemble de documents $\mathcal{T}$ sont très dépendantes des particularités de $\mathcal{T}$. Cela engendre des biais de modélisation, présents même pour des ensembles $\mathcal{T}$ de grande cardinalité (données massives). Pour s'affranchir de ces biais, l'analyse sémantique explicite (*Explicit Semantic Analysis*, ESA *[GM07]* (page 132)) considère un corpus très grand, pouvant faire référence (Wikipedia) pour construire une matrice documents-termes. Cela permet d'obtenir des représentations générales et très riches des termes, indépendantes d'ensembles spécifiques (et souvent beaucoup plus petits) de documents applicatifs.

Avec ESA, chaque terme est représenté par un vecteur dont la dimension est donnée par le nombre de pages (ou de « concepts ») Wikipedia. Chaque composante du vecteur correspond à une page Wikipedia. Si le terme y est absent, la valeur est 0. Si le terme y est présent, la valeur est égale à sa pondération TF-IDF. Tout autre document est représenté par le centre de gravité de l'ensemble des termes qu'il contient ; le calcul du centre de gravité tient compte des pondérations TF-IDF par rapport au document. Contrairement aux résultats de LSA (ou de PLSA, ou de LDA), où les « concepts latents » sont en nombre relativement faible et ne sont pas toujours interprétables par un humain, avec ESA les dimensions des vecteurs sont *interprétables* car chaque dimension correspond à un concept précis (une page) de Wikipedia.

Nous remarquerons qu'un corpus multi-langues comme Wikipedia, ou de nombreux concepts (surtout les plus importants) ont des pages dans plusieurs langues, permet de définir des représentations de documents *indépendantes de la langue* car exprimées comme des vecteurs dans l'espace de ces concepts ; on parle de *Cross-language* ESA.

Il est important de noter que Wikipedia comporte également des biais (certains domaines sont plus présents que d'autres, certains mots sont préférés à d'autres, etc.) et évolue constamment (le nombre de concepts augmente, mais les pages existantes changent également). Aussi, le nombre de concepts décrits et le niveau de détail varient beaucoup d'une langue à une autre. Cela a un impact sur les représentations issues de l'analyse sémantique explicite.

Word2Vec

Les représentations vectorielles précédentes sont basées sur la description d'un texte par l'ensemble des termes (le plus souvent mots individuels) qu'il contient. Dans un tel « sac de mots » (*bag of words*) toute information liée au contexte des mots est ignorée. Or, le contexte d'un mot dans une phrase caractérise assez bien le mot à la fois sur l'aspect syntaxique et sur l'aspect sémantique. Il est alors utile d'exploiter le contexte pour construire des représentations vectorielles plus « raffinées ».

C'est ce que propose la représentation *Word2Vec* (voir *[MCC13]* (page 132), *[MSC13]* (page 132)), sur la base de ressources textuelles très volumineuses. Avec le modèle Skip-gram *[MCC13]* (page 132), l'objectif est de trouver des représentations permettant de prédire le mieux possible le contexte des mots. Plus précisément, étant donnée une séquence de mots $w_1, w_2, \ldots, w_T$, on cherche à maximiser

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j=-k}^{j=k}\log p(w_{t+j}|w_t)$$

où k est la largeur du contexte autour de chaque mot w_t. Les probabilités conditionnelles sont définies avec la fonction softmax

$$p(w_i|w_j) = \frac{\exp(u_{w_i}^\top v_{w_j})}{\sum_{l=1}^{V}\exp(u_l^\top v_{w_j})}$$

V étant le nombre de mots du vocabulaire, u_{w_i} la représentation « de sortie » de w_i et v_{w_j} la représentation « d'entrée » de w_j. Une reformulation du problème permet de remplacer softmax et réduire ainsi le coût des calculs (voir *[MSC13]* (page 132)).

Avec cette représentation, les mots se regroupent par similarité de contexte qui reflète à la fois une similarité syntaxique et une similarité sémantique. Aussi, on constate une forme d'additivité, par exemple la représentation la plus proche du résultat du calcul $v_{Madrid} - v_{Spain} + v_{France}$ est v_{Paris}.

Contrairement aux représentations de base pondérées (*TF-IDF*) ou ESA, les représentations Word2Vec sont d'assez faible dimension (par ex. 300) et denses. Il est possible de développer de telles représentations non seulement pour des mots individuels mais également pour des locutions, des entités nommées et plus généralement pour des phrases courtes. Pour un document textuel court (phrase, liste de *tags*), une assez bonne représentation est le centre de gravité des vecteurs correspondant aux mots du document.

Ces représentations sont utilisées avec de bons résultats pour résoudre des problèmes très divers : désambiguïsation, traduction de textes, classification de textes, illustration de textes, etc. Sur https://code.google.com/archive/p/word2vec/ sont mises en libre accès des représentations Word2Vec pour de grands nombres de mots et d'entités nommées.

Par ailleurs, nous vous proposons une séance complémentaire en ligne de travaux pratiques sur la classification automatique de tweets avec des représentations Word2Vec.

Environnements logiciels

Des ressources libres sont disponibles pour différentes étapes spécifiques à la fouille de données textuelles.

Apache propose *Unstructured Information Management applications* (UIMA (https ://uima.apache.org)), un environnement général pour le développement et l'organisation de services de gestion de données non structurées. Dans ce cadre, OpenNLP (https ://opennlp.apache.org) (NLP pour *Natural Language Processing*) est une suite intégrée en java qui propose la segmentation (*tokenizer*), l'étiquetage (*tagger*), la lemmatisation, l'analyse syntaxique, l'extraction d'entités nommées, le traitement des coréférences, la catégorisation de documents, etc. Tous ces outils sont proposés pour l'anglais, certains aussi pour d'autres langues (espagnol, allemand, etc.). Pour le français sont disponibles la segmentation et l'étiquetage. L'extraction d'entités nommées est proposée comme un complément par Nuxeo (http ://www.nuxeo.com/blog/mining-wikipedia-with-hadoop-and-pig-for-natural-language-processing/). D'autres outils complémentaires sont disponibles ici (https ://sites.google.com/site/nicolashernandez/resources/opennlp). La suite OpenNLP est intégrée à UIMA mais peut être utilisée de façon indépendante.

Une autre suite intégrée en java, bien développée, est Stanford CoreNLP (http ://nlp.stanford.edu/software/). Les outils proposés concernent principalement l'anglais, l'espagnol et le chinois. Sont ainsi disponibles la segmentation (*tokenizer*), l'étiquetage (*tagger*), la lemmatisation, l'analyse syntaxique, l'extraction d'entités nommées, le traitement des coréférences, etc. Cette suite est employée également dans notre séance de travaux pratiques.

Opérations liées à la fouille de textes dans Spark

Spark propose plusieurs outils intégrés pour différentes opérations : construction et manipulation de représentations vectorielles avec pondérations TF-IDF (`HashingTF, IDF`), sélection de mots par test du χ^2 (`ChiSqSelector`), l'allocation de Dirichlet latente (`LDA, DistributedLDAModel`), les représentations vectorielles Word2Vec (`Word2Vec, Word2VecModel`). Pour l'analyse sémantique latente, une implémentation externe utilisant `SVD` est accessible (voir la séance de travaux pratiques).

CHAPITRE 7

Cours - Fouille de flux de données

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/7fouilleFluxDonnees.pdf)]

La « vélocité » est un des multiples V qui caractérisent les données massives. Dans de nombreux cas, de nouvelles données arrivent sous forme de flux et doivent être traitées dans des délais cohérents avec le(s) flux. Cela impose de nouvelles exigences sur la *latence* des opérations de construction de modèle ou de prise de décision grâce à un modèle. Par latence on entend ici l'intervalle de temps entre le moment où le traitement démarre (en général, cela correspond au moment où les données deviennent disponibles) et le moment où il se termine.

Par flux on entend en général une succession de données de même type, qui arrivent à intervalles constants ou variables. Pour être traitées, ces données peuvent être découpées en « tranches » (*slices*), soit à partir des intervalles d'arrivée des données, soit à partir de contraintes liées au traitement (intervalles de durée fixée, tranches de volume fixé, etc.).

On peut considérer que les traitements réalisés avec des données en flux sont synchrones (« continus ») ou asynchrones. Les traitements synchrones doivent être réalisés en « temps réel » ou, du moins, leur latence doit être compatible avec le rythme du flux. Un premier type de traitement synchrone consiste à appliquer régulièrement sur les données du flux un modèle existant, pré-estimé, afin de caractériser l'évolution du flux. Dans un second type de traitement synchrone, on estime (ou on met à jour) un modèle régulièrement sur les données du flux, en général sur les dernières tranches ; ce modèle est ensuite appliqué aux données du même flux (par ex. de la tranche suivante) ou d'un autre flux. Lorsqu'elle est faite à partir de données en flux, la modélisation doit être *incrémentale* : il faut pouvoir mettre à jour le modèle existant sans avoir à le ré-estimer complètement (avec toutes les données accumulées) car le coût de traitement de toutes ces données serait excessif.

Les traitements asynchrones répondent à des demandes uniques ou peu fréquentes et non synchronisées avec le rythme du flux ; ils sont appliqués soit à la totalité des données du flux (si celles-ci ont été stockées et si le coût est acceptable), soit à des « résumés » (synthèses) génériques ou plus ciblés de ces données. Ces traitements peuvent être coûteux mais sont en général moins contraignants en termes de latence que les traitements synchrones. Ils peuvent toutefois reposer en partie sur certains traitements synchrones comme la création et la mise à jour de résumés du flux. Les résumés de flux peuvent être également utiles aux traitements synchrones lorsque chaque tranche du flux dépasse les capacités de stockage en mémoire vive et l'accès au stockage de masse est trop lent pour être compatible avec la latence attendue de ces traitements.

Il faut noter que la fouille de flux de données ne mène pas nécessairement à une problématique spécifique aux données massives. Par exemple, les ressources d'un ordinateur de base suffisent largement pour effectuer une fois par heure la classification automatique des quelques centaines ou milliers de *tweets* arrivés durant les dernières 24 heures sur certains *hashtags*. On considère que les traitements relèvent de la fouille de données massives de flux si, en raison à la fois du *débit* très élevé du flux de données, du coût des traitements à appliquer à ces données et des contraintes sur la latence, il n'est pas envisageable de traiter les données régulièrement de façon exhaustive sur une plate-forme centralisée.

Exemples d'applications

Avant de regarder de plus près l'architecture d'un système de traitement de flux et différents problèmes liés à ces traitements, il est utile de considérer quelques applications actuelles ou envisageables avec, à chaque fois, une

indication de la volumétrie correspondante.

— Données issues de capteurs ou de réseaux de capteurs. Les capteurs sont de plus en plus utilisés dans différents secteurs industriels pour mesurer des vibrations, des bruits, des températures ou des noxes. Les données ainsi obtenues doivent permettre d'optimiser le fonctionnement des machines, d'améliorer la planification des maintenances ou de prévoir les pannes. La volumétrie peut être très élevée, par exemple les différents capteurs présents dans un seul moteur d'avion de ligne génèrent env. 1 To de données par heure de vol. Des réseaux de capteurs sont déployés pour mesurer le trafic routier dans une ville, suivre la densité de polluants atmosphériques, etc. Cela peut permettre de mieux réguler le trafic et respectivement de choisir l'emplacement ou de paramétrer des systèmes de dépollution.

— Logs de moteurs de recherche génériques ou spécialisés. Ces données contiennent les requêtes, les adresses IP d'où ces requêtes ont été envoyées, éventuellement l'identité des utilisateurs ayant envoyé les requêtes (si ils se sont identifiés sur le site) et d'autres informations issues des *cookies*. L'analyse de ces données peut permettre de cibler les publicités, de prévoir l'évolution d'épidémies (par ex. grippe), d'estimer l'intérêt pour différents sujets ou l'impact de divers événements, de prévoir la demande de produits ou d'informations spécifiques, etc. Pour certaines applications (comme le ciblage de publicités) la latence doit être très faible. La volumétrie est variable, par ex. Google reçoit env. 50.000 requêtes par seconde.

— Données entrantes sur des sites de partage de contenus multimédia. Les données sont des contenus multimédia (images, vidéos, séquences sonores) avec d'éventuelles métadonnées (géolocalisation, date et heure, paramètres de prise de vue, etc.), les adresses IP d'où ces données sont envoyées, l'identité déclarée des utilisateurs (si ils se sont identifiés sur le site). L'analyse peut parfois se satisfaire des données non multimédia, mais de plus en plus se développent des méthodes qui travaillent aussi sur les contenus multimédia car les autres données ne sont pas suffisantes. Comme applications nous pouvons mentionner l'estimation de popularité ou d'impact, mais également la détection d'événements (au sens très général du terme) ou le filtrage (contenus protégés, dégradants, etc.). Lorsque le contenu multimédia est analysé, la volumétrie est importante et les 300 heures de vidéo mises en ligne chaque minute sur YouTube indiquent probablement une limite supérieure de ce qu'on peut atteindre aujourd'hui sur un site de ce type.

— Données de vidéosurveillance. Ces données sont des contenus vidéo et audio, complétées par des informations de localisation, éventuellement d'orientation et de focale des caméras. L'analyse du contenu multimédia de ces flux doit permettre de détecter des événements localisés, ainsi que d'éventuelles corrélations entre faits observés par des caméras différentes, etc. La volumétrie est variable mais, par exemple, à Londres on trouve près de 500.000 flux continus de vidéosurveillance.

— Imagerie satellite et aérienne. Ces flux de données, en fort développement ces dernières années, sont issus de satellites souvent en orbite héliosynchrone ou dont le positionnement est sur demande, ainsi que de drones à usage professionnel. Les données sont, en général, des séquences d'images optiques (dans le spectre visible ou non) ou radar pour les satellites et parfois des flux vidéo pour les drones. Les applications sont très diverses et incluent le suivi de l'impact de catastrophes (naturelles ou non) et la détection d'événements. Comme exemple de volumétrie nous pouvons mentionner les 2 To de données qu'envoient les deux satellites Pleiades (résolution 50 cm) chaque jour.

Architecture d'un système de traitement de flux de données

Le traitement d'un flux de données inclut plusieurs opérations, dont la création et le maintien de résumés du flux d'entrée, l'application de modèles sur le flux, la construction et mise à jour de modèles à partir du flux. La figure suivante montre l'architecture générale d'un système de traitement de flux de données.

Les différentes opérations sur le flux (résumés, construction et application de modèles) sont réalisées dans la partie « Gestion du flux ». Un système de traitement de flux doit pouvoir répondre à deux type de requêtes :

— Requêtes « continues », qui impliquent des traitements synchrones avec le flux. Par exemple, à partir du flux de requêtes reçues par un moteur de recherche, obtenir le nombre de requêtes par heure concernant une thématique spécifique. Les réponses aux requêtes continues constituent un flux de données de sortie du système de traitement.

— Requêtes asynchrones, qui impliquent des traitements asynchrones par rapport au flux. Par exemple, à partir des mesures réalisées par un réseau de capteurs de pollution qui couvre la région parisienne, obtenir la concentration moyenne de dioxyde d'azote sur les trois derniers jours à Paris. La réponse à une requête asynchrone a en général peu de contraintes de latence. Le traitement des requêtes asynchrones peut mobiliser ponctuellement des ressources supplémentaires, non utilisées par les traitements synchrones.

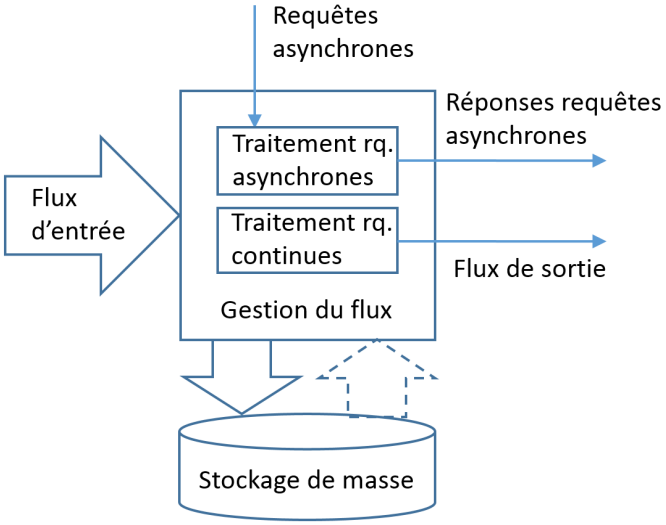


Fig. 7.1 – Traitement d'un flux de données

La construction et la mise à jour de résumés de flux ne sont pas explicitement représentées sur la figure, mais impliquent également des traitements synchrones, tout comme le traitement des requêtes continues.

Les données du flux d'entrée sont parfois conservées sur un stockage de masse, en totalité ou sous forme de résumés. Ce stockage est en général fait dans un but d'archivage, pour permettre de retrouver certaines données ultérieurement. Les données archivées ne sont pas utilisées dans des traitements synchrones car l'accès au stockage de masse est trop lent pour être compatible avec les contraintes de latence de ces traitements (le volume de données peut également être excessif).

Dans la suite nous nous intéressons à certains traitements synchrones typiques : échantillonnage dans un flux, filtrage de flux, modélisation à partir de flux et application de modèle sur un flux.

Échantillonnage dans un flux

Si un flux à un débit élevé, employer la totalité de ses données pour répondre à des requêtes (synchrones ou asynchrones) n'est pas possible en général car le coût des calculs et la latence imposée par l'accès au stockage de masse seraient excessifs. Parfois, il n'est même pas envisageable de stocker la totalité des données du flux pour archivage. La question de l'échantillonnage des données du flux se pose alors. L'objectif est d'extraire du flux un échantillon qui soit suffisamment représentatif pour permettre de répondre aux différentes requêtes prévues et d'assez faible volume pour pouvoir être conservé sur une durée relativement longue en stockage rapide (par ex. dans la mémoire vive des nœuds de calcul, même si un stockage non volatile est aussi assuré).

En général, chaque donnée (ou observation) d'un flux comporte des valeurs pour plusieurs attributs (ou variables) :

1. Réseau de capteurs : l'identifiant du capteur (ou sa position GPS), une étiquette temporelle de la mesure, la (les) valeur(s) du (des) paramètre(s) mesuré(s). Le flux est une succession de mesures envoyées par les capteurs disponibles, dans l'ordre d'arrivée, qui ne correspond pas nécessairement à un tri des valeurs d'un des attributs (pas même à l'étiquette temporelle de la mesure, car la durée du transit entre le capteur et le système de traitement peut être variable).
2. Log d'un moteur de recherche : l'adresse IP de l'utilisateur, une étiquette temporelle, les mots clés de la recherche, le type de navigateur employé, etc. Le flux est une succession de requêtes, dans l'ordre dans lequel elles ont été enregistrées dans le log du moteur de recherche.
3. Imagerie satellite : la position du satellite, les paramètres de prise de vue (domaine spectral, orientation du capteur, focale, résolution, etc.), une étiquette temporelle de la prise de vue, l'image captée, etc. Le flux est une succession d'images avec des méta-données, en général dans l'ordre des étiquettes temporelles.

L'échantillonnage simple d'un flux considère une même probabilité de sélection p_s pour chaque donnée du flux, indépendamment des valeurs des différents attributs. Regardons maintenant, en revenant aux trois exemples de flux mentionnés, dans quelle mesure un tel échantillonnage permet de répondre à certaines requêtes.

1. Réseau de capteurs : on souhaite estimer, à partir de l'échantillon, le pourcentage de capteurs qui indiquent une variation journalière supérieure à un seuil (θ) du paramètre mesuré. Si la probabilité de sélection est $p_s = 0,1$ (ou même $p_s = 0,01$), si chaque capteur transmet une mesure chaque seconde et si le paramètre mesuré varie lentement (par ex., la température ambiante), alors l'échantillon devrait permettre d'obtenir une bonne estimation. Mais les capteurs isolés sont souvent soumis à des contraintes fortes de consommation d'énergie, dans ce cas ils réalisent (et transmettent) seulement une mesure par heure ou même moins. Avec $p_s = 0,1$, pour bon nombre de capteurs nous ne trouvons plus qu'une seule mesure par jour dans l'échantillon, il n'est donc plus possible de mesurer une variation pour ces capteurs et l'estimation est fortement affectée. La solution est de sélectionner dans l'échantillon 10% des *capteurs* pour lesquels on conserve toutes les mesures, plutôt que 10% des *mesures* sans tenir compte du capteur ; le volume de données de l'échantillon est le même (10% du flux) mais cet échantillon permet de bien répondre à la requête.
2. Logs de moteurs de recherche : on cherche à estimer, à partir de l'échantillon, le pourcentage des requêtes de l'utilisateur typique qui sont *répétées* durant un intervalle d'un mois (voir aussi *[LRU11]* (page 132)). Supposons que la probabilité de sélection est $p_s = 0,1$ (nous souhaitons avoir un échantillon de 10% des requêtes) et que cette sélection est faite sans tenir compte des valeurs des attributs (adresse IP, étiquette temporelle, mots-clés de la recherche, etc.). Supposons que l'utilisateur typique fait en moyenne s requêtes non répétées par mois, d requêtes répétées une fois et un nombre négligeable de requêtes répétées plusieurs fois. Sur les s requêtes non répétées dans le flux, $s/10$ seront présentes (une fois) dans l'échantillon. Sur les d requêtes répétées (une fois) dans le flux, seulement $1/100$ ($= \frac{1}{10} \cdot \frac{1}{10}$) seront répétées (une fois) dans l'échantillon. En effet, chaque requête répétée dans le flux a deux occurrences, chacune de ces occurrences a une probabilité de $\frac{1}{10}$ d'être retenue dans l'échantillon ($p_s = 0,1$), pour que la requête soit répétée dans le flux il faut que chacune de ses deux occurrences soit retenue et les tirages correspondants sont indépendants. Aussi, sur les d requêtes répétées une fois dans le flux, $18/100$ ($= \frac{1}{10} \cdot \frac{9}{10} + \frac{9}{10} \cdot \frac{1}{10}$) seront présentes dans l'échantillon *une seule fois* (soit la première occurrence est présente et la seconde absente, soit la première est absente et la seconde présente). A partir de l'échantillon nous obtenons donc une estimation $\frac{\frac{d}{100}}{\frac{s}{10}+\frac{18d}{100}+\frac{d}{100}} = \frac{d}{10s+19d}$, très éloignée de la valeur obtenue à partir de toutes les requêtes, qui est de $\frac{d}{s+d}$. La solution consiste à sélectionner dans l'échantillon non 10% des requêtes mais plutôt 10% des utilisateurs, pour lesquels on conserve toutes les requêtes ; ici encore, le volume de données de l'échantillon reste le même (10% du flux) mais l'échantillon obtenu en sélectionnant les utilisateurs nous donne la possibilité de répondre à la requête. Par ailleurs, cet échantillon de 10% des utilisateurs permet de répondre également à d'autres questions concernant le comportement des utilisateurs.
3. Imagerie satellite : on souhaite estimer, à partir de l'échantillon, le pourcentage de positions pour lesquelles les paramètres de prise de vue sont identiques. La relation recherchée (ici, identité entre paramètres de prise de vue) ne peut pas être présente à un taux proche dans la population entière et dans un échantillon réduit ($p_s = 0,1$, par ex.) si celui-ci est constitué sans tenir compte des valeurs des attributs concernés par la relation (ici, les paramètres de prise de vue). Dans cet exemple, il sera en revanche possible d'obtenir la réponse si l'échantillon est constitué en sélectionnant $p_s \cdot 100\%$ des valeurs prises par l'attribut « paramètres de prise de vue ».

Pour pouvoir répondre à des requêtes à partir de l'échantillon, il est donc souvent nécessaire de *tenir compte des valeurs d'un ou plusieurs attributs* pour réaliser l'échantillonnage d'un flux. Dans le premier exemple il est nécessaire de sélectionner 10% des capteurs, dans le deuxième exemple 10% des utilisateurs et dans le troisième 10% des valeurs prises par l'attribut « paramètres de prise de vue ».

Cette sélection peut être réalisée soit par la recherche de la valeur dans une liste, soit en utilisant une méthode de hachage. La liste des valeurs qui doivent se retrouver dans l'échantillon du flux peut être obtenue *a priori*, en choisissant un échantillon des valeurs possibles lorsque leur domaine est bien connu. Cette liste peut également être construite progressivement, en y insérant avec une même probabilité p_s chaque valeur nouvellement observée.

Si la conservation de la liste des valeurs et la recherche dans cette liste sont trop coûteuses, il est possible d'utiliser une méthode de hachage. On définit une fonction de hachage qui a comme domaine l'ensemble des valeurs possibles de l'attribut visé et peut prendre $1/p_s$ valeurs de *hash* différentes ; on conserve une valeur de l'attribut dans l'échantillon si le *hash* est égal à une valeur fixée *a priori* (par ex. 0). Bien entendu, la fonction de hachage doit prendre chacune des valeurs de *hash* avec une même probabilité p_s.

Cette solution de hachage peut être facilement étendue pour permettre, entre autres :

— Un réglage plus fin de p_s (pour enlever la contrainte, implicite ci-dessus, que $1/p_s$ soit une valeur entière). Ainsi, avec 100 valeurs de *hash* différentes, en conservant la valeur de l'attribut si la valeur de *hash* fait

partie d'un ensemble fixé *a priori* de x valeurs, un échantillon de $x\%$ est obtenu ($p_s = x/100$).
— Une variation dans le temps du pourcentage lorsque, par exemple, le débit augmente. Dans un tel cas, p_s doit diminuer pour que le débit des données sélectionnées dans l'échantillon reste constant. Un hachage choisi pour obtenir un réglage fin permet facilement de réduire progressivement p_s (on choisit un ensemble de $x - 1$ valeurs sur les 100 lorsque le débit augmente, puis de $x - 2$ valeurs sur les 100 lorsque le débit augmente encore, et ainsi de suite).
— Un sélection tenant compte des valeurs de *plusieurs* attributs. Si un hachage est mis en place pour la sélection par rapport à chaque attribut dont la valeur compte, il est possible de construire des échantillons par rapport à des attributs individuels ou des groupes d'attributs (à condition que les valeurs des différents attributs puissent être choisies de façon indépendante).

Filtrage dans un flux et filtre de Bloom

Il est souvent nécessaire d'appliquer des traitements particuliers aux données issues d'un flux et qui satisfont certaines conditions. Un filtrage qui vérifie ces conditions doit être mis en place. Les conditions dont la vérification est peu coûteuse (comme par ex. le fait que la valeur d'un attribut soit dans un intervalle spécifique, ou la valeur d'un autre attribut dans un ensemble prédéfini réduit de valeurs) ne posent pas de difficulté particulière. En revanche, pour celles dont la vérification est très coûteuse, des solutions simples doivent être trouvées, au prix d'éventuelles approximations.

Considérons le cas où la condition de filtrage est l'appartenance de la valeur d'un attribut à un très grand ensemble de valeurs. Lorsque cette condition ne peut être vérifiée que par comparaison aux valeurs de cet ensemble ou à une partie des valeurs de l'ensemble, le coût (en termes de temps et de capacité mémoire mobilisée) peut être excessif par rapport aux exigences du traitement du flux. Ainsi, si pour résoudre ce problème on applique une méthode classique de hachage employée dans les bases de données relationnelles pour la recherche par identité, il sera nécessaire de comparer la valeur recherchée à toutes les valeurs stockées dans la « page » identifiée par le *hash* de cette valeur. Si la table de hachage est stockée sur disque il faudra d'abord lire la page concernée (après un temps d'accès comparativement élevé), si la table est conservée en mémoire alors l'occupation mémoire risque d'être excessive.

Considérons l'exemple d'un site de partage vidéo qui souhaite interdire, pour diverses raisons (présence d'incitations à la haine, contenus dégradants, non respect du droit d'auteur, etc.), la mise en ligne de vidéos **répertoriées** au préalable (de nouvelles vidéos peuvent néanmoins être ajoutées à cet ensemble). Sachant que les données textuelles accompagnant une vidéo peuvent être falsifiées par celui qui dépose la vidéo afin d'empêcher une détection facile par titre ou par mots clés, nous considérerons que les vidéos sont identifiées par un ensemble de descripteurs extraits de certaines trames vidéo particulières. Pour chaque vidéo, l'ensemble des descripteurs (nous l'appellerons « signature » de la vidéo) est volumineux. Aussi, supposons que le nombre de vidéos « interdites » est très élevé.

L'objectif est, dans ce cas, d'effectuer le filtrage
— Rapidement et avec relativement peu de ressources mémoire, afin de pouvoir traiter à un coût acceptable les données du flux. Pour avoir une idée de l'ordre de grandeur, rappelons que 300 h de vidéo sont mises en ligne chaque minute sur YouTube.
— En garantissant que les vidéos « interdites » ne se retrouvent pas en ligne. Dans le contexte décrit plus haut, cette garantie ne pourra toutefois être apportée que si le contenu des vidéos n'est pas modifié ou l'est suffisamment peu pour que la « signature » reste inchangée.
— En acceptant éventuellement que certaines autres vidéos (« bénignes ») soient également supprimées. Les cas litigieux peuvent éventuellement être résolus par des échanges explicites entre l'utilisateur qui souhaite déposer une vidéo et le service de modération du site.

Examinons maintenant ce qu'est un filtre de Bloom et de quelle manière il permet d'atteindre cet objectif.

La figure suivante illustre le fonctionnement d'un filtre de Bloom. Supposons que nous cherchons à filtrer certaines valeurs d'un attribut décrivant chaque donnée du flux et que la zone mémoire disponible pour réaliser le filtrage a une capacité de N bits ($N/8$ octets). Considérons une fonction de hachage qui prend en entrée une valeur de cet attribut (dans la figure, `p1`, `p2`, `p3`, `p4`, `n1`, `fp1`, etc.) et lui associe un *hash* entre 0 et $N - 1$. Appliquée à une même valeur de l'attribut, la fonction de hachage produira le même *hash*. Il suffit alors de positionner au départ à 1 les bits identifiés par les *hash* des valeurs à filtrer (en rouge dans la figure) et à 0 les autres bits (en blanc dans la figure) pour pouvoir facilement filtrer les données. Appliquée à une valeur qui fait partie de l'ensemble à filtrer (`p1`, `p2`, `p3` et `p4` dans la figure), la fonction de hachage produit un *hash* qui correspond à un bit à 1 et permet

ainsi de détecter l'appartenance à l'ensemble à filtrer. Appliquée à une autre valeur, la fonction de hachage produit en général (comme pour `n1` dans la figure) un *hash* qui correspond à un bit à 0 et indique que la valeur ne doit pas être filtrée. Dans certains cas (comme pour `fp1` dans la figure), il y a néanmoins « collision » avec un *hash* de valeur à filtrer, par ex. la valeur `fp1` est filtrée (car de même *hash* que `p1` ici) bien que *n'appartenant pas* à l'ensemble à filtrer ; nous sommes en présence d'un « faux positif ».

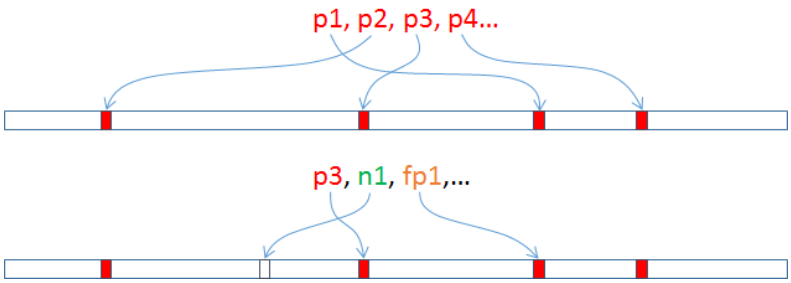


Fig. 7.2 – Filtre de Bloom

Pour réduire la probabilité de collision et donc le nombre de faux positifs il faut soit modifier la fonction de hachage pour augmenter le nombre de valeurs de *hash* différentes, soit employer plusieurs fonctions de hachage indépendantes. Dans ce dernier cas, une valeur est considérée appartenir à l'ensemble à filtrer si et seulement si pour chaque fonction de hachage le bit identifié par le *hash* est 1. Dans les deux cas, il est nécessaire d'utiliser plus d'espace mémoire pour réaliser le filtrage. On peut montrer que, pour M valeurs à filtrer et k fonctions de hachage, chacune avec N valeurs de *hash* différentes, le taux de faux positifs est $\approx (1 - e^{-kM/N})^k$ pour une occupation mémoire de kN bits (N bits par fonction). Le temps nécessaire pour filtrer une valeur correspond aux calculs des k fonctions de hachage pour obtenir les k *hash*, à la lecture des k cases mémoire identifiées et au calcul d'un ET logique entre leurs valeurs.

Revenons à l'exemple décrit plus haut, celui du filtrage de vidéos « interdites ». Pour réaliser ce filtrage avec un filtre de Bloom, il faut définir une fonction de hachage qui prend en entrée une signature de vidéo et lui associe un *hash* entre 0 et $N - 1$, où N est le nombre de bits de la zone mémoire utilisée pour le filtrage. Tous les bits de cette zone mémoire sont mis à 0. La fonction de hachage est ensuite appliquée à la signature de chaque vidéo interdite et le bit identifié par le *hash* obtenu est à chaque fois mis à 1. Supposons que la fraction de bits à 1 est $0 < r \ll 1$ ($r = M/N$, M étant le nombre de vidéos interdites).

Lorsqu'une vidéo arrive (par *upload*), sa signature est calculée et ensuite la fonction de hachage est appliquée à cette signature. Si la valeur du bit identifié par le *hash* obtenu est 1 alors la vidéo n'est pas mise en ligne, si la valeur est 0 alors la vidéo est mise en ligne.

Pour toute vidéo qui fait partie de l'ensemble des vidéos interdites, le bit identifié par le *hash* est 1 **par construction**, il n'y a donc **pas de faux négatifs** ! Pour une vidéo qui ne fait pas partie de l'ensemble, la probabilité pour que le bit identifié par le *hash* soit 1 (et donc que la vidéo soit à tort interdite) n'est pas nulle mais proche de r, il y a donc des faux positifs. En augmentant le nombre de fonctions de hachage, comme expliqué plus haut, on peut diminuer le taux de faux positifs (sans impact sur le taux de faux négatifs qui sera toujours de 0) au prix d'une augmentation de la consommation mémoire. Le temps nécessaire pour le filtrage d'une vidéo correspond au calcul de la signature, au calcul de la (des) fonction(s) de hachage du filtre de Bloom et à la lecture de la (des) case(s) mémoire identifiée(s).

Fenêtres sur un flux

Que ce soit pour des traitements synchrones ou pour répondre à des requêtes asynchrones, on pourrait considérer nécessaire de traiter toutes les données du flux (ou d'un échantillon extrait du flux). Il y a au moins deux bonnes raisons pour ne pas employer systématiquement la totalité de l'historique conservé.

1. Même avec un échantillonnage du flux, le volume de l'échantillon accumulé sur une longue période peut être tel que le coût de traitement soit excessif. Ce sera vraisemblablement le cas, par exemple, pour le flux de requêtes reçues par un site très populaire ou pour le flux d'images issues d'un satellite d'observation de la Terre.

2. Les distributions des valeurs des différents attributs qui décrivent les données du flux sont souvent non stationnaires, remonter loin dans l'historique peut être non pertinent pour l'indication recherchée. Par exemple, pour estimer la popularité d'un sujet d'actualité ou celle d'un film sorti en salles, cumuler depuis le début les nombres de messages échangés par heure ou respectivement le nombre d'entrées par jour ne permet pas d'obtenir une indication très utile. Mieux vaut s'intéresser (ou du moins privilégier) la période récente (les deux derniers jours pour le sujet d'actualité ou la dernière semaine pour le film sorti en salles).

La solution est de se limiter au traitement des données du flux qui sont situées dans une fenêtre temporelle glissante. Si l est la longueur de la fenêtre, les données des tranches $i, i-1, i-2, \ldots, i-l+1$ du flux sont prises en compte (la tranche i est la dernière tranche reçue) alors que les données des tranches qui précèdent $(i-l, i-l-1, \ldots)$ sont ignorées.

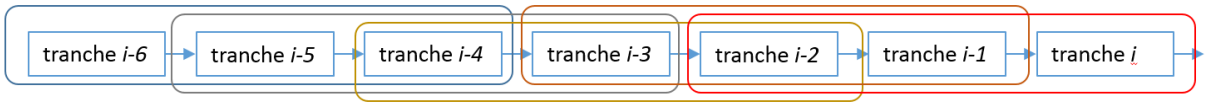


Fig. 7.3 – Fenêtre temporelle glissante appliquée à un flux de données

La fenêtre glissante peut être
- de longueur fixe l prédéfinie : techniquement la solution la plus simple, bien adaptée quand c'est la non stationnarité qui motive l'emploi de fenêtres temporelles ;
- de volume fixe (la longueur l_i variera donc suivant les volumes de données des tranches successives) : cas préférable lorsque c'est plutôt le trop grand volume de données qui justifie l'utilisation de fenêtres.

Les fenêtres définies plus haut sont « nettes », une tranche est soit dans la fenêtre (donc prise en compte), soit en dehors de la fenêtre (donc ignorée). Il peut être utile de *pondérer* les tranches, afin de prendre plus en compte les tranches récentes et moins les tranches plus anciennes (tout en évitant de les ignorer complètement). Nous obtenons ainsi des **fenêtres avec oubli** (*decaying windows*) : la pondération d'une tranche diminue avec l'augmentation de l'écart temporel entre cette tranche et la tranche courante (dernière tranche reçue). Cette diminution continue des pondérations a également comme conséquence un « lissage » des estimations obtenues à partir des fenêtres du flux.

Un cas fréquent est celui de la décroissance exponentielle des pondérations. Prenons l'exemple de l'estimation de la popularité d'un sujet d'actualité ou d'un film sorti en salles. Considérons que chaque tranche est caractérisée par une valeur numérique, m_i pour la tranche i, correspondant au nombre de messages échangés par heure (pour un sujet d'actualité) ou respectivement au nombre d'entrées par jour (pour un film sorti en salles). Si on note par s_i la mesure de popularité obtenue après l'arrivée de la tranche i (donc de la valeur m_i), une décroissance exponentielle des pondérations correspond à la relation de définition suivante (pour $i \geq 1$) :

$$s_i = \sum_{j=1}^{i} d^{i-j} \cdot m_j$$

où $0 < d < 1$ est le « facteur d'oubli » (*decay factor*). La borne inférieure $j = 1$ indique que la somme est calculée depuis le début du flux. Heureusement, pour faire ce calcul avec une décroissance exponentielle des pondérations il n'est pas nécessaire de conserver les données depuis le début du flux car la relation de récurrence suivante permet d'arriver au même résultat (la vérification est facile) :

$$s_i = m_i + s_{i-1} \cdot d, \quad s_0 = 0$$

Mise à jour et application de modèles : l'exemple de *Streaming K-means*

Deux traitements synchrones importants sur les données d'un flux sont
- l'application d'un modèle existant, pré-estimé, sur les tranches successives du flux,
- la mise à jour d'un modèle en utilisant les données de chaque tranche du flux.

Lorsqu'elle est faite à partir de données en flux, la construction d'un modèle doit être *incrémentale* : il faut pouvoir mettre à jour le modèle existant avec les données d'une tranche sans avoir à le ré-estimer complètement car le coût de calcul avec toutes les données du flux (depuis le début) serait incompatible avec un traitement synchrone.

Par ailleurs, si les données du flux ont une distribution qui évolue dans le temps (n'est pas stationnaire), il peut être intéressant de donner des pondérations plus fortes aux données récentes du flux et des pondérations plus faibles aux données plus anciennes. Pour que cela soit compatible avec la mise à jour du modèle à partir des données de la dernière tranche, il faut pouvoir sous-pondérer le modèle antérieur (obtenu après la tranche précédente, qui intègre donc l'impact des données plus anciennes) dans la construction du nouveau modèle.

Considérons un cas concret simple, celui de la classification automatique avec *k-means* des données d'un flux. On souhaite pouvoir appliquer le modèle courant aux données de la dernière tranche afin de les affecter aux groupes déjà trouvés et ensuite mettre à jour ce modèle en tenant compte de ces données.

Rappelons-nous que pour *k-means* un modèle consiste en l'ensemble des centres de gravité des k groupes, $\mathcal{C} = \{\mathbf{m}_j, 1 \leq j \leq k\}$. Le flux correspond à une séquence d'ensembles $\mathcal{E}_\sqcup$ de n_t données de $\mathbb{R}^p$, où $t = 1, 2, \ldots$ est l'indice des tranches successives du flux. Nous cherchons à affecter chaque donnée de chaque tranche à un groupe (celui du centre le plus proche de la donnée) et ensuite à mettre à jour le modèle (les centres) à partir des données de la dernière tranche. L'algorithme global sera le suivant :

`Entrées` : tranches du flux, c'est à dire ensembles $\mathcal{E}_t$ de n_t données de $\mathbb{R}^p$, pour $t = 1, 2, \ldots$;
`Sorties` : après chaque tranche, k groupes (*clusters*) disjoints $\mathcal{E}_{1,t}, \ldots, \mathcal{E}_{k,t}$ (qui sont des sous-ensembles de $\mathcal{E}_t$) et ensemble $\mathcal{C}_\sqcup$ de leurs centres ;

1. Initialisation des centres $\mathbf{m}_{j,1}$, $1 \leq j \leq k$, à partir des données de la première tranche (par exemple, avec *k-means*‖) ;
2. `pour` (chaque tranche $\mathcal{E}_t$ du flux) `faire`
   (a) Affectation de chaque donnée de la tranche au groupe du centre le plus proche ;
   (b) Mise à jour des centres et remplacement des anciens centres par les nouveaux ;

`fin pour`

L'affectation de chaque donnée de chaque tranche au groupe du centre le plus proche (parmi les centres courants, donc de $\mathcal{C}_\sqcup$), étape 2.1 de l'algorithme, ne pose pas de difficulté particulière, toutes les distances entre les données de la tranche et les centres courants doivent être calculées, comme pour *k-means*.

En revanche, la mise à jour des centres (étape 2.2) doit se faire uniquement avec les données de la dernière tranche, tout en tenant compte des données plus anciennes indirectement, à travers les anciens centres. La relation de mise à jour est

$$\mathbf{m}_{j,t+1} = \frac{\alpha N_{j,t}\mathbf{m}_t + n_{j,t}\mathbf{x}_{j,t}}{\alpha N_{j,t} + n_{j,t}}$$
$$N_{j,t+1} = N_{j,t} + n_{j,t}$$

où $\mathbf{m}_{j,t+1}$ est le centre j mis à jour dans l'itération courante, $\mathbf{m}_{j,t}$ est le centre j avant sa mise à jour, $n_{j,t}$ est le nombre de données de la tranche courante affectées à ce centre, $N_{j,t}$ est le nombre de données affectées au centre j depuis le début, $\mathbf{x}_{j,t}$ est le centre correspondant calculé uniquement avec les données de la tranche courante et $0 \leq \alpha \leq 1$ est le « facteur d'oubli ».

Après l'affectation des données de la dernière tranche aux centres dans l'étape 2.1, pour chaque groupe j on détermine d'abord le centre de gravité $\mathbf{x}_{j,t}$ des données de cette tranche affectées à ce groupe. Ensuite, le nouveau centre $\mathbf{m}_{j,t+1}$ du groupe j est calculé comme une moyenne pondérée entre l'ancien centre $\mathbf{m}_{j,t}$ (contribution des données des tranches précédentes) et $\mathbf{x}_{j,t}$ (contribution des données de la dernière tranche).

Si $\alpha = 1$, les anciennes données du flux « pèsent de tout leur poids » sur l'évolution des centres, si les rapports $\frac{n_{j,t}}{N_{j,t}}$ sont faibles alors les centres évoluent très peu d'une itération à la suivante (sauf au démarrage du flux). Si $\alpha = 0$ alors les anciens centres sont ignorés dans le calcul des nouveaux centres, l'impact des données de la tranche précédente se retrouve seulement dans l'étape d'affectation des nouvelles données aux centres issus de l'itération précédente (étape 2.1 de l'algorithme).

Les quatre illustrations suivantes montrent un cas simple de classification automatique avec *streaming K-means* d'un flux de données de $\mathbb{R}^3$. Les données ont été générées à partir de sept lois d'assez faible variance, ensuite sérialisées dans les tranches successives d'un flux. La distribution des données est ici stationnaire et $\alpha = 0.5$. Une couleur différente est utilisée pour chaque groupe.

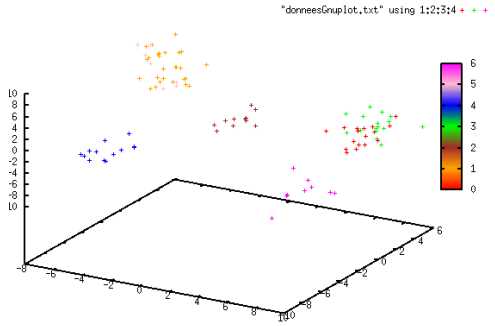


Fig. 7.4 – Groupes après l'arrivée des premières données

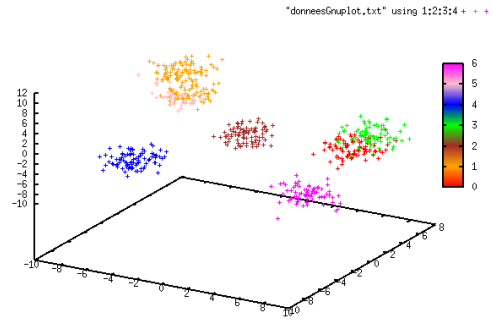


Fig. 7.5 – Groupes après l'arrivée de 30% des données

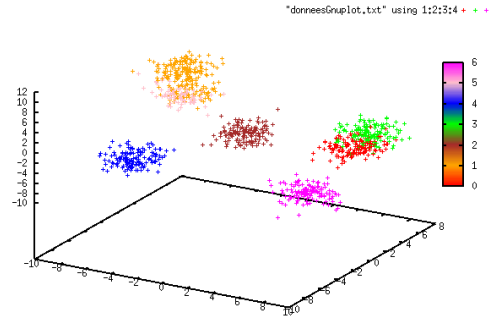


Fig. 7.6 – Groupes après l'arrivée de 60% des données

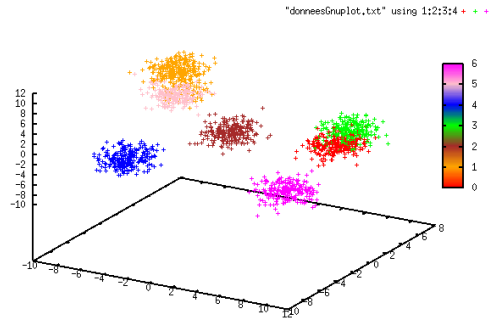


Fig. 7.7 – Groupes après l'arrivée de toutes les données

Spark Streaming

Spark Streaming est une interface de programmation qui permet de traiter des flux de données sur une plate-forme Spark. Les données peuvent être reçues de différentes façons : transferts dans le système de fichiers (voir l'exemple donné dans la séance de travaux pratiques), réception sur des *sockets* TCP (connexions réseau génériques), ou obtenues de Twitter (voir aussi cette séance complémentaire de travaux pratiques), Kafka, Flume, etc.

Un flux de données correspond dans *Spark Streaming* à une abstraction appelée flux discrétisé (`DStream`). Chaque `DStream` est représenté par une séquence de RDD (structure de données distribuée générique de Spark). Un `DStream` est créé à partir d'un `StreamingContext`, en précisant la durée d'une tranche (*slice*). Les données reçues (par un des moyens mentionnés plus haut) sont découpées en tranches en respectant cette durée. Chaque tranche est un RDD qui peut être traité ou transformé comme un RDD quelconque.

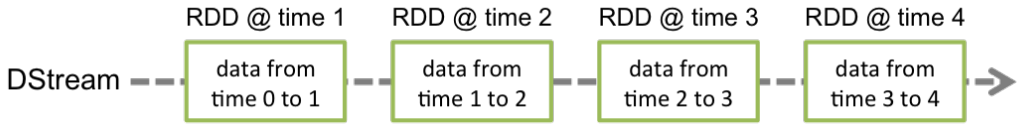


Fig. 7.8 – Un `DStream` est une séquence de RDD (figure issue de la documentation de Spark)

Plusieurs opérations peuvent être directement appliquées aux flux, chaque flux étant représenté par un `DStream` : transformation d'un flux pour obtenir un autre flux, fusion de plusieurs flux en un seul, jointure de flux, jointure entre un flux et un RDD unique, filtrage d'un flux à partir d'un autre flux, mise à jour d'un « état » à partir d'un flux, etc. Ces opérations peuvent être appliquées aussi bien à travers `spark-shell` qu'à partir d'un programme.

Toute source de données en flux (à l'exception de la source « système de fichiers ») doit avoir un `Receiver` associé pour récupérer les données reçues, les découper en tranches et distribuer le contenu de chaque tranche (représentée par un RDD) aux nœuds de calcul.

Spark Streaming permet d'employer des fenêtres glissantes sur un flux. Les fenêtres sont définies par deux valeurs (entières) : le nombre de tranches qui se trouvent dans la fenêtre à chaque moment (longueur de la fenêtre) et le nombre de tranches de décalage à chaque déplacement de la fenêtre (pas du glissement). Les RDD correspondant aux tranches qui sont dans la fenêtre sont regroupés avant d'être traités ; les opérations appliquées sur les RDD individuels (tranches) sont étendues aux fenêtres (par ex. `reduceByKeyAndWindow`).

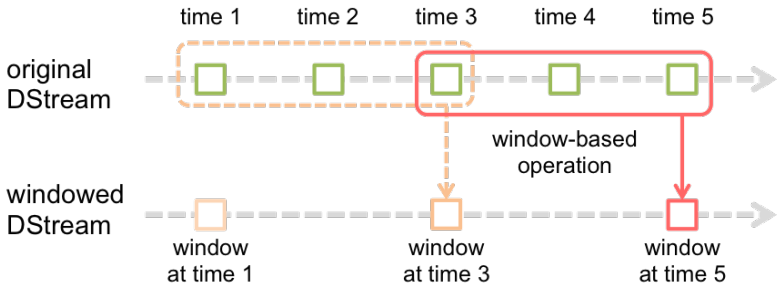


Fig. 7.9 – Utilisation de fenêtres glissantes dans Spark (figure issue de la documentation de Spark)

Cours - Apprentissage supervisé à large échelle

[Diapositives du cours (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/8apprentissageLargeEchelle.pdf)]

Un modèle décisionnel doit permettre de prédire, pour chaque nouvelle donnée (ou observation), la valeur d'une variable *expliquée* (ou « dépendante », ou « de sortie ») à partir des valeurs prises par les autres variables (« explicatives », ou « d'entrée ») pour cette donnée. Dans un problème de *discrimination* entre deux ou plusieurs classes, la variable expliquée est une variable nominale, dont chaque modalité correspond à une des classes possibles. Dans un problème de *régression*, la variable expliquée est une variable quantitative. Dans un problème de *prédiction structurée*, la variable expliquée prend des valeurs dans un domaine de données structurées (par ex. de graphes).

Un modèle décisionnel peut être construit analytiquement, à partir d'une parfaite compréhension du phénomène observé, ou par des moyens statistiques, sur la base d'un ensemble d'observations disponibles. Parfois on a une compréhension *partielle* du phénomène observé, qui permettra de définir des *a priori* utiles pour l'identification statistique du modèle. L'apprentissage supervisé vise à construire un modèle décisionnel à partir de données pour lesquelles une information de « supervision » est disponible. Cette information correspond en général, pour chaque donnée, à la valeur prise par la variable expliquée.

Avant d'aller plus loin, rappelons que la prise de décision peut être faite à partir d'un ensemble d'observations disponibles *sans* construction préalable d'un modèle. Pour décider de la valeur de la variable expliquée pour une nouvelle donnée il est possible d'utiliser, par exemple, directement les k plus proches observations pour lesquelles la valeur de la variable expliquée est connue.

La figure suivante illustre un problème de discrimination entre deux classes dans $\mathbb{R}^2$. La variable expliquée possède une modalité pour chaque classe. Les points clairs sont les données ayant permis de construire de modèle et le trait noir est la frontière de discrimination à laquelle peut se réduire ici le modèle décisionnel. Toute nouvelle donnée se situera d'un côté ou de l'autre de cette frontière et sera ainsi affectée par le modèle à une des classes. Ce modèle peut éventuellement être complété par des critères de *rejet* (refus d'affectation), par ex. on peut refuser de classer les données trop proches de la frontière (rejet d'ambiguïté) ou trop éloignées des données ayant permis de construire de modèle (rejet de non représentativité).

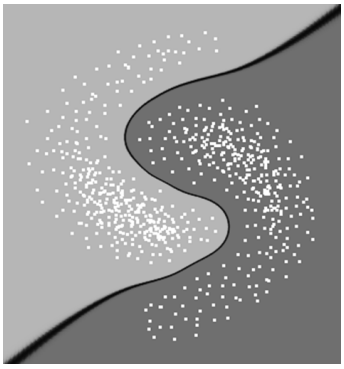


Fig. 8.1 – Exemple de discrimination entre deux classes dans $\mathbb{R}^2$. Les points clairs sont les données d'apprentissage et le trait noir la frontière de discrimination obtenue.

L'apprentissage supervisé n'est pas la seule approche de construction de modèles à partir de données. Nous pouvons mentionner, entre autres :

— L'apprentissage non supervisé. Dans ce cas, aucune information de supervision n'est disponible, par ex. on ne connaît la classe d'appartenance pour aucune des données disponibles. Souvent, on ne sait même pas quelles sont les classes pertinentes pour caractériser les données. *La classification automatique* (page 53) fait partie de cette famille. Le modèle obtenu par apprentissage non supervisé est en général employé comme modèle *descriptif* et aide à mieux définir un problème décisionnel qui est traité ultérieurement (par ex., à mettre en évidence des groupes ou *clusters* qui ont un rapport étroit avec une ou plusieurs classes dont la détection serait utile).

— *L'apprentissage semi-supervisé* (page 29) (voir par ex. *[CSZ06]* (page 132)). Dans ce cas, l'information de supervision est disponible sous une forme partielle, par ex. pour une (faible) partie des données d'apprentissage. Par ailleurs, la connaissance *a priori* du problème traité permet éventuellement de valider des hypothèses sur le lien entre la distribution des données et la variable expliquée (par ex., la séparation entre classes correspond à des régions de faible densité, comme dans la figure précédente). Ces hypothèses permettent de se servir des données qui n'ont pas d'information de supervision afin d'améliorer la modélisation décisionnelle par rapport à l'utilisation exclusive des données pour lesquelles la supervision est présente.

Dans la suite de ce chapitre nous nous intéresserons d'abord à la formulation d'un problème d'apprentissage supervisé et à l'évaluation des performances de généralisation du modèle décisionnel ainsi obtenu. Seront étudiées également les questions de passage à l'échelle et leurs implications pour l'apprentissage supervisé. Nous examinerons ensuite une famille très populaire d'outils de modélisation décisionnelle, les machines à vecteurs de support. Pour trouver le modèle à partir des données d'apprentissage il est en général nécessaire de choisir les valeurs d'un ou plusieurs hyperparamètres (qui permettent de mieux définir la famille de modèles candidats) et de procéder à une optimisation pour déterminer les paramètres du modèle. Les hyperparamètres ont un impact significatif sur les résultats, nous regarderons comment faire la recherche des meilleures valeurs pour ces hyperparamètres. Enfin, nous passerons brièvement en revue les outils proposés par la plate-forme Spark pour l'apprentissage supervisé. L'utilisation de machines à vecteurs de support dans Spark sera abordée dans la séance de travaux pratiques.

Apprentissage supervisé et généralisation

Considérons un espace d'entrée $\mathcal{X}$, dans lequel prennent des valeurs les variables explicatives, et un espace de sortie $\mathcal{Y}$ dans lequel prend des valeurs la variable expliquée (ou variable « dépendante »).

Les variables explicatives numériques sont en général représentées par des variables réélles et l'espace d'entrée est $\mathbb{R}^p$ (p étant le nombre de variables explicatives numériques). Les variables explicatives nominales peuvent être représentées par des variables numériques, à condition de ne pas « travestir » leur nature. Par exemple, entre les modalités d'une variable nominale « catégorie socio-professionnelle » il n'y a pas de relation d'ordre ni de similarité directement quantifiable. Représenter ces modalités par des valeurs différentes d'une même variable numérique introduirait à la fois une relation d'ordre et des similarités quantifiables, étrangères à la nature de la variable de départ et avec un impact négatif fort sur la modélisation. Une variable nominale doit plutôt être représentée à travers un codage disjonctif, avec une variable binaire par modalité (1 pour la modalité présente, 0 pour les autres). Il est également possible d'appliquer en amont une analyse des correspondances multiples aux données décrites par des variables nominales afin de les représenter par les variables quantitatives qui sont les facteurs issus de l'analyse (en nombre en général bien plus faible que les variables nominales de départ).

Lorsque les valeurs des variables explicatives sont des données structurées (par ex. des arbres d'analyse grammaticale ou des graphes de structure de protéines), leur représentation par des ensembles de variables quantitatives est plus problématique. Des outils de modélisation comme les machines à vecteurs de support peuvent employer des fonctions-noyau spécifiques à de tels types de données, sans passer par une représentation vectorielle intermédiaire.

Pour un problème de discrimination entre deux classes, l'espace de sortie est souvent $\{-1; 1\}$. Lorsque le nombre de classes est supérieur à deux, le problème est le plus souvent décomposé en un ensemble de problèmes à deux classes, comme nous le verrons plus loin. Pour un problème de régression, l'espace de sortie est en général $\mathbb{R}$.

Pour la modélisation on dispose des données $\mathcal{D}_N = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq N}$, l'information de supervision correspondant aux valeurs $\{y_i\}_{1 \leq i \leq N}$. On considère en général que ces données sont des réalisations des variables aléatoires

$\mathcal{D}_N = \{(X_i, Y_i)\}_{1 \leq i \leq N}$, où $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$, (X_i, Y_i) respectent la loi (inconnue) P pour tout i et (X_i, Y_i) sont indépendantes de (X_j, Y_j) pour $i \neq j$.

L'objectif de l'apprentissage supervisé est d'obtenir le modèle décisionnel qui présente la meilleure *généralisation*, à partir de données pour lesquelles l'information de supervision est disponible. Cela revient en général à trouver, dans une famille de fonctions $\mathcal{F}$, une fonction $f : \mathcal{X} \to \mathcal{Y}$ qui « prédit » y à partir de $\mathbf{x}$ en minimisant le risque *espéré* (ou risque théorique, ou erreur de généralisation) :

$$R(f) = E_P[L(X, Y, f)] \tag{8.1}$$

Dans cette expression, E_P est l'espérance par rapport à la loi inconnue P et L est la fonction de « perte » ou d'erreur.

Les fonctions de « perte » (ou d'erreur) L les plus fréquemment employées sont :

— La perte quadratique pour la régression : $L_Q(\mathbf{x}, y, f) = [f(\mathbf{x}) - y]^2$, $\mathbf{x}$ étant l'observation d'entrée, $f(\mathbf{x})$ la prédiction du modèle et y l'information de supervision associée à cette observation. L'erreur est donc le carré de la différence entre la valeur prédite par le modèle et la valeur observée.

— La perte 0-1 pour la discrimination entre 2 classes : $L_{0-1}(\mathbf{x}, y, f) = \mathbf{1}_{f(\mathbf{x}) \neq y}$, $y \in \{-1; 1\}$. L'erreur est nulle si la classe prédite est la même que la classe observée, et égale à 1 sinon.

— *Hinge loss* pour la discrimination entre 2 classes en maximisant la marge (nous y reviendrons dans la section qui traite les machines à vecteurs de support) : $L_H(\mathbf{x}, y, f) = \max\{0, 1 - yf(\mathbf{x})\}$, $y \in \{-1; 1\}$. Contrairement à la perte 0-1, *hinge loss* est continue et différentiable partout sauf dans $yf(\mathbf{x}) = 1$. Des variantes différentiable partout (« lissées ») existent également.

Détermination du modèle

Pour obtenir un modèle décisionnel à partir des données de $\mathcal{D}_N$, il est nécessaire de choisir une famille paramétrique $\mathcal{F}(\mathbf{w})$ de modèles (par ex. les modèles linéaires) et ensuite d'appliquer un algorithme d'optimisation pour déterminer le (vecteur de) paramètre(s) optimal $\mathbf{w}^*$ qui définit le modèle dans la famille $\mathcal{F}(\mathbf{w})$.

L'objectif est de trouver le modèle qui présente la meilleure généralisation. Comme le risque espéré $R(f)$ d'un modèle f ne peut pas être directement évalué car P est inconnue, il n'est pas envisageable de déterminer le modèle en minimisant directement son risque espéré. En revanche, il est possible de mesurer le risque empirique de f, c'est à dire l'erreur de f sur les données de $\mathcal{D}_N$:

$$R_{\mathcal{D}_N}(f) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}_i, y_i, f) \tag{8.2}$$

Le modèle $f \in \mathcal{F}$ étant défini par un paramètre $\mathbf{w}$, le risque empirique $R_{\mathcal{D}_N}(f)$ est également directement exprimable comme une fonction de ce paramètre et il est alors envisageable de chercher le paramètre optimal $\mathbf{w}^*$ qui minimise le risque empirique :

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} R_{\mathcal{D}_N}(f(\mathbf{w})) \tag{8.3}$$

Dans quelle mesure cette **minimisation du risque empirique** permet d'obtenir un modèle qui a un risque théorique minimal ? Il faut examiner deux aspects différents :

1. La cohérence (*consistency*) de la minimisation du risque empirique (MRE) : quand le nombre N de données d'apprentissage tend vers l'infini, le risque empirique (l'erreur d'apprentissage) converge-t-il vers le risque espéré (l'erreur de généralisation) ? Différents résultats existent concernant cette question (voir par ex. *[BBL99]* (page 132)) ; on peut ainsi montrer que la MRE est cohérente si et seulement si la VC-dimension (la dimension de Vapnik-Chervonenkis) de la famille $\mathcal{F}$ (dans laquelle on cherche f) est finie.
2. Si le nombre N de données d'apprentissage est fini, est-ce possible de borner l'écart entre le risque espéré et le risque empirique pour un modèle f ? On peut montrer (voir par ex. *[Vap98]* (page 132), *[BBL99]* (page 132)) que de telles bornes existent et dépendent à la fois de la « capacité » de $\mathcal{F}$ (par ex. de sa VC-dimension) et du nombre N de données d'apprentissage : $R(f) \leq R_{\mathcal{D}_N}(f) + B(N, \mathcal{F})$. La borne $B(N, \mathcal{F})$ diminue (est plus stricte) lorsque $\mathcal{F}$ est de « capacité » plus faible et lorsque N augmente, mais reste en général trop élevée pour permettre de borner $R(f)$ de façon utile.

Ces résultats sont intéressants d'un point de vue théorique mais leur utilité pratique est limitée. Avec un ensemble de données d'apprentissage $\mathcal{D}_N$ fini, la minimisation du risque empirique peut avoir comme résultat un modèle qui généralise mal (sur-apprentissage ou apprentissage « par cœur ») même pour une famille $\mathcal{F}$ de capacité finie et relativement faible. Les deux figures suivantes illustrent un tel cas où le modèle (la frontière de discrimination entre deux classes) issue de la MRE a un risque empirique nul mais une erreur de généralisation (sur des données non employées pour l'apprentissage) comparativement élevée.

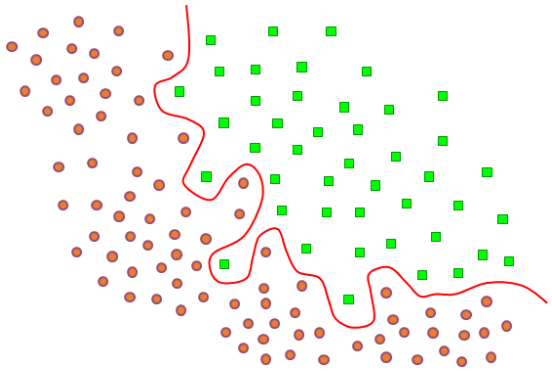


Fig. 8.2 – Risque empirique nul

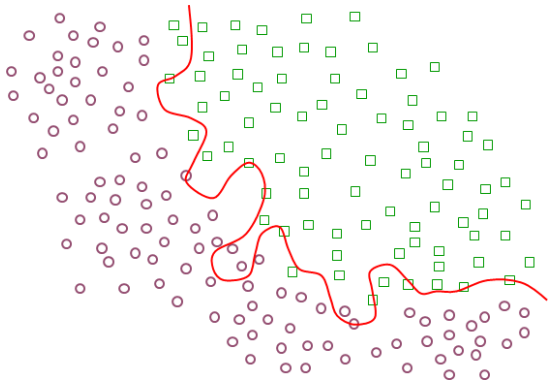


Fig. 8.3 – Erreur comparativement élevée sur des données de test (non utilisées pour l'apprentissage)

Une solution fréquemment adoptée est la **minimisation du risque empirique régularisé** (MRER), qui cherche à réduire le risque empirique tout en pénalisant la « complexité » du modèle :

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left[R_{\mathcal{D}_N}(f(\mathbf{w})) + \text{Reg}(f(\mathbf{w})) \right] \tag{8.4}$$

Si la minimisation du seul risque empirique mène à une augmentation de la complexité du modèle, le terme de régularisation $\text{Reg}(f(\mathbf{w}))$ doit pénaliser plus ce modèle. Le résultat est un compromis entre la réduction de l'erreur d'apprentissage et l'augmentation de la complexité du modèle. Le terme de régularisation cherche, par exemple, à privilégier les modèles « lisses » et son expression exacte peut dépendre de la nature de la famille $\mathcal{F}$.

Une approche alternative est la **minimisation du risque structurel** (MRS) : on considère une séquence $\mathcal{F}_d$, $d \in \mathbb{N}$ de modèles de « capacité » d croissante, dans chaque famille on minimise le risque empirique, ensuite on pénalise la capacité de la famille. Sont ainsi obtenus plusieurs modèles dans des familles de capacité croissante, le modèle retenu est celui qui présente le meilleur compromis entre risque empirique et capacité :

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left[R_{\mathcal{D}_N}(f(\mathbf{w})) + \text{pen}(N, d) \right] \tag{8.5}$$

Sans garanties *a priori* sur le risque espéré (l'erreur de généralisation) d'un modèle obtenu par MRE ou MRER, il est important de pouvoir estimer ce risque afin de savoir dans quelle mesure le modèle peut être employé pour la prise de décision.

Estimation des performances de généralisation

L'estimation directe du risque espéré par le risque empirique (estimation *in-sample*) serait excessivement optimiste pour des familles $\mathcal{F}$ de capacité élevée (ou infinie) et doit donc être évitée. Lorsque la capacité de $\mathcal{F}$ est faible et le nombre N de données d'apprentissage très élevé, des bornes de généralisation $B(N, \mathcal{F})$ suffisamment réduites pour majorer de façon utile le risque espéré peuvent éventuellement être obtenues : $R(f^*_{\mathcal{D}_N}) \leq R_{\mathcal{D}_N}(f^*_{\mathcal{D}_N}) + B(N, \mathcal{F})$. Cette situation est toutefois rare, même pour des données massives.

Une méthode générale pour estimer le risque espéré est celle de l'échantillon-test (estimation *out-of-sample*) :

1. L'ensemble de données disponibles $\mathcal{D}_N$ est partitionné en deux ensembles mutuellement exclusifs par sélection aléatoire, les données d'apprentissage $\mathcal{A}$ et les données de validation $\mathcal{V}$.
2. L'apprentissage du modèle est réalisé sur les données de $\mathcal{A}$ en utilisant une des approches mentionnées la MRE ou la MRER.
3. Le risque espéré du modèle résultant est estimé sur les données de $\mathcal{V}$ (estimation *out-of-sample*).

Apprendre sur une partie seulement des données disponibles ($\mathcal{A}$) peut être un inconvénient sérieux lorsque N est faible. Même pour les données massives, certaines classes (ou certains comportements intéressants) peuvent être relativement rares dans les données, diminuer encore le nombre d'observations appartenant à ces classes (ou présentant ces comportements) a un impact négatif sur la qualité du modèle résultant.

Aussi, l'estimateur ainsi obtenu pour le risque espéré a une variance élevée (un autre découpage de $\mathcal{D}_N$ en $\mathcal{A}'$ et $\mathcal{V}'$ peut produire un résultat assez différent). Pour réduire la variance de l'estimateur il faudrait moyenner les résultats issus de plusieurs découpages différents de $\mathcal{D}_N$, c'est ce que proposent les méthodes de validation croisée.

Avant d'examiner de plus près ces méthodes, il est important de noter qu'un problème de validité important rencontré dans la pratique est la *non stationnarité* : les données d'apprentissage deviennent de moins en moins représentatives au fil du temps car la loi conjointe inconnue P évolue. Il est donc important d'estimer régulièrement l'erreur sur des données *récentes* afin de mettre en évidence une éventuelle divergence du modèle et donc la nécessité de le mettre à jour.

La validation croisée

Afin de réduire la variance de l'estimation du risque espéré obtenue sur un échantillon-test, plusieurs partitionnements différents de $\mathcal{D}_N$ en $\mathcal{A}_i$ et $\mathcal{V}_i$ sont réalisés, avec $i \in 1, \ldots, k$, $\mathcal{D}_N = \mathcal{A}_i \cup \mathcal{V}_i$ et $\mathcal{A}_i \cap \mathcal{V}_i = \emptyset$, à chaque fois un modèle f_i est appris sur $\mathcal{A}_i$, son erreur $L(\mathcal{V}_i, f_i)$ est calculée sur $\mathcal{V}_i$, et enfin le risque espéré est estimé par la moyenne $\frac{1}{k}\sum_{i=1}^{k} L(\mathcal{V}_i, f_i)$.

Suivant les partitionnements réalisés, les méthodes peuvent être

1. Exhaustives, lorsque tous les partitionnements possibles respectant certains effectifs sont utilisés :
   — *Leave p out* : $N-p$ données sont employées pour l'apprentissage ($\mathcal{A}_i$) et p pour la validation ($\mathcal{V}_i$), tous les C_N^p partitionnements possibles avec ces effectifs sont utilisés. Il est donc nécessaire d'apprendre C_N^p modèles différents, ce qui implique en général un coût excessif.
   — *Leave one out* : $N-1$ données sont employées pour l'apprentissage ($\mathcal{A}_i$) et *une seule* pour la validation ($\|\mathcal{V}_i\| = 1$), il y a donc $C_N^1 = N$ partitionnements possibles et il faut apprendre N modèles différents. Le coût reste excessif pour des données massives (N très élevé).
2. Non exhaustives :
   — *k-fold* : l'ensemble $\mathcal{D}_N$ est partitionné en k parties, chaque modèle est évalué sur une des partitions et les $k-1$ autres sont employées pour son apprentissage. Sont appris k modèles, chacun évalué sur une partition différente parmi les k. Souvent $k = 10$, mais le choix de k dépend des ressources disponibles car le coût augmente linéairement avec k. La méthode *leave one out* peut être vue aussi comme un cas particulier de *k-fold*, pour $k = N$.
   — Échantillonnage répété : un échantillon aléatoire de n données est utilisé pour la validation, les autres $N-n$ données étant employées pour l'apprentissage du modèle, on répète cela k fois pour obtenir k modèles. Cette méthode permet d'éviter la contrainte $k \cdot n = N$, mais présente le désavantage de trouver certaines données dans plusieurs ensembles de validation différents alors que d'autres ne seront présentes dans aucun échantillon.

Même les méthodes non exhaustives sont coûteuses, car il est nécessaire d'apprendre (et ensuite évaluer) k modèles plutôt qu'un seul. Il faut toutefois noter que ces k apprentissages (et les k évaluations correspondantes) peuvent être réalisées en parallèle.

L'estimateur ainsi obtenu pour le risque espéré est asymptotiquement (lorsque $k = N$ tend vers l'infini) sans biais. Pour $k = N$ fini, cet estimateur présente néanmoins un léger biais, il surestime le risque espéré car l'apprentissage se fait à chaque fois sur moins de N données. Le biais sera ainsi plus faible pour *leave one out* que pour *k-fold*. La variance de l'estimateur reste relativement élevée, en partie à cause de l'utilisation de relativement peu de données pour évaluer chaque modèle. On considère que la validation croisée présente un bon compromis entre le biais et la variance de l'estimateur.

Une lecture attentive aura permis de remarquer que, pour répondre à l'objectif d'estimation du risque espéré pour *un* modèle, nous avons choisi d'en développer (et évaluer) k. Pour lequel de ces modèles est valide cette estimation du risque espéré ? En général, pour traiter un problème de modélisation, avec un même ensemble $\mathcal{D}_N$ sont employées plusieurs « procédures » de modélisation, chacune définie par une famille $\mathcal{F}$, un critère de régularisation $\text{Reg}(f(\mathbf{w}))$, une pondération pour le terme de régularisation, etc. La validation croisée sert ensuite à *choisir*, entre ces « procédures » de modélisation, celle pour laquelle le risque espéré estimé a la valeur la plus faible. La variance de l'estimateur étant élevée, ce choix n'est pas nécessairement optimal. La « procédure » choisie est alors employée pour apprendre un modèle sur la totalité des données de $\mathcal{D}_N$. C'est à ce modèle que s'applique l'estimation du risque espéré obtenue pour la « procédure » dont le modèle est le résultat. Nous reviendrons plus loin sur les questions de sélection de modèle (ou de « procédure » de modélisation).

Problématique de l'échelle en apprentissage supervisé

La complexité du processus de construction de modèle par apprentissage supervisé et la complexité de la prise de décision avec un tel modèle peuvent varier suivant la famille $\mathcal{F}$ et la méthode de modélisation considérée. La complexité fait partie des critères de choix, surtout dans le cas des données massives.

Considérons que la modélisation doit se faire à partir des N données de l'ensemble $\mathcal{D}_N$ où chaque observation est décrite par p variables explicatives. Une fois fixée la famille paramétrique $\mathcal{F}(\mathbf{w})$ de modèles, la complexité de l'algorithme d'optimisation permettant de déterminer le (vecteur de) paramètre(s) optimal $\mathbf{w}^*$ peut être :

- Pour les méthodes linéaires (régression linéaire, régression logistique, SVM linéaires), en général $O(N \times p)$ car le nombre de paramètres est de $O(p)$.
- Pour les réseaux de neurones comme les perceptrons multicouches (non linéaires), $O(N \times p^2)$ car le nombre de paramètres est de $O(p^2)$ (avec p ou $O(p)$ entrées et $O(p)$ neurones dans une couche cachée, il y a $O(p^2)$ connexions).
- Pour les machines à vecteurs de support à noyaux non linéaires en général, la résolution directe du problème d'optimisation quadratique sous contraintes d'inégalité a une complexité de $O(N^3 \times p)$, mais des méthodes de complexité $O(N^2 \times p)$ sont couramment employées. Si l'espace d'arrivée est de dimension finie d et la fonction de transposition (*feature map*) de l'espace de départ vers l'espace d'arrivée, associée au noyau choisi, peut être explicitée, il peut être préférable de faire directement la modélisation linéaire dans l'espace d'arrivée, la complexité résultante sera $O(N \times d)$.

Il faut également noter ici que certaines méthodes d'optimisation sont plus efficacement parallélisables que d'autres.

Avec le modèle $f(\mathbf{w}^*)$ obtenu, la complexité de chaque prise de décision pour une nouvelle donnée $\mathbf{x}$ peut être :

- Pour les méthodes linéaires $O(p)$, le nombre de paramètres étant de $O(p)$.
- Pour les perceptrons multicouches $O(p^2)$, le nombre de paramètres étant de $O(p^2)$.
- Pour les machines à vecteurs de support à noyaux non linéaires, $O(v \times p)$, où v est le nombre de vecteurs de support (on peut avoir $v \propto N$).

Nous examinons dans la suite les machines à vecteurs de support (SVM) et les solutions adoptées pour leur apprentissage lorsque N est élevé.

Machines à vecteurs de support

Considérons d'abord un problème de discrimination entre deux classes linéairement séparables, c'est à dire séparables parfaitement par une frontière linéaire. La figure suivante présente un exemple avec des données de $\mathbb{R}^2$. Les données de $\mathcal{D}_N$ sont les points rouges (classe 1) et bleus (classe 2). Dans cet exemple, une *infinité* de droites différentes séparent parfaitement les données de $\mathcal{D}_N$ appartenant aux deux classes. Parmi ces droites, laquelle préférer comme modèle et pourquoi ?

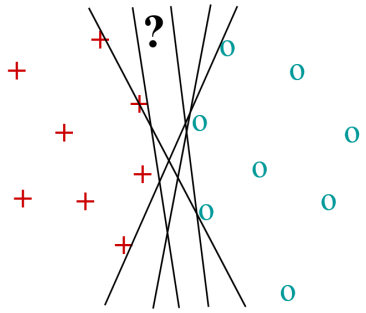


Fig. 8.4 – Quelle frontière pour séparer les classes ?

L'analyse discriminante linéaire propose une solution à ce problème, sur la base de la règle de décision bayésienne et le respect d'hypothèses de normalité et de homoscédasticité des classes (chaque classe est issue d'une loi normale multidimensionnelle et les matrices de variances-covariances sont identiques pour toutes les classes). Ces hypothèses étant rarement satisfaites, il est utile de chercher d'autres approches.

Les machines à vecteurs de support (*Support Vector Machines*, SVM, introduites par Vapnik, voir par ex. *[ScS02]* (page 132)) proposent plutôt de préférer la frontière (le modèle) qui **maximise la marge**. Par marge on entend la distance entre la frontière et les données d'apprentissage les plus proches de chaque classe. Dans l'illustration suivante, la frontière qui maximise la marge est la droite située entre deux autres droites parallèles qui marquent le « bord » de chaque classe. Il est très important de noter que la position de cette frontière dépend exclusivement (pour des classes linéairement séparables) des données de $\mathcal{D}_N$ situées sur ces « bords » de classes. Ces données sont appelées « vecteurs (de) support ». Dans l'illustration, ce sont les quatre points entourés.

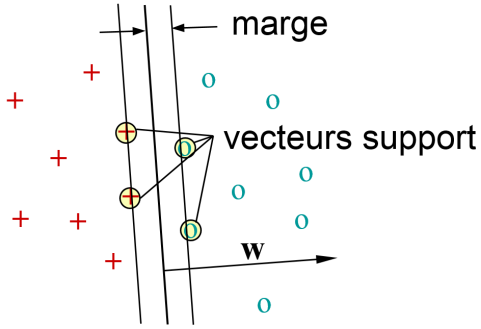


Fig. 8.5 – La frontière qui maximise la marge

En français on emploie fréquemment la traduction « séparateurs à vastes marges », qui a l'avantage de présenter les mêmes initiales SVM. Nous avons préféré ici la traduction « machines à vecteurs (de) support » car les SVM ne sont pas les seuls séparateurs à vastes marges et d'ailleurs les SVM ne sont pas uniquement des « séparateurs », vu qu'il est possible de s'en servir pour la régression.

Pourquoi chercher à maximiser la marge ? Deux justifications sont en général mises en avant :

1. Les SVM sont issus des travaux théoriques de Vapnik et Chervonenkis sur une mesure de « capacité » de $\mathcal{F}$ qui porte le nom de dimension de Vapnik-Chervonenkis (VC-dimension). On peut montrer que, pour des données vectorielles à l'intérieur d'une hypersphère de rayon r, la VC-dimension h de la famille $\mathcal{F}_m$ de hyperplans de marge $\frac{1}{\|\mathbf{w}\|} \geq m$ est bornée, $h \leq \left(\frac{r}{m}\right)^2 + 1$ (noter que cette borne ne dépend pas de la dimension des données vectorielles considérées !). Or, l'écart entre le risque empirique et le risque espéré d'un modèle $f \in \mathcal{F}_m$ est borné par $B(N, h)$ qui diminue quand h diminue. Augmenter la marge permet donc de réduire h et donc de diminuer la borne $B(N, h)$, donc d'attendre de meilleures performances de généralisation.
2. Le risque espéré peut être estimé en utilisant la validation croisée *leave one out*. Or, nous avons remarqué que pour une SVM la frontière de discrimination dépendait exclusivement des vecteurs de support. Lorsque la validation croisée *leave one out* est appliquée, un modèle est appris sur toutes les donnée sauf une et est ensuite évalué sur la donnée mise de côté. Si cette donnée n'est pas un vecteur de support, le modèle

résultant reste le même et classe bien la donnée mise de côté (erreur nulle). En revanche, si cette donnée est un vecteur de support, le modèle résultant est probablement différent et peut classer mal (ou non) la donnée mise de côté. Si v est le nombre de vecteurs support, sur les N modèles différents obtenus par validation croisée *leave one out*, au maximum v font une erreur lors de leur évaluation (sur la donnée exclue de leurs données d'apprentissage). On peut ainsi montrer que le risque espéré est borné par le rapport entre le nombre de vecteurs support et le nombre total de données d'apprentissage, $R(f) \leq \frac{E[v(N)]}{N}$, où $E[v(N)]$ est l'espérance du nombre de vecteurs de support avec N données d'apprentissage. Cette borne est surtout intéressante si $v \ll N$. Un tel résultat est rendu possible par le choix de maximiser la marge. Cet aspect est approfondi dans *[Joa99]* (page 132), où sont proposés d'autres estimateurs de l'erreur de généralisation pour les SVM.

SVM : problème d'optimisation pour une séparation linéaire

Une frontière de séparation linéaire est, dans un cas général, un hyperplan défini de façon unique par un vecteur normal $\mathbf{w}$ (pour son « orientation ») et une constante b (pour sa position exacte) : un vecteur $\mathbf{x}$ est situé sur l'hyperplan si et seulement si $\mathbf{w}^T\mathbf{x}_i + b = 0$.

Si les données de $\mathcal{D}_N$ sont linéairement séparables, il est possible de trouver des hyperplans tels que les données $(\mathbf{x}_i, y_i) \in \mathcal{D}_N$ appartenant à une des classes (par ex. $y_i = 1$) soient d'un côté de l'hyperplan ($\mathbf{w}^T\mathbf{x}_i + b > 0$) et les données appartenant à l'autre classe ((par ex. $y_i = -1$) de l'autre côté de l'hyperplan ($\mathbf{w}^T\mathbf{x}_i + b < 0$). Parmi les hyperplans qui satisfont cette condition, il faut identifier celui qui maximise la marge, caractérisé par $\mathbf{w}^*$ et b^*.

On observe que l'équation d'un hyperplan, $\mathbf{w}^T\mathbf{x}_i + b = 0$, reste satisfaite pour $\mathbf{w} \to k\mathbf{w}$, $b \to kb$, avec $k \in \mathbb{R}$. Pour que le problème de détermination de $\mathbf{w}^*$, b^* soit bien posé, il faut donc ajouter une condition de normalisation : pour tout vecteur de support $\mathbf{x}_{SV}$, on impose $\left|\mathbf{w}^T\mathbf{x}_{SV} + b\right| = 1$. Avec cette condition, on constate que la marge est $\frac{1}{\|\mathbf{w}\|}$.

La recherche de l'hyperplan qui maximise la marge revient alors à un problème d'optimisation sous contraintes d'inégalité :

$$\begin{cases} \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \\ y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1,\ 1 \leq i \leq N \end{cases} \tag{8.6}$$

La condition d'optimisation correspond à la maximisation de la marge ($= \frac{1}{\|\mathbf{w}\|}$), les inégalités indiquent que seuls les vecteurs de support (non connus au départ, identifiés par le processus d'optimisation) se situent sur les « bords » des classes, $y_i\left(\mathbf{w}^T\mathbf{x}_{SV} + b\right) = 1$, les autres données de $\mathcal{D}_N$ doivent être au-delà (plus loin de l'hyperplan recherché), $y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) > 1$.

Les résultats obtenus avec une séparation linéaire peuvent être suffisamment bons même lorsque les classes ne sont pas tout à fait linéairement séparables. Pour trouver le meilleur modèle linéaire avec des données de $\mathcal{D}_N$ qui ne sont pas linéairement séparables, il est nécessaire d'utiliser une fonction d'erreur (de perte) adaptée. La fonction *hinge loss*, $L_H(\mathbf{x}, y, f) = \max\{0, 1 - yf(\mathbf{x})\}$, $y \in \{-1; 1\}$, illustrée dans la figure suivante, est bien adaptée.

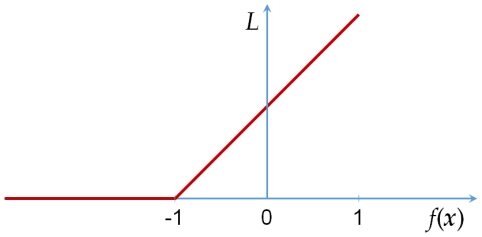


Fig. 8.6 – Fonction d'erreur *hinge loss*

L'application de cette fonction pour les données des deux classes est représentée dans la figure suivante. Tant qu'une donnée reste du bon côté du « bord » de sa classe, l'erreur est nulle ; lorsqu'une donnée est du mauvais côté du « bord » de sa classe, l'erreur augmente linéairement avec l'éloignement de ce « bord » :

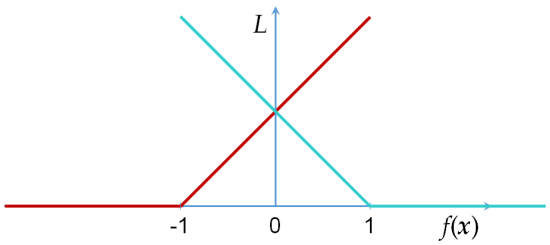


Fig. 8.7 – *Hinge loss* appliquée aux données de chaque classe

Le modèle recherché correspondra à un compromis entre la maximisation de la marge et la minimisation du nombre de données de $\mathcal{D}_N$ se trouvant du mauvais côté des « bords » des classes (et leur éloignement du bord correspondant). Il est nécessaire d'introduire un paramètre pour pondérer un de ces deux critères par rapport à l'autre. Le critère à optimiser employé sera :

$$\begin{cases} \min_{\mathbf{w},b} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \right) \\ y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) \geq 1 - \xi_i, \ \xi_i \geq 0, \ 1 \leq i \leq N \end{cases} \tag{8.7}$$

où ξ_i correspond à l'erreur faite pour la donnée i de $\mathcal{D}_N$ et C est la pondération de l'erreur totale dans la fonction à minimiser. C est considéré comme un paramètre de régularisation, car il permet de contrôler l'importance donnée à la maximisation de la marge (qui vise à reduire la capacité de la famille $\mathcal{F}$ de modèles). Il faut toutefois noter que C pondère l'erreur et non le terme de régularisation : plus la valeur de C est élevée, plus le poids de la minimisation de l'erreur d'apprentissage est élevé par rapport à celui de la maximisation de la marge (et donc plus *faible* est la régularisation).

Bien entendu, ce critère combiné peut être utilisé même lorsque les données de $\mathcal{D}_N$ *sont* linéairement séparables car cela permet d'élargir la marge, quitte à ce que certaines des données de $\mathcal{D}_N$ se trouvent du mauvais côté des « bords » des classes ou même de la frontière de séparation, comme dans la figure suivante :

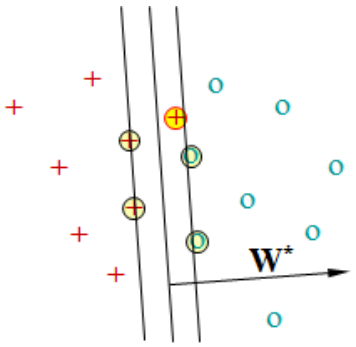


Fig. 8.8 – Donnée d'apprentissage « dans la marge »

La résolution du problème d'optimisation permet de déterminer le modèle, caractérisé par $\mathbf{w}^*$ et b^*. La fonction de décision correspondante, appliquée à une donnée (observation) $\mathbf{x}$, est :

$$f^*(\mathbf{x}) = \mathbf{w}^{*T}\mathbf{x} + b^* \tag{8.8}$$

Cette fonction de décision s'annule sur l'hyperplan qui est la frontière de discrimination. L'observation $\mathbf{x}$ est affectée à une des classes suivant le signe de la fonction de décision.

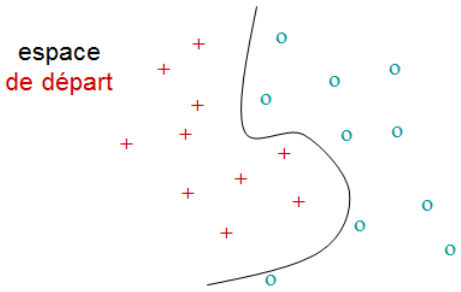


Fig. 8.9 – Frontière dans l'espace de départ

SVM : séparation non linéaire

Comment procéder si une séparation linéaire des classes est vraiment inadaptée (l'erreur obtenue est trop élevée), comme dans l'exemple de la figure suivante ?

La solution consiste à transposer les données dans un autre espace, de dimension supérieure (éventuellement infinie), dans lequel une séparation linéaire donne de bons résultats, comme dans la figure suivant :

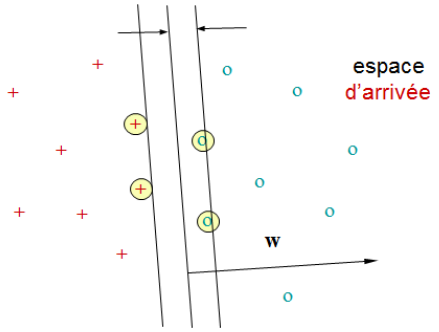


Fig. 8.10 – Frontière dans l'espace d'arrivée

A chaque donnée $\mathbf{x}$ de l'espace de départ, par ex. $\mathbb{R}^p$, la fonction de transposition (*feature map*) $\phi : \mathbb{R}^p \rightarrow \mathcal{H}$ associe une image $\phi(\mathbf{x})$ dans l'espace « d'arrivée » $\mathcal{H}$.

Définir et résoudre le problème d'optimisation (8.7), appelé problème *primal*, dans l'espace d'arrivée $\mathcal{H}$ peut être impossible (par ex. si la fonction ϕ ne peut pas être exprimée sous une forme analytique) ou peu pratique (par ex. si la dimension de l'espace d'arrivée n'est pas finie). Heureusement, si la fonction de transposition ϕ peut être associée à une fonction noyau K **valide** (nous expliciterons cette notion dans la section suivante) par la relation $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = K(\mathbf{x}_i, \mathbf{x}_j), \forall \mathbf{x}_i, \mathbf{x}_j$, où $\langle \cdot, \cdot \rangle$ est un produit scalaire dans $\mathcal{H}$, alors tous les calculs impliquant des produits scalaires dans l'espace d'arrivée $\mathcal{H}$ peuvent être exprimés par des calculs de noyau K dans l'espace de départ. Or, le problème d'optimisation (8.7) fait appel uniquement à de tels produits scalaires. On appelle cela l'« astuce du noyau » (*kernel trick*).

Le problème d'optimisation *primal* dans l'espace d'arrivée est ainsi traduit, par l'emploi de la fonction noyau, en un problème d'optimisation quadratique *dual* dans l'espace de départ :

$$\begin{array}{l} \max_\alpha \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ 0 \leq \alpha_i \leq C, 1 \leq i \leq N \\ \sum_{i=1}^N \alpha_i y_i = 0 \end{array} \tag{8.9}$$

Le modèle résultant de l'algorithme d'optimisation est défini par les coefficients $\alpha_i^*, 1 \leq i \leq N$ et b^*. Seuls les vecteurs de support ont des coefficients α_i^* non nuls.

La fonction de décision correspondante appliquée à une donnée $\mathbf{x}$ est, lorsqu'on l'écrit en utilisant le noyau :

$$f^*(\mathbf{x}) = \sum_{i=1}^{v} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b^* \tag{8.10}$$

où $\mathbf{x}_i$ sont les vecteurs de support et v leur nombre. La fonction de décision dépend donc exclusivement des vecteurs support.

SVM : fonctions noyau

Par noyau « valide » on entend un noyau qui garantit l'existence d'un espace d'arrivée $\mathcal{H}$ et de la fonction de transposition ϕ tels que $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, $\langle \cdot, \cdot \rangle$ étant le produit scalaire de $\mathcal{H}$. Pour des données vectorielles, le résultat suivant permet de définir une condition suffisante.

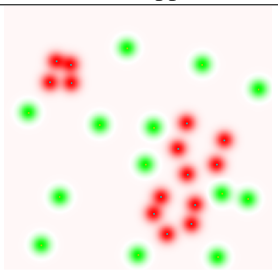
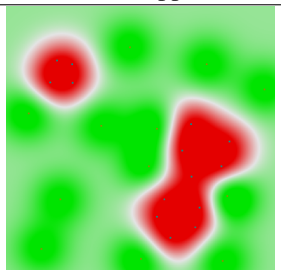
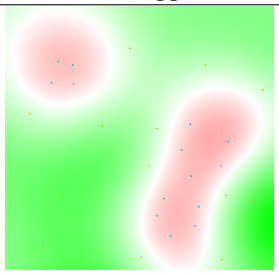
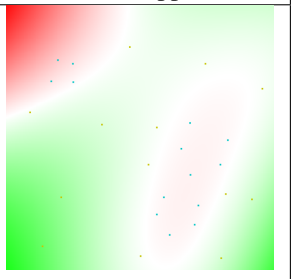
Soit $\mathcal{X}$ compact dans $\mathbb{R}^p$ et $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, symétrique, tel que $\forall f \in L_2(\mathcal{X})$, $\int_{\mathcal{X}^2} K(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$ (noyau *défini positif*[1], ou condition de Mercer). Alors il existe un espace de Hilbert $\mathcal{H}$ et $\phi : \mathcal{X} \to \mathcal{H}$ tels que $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, $\forall \mathbf{x}_i, \mathbf{x}_j$, où $\langle \cdot, \cdot \rangle$ est le produit scalaire de $\mathcal{H}$. L'existence de $\mathcal{H}$ et de ϕ étant ainsi garanties, il n'est pas nécessaire de connaître $\mathcal{H}$ ou d'exprimer ϕ pour pouvoir faire les calculs de (8.9) et (8.10) en employant K.

Différents noyaux valides sont couramment employés pour des données vectorielles : le noyau linéaire, le noyau gaussien (ou RBF), noyaux polynomiaux, le noyau angulaire (*conditionnellement* défini positif), etc. Pour certains noyaux (polynomiaux, par ex.), la fonction ϕ peut être explicitée et les calculs facilement réalisés dans l'espace d'arrivée, il peut donc être préférable de résoudre directement le problème d'optimisation *primal* dans l'espace d'arrivée.

Un intérêt majeur des SVM est la possibilité de travailler directement sur des données non vectorielles (par ex. séquences de longueur variable, arbres, graphes) en choisissant un noyau adapté au type de données. L'espace de Hilbert $\mathcal{H}$, la fonction de transposition ϕ et le noyau $K(\cdot, \cdot) = \langle \phi(\cdot, \cdot) \rangle$ sont souvent construits explicitement. Voir par exemple *[Gär03]* (page 132) pour un passage en revue de noyaux pour données structurées.

Enfin, il est possible de construire des noyaux valides en combinant d'autres noyaux valides, par exemple pour obtenir des noyaux pour données hybrides (décrites par plusieurs variables de nature différente) ; on parle de l'« ingénierie des noyaux ». Quelques détails supplémentaires sur les SVM pour la discrimination, pour la régression, pour l'estimation du support d'une distribution, ainsi que sur l'analyse factorielle à noyaux et sur l'ingénierie des noyaux peuvent être trouvés par ex. ici (http ://cedric.cnam.fr/~crucianm/src/ML7.pdf).

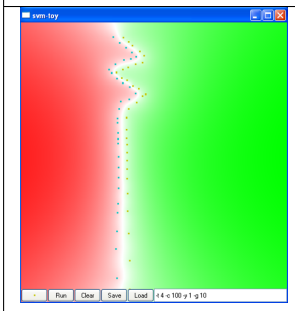
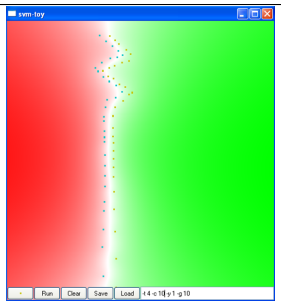
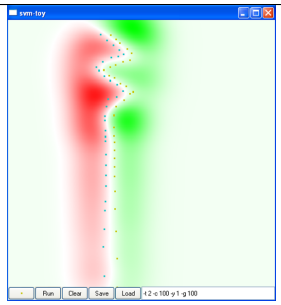
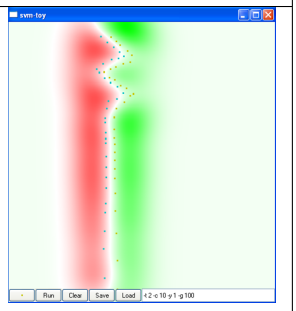
La figure suivante montre l'impact de la variance du noyau RBF sur la discrimination obtenue dans un problème à deux classes dans $\mathbb{R}^2$, ainsi que sur le nombre de vecteurs de support. L'intérieur de claque classe est représenté en rouge et respectivement en vert, la frontière de séparation est représentée en blanc (cliquez sur une image pour la voir plus grande) :

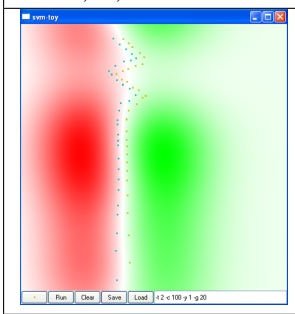
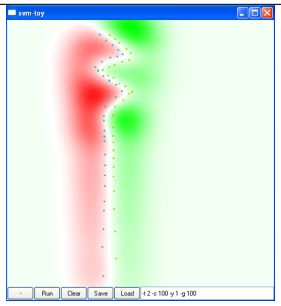
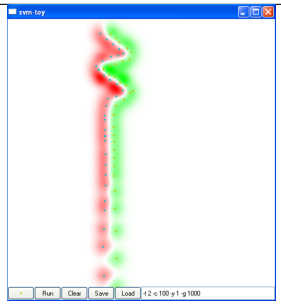
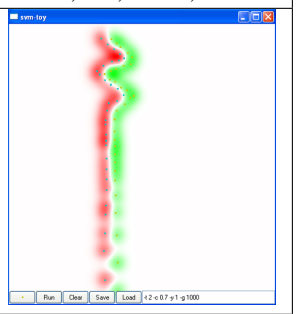
Variance 0,001	Variance 0,01	Variance 0,1	Variance 1,0
27 vecteurs support	26 vecteurs support	12 vecteurs support	10 vecteurs support
			

Par « sur-apprentissage » on entend un cas où le risque empirique (l'erreur d'apprentissage) est faible mais le risque espéré (l'erreur de généralisation) élevé. Par « sous-apprentissage » on entend un cas où le risque empirique reste élevé. On peut constater que la variance du noyau RBF a un impact très significatif sur les résultats.

La figure suivante montre l'impact du type de noyau, de la variance du noyau (pour le noyau RBF, car le noyau angulaire n'a pas de paramètres) et de la constante C sur la discrimination obtenue dans un problème à deux classes dans $\mathbb{R}^2$ (cliquez sur une image pour la voir plus grande) :

1. Une condition équivalente pour que le noyau soit défini positif est la suivante : $\forall n \in \mathbb{N}$ et $\{\mathbf{x}_i\}_n \subset \mathcal{X}$, la matrice de Gram $\mathbf{K}_n$ définie par $\{\mathbf{K}_n\}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, $1 \leq i, j \leq n$, est définie positive, c'est à dire $\forall \mathbf{c}_n \in \mathbb{R}^n$ différent du vecteur nul de $\mathbb{R}^n$, on a $\mathbf{c}_n^T \mathbf{K}_n \mathbf{c}_n > 0$. Par ailleurs, on peut montrer qu'un noyau simplement *conditionnellement* défini positif suffit pour la discrimination et la régression avec SVM. Pour qu'un noyau soit conditionnellement défini positif, la condition précédente est remplacée par : $\forall \mathbf{c}_n \in \mathbb{R}^n$ différent du vecteur nul de $\mathbb{R}^n$ et tel que $\sum_{i=1}^n c_i = 0$, on a $\mathbf{c}_n^T \mathbf{K}_n \mathbf{c}_n > 0$.

Noyau angulaire		Noyau RBF	
C = 100	C = 10	Var. 0,01, C = 100	Var. 0,01, C = 10
			

Noyau RBF		Noyau RBF	
Var. 0,05, C = 100	Var. 0,01, C = 100	Var. 0,001, C = 100	Var. 0,001, C = 0,7
			

Une valeur élevée de C accorde plus de poids à la minimisation du risque empirique et force ainsi le modèle résultant à suivre de plus près les « détails » de la séparation entre les classes sur les données d'apprentissage. On constate aussi qu'il est nécessaire de tenir compte de plusieurs « hyperparamètres » (type du noyau, variance du noyau, constante C) pour rechercher la meilleure solution. Nous y reviendrons dans la section dans la section Hyperparamètres et sélection de modèle plus loin.

SVM et apprentissage à large échelle

L'apprentissage des SVM linéaires est réalisé par la résolution du problème d'optimisation (8.7) dans l'espace de départ avec par ex. la descente de (sous-)gradient stochastique, facilement parallélisable, comme nous le verrons plus loin.

Lorsque le problème exige des SVM (à noyaux) non linéaires, plusieurs possibilités existent :

1. Si le noyau permet de déterminer une représentation vectorielle explicite et de dimension finie dans l'espace d'arrivée (c'est le cas, par ex., pour les noyaux polynomiaux), il est possible d'exprimer et de résoudre le problème d'optimisation *primal* (8.7) *dans l'espace d'arrivée*. Les algorithmes de résolution des modèles linéaires peuvent alors être employés, avec leurs avantages de complexité linéaire en N et de parallélisation facile.
2. Pour les noyaux qui ne présentent pas ces caractéristiques (par ex. le noyau RBF), il faut résoudre le problème *dual* (8.9) qui exige des solveurs quadratiques dont les opérations ne sont pas parallélisables de façon efficace. Pour passer à l'échelle, différentes pistes sont explorées(voir également *[BCDW07]* (page 132)) :
   (a) L'approximation du noyau original par un noyau permettant d'expliciter une représentation d'arrivée de dimension finie. Les résultats sont néanmoins différents de ceux qui seraient obtenus avec le noyau original.
   (b) L'emploi de méthodes de réduction de la complexité pour noyaux à décroissance rapide (par ex. RBF), comme indiqué dans le paragraphe sur l'apprentissage supervisé dans le cours sur la réduction de l'ordre de complexité. Cela est intéressant si la variance du noyau employé est faible par rapport à l'étendue du domaine couvert par les données de $\mathcal{D}_N$.

Plus généralement, pour réduire le coût de l'apprentissage supervisé (par SVM ou autre méthode) il est envisageable d'effectuer cet apprentissage en plusieurs étapes successives et, lors de chaque étape, sur un volume de données comparativement faible. Cela est possible (entre autres) par :

1. Apprentissage incrémental : le principe est de réaliser un premier apprentissage sur un échantillon de $\mathcal{D}_N$, ensuite un deuxième apprentissage sur la réunion entre un nouvel échantillon de $\mathcal{D}_N$ et les vecteurs de support issus du premier apprentissage, et ainsi de suite jusqu'à épuisement de $\mathcal{D}_N$ ou absence de changement dans l'ensemble de vecteurs de support. Ainsi, à chaque itération on apprend sur un faible volume de données, à condition que le nombre de vecteurs de support issus de l'itération précédente soit bien plus faible que la taille de l'échantillon.
2. *Apprentissage actif* (page 29) : le principe est de faire un apprentissage initial sur un échantillon de $\mathcal{D}_N$, ensuite d'ajouter progressivement, lors d'itérations successives, les données de $\mathcal{D}_N$ qui peuvent améliorer le plus le modèle courant.

Pour une machine à vecteurs de support, la complexité de la prise de décision pour une nouvelle donnée $\mathbf{x}$ est $O(v \times p)$, v étant le nombre de vecteurs de support (qui peut être $v \propto N$). Pour réduire le coût de la prise de décision lorsque v est très élevé, on peut envisager :

1. L'approximation de la solution avec nettement moins de vecteurs support, voir par ex. *[DT05]* (page 132).
2. L'application de méthodes de réduction de la complexité pour noyaux à décroissance rapide, comme indiqué dans le paragraphe sur la prise de décision dans le cours sur la réduction de l'ordre de complexité.

SVM linéaires et descente de gradient stochastique

Pour les SVM linéaires, dans le problème d'optimisation à résoudre, donné par (8.7), le terme à minimiser peut être écrit sous la forme suivante :

$$\min_{\mathbf{w},b} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} L_H(\mathbf{x}_i, y_i; \mathbf{w}, b) \right)$$

où $L_H(\mathbf{x}_i, y_i; \mathbf{w}, b) = \max\{0, 1 - y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right)\}$ est le *hinge loss*. Il est possible d'inclure le seuil b comme composante supplémentaire de $\mathbf{w}$, en ajoutant une composante supplémentaire correspondante, toujours égale à 1, à l'entrée $\mathbf{x}$. Le problème d'optimisation devient $\min_{\mathbf{w}} g(\mathbf{w})$, où

$$g(\mathbf{w}) = \left(\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} L_H(\mathbf{x}_i, y_i; \mathbf{w}) \right)$$

avec $L_H(\mathbf{x}_i, y_i; \mathbf{w}) = \max\{0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i)\}$.

Pour minimiser une fonction différentiable $l(\mathbf{w})$, une solution simple est la descente de gradient :

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\partial}{\partial \mathbf{w}} l(\mathbf{w}_t), \text{ avec } \eta > 0 \tag{8.11}$$

Notre fonction g n'est pas différentiable partout car le *hinge loss* n'est pas différentiable pour $y_i(\mathbf{w}^T \mathbf{x}_i) = 1$. Il est en revanche possible d'employer pour g la descente de *sous-gradient*, suivant la même équation que (8.11) mais où le gradient est remplacé par le sous-gradient calculé suivant :

1. Minimisation de l'inverse de la marge (terme différentiable) :

$$\frac{\partial}{\partial \mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 = \mathbf{w}$$

1. Minimisation de l'erreur d'apprentissage (terme différentiable partout sauf pour $y_i \mathbf{w}^T \mathbf{x}_i = 1$) :

$$\frac{\partial}{\partial \mathbf{w}} L_H(\mathbf{x}_i, y_i; \mathbf{w}) \leftarrow \begin{cases} -y_i \mathbf{x}_i & \text{si } y_i \mathbf{w}^T \mathbf{x}_i < 1 \\ 0 & \text{sinon} \end{cases}$$

De plus, la composante $\sum_{i=1}^{N} L_H(\mathbf{x}_i, y_i; \mathbf{w})$ étant additive par rapports aux données $(\mathbf{x}_i, y_i)$, une version stochastique de la descente de sous-gradient peut être définie : à chaque itération t un i est choisi et $\mathbf{w}_{t+1}$ est mis à jour sur la base de la seule contribution de $\frac{\partial}{\partial \mathbf{w}} L_H(\mathbf{x}_i, y_i; \mathbf{w}_t)$. On préfère parfois une variante *mini-batch*, dans laquelle un échantillon faible est choisi à chaque itération plutôt qu'une seule donnée.

L'implémentation MapReduce de la descente de (sous-)gradient stochastique *mini-batch* est facile :

`Entrées` : ensemble $\mathcal{D}_N$ de N données $(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \{-1; 1\}$, taux d'échantillonnage s, pas initial η ;

`Sorties` : Vecteur $\mathbf{w}$ et seuil b ;

1. $\mathcal{D}_N$ est découpé en fragments (partitions) distribués aux nœuds de calcul ; un fragment doit tenir dans la mémoire d'un nœud ;
2. Une solution initiale est choisie pour $\mathbf{w}_0$ et b_0 ;
3. `tant que` (nombre maximal d'itérations non atteint) `faire`
   (a) La solution courante $\mathbf{w}_t$ et b_t est transmise à tous les nœuds de calcul ;
   (b) Chaque tâche *Map* choisit un échantillon de données $(\mathbf{x}_i, y_i)$ de son fragment, en respectant le taux d'échantillonnage s ;
   (c) Chaque tâche *Map* calcule la contribution de cet échantillon à $\mathbf{w}_{t+1}$ et b_{t+1} ;
   (d) Les résultats des tâches *Map* sont transmis aux tâches *Reduce* qui calculent $\mathbf{w}_{t+1}$ et b_{t+1} en ajoutant également la contribution du terme de maximisation de la marge ;

   `fin tant que`

Dans l'étape 3.4, le partitionnement pour les tâches *Reduce* peut être fait, par exemple, par groupe de dimensions de $\mathbb{R}^p$.

Évaluation pour problèmes avec une classe d'intérêt

L'évaluation de base d'un modèle discriminant sur un ensemble de données $\mathcal{E}$ est réalisée à travers le taux de mauvais classements, qui correspond à l'erreur moyenne obtenue en utilisant la perte 0-1. Une telle évaluation est toutefois réductrice car la perte 0-1 fait l'hypothèse d'un coût de mauvais classement symétrique : le coût est égal à 1 quelle que soit l'erreur d'affectation. Or, dans de nombreux cas les erreurs d'affectation ne sont pas équivalentes. Par exemple, ne pas détecter la maladie grave d'un patient à partir de mesures cliniques est dramatique, alors que signaler une maladie grave pour un patient qui n'en est pas atteint oblige simplement à effectuer un ou plusieurs examens complémentaires. Bien entendu, si cette asymétrie peut être quantifiée alors elle peut être prise en compte dès le départ, dans la construction du modèle décisionnel. Il est toutefois utile de pouvoir comparer plusieurs modèles discriminants sans fixer *a priori* un « degré » d'asymétrie.

Les courbes ROC (initiales provenant de *receiver operating characteristic*, ou « caractéristique de fonctionnement du récepteur » en traduction littérale) répondent à cet objectif. On considère un problème de discrimination dans lequel une des classes est la « classe d'intérêt » et un modèle discriminant appris à partir des données est vu comme le « détecteur » de la classe d'intérêt.

Un modèle discriminant est en général décrit par un vecteur de paramètres (par ex. $\mathbf{w}$, qui indique l'orientation de l'hyperplan de séparation pour une SVM linéaire) et un seuil de détection (b, la position de l'hyperplan pour une SVM linéaire). Le principe d'une courbe ROC est de balayer la totalité du domaine utile de variation de ce seuil et d'examiner, pour chaque valeur du seuil, les résultats obtenus sur les données d'évaluation, représentés par la matrice de confusion :

	Classe présente	**Classe absente**
Classe détectée	Vrai Positif	**Faux Positif**
Classe non détectée	**Faux Négatif**	Vrai Négatif

Chaque donnée de $\mathcal{E}$ peut appartenir à la classe d'intérêt (**Classe présente**) ou à une autre classe (**Classe** [d'intérêt] **absente**) ; cette information est supposée connue pour toutes les données de $\mathcal{E}$ (« vérité terrain »). Chaque donnée peut être affectée par le modèle à la classe d'intérêt (*Classe détectée*) ou à une autre classe (*Classe non détectée*). Si une donnée fait partie de la classe d'intérêt mais est affectée par le modèle à une autre classe, on parle d'un « faux négatif ». Si une donnée est affectée par le modèle à la classe d'intérêt alors qu'elle fait partie d'une autre classe, on parle d'un « faux positif ».

A partir du contenu d'une matrice de confusion, on définit les mesures suivantes :

$$\text{Sensibilité} = \frac{\text{Vrais Positifs}}{\text{Total Positifs}} = \frac{\text{VP}}{\text{VP + FN}}$$
$$1 - \text{spécificité} = \frac{\text{Faux Positifs}}{\text{Total Négatifs}} = \frac{\text{FP}}{\text{VN + FP}} = 1 - \frac{\text{VN}}{\text{VN + FP}}$$

La sensibilité, ou le taux de vrais positifs, mesure la capacité du modèle à détecter les vrais positifs. Si tous les vrais positifs sont détectés (s'il n'y a pas de faux négatifs, c'est à dire des positifs non détectés comme positifs), alors la sensibilité est égale à 1.

La 1 - spécificité, ou le taux de faux positifs, mesure la capacité du modèle à détecter *seulement* les vrais positifs. Si aucun négatif n'est détecté comme positif (s'il n'y a pas de faux positifs), alors la 1 - spécificité est égale à 0.

Dans la figure suivante sont représentées des données de $\mathbb{R}^2$. Celles de la classe d'intérêt sont en vert et celles des autres classes en bleu. Considérons un modèle discriminant correspondant à une frontière de séparation verticale (le vecteur $\mathbf{w}$, orthogonal à cette frontière, est horizontal). Les points rouges correspondent en partie à des données de la classe d'intérêt situées, par rapport à une frontière verticale, du côté des données des autres classes, et en partie à des données des autres classes situées du côté de la classe d'intérêt. Dans cet exemple, la classe d'intérêt n'est jamais parfaitement séparée des autres classes, quelle que soit la position de la frontière verticale.

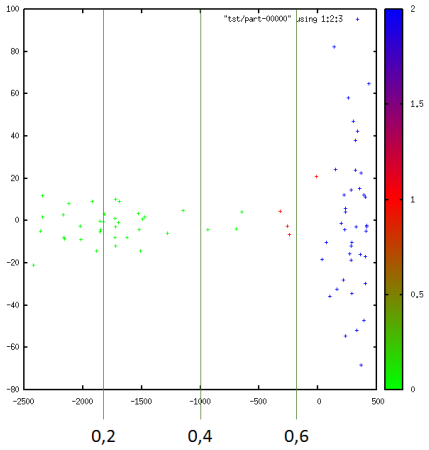


Fig. 8.11 – Positions de la frontière verticale de discrimination entre deux classes de $\mathbb{R}^2$

En explorant la totalité du domaine de variation utile du seuil (b, pour une SVM linéaire) et en enregistrant à chaque fois la sensibilité et 1 - spécificité, on obtient le graphique suivant :

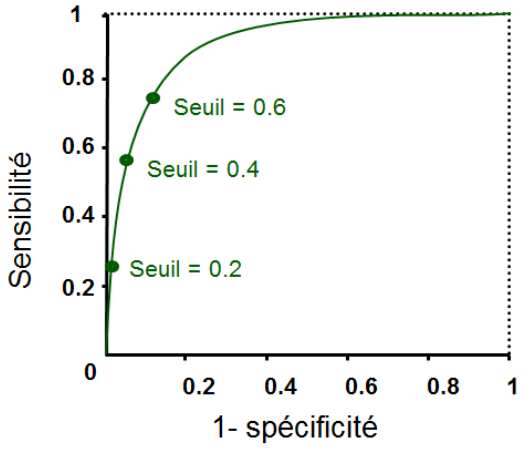


Fig. 8.12 – Courbe ROC résultante

Lorsque la frontière est très à gauche, toutes le données sont du côté « négatif ». Aucun vrai positif n'est détecté et aucun négatif n'est détecté comme positif. La sensibilité est donc nulle et la 1 - spécificité est nulle également.

Au fur et à mesure que la frontière se déplace vers la droite, de plus en plus de positifs sont détectés mais aucun négatif n'est détecté comme positif. La sensibilité augmente et la 1 - spécificité reste nulle. A partir d'une valeur du seuil, des négatifs commencent à être détectés comme positifs. La sensibilité commence donc à augmenter. Si la frontière continue son déplacement vers la droite, à partir d'une autre valeur du seuil il n'y a plus aucun positif du mauvais côté de la frontière, tous sont détectés comme des positifs. La sensibilité atteint donc 1. En revanche, comme de plus en plus de négatifs sont détectés comme positifs, la 1 - spécificité augmente.

Lorsque la frontière est suffisamment à droite, toutes le données sont du côté « positif ». Tous les positifs sont détectés et tous les négatifs sont détectés comme positifs. La sensibilité est donc égale à 1 et la 1 - spécificité égale à 1 aussi.

Il est utile de considérer deux autres modèles discriminants et d'examiner leurs courbes ROC. Un premier modèle est linéaire aussi mais correspond à des frontières inclinées de 30 °), comme dans la figure suivante :

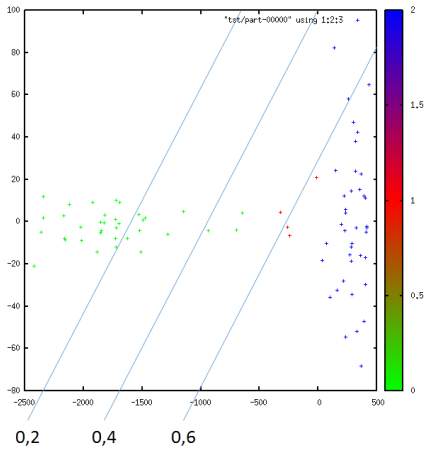


Fig. 8.13 – Positions de frontières obliques de discrimination entre deux classes de $\mathbb{R}^2$

Un second modèle est non linéaire, sur la région considérée de $\mathbb{R}^2$ les frontières correspondantes ont la forme de segment de paraboles, comme dans la figure suivante :

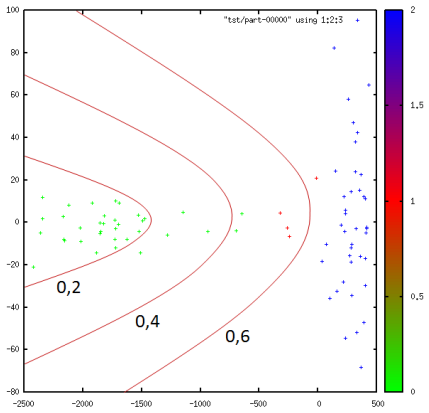


Fig. 8.14 – Positions de frontières non linéaires de discrimination entre deux classes de $\mathbb{R}^2$

Peut-on comparer ces modèles à partir de leurs courbes ROC, pour constater éventuellement qu'un est plus intéressant (ou moins intéressant) que les autres sur la totalité du domaine de variation du seuil ?

La figure suivante montre que la courbe ROC du modèle oblique se situe systématiquement en-dessous des deux autres, en revanche les courbes ROC du modèle vertical et du modèle non linéaire se croisent. Une mesure couramment employée pour comparer deux modèles sur la totalité du domaine de variation utile du seuil est l'aire sous la courbe ROC, notée en général AUC (*area under curve*). Plus sa valeur est élevée, plus le modèle est intéressant

(performant). Si deux modèles ont des valeurs AUC très proches, le choix doit être fait à partir d'autres critères, par exemple en comparant les sensibilités à spécificité fixée.

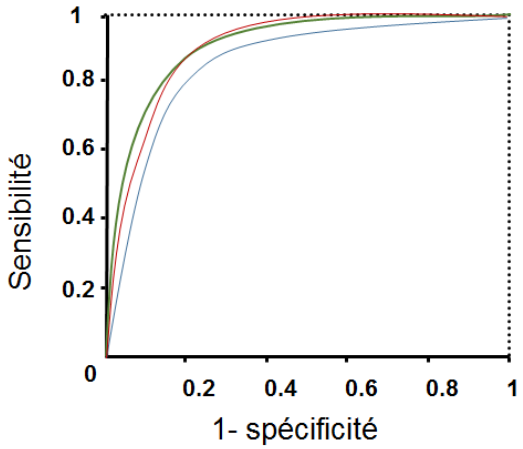


Fig. 8.15 – Comparaison de modèles utilisant les courbes ROC

Question :

Supposons que, pour les données de l'ensemble d'évaluation $\mathcal{E}$ et pour au moins une position de la frontière de discrimination (une valeur du seuil), les deux classes sont parfaitement séparées. Que pouvons-nous dire de la forme de la courbe ROC résultante ?

Hyperparamètres et sélection de modèle

Le traitement d'un problème de modélisation décisionnelle à partir d'un ensemble de données $\mathcal{D}_N$ passe en général par l'évaluation de *plusieurs* « procédures » de modélisation différentes, chacune définie par une famille de modèles $\mathcal{F}$, un critère de régularisation $\text{Reg}(f(\mathbf{w}))$, une pondération pour le terme de régularisation, etc. Ces « procédures » de modélisation sont ensuite comparées en utilisant la validation croisée afin de retenir celle pour laquelle le risque espéré (l'erreur de généralisation) estimé a la valeur la plus faible.

Les paramètres numériques dont dépend une procédure de modélisation sont en général appelés « hyperparamètres » (terme employé à l'origine pour les paramètres des distributions *a priori* dans les statistiques bayesiennes). Par exemple, sont considérés hyperparamètres :

— Pour les SVM linéaires, la valeur de C dans (8.7) (appelé aussi paramètre de régularisation).
— Pour les SVM à noyau RBF, la valeur de C dans (8.7), mais aussi la variance du noyau RBF employé. Nous avons vu plus haut, pour des exemples simples, l'impact de la valeur de C et de la variance du noyau sur les résultats obtenus.

Parfois, le terme « hyperparamètres » est étendu aux variables nominales définissant le famille de modèles $\mathcal{F}$, comme par exemple le *type* de noyau employé pour des SVM.

Grid search pour le choix des hyperparamètres

Pour trouver le meilleur modèle décisionnel il est nécessaire de comparer plusieurs « procédures » de modélisation différentes. Chaque « procédure » de modélisation est définie par une valeur spécifique pour chaque hyperparamètre. Une fois fixées les valeurs des hyperparamètres (par ex. de C pour les SVM linéaires), l'algorithme d'optimisation (par ex. la descente de sous-gradient stochastique) opère pour déterminer le modèle (par ex. $\mathbf{w}^*$ et b^* pour une SVM linéaire).

Les meilleures valeurs des hyperparamètres sont identifiées de la manière suivante :

1. Définition d'intervalles de variation pour les hyperparamètres et d'une procédure d'exploration de cet espace.
2. Exploration de l'espace des hyperparamètres suivant la procédure choisie. Pour chaque valeur considérée des hyperparamètres, évaluation des modèles résultants par application de la validation croisée.
3. Comparaison des résultats de validation croisée, sélection des valeurs des hyperparamètres qui mènent aux meilleures performances (à l'erreur de généralisation estimée la plus faible).

Une procédure d'exploration simple est la recherche dans une grille (*grid search*) : si on dispose de m hyperparamètres, on définit pour chaque hyperparamètre un intervalle et un pas de variation, on obtient ainsi une grille (de dimension m) dont les nœuds correspondent aux valeurs à tester. Pour chaque nœud de la grille, la validation croisée est appliquée afin d'obtenir une estimation de l'erreur de généralisation des modèles correspondants. Bien entendu, tous les nœuds de la grille peuvent être explorés **en parallèle**.

Avec une grille trop fine, le nombre de modèles à explorer peut être excessif. Avec une grille trop grossière, l'exploration de l'espace des paramètres peut être trop grossière et ne pas permettre de trouver les meilleurs modèles. Une solution souvent adoptée est celle d'une grille qui a plusieurs niveaux de « finesse », comme dans la figure suivante. Dans un tel cas, la recherche est hiérarchique : sont d'abord explorées les valeurs correspondant à la grille grossière, ensuite des grilles plus fines sont définies dans les intervalles de la grille grossière où les meilleurs résultats ont été trouvés, et sont explorées enfin les valeurs issues de ces grilles plus fines. Le procédé peut être répété à plusieurs niveaux. A chaque niveau, les nœuds de la grille peuvent être explorés en parallèle.

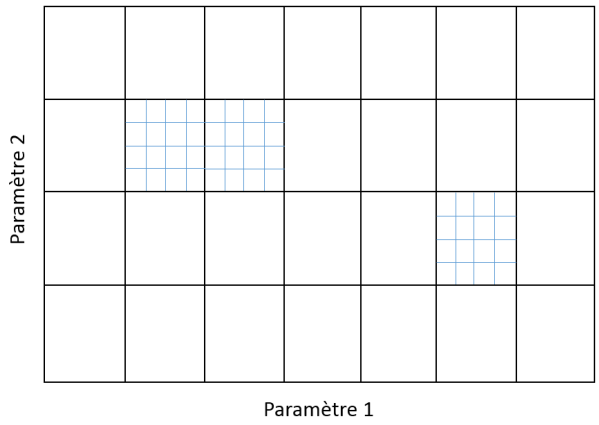


Fig. 8.16 – Grille bidimensionnelle à deux niveaux de granularité

Il est important de noter que pour des méthodes d'apprentissage spécifiques il est parfois possible de définir des procédures d'exploration plus efficaces.

Enfin, pour estimer le risque espéré (l'erreur de généralisation) du modèle retenu *in fine* il faut employer des données qui n'ont été utilisées ni pour l'apprentissage du modèle (pour l'algorithme d'optimisation des paramètres), ni pour la recherche des meilleures valeurs des hyperparamètres !

Apprentissage supervisé avec *Spark*

Spark ML est une interface de programmation (API) de niveau supérieur à MLlib, conçue pour permettre de définir et évaluer des *pipelines* (chaînes de traitement) de plusieurs outils de modélisation et apprentissage statistique. Plusieurs notions sont importantes dans ce contexte :

— *Dataset* : structure de données distribuée représentée par une `DataFrame` qui généralise la notion de `RDD` à des **relations** définies par des « colonnes » nommées (attributs) pouvant être de types différents.
— *Transformer* : méthode qui transforme une `DataFrame` en une autre `DataFrame`. Par exemple, une méthode de réduction de dimension qui modifie la représentation des observations, ou un modèle décisionnel qui fait des prédictions à partir de données d'entrée.
— *Estimator* : algorithme qui produit un *Transformer* à partir d'une `DataFrame`. Par exemple, un modèle décisionnel est obtenu par apprentissage sur les données représentées par une `DataFrame`.

— *Pipeline* : chaîne de traitement composée d'une succession de *Estimator* et *Transformer*.
— *Param* : outil de spécification de paramètres, ou de la recherche de paramètres, pour un ou plusieurs *Estimator* et *Transformer*.

Spark ML permet l'utilisation de la recherche en grille (voir `ParamGridBuilder`) et la sélection de modèle (ou de « procédure » de modélisation) par validation croisée (*k-fold*, voir `CrossValidator`).

Notes

Cours - Fouille de graphes et réseaux sociaux (1)

Diapositives du cours 1 (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/RCP216-ReseauxSociaux1.pdf)

Diapositives du cours 2 (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/RCP216-ReseauxSociaux2.pdf)

La rédaction de ce cours est récente, merci de votre indulgence et de vos retours constructifs.

Introduction

Un réseau social est un ensemble d'individus qui interagissent. L'étude scientifique de ces groupes a débuté dans les années 50, avec les travaux des anthropologues J.A. Barnes, M. Gluckman suivis par le sociologue M. Granovetter *[G73]* (page 132).

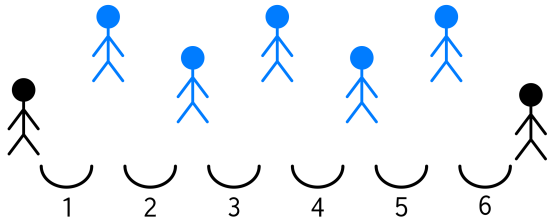
Une des premières expériences est celle de S. Milgram *[M67]* (page 132), un psychologue américain, appelée « l'expérience des six degrés de séparation » (1967). Elle vise à étudier les degrés qui séparent des individus, donc les interconnexions entre leurs réseaux sociaux. L'objectif de l'expérience : faire transiter une lettre depuis Omaha (Nebraska) jusqu'à Boston (Massachussetts). Plusieurs expériences ont été menées. Dans l'une de celles-ci, 296 lettres partent d'Omaha (remise entre les mains de volontaires), qui doivent les faire parvenir à une personne donnée, résidant à Boston. Chacune des personnes qui reçoit la lettre doit la remettre en mains propres à une personne de son choix, supposée permettre de rapprocher la lettre de l'objectif.

Fig. 9.1 – Carte des USA avec les États de départ et d'arrivée des lettres de Milgram

Les résultats furent les suivants : 64 lettres sur 296 arrivèrent, et les chemins eurent en moyenne 5,2 intermédiaires, ce qui amena l'expression « six degrés de séparation ». Plusieurs observations ont été tirées de cette expérience, par Milgram lui-même ou d'autres depuis :

— il existe des « chemins courts » entre des personnes lointaines (une lettre ne mis que 4 jours pour arriver). Ce phénomène, déjà postulé par F. Karinthy en 1929, s'appelle aussi « phénomène du petit monde » ;
— les membres du réseau parviennent à trouver ces chemins sans connaissance exhaustive de l'organisation du réseau (qui est ici de taille très grande, avec 300 millions d'habitants aux USA) : leur connaissance essentiellement *locale* leur permet de trouver des chemins *globaux* ;

— pour les lettres qui ne sont pas arrivées (232, tout de même), l'interruption de la chaîne ne veut pas nécessairement dire qu'il n'existait pas de chemin ;
— les lettres arrivées nous permettent de connaître des chemins de longueur x, ce qui ne veut pas dire qu'il n'existait pas de chemin de longueur inférieure (mais seulement que la transmission ne l'a pas empruntée à ce moment-là).



Depuis 1967, cette expérience a été reproduite plusieurs fois, avec des conditions expérimentales différentes, avec des courriers électroniques notamment, mais aussi en 2011 par les chercheurs de Facebook. Les résultats sont sensiblement les mêmes, bien que la valeur moyenne de la longueur des chemins varie un peu (4,74 dans le cas de Facebook).

Une partie importante de ces travaux de recherche a consisté à proposer des *modèles* pour formaliser et reproduire l'expérience de Milgram sur des graphes aléatoires. En particulier, cela a conduit à développer des outils mathématiques dérivant de la théorie des graphes pour pouvoir analyser les phénomènes observés. D. Watts, associé à S. Strogatz, puis J. Kleinberg ont proposé des modélisations, reposant sur l'existence de distances connues localement et d'autres globalement.

Le modèle de Kleinberg *[K00]* (page 132) présente une grille, régulière, supposée connue de tous (connaissance *globale*). Les liens sont orientés, la connaissance des uns et des autres n'est pas forcément mutuelle. On ajoute q voisins quelconques à chaque sommet, pointant vers un nœud éloigné du réseau. Ces liens représentent les *connaissances locales* du nœud (il est le seul de son voisinage à connaître ces personnes éloignées). Ces liens sont distribués aléatoirement dans le réseau, en suivant une loi de probabilité proportionnelle à l'inverse de la distance entre les points (élevée à une puissance s à paramétrer) : $p_{(u,v)} \propto \frac{1}{d^s}$.

On peut voir la grille comme une modélisation du réseau social géographiquement proche, les liens supplémentaires comme des relations éloignées. Un individu connaît quelques personnes loin de lui, et celles-ci ne sont pas connues des gens qui vivent dans la même ville que lui (exemple : sa famille éloignée, des anciens collègues).

Les auteurs de l'expérience proposent de faire circuler les messages selon un algorithme glouton (https ://fr.wikipedia.org/wiki/Algorithme_glouton), c'est-à-dire ici qu'il choisira toujours le voisin le plus proche de la cible (en fonction d'une distance sur la grille).

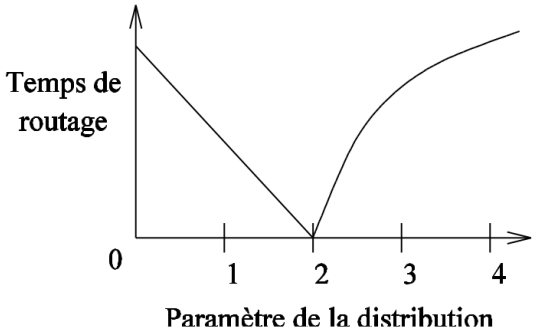
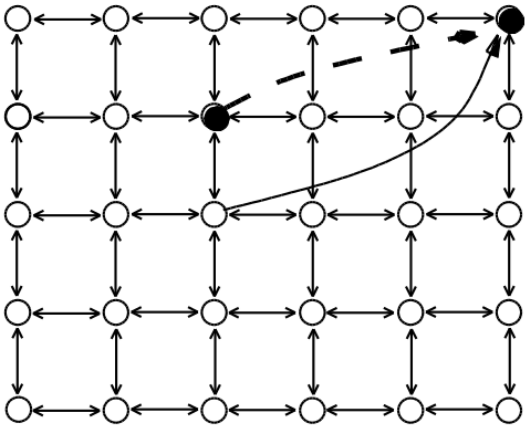


Fig. 9.2 – Courbe de l'approximation du (log du) « temps » de routage en fonction de s

Le point surprenant de l'étude est l'effet de seuil observé sur la valeur de s : avec une valeur de $s = 2$, l'acheminement du message requiert en moyenne $log(n)$ sauts. C'est la seule valeur pour laquelle cela se produit, avec des valeurs significativement supérieures dans les autres cas ($n^{(2-s)/3}$ pour $0 \leq s < 2$, $n^{(s-2)/(s-1)}$ pour $s > 2$). Voir la *Courbe de l'approximation du (log du) « temps » de routage en fonction de s* (page 106). En fait, quand s est trop petit, la probabilité de pointer sur des liens distants est forte : l'algorithme « saute » souvent à longue

distance, même quand il est proche de la destination, rallongeant la marche. Avec s grand, les liens aléatoires ne permettent pas de faire de « grand saut » vers la destination, le chemin reste long.



Ce phénomène de « petit monde » a été généralisé ensuite à des réseaux de dimension $d > 2$. Il a été aussi mis en évidence expérimentalement dans des réseaux variés, qu'il s'agisse des neurones de *C. elegans* ou de la distribution d'électricité aux USA.

Des exemples de réseaux sociaux et de graphes d'interaction

Depuis l'arrivée, au début des années 2000, de technologies ayant permis le Web 2.0 (participatif), les « réseaux sociaux en ligne » se sont multipliés. Il s'agit de l'informatisation de modélisations parfois existantes, qui a rendu visible (et grand public) la notion de « réseau social ». Les membres de ces plateformes peuvent interagir avec leur « réseau » : ajouter des amis (Facebook, LinkedIn), suivre des personnes (Twitter), créer des groupes.

Les réseaux sociaux constituent des réseaux d'interaction particuliers, entre individus. Examinons quelques exemples célèbres de tels réseaux.

Le nombre d'Erdős

Paul Erdős fut un mathématicien hongrois (1913-1996) extrêmement prolifique, qui co-signa des articles avec plus de 500 collaborateurs directs. Pour lui rendre hommage, le graphe des chercheurs en mathématiques a été étudié et chaque membre peut évoquer son « nombre d'Erdős », qui évalue la distance à Erdős dans ce graphe de collaboration. Le nombre d'Erdős est donc défini comme suit :

- 0 : Pál Erdős
- 1 : ses collaborateurs
- 2 : les collaborateurs de ses collaborateurs
- etc.

Voir l'Erdős Number project sur http://www.oakland.edu/enp/.

En pratique : il suffit de récupérer une liste d'articles et de calculer les *plus courts chemins* entre chaque chercheur et Erdős. On pourrait bien sûr calculer en prenant un autre scientifique pour référence.

Le Kevin Bacon Game

Kevin Bacon (1958–) est acteur américain et le Kevin Bacon Game (ou « Six Degrees of Kevin Bacon ») est une transposition du concept de nombre d'Erdős à l'univers d'Hollywood. En 1994, sur un forum d'Internet, des étudiants se sont demandés (alors que des bases de données n'existaient pas encore) dans combien de films il avait joué et le nombre de personnes avec lesquelles il avait joué. Rapidement, c'est devenu un jeu pour cinéphiles que d'essayer de relier des acteurs au hasard avec Kevin Bacon. Deux acteurs sont reliés s'ils ont joué dans un même film.

Il existe maintenant une plateforme en ligne où l'on peut vérifier les résultats : http://oracleofbacon.org/. On peut également y calculer les distances entre acteurs (en dehors de Kevin Bacon) :

— Distance entre Tom Cruise et Clint Eastwood ? 2 (un acteur commun entre Space Cowboys et Eyes Wide Shut)
— Distance entre Mickey Mouse et Omar Sy ? 4

Il existe quelques acteurs qui ne peuvent pas être reliés à Kevin Bacon.

Réseaux de connaissances professionnelles

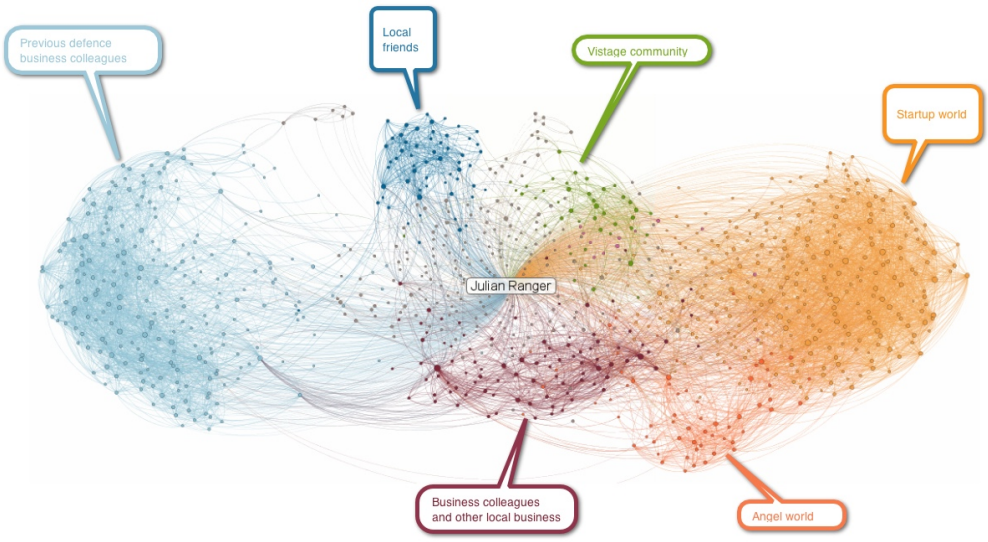


Fig. 9.3 – Réseau professionnel égocentré

Réseaux de communication

Les communications entre individus requièrent des *réseaux de communication* (réseau téléphonique, Internet, etc.). On peut collecter et analyser les échanges par le biais de ces réseaux. Internet n'est pas constitué que du Web, d'autres protocoles y existent : mail (SMTP/POP/IMAP), réseaux *peer-to-peer* (BitTorrent, Gnutella, etc.).

En 2011, Friggeri et Latapy ont par exemple observé la diffusion d'un fichier de proche en proche dans un réseau P2P (https ://vimeo.com/24528214). Comme on le verra plus tard, étudier Internet pose un certain nombre de questions (*biais de mesure*), même si cela fournit souvent des données nombreuses et extrêmement riches.

Le projet Senseable (http://senseable.mit.edu/realtimerome/) a collecté diverses informations sur le réseau téléphonique et présenté des analyses (type d'appelant, comportement de mobilité), durant la finale de la coupe du monde de football en 2006 :

Encore d'autres réseaux

D'autres réseaux présentent des caractéristiques communes et soulèvent des problématiques d'études similaires :
- informatique : pages Web, routeurs, P2P, etc.
- biologie : protéines, neurones cérébraux, etc.
- sciences sociales : amitiés, collaboration, contacts sexuels, etc.
- économie : échanges financiers
- histoire : mariages
- linguistique : synonymie, co-occurrence
- transports : réseau aérien, électrique

Éléments de théorie des graphes

En vue de permettre l'étude des graphes et réseaux sociaux, quelques rapides rappels de définitions et terminologies issues de la théorie des graphes.

Un graphe est défini par un couple $G = (V, E)$ tel que :
- V (pour l'anglais *vertices*) est un ensemble fini de sommets
- E (pour l'anglais *edges*) est un ensemble fini de arêtes

Un graphe peut être orienté, ou non :
- si oui, les couples $(v_i, v_j) \in E$ sont ordonnés, v_i est le sommet initial, v_j est le sommet terminal. on appelle alors le couple (v_i, v_j) un *arc*, représenté graphiquement par $v_i \to v_j$.
- si non, les couples ne sont pas orientés et (v_i, v_j) est équivalent à (v_j, v_i), et on l'appelle *arête*, représenté par $v_i - v_j$

Terminologie :
- l'**ordre** d'un graphe, c'est son nombre de sommets (souvent désigné par n)
- une **boucle** est un arc/une arête reliant un sommet à lui-même
- un graphe dépourvu de boucle est dit **élémentaire**
- un graphe **simple** ne comporte pas de boucle et au plus une arête entre deux sommets
- un graphe **partiel** est le graphe obtenu en supprimant certains arcs ou arêtes
- un **sous-graphe** est le graphe obtenu en supprimant certains sommets et tous les arcs/arêtes incidents aux sommets supprimés.
- un graphe est dit **complet** s'il comporte une arête (v_i, v_j) pour toute paire de sommets $(v_i, v_j) \in E^2$.
- un sommet v_i est dit **adjacent** à un autre s'il existe une arête entre eux (on parle alors de sommets **voisins**).
- le **degré** d'un sommet est le nombre d'arêtes incidentes à ce sommet (il peut y avoir un degré entrant et un degré sortant, pour les graphes orientés)

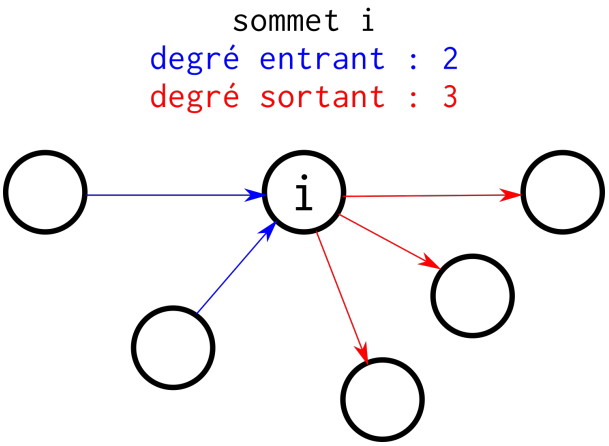
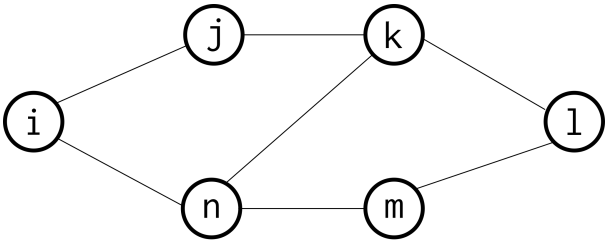


Fig. 9.4 – Degré entrant, degré sortant. Si les liens de ce graphe n'étaient pas orientés, le degré de i serait de 5.

— le **plus court chemin entre deux nœuds** est la valeur qui minimise la somme des valuations des arêtes empruntées pour parcourir le chemin entre ces nœuds. Dans un graphe non valué, il s'agit simplement du plus petit nombre d'arêtes à emprunter pour passer d'un nœud à l'autre. L'algorithme de Djikstra permet de calculer le plus court chemin entre deux nœuds avec une complexité polynomiale (non détaillé ici, voir par exemple l'article Wikipedia consacré (https ://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra)).



i-j-k-l et i-n-m-l sont les plus courts chemins de i à l (de longueur 3)

Fig. 9.5 – Entre les noeuds i et l, il y a deux plus courts chemins : i-j-k-l et i-n-m-l. Ils ont pour longueur 3.

— le **diamètre d'un graphe** (ou excentricité maximale) est la valeur maximale des distances entre les nœuds. Pour l'obtenir, on calcule les plus courts chemins entre chaque paire de nœuds et on garde le maximum. L'excentricité minimale, ou rayon du graphe, est la plus petite distance à laquelle puisse se trouver un sommet de tous les autres.

Étudier de tels réseaux

L'objectif est de comprendre le comportement des entités qui interagissent dans le système étudié, les lois qui les gouvernent. On cherche donc :

— quelle est la structure des graphes ?
— quelle est l'évolution de cette structure ?
— quels sont les phénomènes reposant sur l'existence de ce réseau ?

Les applications sont multiples. En informatique/télécommunications, comme la conception et la mise en place de réseaux est un enjeu crucial, on étudie les réseaux existants en vue de les améliorer (en amendant les protocoles existants ou en élaborant de nouveaux), que ce soit pour des questions de performances ou de sécurité. En sciences humaines, l'étude de réseaux d'interaction permet de comprendre la diffusion d'innovations (économie), de rumeurs (sociologie). En médecine, les réseaux sont fondamentaux pour comprendre les interactions entre protéines mais aussi la propagation de virus au sein des populations (et les effets des campagnes de vaccinations, par exemple).

On propose dans ce cours une méthode d'étude générale, qui est ensuite affinée par les praticiens des différentes disciplines en fonctions de leurs besoins. Les outils sont souvent les mêmes :

— Théorie des graphes
— Analyse statistique
— Modélisation probabiliste

On procède parfois à de la simulation sur des données synthétiques que l'on compare à des données réelles. Les problématiques rencontrées dans l'étude des réseaux sociaux peuvent être classées en quatre catégories : Mesure, Modélisation, Analyse et Algorithmique (il faudra souvent faire des allers-retours entre ces domaines, par exemple en améliorant le modèle après des analyses préliminaires peu satisfaisantes, voir refaire une expérience de captation de données).

Mesure

La mesure, ou métrologie, désigne ici la capture des données. Les réseaux sociaux que l'on cherche à étudier sont généralement issus d'une expérience de mesure d'un système donné, vu comme un graphe. Celle-ci est particulièrement déterminante : une mauvaise capture peut rendre complètement fausses les analyses subséquentes. Il faut

donc, dès l'obtention des données, se poser la question des "biais d'observation", en ayant des réponses précises aux questions suivantes :

— quelle proportion du système a été mesurée ?
— combien de temps la mesure a-t-elle duré ?
— quelles étaient les contraintes techniques ?
— qui a fait la mesure ?
— la mesure peut-elle être reproduite ?

Compte tenu de la taille des systèmes concernés (Facebook et Twitter comptent plusieurs centaines de millions d'utilisateurs), de leur caractère dynamique (les réseaux P2P ont des utilisateurs qui ne restent dans le système que quelques heures au maximum), ces problématiques jouent fortement sur la qualité des représentations qui seront obtenues. En effet, un graphe qui ne modéliserait que 1000 utilisateurs de Facebook, par exemple, ne pourrait évidemment prétendre décrire le comportement global des membres de cette plateforme. De même, une mesure d'une heure sur un réseau P2P très populaire ne mènera qu'à des analyses limitées. Enfin, la connaissance du protocole de mesure est importante pour estimer si des informations observables ne l'ont pas été.

Par exemple, l'utilisation de l'outil *traceroute*, souvent employé pour établir des « cartes physiques » de l'Internet, présente un certain nombre de caractéristiques qu'il faut maîtriser. Cet outil enregistre les chemins qu'empruntent des paquets de données pour atteindre une adresse IP spécifique, à partir d'une source. Ces deux paramètres, une adresse IP de départ et une d'arrivée, sont des contraintes fortes sur la mesure, et on doit en tenir compte : pour cartographier une zone de l'Internet (par exemple, l'environnement d'une machine donnée), il est donc nécessaire de multiplier les points "source" à partir desquels on tentera d'atteindre cette machine. Il faut noter que certaines zones seront inaccessibles (par exemple des routeurs de degré 1, sauf s'ils sont sources ou destinations), et que la répartition de charge (*load-balancing*) peut amener des routeurs en milieu de chemin à faire passer les paquets par une branche d'une fourche plutôt qu'une autre (cf les figures *Nœud accessible par un seul chemin.* (page 111) et *Équilibrage de charge sur une fourche.* (page 111)). Ces deux phénomènes conduisent à ne pas observer certains nœuds et certains liens du réseau, ce qui peut changer considérablement les caractéristiques du réseau mesuré par rapport au réseau original (cf la figure *Distribution de degré observée et distribution réelle* (page 112), où la distribution de degré observée n'est pas la même que celle qu'on attendrait : il y a des contraintes techniques qui ont un impact significatif sur la mesure.).

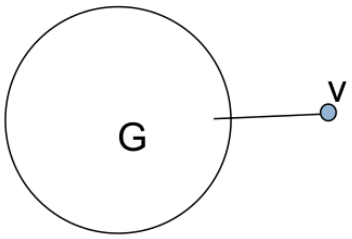


Fig. 9.6 – Nœud accessible par un seul chemin.

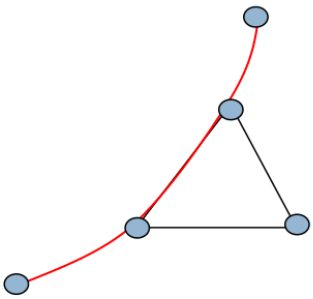


Fig. 9.7 – Équilibrage de charge sur une fourche.

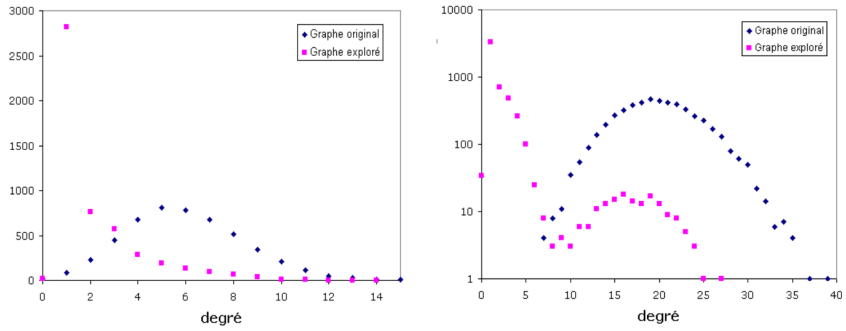


Fig. 9.8 – Distribution de degré observée et distribution réelle

Analyse

Une fois que les données sont collectées, il faut procéder à l'analyse de celles-ci. L'objectif est d'obtenir des informations pertinentes sur le système, à l'aide d'une description statistique que l'on interprète ensuite. L'analyse des graphes peut passer d'abord par la détermination des valeurs de certaines propriétés "classiques" (distribution de degrés, longueurs des chemins, etc.), des propriétés spécifiques pertinentes pour le graphe en question (coefficient de *clustering*, existence et taille de communautés, etc.). L'analyse peut aussi requérir une phase préalable de modélisation (cf *Modélisation* (page 113)), permettant de comparer le graphe étudié avec un ou des graphes aléatoires "similaires" (ou supposés tels). Enfin, un aspect important mais sur lequel on s'étend peu dans ce cours, c'est l'évolution dynamique des graphes : quels sont les changements des valeurs des propriétés au cours du temps.

Quelques propriétés classiques

Les propriétés classiques que l'on peut observer sur des graphes proviennent de la théorie des graphes.

- la "longueur moyenne des (plus courts) chemins" : on la définit comme la moyenne, pour toutes les paires de sommets du graphe, des plus courts chemins. Formellement, pour un graphe $G(V, E)$ où $d(v_1, v_2)$ désigne le plus court chemin entre v_1 et v_2 ($v_1, v_2 \in V$), la longueur moyenne l_G vaut : $l_G = \frac{1}{n(n-1)} \sum_{i \neq j} d(v_1, v_2)$.
- le coefficient de *clustering*. Il existe une version globale, parfois appelée "transitivité", et une version locale, provenant du travail de Watts et Strogatz *[WS98]* (page 133).
  La version globale est définie comme : $C = \frac{3 \times \text{nb triangles}}{\text{nb triplets connectés}}$. Un triplet connecté étant soit un triangle (triplet fermé), soit un triangle privé d'une arête (triplet ouvert).
  La version locale mesure, pour un sommet donné, à quel point son voisinage se rapproche d'une clique (voir def-clique). Notons $N(i)$ le voisinage du sommet i : $N(i) = \{v_j : e_{ij} \in E\}$ (ensemble des sommets connectés à i par au moins une arête). On définit le coefficient de *clustering* comme étant le rapport entre le nombre d'arêtes existant dans le voisinage et le nombre de celles qu'il pourrait y avoir s'il s'agissait d'une clique. Pour un sommet avec k_i voisins, il pourrait y avoir $\frac{k_i(k_i-1)}{2}$ arêtes. Donc, la valeur du coefficient de *clustering* local est :
  $c(i) = 2 \times \frac{|e_{xy} \in E / x, y \in N(i)|}{k_i(k_i-1)}$
- la distribution de degrés $P(k)$. Pour un graphe donné, il s'agit de la fraction de sommets du graphe de degré k. S'il y a n sommets, et que n_k d'entre eux ont pour degré k, $P(k) = \frac{n_k}{n}$. Attention, il y a parfois une confusion de faite avec la distribution *cumulative* des degrés, désignant la fraction de sommets qui ont un degré *au moins égal à* k.
- les composantes connexes. C'est un ensemble maximal de sommets tel qu'il existe un chemin entre toute paire de sommets dans l'ensemble. Certains graphes sont entièrement connexes. D'autres non, auquel cas il est intéressant de mesurer la taille et le nombre des composantes connexes
- les communautés. Ce sont des ensembles de sommets très liés entre eux et peu liés vers le reste du graphe. Nous étudierons en détail dans la seconde partie du cours quelles communautés on peut rechercher dans un graphe et les principales manières de le faire.
- la centralité. Pour chaque sommet du graphe, on peut définir plusieurs mesure de centralité, qui estime à quel point il est "central" dans le graphe :
  - centralité de degré, égale au degré.

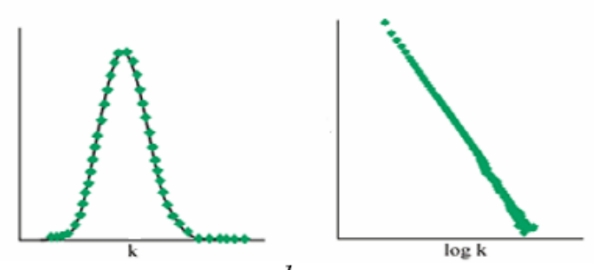


Fig. 9.9 – Distribution gaussienne (à gauche), en loi de puissance à droite.

— centralité d'intermédiarité (betweenness centrality) : nombre de plus courts chemins passant par un sommet : $C(v) = \sum_{s \neq t \neq v \in V} \frac{s_{st}(v)}{s_{st}}$
— centralité de proximité : inverse de la distance à tous les autres sommets $C(x) = \frac{1}{\sum_y d(y,x)}$

Les réseaux réels ont souvent des propriétés assez différentes des graphes aléatoires que l'on peut générer (cf section suivante) :

— faible densité
— fort *clustering*
— faible distance moyenne
— distribution de degré très hétérogène
— composante connexe géante
— présence de communautés

Modélisation

La modélisation, dans notre contexte, consiste à proposer des modèles de graphes afin de comparer les comparer au graphe étudié et d'estimer si les propriétés observées étaient attendues, ou non. On utilisera souvent des graphes "aléatoires", c'est-à-dire générés par tirage aléatoire en respectant éventuellement une loi (contraignant la distribution de degrés, la densité, etc.).

On peut, dans un premier temps, utiliser un modèle uniforme, en se donnant n sommets et en décidant que chaque arête a une probabilité $0 < p < 1$ d'exister (modèle de Gilbert, noté $G(n, p)$). On observe alors en général une composante connexe géante, un diamètre proche de $log(n)$, un *clustering* très faible, une distribution de degrés homogène, pas de structure communautaire (cf liste précédente sur les réseaux réels).

Il existe de nombreux autres modèles pour générer des graphes et permettre d'affiner l'analyse.

Le modèle de **l'attachement préférentiel**, qui permet de générer des réseaux "sans échelle" (*scale-free*) a été proposé par Barabási et Albert *[AB02]* (page 132). On débute la création du graphe aléatoire avec m_0 sommets, puis on ajoute successivement chacun des nœuds. Ceux-ci sont connectés à m sommets, avec une probabilité qui dépend du degré de chacun des nœuds. Formellement, la probabilité p_i que le nouveau nœud soit connecté au nœud i est : $p_i = \frac{k_i}{\sum_j k_j}$. Cette règle de génération fait que les nœuds qui ont déjà des connexions ont tendance à en recevoir toujours davantage, ce qui crée des "hubs", et une distribution de degrés qui est asymptotiquement une "loi de puissance".

Le **modèle configurationnel** permet d'obtenir une distribution de degrés fixée : on prend n sommets, on fixe le degré de chaque sommet, on ajoute des liens au hasard en respectant les degrés.

Parmi ces modèles, celui de l'attachement préférentiel donne un *clustering* significativement supérieur (les autres étant très faibles).

Quelques applications de la modélisation

La modélisation, et les simulations qu'elle autorise, permettent d'étudier de nombreux phénomènes reposant sur des graphes, et de comprendre ce qui se passe sur des réseaux réels.

Citons en particulier :

— les phénomènes de diffusion épidémique, que ce soit pour des virus informatique ou pour de la médecine. Voir les modèles SI / SIR pour plus de détails.

— les phénomènes d'atteintes aux réseaux, que ce soit des pannes aléatoires ou des attaques ciblées

On examine en particulier **pourquoi** et **où** peuvent se produire des ruptures de connexité dans le graphe.

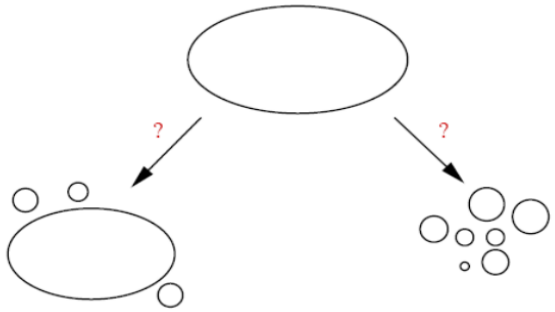
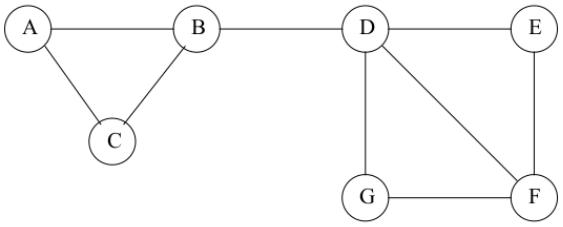


Fig. 9.10 – Ruptures de connexité liées à une atteinte au système. Il peut résulter un certaine stabilité (grande composante connexe, peu d'îlôts créés, à gauche) ou un morcellement du réseau (à droite)

Algorithmique

La représentation informatique des graphes peut se faire sous différentes formes. On peut utiliser des *listes d'adjacence*, c'est-à-dire des listes chaînées ou des tableaux de longueurs variables, ce qui présente l'intérêt d'accéder avec un coût faible aux successeurs d'un nœud dans un graphe orienté (mais rend plus complexe la recherche de ses prédécesseurs).

Une représentation privilégiée reste bien sûr la matrice d'adjacence A, dont les coefficients binaires valent 0 quand il n'y a pas d'arête entre le nœud i et le nœud j, 1 si oui.



$$
\begin{array}{c} \\ \mathrm{A} \\ \mathrm{B} \end{array}
\begin{array}{c}
\begin{array}{ccccccc} \mathrm{A} & \mathrm{B} & & & & & \end{array} \\
\left[\begin{array}{ccccccc}
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}\right]
\end{array}
$$

Attention cependant, la taille de la matrice est n^2, ce qui peut la rendre énorme pour des graphes de taille importante. Elle est néanmoins souvent creuse et les *frameworks* récents disposent d'optimisations intéressantes pour stocker et traiter de telles matrices (mais cela peut ne pas être suffisant).

On définit également la matrice des degrés D, diagonale, comportant le degré du nœud i sur la i-ème ligne. Enfin, on verra dans le prochain cours l'utilisation de la matrice *laplacienne*, définie comme $L = D - A$.

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Difficultés algorithmiques

Les graphes et réseaux sociaux étudiés sont régulièrement de grande taille et posent des difficultés, il n'est pas toujours possible de stocker l'entièreté du graphe ou d'effectuer des calculs dessus (qui aboutissent en un temps raisonnable). Compter les triangles du graphes, par exemple, se fait naïvement avec une complexité $O(n^3)$, mais des algorithmes efficaces ont été construits pour pouvoir le faire avec une complexité $O(m * n^{1/a})$ dans les cas où la distribution de degrés est en loi de puissance d'exposant a. De même, calculer le diamètre d'un graphe a une complexité théorique de $O(nm)$, mais on peut avoir une bonne estimation en $O(m)$. Beaucoup de problèmes (et de calculs) sont NP-complets et nécessitent des heuristiques appropriées.

Deux algorithmes classiques

Pour parcourir un graphe, on dispose de deux possibilités, le parcours en largeur et le parcours en profondeur. On les utilise souvent au sein d'autres algorithmes. On les illustre ici sur le graphe suivant :

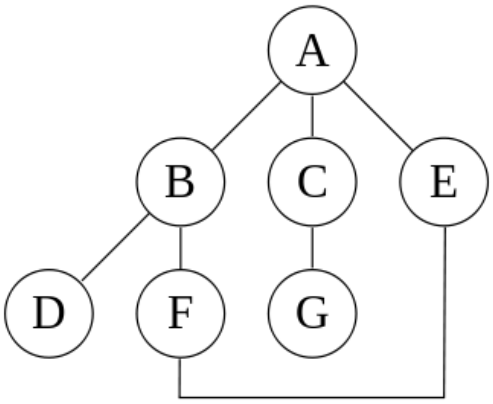


Fig. 9.11 – Graphe pour présenter les parcours en profondeur et en largeur

Pour le parcours en profondeur (DFS, ou *Depth-first search*) :
- à partir d'un sommet S, appelé racine, on appelle récursivement cet algorithme pour tous les voisins de S
- pour chaque sommet i, on prend le premier sommet voisin et on explore tous les sommets (non marqués) avant de revenir à i
- sur notre graphe, l'ordre de visite est donc : A, B, D, F, E, C, G.
- l'implémentation de l'algorithme repose sur une pile.

Pour le parcours en largeur (BFS, ou *Breadth-first search*) :
- pour chaque sommet, on repère tous ses voisins, on stocke tous ceux qui ne sont pas marqués dans une file.
- sur notre graphe, l'ordre de visite est donc : A, B, C, E, D, F, G.
- cet algorithme permet de calculer des distances entre nœuds (cf deuxième partie)

Logiciels et outils

Avec l'explosion de la taille moyenne des graphes étudiés et les avancées significatives dans leur traitement, de nombreux *frameworks* et bibliothèques dédiés sont apparus aux cours des dix dernières années.

Le projet GraphLab, né à Carnegie Mellon University en 2009, repose sur des routines en C++ et dispose de très bonnes performances. Le calcul est distribué et il est disponible sous licence Apache (open source). Nous utilisons en travaux pratiques GraphX, qui en l'implémentation dans Spark.

Google, dont le cœur de métier repose sur le calcul du PageRank sur le graphe du Web, a développé de son côté Pregel *[MABDHLC10]* (page 132). Les calculs reposent sur des passages de messages entre nœuds. Plusieurs implémentations existent au-dessus d'Hadoop.

D'autres frameworks plus ou moins stables existent, Giraph, FlockDB, AllegroGraph, etc. Une base de données « orientée graphes » a beaucoup de succès en ce moment, Neo4j. Elle est open source.

Quelques logiciels dédiés sont également disponibles, dont Gephi (http ://gephi.org), commencé en projet étudiant à l'Université Technologique de Compiègne et open source. Il permet l'analyse de graphes de tailles raisonnables, intègre de nombreux algorithmes et traitements classiques (PageRank, modularité, plus courts chemins, etc.), le tout en permettant un ajustement fin de la visualisation définitive.

Références

Ce cours repose sur les travaux de l'équipe ComplexNetworks (http ://www.complexnetworks.fr) du LIP6 (http ://www.lip6.fr) au sein duquel l'auteur a effectué son travail de doctorat. En particulier, les cours de Jean-Loup Guillaume (http ://jlguillaume.free.fr/www/) ont été une source d'inspiration précieuse, de même que le livre *Mining massive datasets [MMDS]* (page 133) et le matériel associé.

CHAPITRE 10

Cours - Fouille de graphes et réseaux sociaux (2)

Diapositives du cours 1 (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/RCP216-ReseauxSociaux1.pdf)

Diapositives du cours 2 (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/RCP216-ReseauxSociaux2.pdf)

Dans cette deuxième partie, nous nous intéressons plus spécifiquement à la recherche de communautés dans des graphes. Par *communauté*, on entend ici *sous-ensemble de nœuds qui sont fortement connectés*, c'est-à-dire qu'il existe entre eux un nombre important de connexions, davantage que ce que ces nœuds entretiennent avec les autres membres du réseau. Contrairement à des réseaux générés aléatoirement, les réseaux sociaux (et d'autres graphes) présentent une propriété de *localité* : les nœuds tendent à s'interconnecter en groupes, plus ou moins denses. Cela peut résulter de phénomènes liées aux structures humaines (des élèves regroupés dans une classe, des sportifs dans une équipe), ou de la transitivité (on présente nos amis les uns aux autres, certains vont ensuite devenir amis, alors qu'il n'y aurait pas eu de lien entre eux sans nous).

Les motivations de l'analyse des communautés sont similaires à celle de l'analyse des graphes en général. Comme on l'a vu au chapitre précédent, on cherche à comprendre comment fonctionne le système que l'on a modélisé par un graphe. Quels sont les échanges (d'information, d'argent, etc.) entre les membres ? Pourquoi M. X et Mme Y ont des chances de devenir amis, mais peu de rencontrer un jour Mme Z ? On peut également se servir des communautés pour élaborer des systèmes de recommandations. Mme AA, M. BB et Mme CC ont l'air d'avoir des centres d'intérêts en commun, si je recommande à Mme CC les films qu'apprécient AA et BB, ça devrait aussi lui plaire.

Il existe différents types de communautés. On peut dans un premier temps rechercher un partitionnement du graphe (c'est-à-dire que chaque nœud appartient à une et une seule communauté). On peut alors utiliser des techniques de *clustering* classiques, ou des algorithmes dédiés aux graphes, comme on le verra dans ce chapitre. Mais, dans un réseau social, les communautés sont souvent recouvrantes : un individu est relié à divers groupes d'amis ou de collègues dont les membres se connaissent peu. La figure *Modules* (page 118) présente un réseau partitionné, avec des communautés non recouvrantes, alors que la figure *Réseau social égocentré* (page 118) montre les communautés auxquelles appartient une personne (appelée "ego u"). Voir aussi le *Réseaux de connaissances professionnelles* (page 108) du premier chapitre.

Note : Nous n'aborderons pas dans ce chapitre l'aspect important mais délicat (car toujours sujet à de nombreuses recherches) du *suivi temporel des communautés*. Après avoir repéré des communautés sur un graphe à un instant *t*, il est souvent difficile de savoir (algorithmiquement) si les communautés détectées à *t+1* sont le résultat de fusion/scission de communautés précédentes. Nous nous contentons donc ici d'oberserver des graphes "figés".

Des sous-graphes intéressants

La théorie des graphes a mis en évidence de nombreuses manières de définir des sous-graphes présentant une cohésion forte, c'est-à-dire des sous-ensemble de nœuds ou d'arêtes particuliers :

- une composante *k-connexe* est un ensemble de sommets tels qu'il existe au moins k chemins disjoints entre chaque paire de sommet. On parle simplement de *composante connexe* quand $k = 1$.
- une clique est un sous-graphe complètement connecté : chaque sommet est connecté à tous les autres.

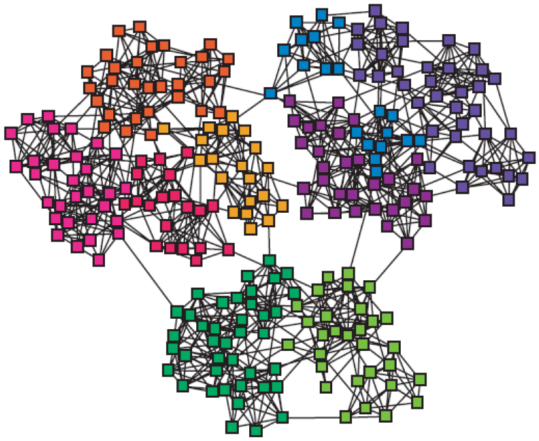
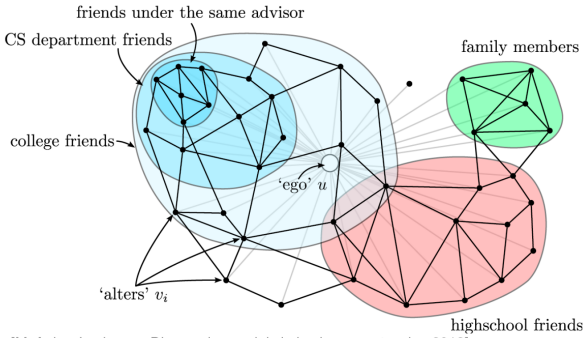
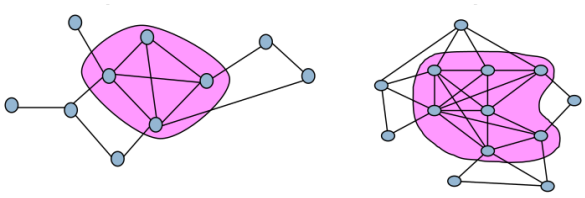


Fig. 10.1 – Modules

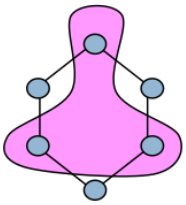
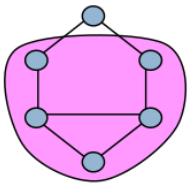


[McAuley, Leskovec: Discovering social circles in ego networks, 2012]

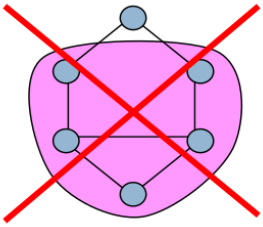
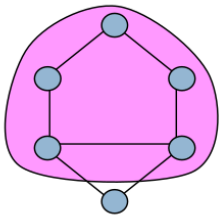
Fig. 10.2 – Réseau social égocentré



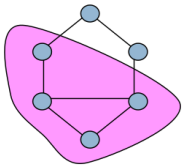
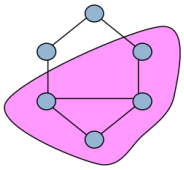
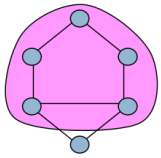
— une n-clique est un sous-graphe G' tel que la distance entre les nœuds de G' soit inférieure à n dans G. L'ensemble G' peut ne pas être connecté ou avoir un diamètre supérieur à n.

— un n-clan est une n-clique de diamètre n au plus.

— Un n-club est un sous-graphe de diamètre maximal n.

On peut lister encore de nombreux autres sous-graphes cohésifs :

— k-degree set : sous-graphe dans lequel tous les sommets ont degré au moins k.
— k-outdegree set : sous-graphe tel qu'aucun sommet n'a plus de k liens sortants.
— k-plex : sous-graphe maximal H tel que chaque sommet est connecté à au moins $|H| - k$ sommets
— LS set : sous-graphe minimal pour le nombre de liens sortants.
— alpha set : sous-graphe tel que chaque sommet a plus de liens internes que sortants.

La recherche de ces zones sur un graphe donné est en général un problème coûteux, en terme de complexité. Dans de nombreux cas, il s'agit de problèmes NP-complet (cliques, k-plex). Parfois, c'est polynomial (LS set). Cela rend bien sûr le passage à l'échelle très délicat. Il en est de même pour la recherche de communautés recouvrantes, souvent coûteuse. Nous allons donc voir d'autres techniques pour trouver des "communautés" (ou groupes, *clusters*, modules).

Clustering hiérarchique

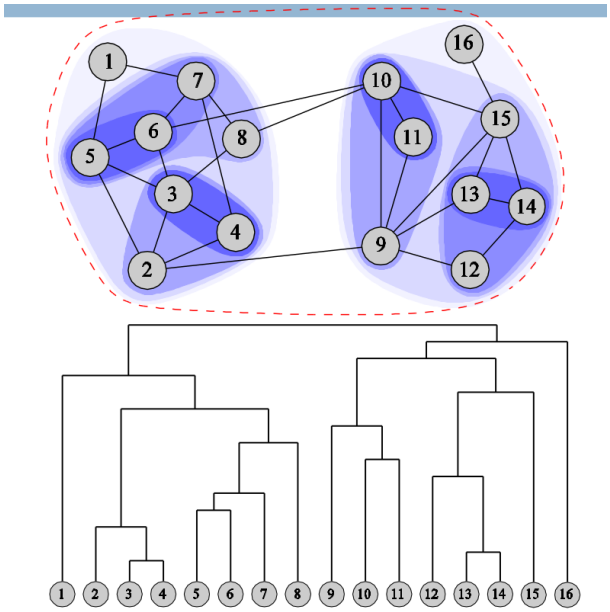
Il existe une manière ascendante et une manière descendante de procéder au regroupement hiérarchique (ou *clustering* hiérarchique).

Approche ascendante

Voici un algorithme générique pour une classification ascendante :

— Chaque sommet est dans une communauté.
— Calculer une distance entre chaque paire de communautés.
— Fusionner les deux plus proches.
— Revenir à 2.

On obtient un dendogramme et un découpage du graphe en communautés, comme le présente la figure Fig. 10.2.1. La distance entre les communautés qui est choisie peut varier, on peut opter pour le minimum, le maximum ou la moyenne des distances entre paires de nœuds (que l'on calcule généralement avant).



Approche divisive

Il existe aussi une approche divisive, proposée par Girvan et Newman en 2002, qui est assez efficace. Elle repose sur l'idée suivante : si un **lien** se trouve fréquemment sur les plus courts chemins entre les nœuds du graphe, alors il est naturel de penser qu'il ne se trouve pas "au sein d'une communauté donnée", mais plutôt qu'il relie des portions distantes du graphe, des communautés distinctes. En retirant progressivement le lien qui a la plus forte "centralité", on obtient un découpage des blocs de notre réseau.

À titre d'analogie, on peut penser au réseau de transport français : la liaison de TGV Paris-Lyon se trouve sur les plus courts chemins entre de nombreuses villes, par exemple Rennes-Marseille, Rennes-Lyon, Rouen-Lyon, Lille-Valence. En revanche, la liaison Rouen-Rennes n'est un plus court chemin qu'entre les villes normandes et bretonnes. Si l'on retire la liaison Paris-Lyon (et quelques autres liaisons Nord-Sud importantes), on va obtenir un partitionnement Nord-Sud du réseau.

Voici le détail de l'algorithme :

— on calcule la centralité (*betweenness*) de chaque lien (c'est une extension de la notion de centralité pour les nœuds vue dans le chapitre précédent)
— on enlève le lien de plus forte centralité
— on recommence jusqu'à ce qu'il n'y ait plus de lien
— les composantes connexes restantes sont les communautés
— on obtient une décomposition hiérarchique du réseau

La figure Fig. 10.4 présente les différentes étapes de l'algorithme, appliquées au graphe de la figure Fig. 10.3. La figure Fig. 10.5 montre le résultat du *clustering* sur le graphe du Karaté-club étudié par Zachary.

Calcul de la centralité

L'algorithme de Girvan-Newman repose sur un calcul de centralité, présenté dans les diapos du cours. Vous pouvez également consulter le livre Mining Massive Datasets *[MMDS]* (page 133), pages 333 et suivantes.

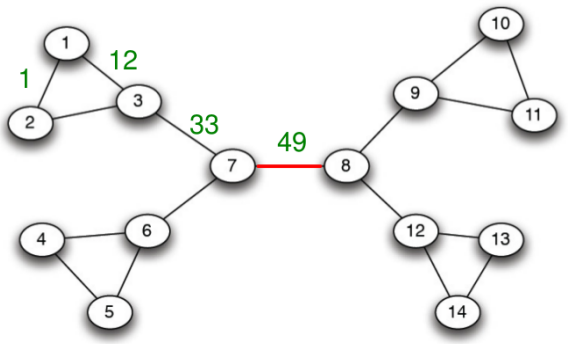


Fig. 10.3 – Graphe pour la présentation de l'algorithme de Girvan-Newman

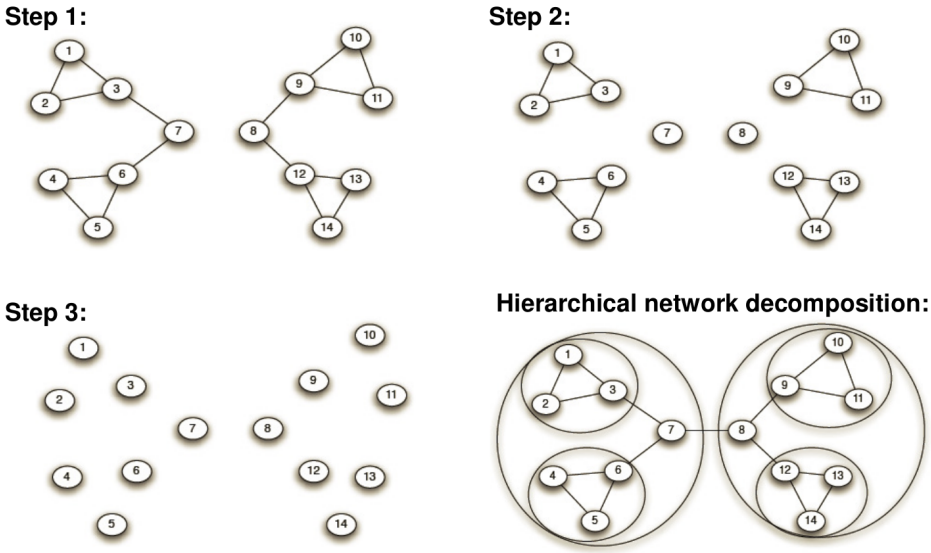


Fig. 10.4 – Étapes de l'algorithme de Girvan-Newman

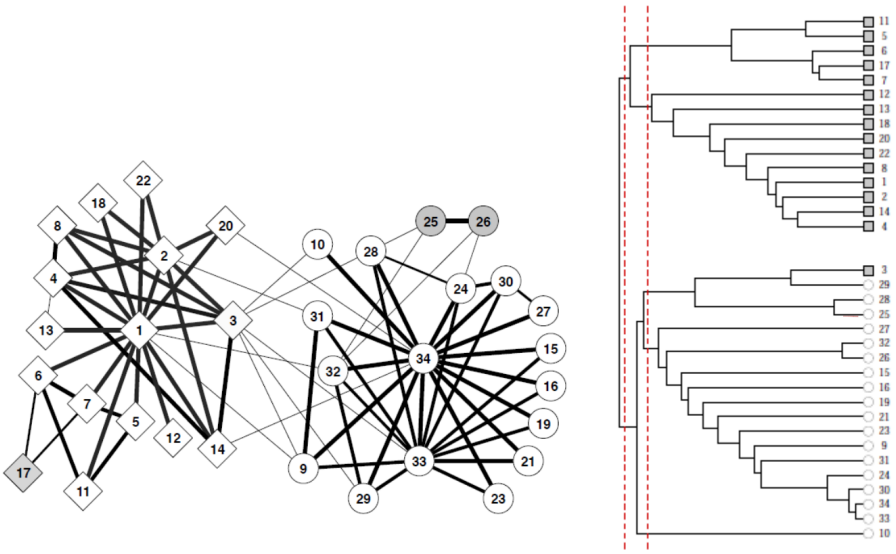


Fig. 10.5 – Application de l'algorithme de Girvan-Newman au graphe du Karaté Club de Zachary

Évaluer la structure communautaire : la modularité

On a défini une communauté comme un sous-graphe dont les sommets sont plus liés entre eux qu'avec le reste du réseau. C'est une notion pour l'instant non formalisée mathématiquement. Il peut donc être intéressant de se donner une métrique pour évaluer la qualité du partitionnement d'un graphe en communauté.

La **modularité**, également introduite par Newman, permet de caractériser cela. Elle est définie comme la différence entre le nombre de liens présents dans un *module* (ou groupe, ou *cluster*, ou communauté), et le nombre de liens attendus dans un graphe aléatoire. La valeur de la modularité est dans l'intervalle [-1 ;1]. On calcule une valeur de modularité *pour un partitionnement donné* du graphe, on note traditionnellement la valeur de la modularité par la lettre *Q*.

Détaillons un peu le calcul. On dispose d'un partitionnement de notre graphe en communautés, notées *s*. On note *S* l'ensemble des communautés. On souhaite que

$$Q \propto \sum_{s \in S} (\text{nb liens dans s} - \text{nb liens attendu dans s}).$$

La fraction du nombre de liens du graphes qui sont "internes" à une communauté dans le graphe est obtenue par la formule :

$$\frac{1}{2m} \sum_{s \in S} \sum_{i \in S} \sum_{j \in S} A_{ij}$$

où $A_{ij} = 1$ si le lien ij existe, 0 sinon.

Il nous faut donc un modèle aléatoire auquel comparer le partitionnement considéré. On se réfère pour cela au modèle configurationnel, qui est une manière de générer un graphe aléatoire à partir d'un graphe existant. Ce modèle conserve la distribution de degrés existante. On fait de chaque lien un "demi lien" partant de chaque nœud et on "recâble" les demi-liens du graphe aléatoirement. La probabilité d'avoir un lien entre les nœuds *v* et *w* est $\frac{k_v k_w}{2m}$, si k_v et k_w sont les degrés de *v* et *w* et *m* le nombre de liens total dans le graphe.

En insérant ce terme dans la formule précédente, on obtient la valeur de la modularité :

$$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in S} \sum_{j \in S} [A_{ij} - \frac{k_i k_j}{2m}]$$

La modularité est positive si le nombre de nœuds dans les modules est supérieur à ce que l'on pourrait atteindre. Elle tend vers 0 si l'on prend un partitionnement aléatoire d'un graphe aléatoire. Au-dessus d'une valeur de 0.3 environ, on peut considérer que l'on a une structure communautaire significative.

En utilisant la valeur de la modularité pour différents partitionnements, on peut décider quel partitionnement est souhaitable pour un graphe donné, comme le montre la figure *Sélection d'un partitionnement avec la modularité* (page 123). On a un graphe partitionné en 4 grandes communautés, elles aussi subdivisées. Mais l'on constate que le meilleur partitionnement, au sens de la modularité, est obtenu lorsque l'on s'arrête à cette division du graphe en 4 groupes.

La complexité de l'algorithme de Girvan-Newman est $O(m^2 n)$. Nous allons voir dans la section suivante une manière différente et de complexité plus faible pour optimiser la modularité.

Optimiser la modularité : l'algorithme de Louvain

En 2008, des chercheurs français et belges de l'Université de Louvain ont proposé un nouvel algorithme, cherchant à maximiser, à chaque étape, la modularité *[BGLL08]* (page 133).

L'algorithme repose sur deux phases distinctes, qui sont répétées itérativement. Au départ, tous les nœuds du graphe sont indépendants : il y a autant de communautés que de nœuds dans le graphe.

Ensuite, pour chaque nœud i du graphe on considère chacun de ses voisins j et l'on évalue le gain de modularité que l'on ferait en retirant i de sa communauté pour le mettre dans celle de j.

Le nœud i est ensuite placé dans la communauté pour laquelle le gain de modularité est maximal, à condition que ce gain soit positif (sinon, i reste dans sa communauté).

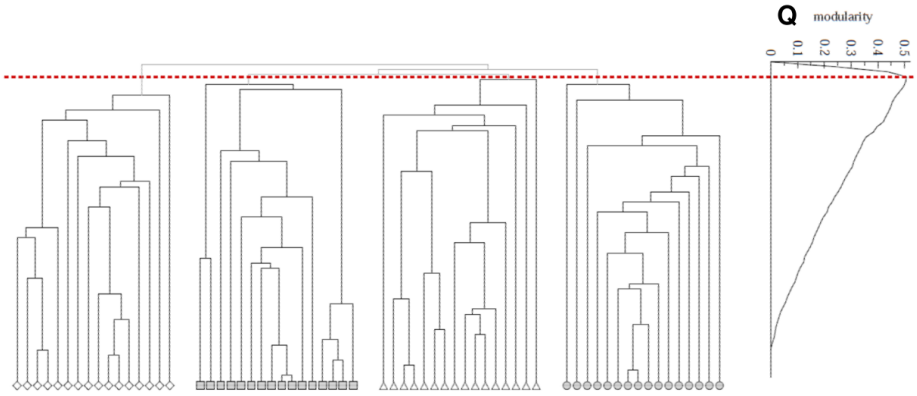
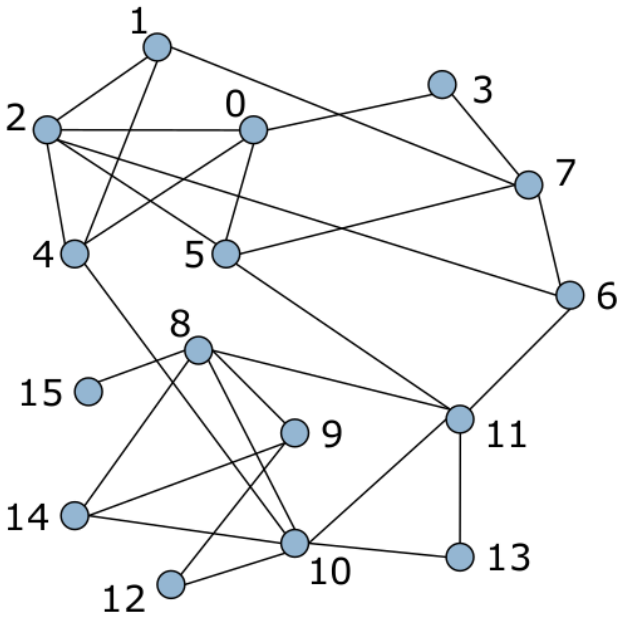
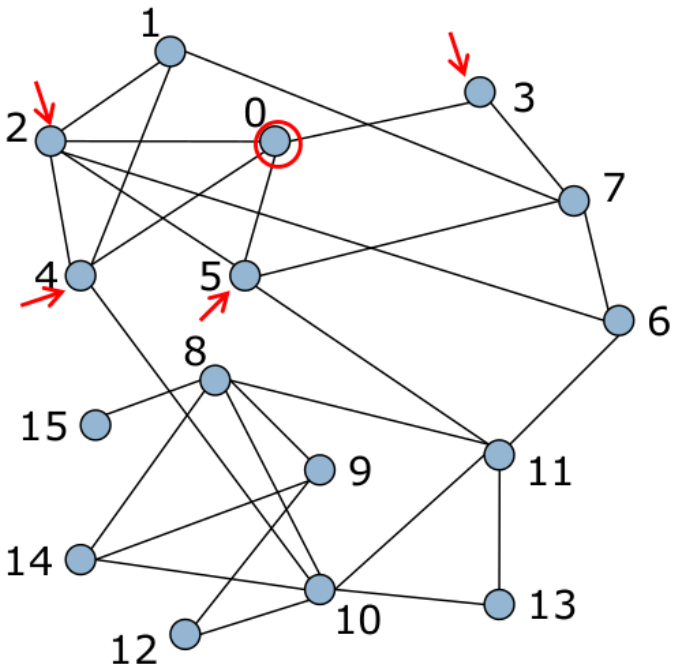
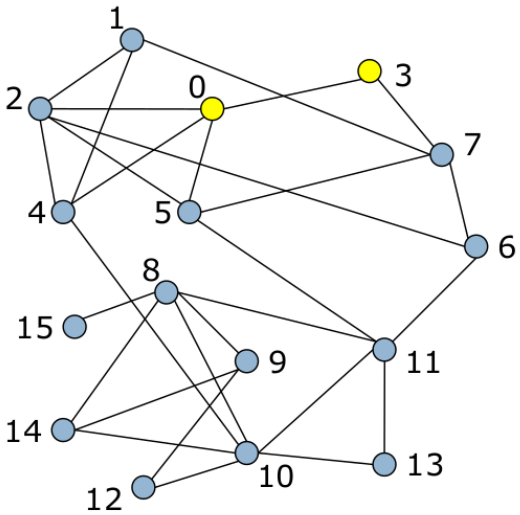


Fig. 10.6 – Sélection d'un partitionnement avec la modularité

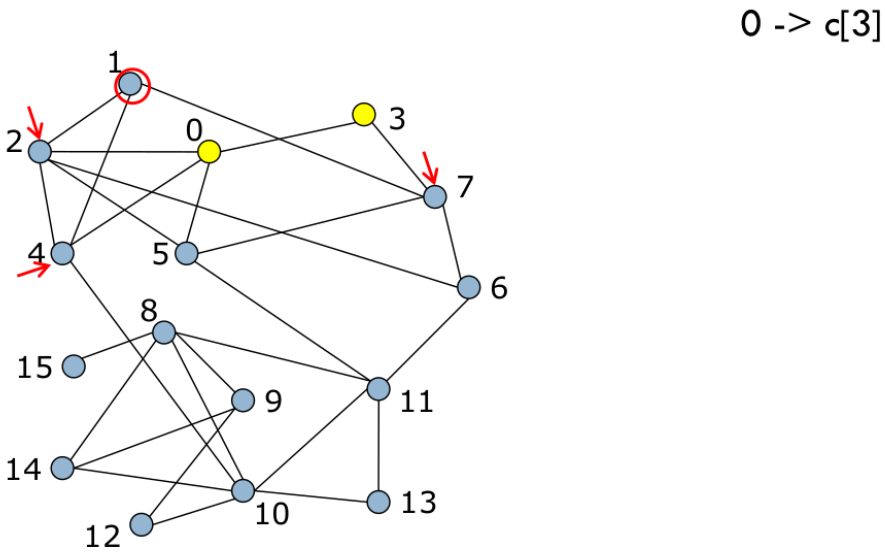


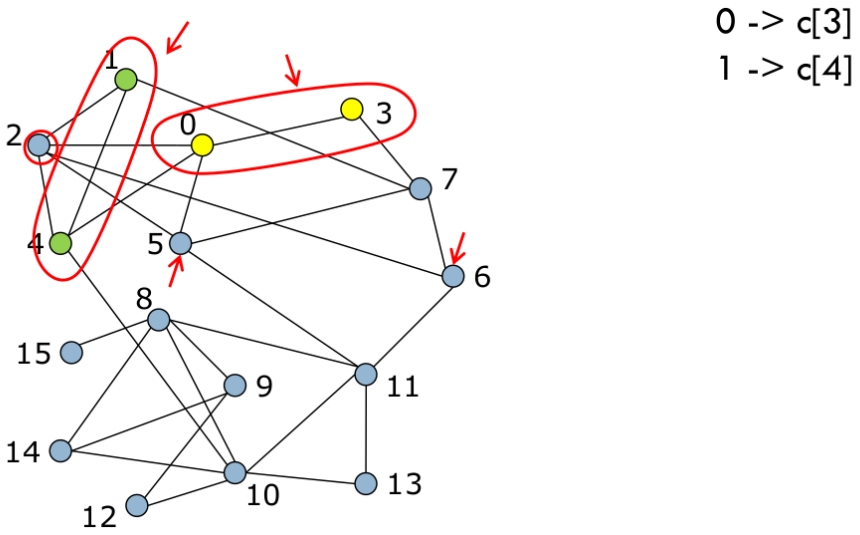


0 -> c[3]



Ce processus est appliqué au nœud suivant (figure `LouvainChoix` et Fig. 10.3), et ainsi de suite pour tous les nœuds (figure Fig. 10.7).





Cette séquence est ensuite répétée (figure Fig. 10.8), jusqu'à ce qu'il n'y ait plus d'amélioration. C'est la fin de la première phase, on a atteint un maximum local de la modularité.

La deuxième phase de l'algorithme consiste à construire un nouveau graphe dont les nœuds sont les communautés trouvées dans la première phase. Les liens entre ces "super-nœuds" sont "valués", avec un poids correspondant au nombre de liens entre les nœuds de chacune des communautés qu'ils représentent. Les liens internes de chaque communauté forment des boucles (voir figure Fig. 10.3, flèche rouge).

Une fois que cette deuxième phase est terminée, cela clôt une "passe" de l'algorithme. Ensuite à la lieu la passe suivante, qui commence donc par la première phase : une détection communautaire sur le nouveau graphe (celui en haut à droite sur la figure Fig. 10.3). On trouve des communautés (fin de phase 1), qui sont ensuite transformées en nœuds (fin de phase 2). Les passes sont répétées, jusqu'à atteindre un maximum de modularité.

Par construction, le nombre de communautés diminue donc à chaque passe, la première passe est donc la plus gourmande en temps de calcul. On obtient une décomposition hiérarchique des communautés du graphe. Au niveau supérieur, de très "grosses" communautés regroupant des communautés des niveaux inférieurs. Le nombre total de niveaux dans la hiérarchie est déterminé par le nombre de passes, qui est généralement faible (moins de 10).

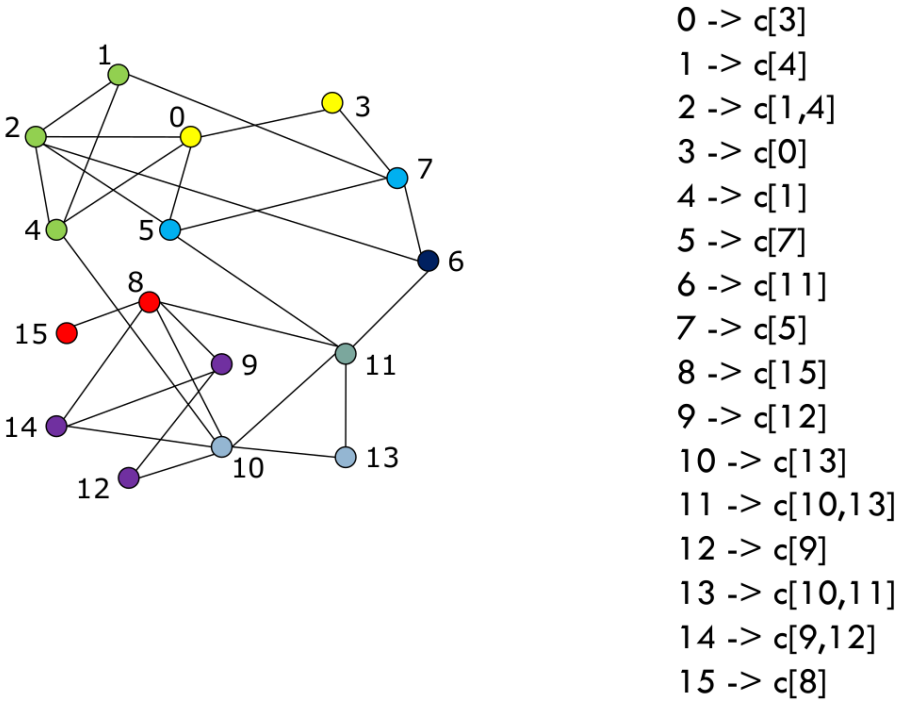


Fig. 10.7 – On répète pour tous les nœuds

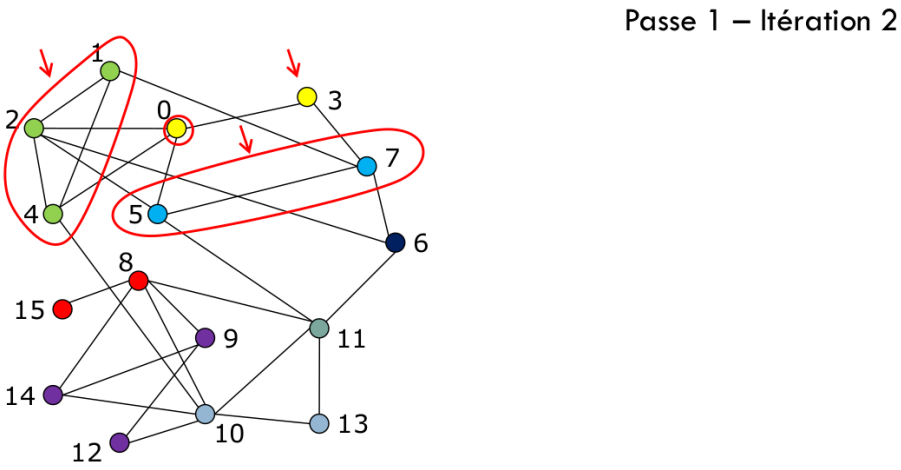
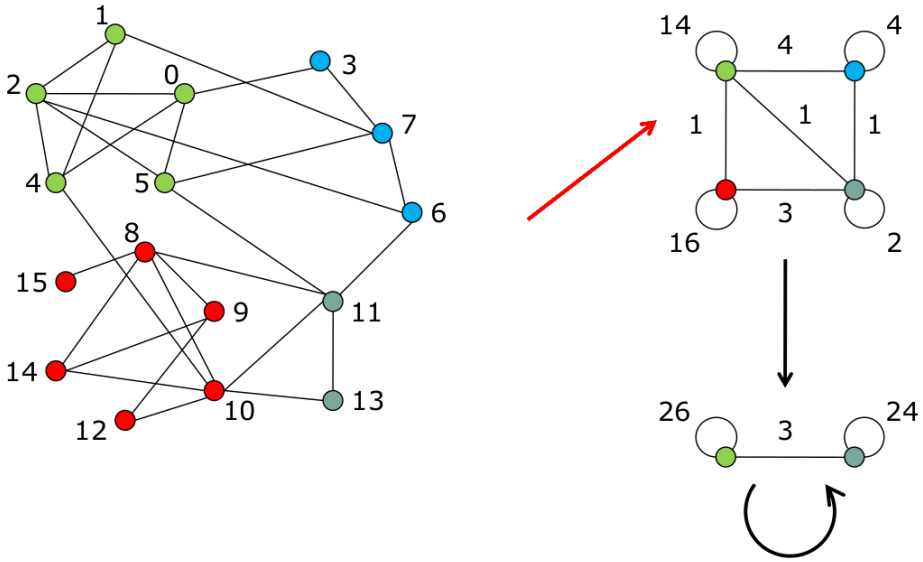


Fig. 10.8 – On itère, en réessayant d'optimiser localement la modularité avec les communautés formées à l'itération précédente



L'algorithme est simple, d'implémentation facile. Son avantage principal, bien sûr, reste son efficacité : comme il nécessite le stockage en mémoire de peu d'informations et que l'essentiel du calcul se fait dans les premières passes, il ne permet de traiter sur des machines traditionnelles des graphes qui auparavant devaient être traitées sur des serveurs de puissance significativement plus importante. On peut améliorer l'algorithme en remarquant que les dernières passes apportent un gain de modularité faible, on peut décider de s'arrêter quand le gain entre une passe et la suivante est inférieur à un certain seuil.

Deux aspects importants à noter. Au début de la première phase, les nœuds sont numérotés aléatoirement, et c'est sur cet ordre que repose le test qui suit de "prendre chaque nœud et tester son déplacement dans une communauté voisine". Ainsi, si l'on exécute deux fois l'algorithme *sur les mêmes données*, le résultat pourra être sensiblement différent. On dit que l'algorithme n'est pas *déterministe*.

D'autre part, il existe un phénomène dit de "limite de résolution" de la modularité, que l'on peut observer sur un exemple. Si l'on considère un graphe formé d'un "anneau de cliques", comme sur la figure Fig. 10.9, on s'attend à ce que la partition de meilleure modularité soit celle où chaque communauté est une clique.

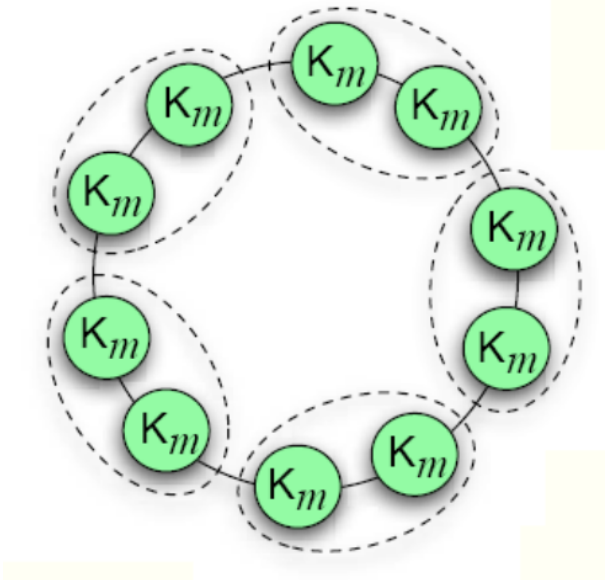


Fig. 10.9 – Anneau de cliques : un cercle vert est une clique de m sommets. Une arête relie un des m sommets de la clique à un autre sommet de la clique voisine.

Pourtant, effectuons le calcul. Il y a n cliques sont de taille m, donc la modularité de la partition en "cliques" donne :

$$Q_{single} = 1 - \frac{2}{m(m-1)+2} - \frac{1}{n}$$
$$Q_{pairs} = 1 - \frac{2}{m(m-1)+2} - \frac{2}{n}$$

Ainsi :

$$Q_{single} > Q_{pairs} \iff m(m-1)+2 > n$$

De sorte que, pour $m = 5$ et $n = 30$, on obtient une modularité plus élevée pour la partition qui regroupe les cliques deux à deux ($Q = 0.888$) que pour la partition où chaque clique est seule dans sa communauté ($Q = 0.876$). Cela permet de caractériser cette "limite de résolution de la modularité" : dans un même graphe avec des communautés de tailles différentes, il est possible que certaines d'entre elles ne soient pas détectées. À noter : c'est un problème qui est lié à l'emploi de la modularité comme métrique, pas à l'algorithme de Louvain lui-même.

Un exemple d'utilisation de l'algorithme de Louvain en Python est disponible dans ce script Python (http ://cedric.cnam.fr/vertigo/Cours/RCP216/docs/louvain-example.py). Le logiciel Gephi (http ://gephi.org) incorpore aussi cet algorithme.

Autres approches

Partitionner un graphe peut aussi se faire avec d'autres approches. Une technique classique consiste à tenter de minimiser ce qu'on appelle le *cut*, c'est-à-dire le nombre de connexions intergroupes coupées par notre partitionnement. On peut définir le *cut* entre deux ensembles A et B de sommets comme "le nombre de liens qui ont exactement un sommet dans chacun des groupes". On le note $cut(A, B)$.

Voyons sur un exemple, avec la figure Fig. 10.10. Si l'on découpe ce graphe en deux sous-ensembles A et B, le cut est de 2.

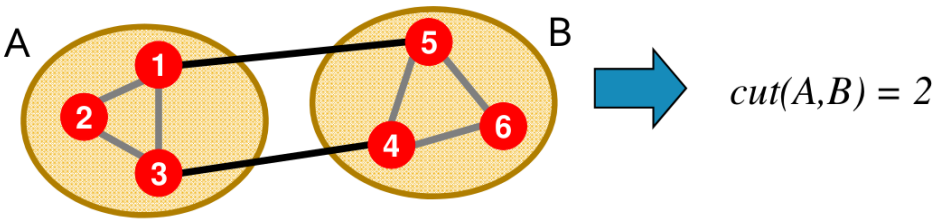


Fig. 10.10 – Calcul du cut

Comme on peut le voir avec la figure Fig. 10.11, minimiser le cut peut cependant poser des problèmes : le minimum est atteint en coupant en deux groupes qui sont de tailles très déséquilibrées, un groupe ne contenant qu'un seul sommet.

On utilise donc généralement une mesure normalisée, le *normalized cut*. Si l'on définit le nombre de liens qui ont une extrémité dans A comme le volume de A (noté $vol(A)$), le *normalized cut* entre deux groupes vaut :

$$ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)}$$

On obtient des partitions de taille plus équilibrées. Une démonstration mathématique un peu longue pour figurer ici (mais tout à fait accessible via l'article *[SM00]* (page 133), ou *[MMDS]* (page 133) pages 344 et suivantes) montre que l'on peut trouver cette valeur minimale de *normalized cut* en recherchant la seconde valeur propre la plus faible de la matrice laplacienne du graphe (la plus faible étant toujours 0) et le vecteur propre associé.

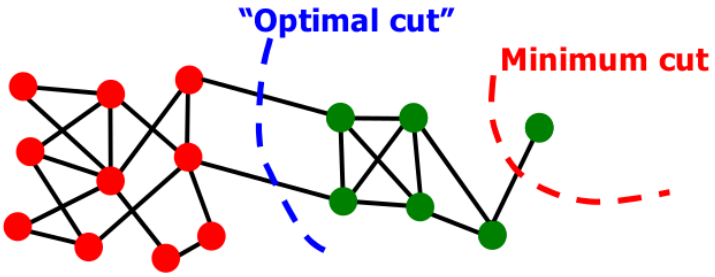


Fig. 10.11 – Prendre le "minimum cut" seul conduit à favoriser des communautés de taille déséquilibrées

Éléments pour des systèmes de recommandations "orientés graphes"

L'analyse de graphes et réseaux sociaux peut aussi être mise à profit pour élaborer des systèmes de recommandation. Ceux-ci visent à filtrer la quantité d'information aujourd'hui disponible dans certains systèmes (catalogue des produits d'un vendeur en ligne, publications scientifiques dans une discipline, films à voir en VOD, etc.). Les techniques classiques reposent sur une analyse de la matrice encodant les informations de l'historique du système : quel utilisateur a vu/aimé/acheté quoi ? Un calcul algorithmique de similarité entre utilisateurs permet ensuite de proposer des éléments nouveaux (films à voir, produits à acheter), qui auraient été perdus dans la masse des résultats avec un moteur de recherche classique.

On peut utiliser une représentation sous forme de graphe biparti pour visualiser les interactions ayant eu lieu dans le système, comme le montre la figure Fig. 10.12. Les utilisateurs sont reliés aux éléments qu'ils ont déjà vus (ou appréciés), avec un poids généralement proportionnel à l'intensité de cette interaction.

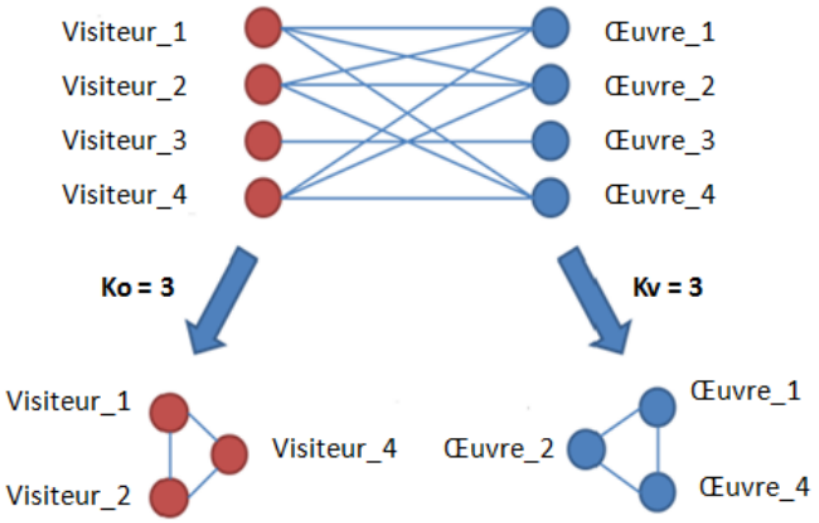


Fig. 10.12 – Graphe biparti d'un système dans lequel interagissent des utilisateurs et des éléments (ici, des visiteurs et des œuvres dans un musée).

S'il existe des techniques spécifiques de calcul de communautés sur des graphes bipartis, il est souvent préférable de procéder à une opération dite de *projection* du graphe biparti. Celle-ci consiste à se ramener à un graphe ne comportant qu'un seul des deux ensembles de nœuds du graphe initial (des utilisateurs ou des articles, donc). Pour cela, on conserve tous les sommets de l'ensemble en question, puis on décide qu'un lien entre deux nœuds existe dans le graphe projeté selon une métrique relative au biparti. Par exemple, ils partagent plus de k voisins. D'autres mesures de similarités peuvent s'avérer utiles, comme le coefficient de Jaccard (https ://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard) (cardinal de l'intersection des voisinages sur

le cardinal de l'union), qui permet de limiter les effets de taille sur la métrique précédente.

Une fois le graphe projeté obtenu, il est possible de procéder à de la détection de communautés. La recommandation d'éléments se fait ensuite sur la base des communautés obtenues, selon des paramètres là encore à choisir : - quels éléments sont retenus (le meilleur de chaque utilisateur, les k meilleurs, etc) - quel ordre leur est affecté pour un utilisateur donné (votes de la communauté, avec ou sans agrégation, avec ou sans pondération par les poids des liens entre les utilisateurs, etc.

Des travaux comme *[THL13]* (page 133) et *[BDFFV14]* (page 133) présentent des états de l'art sur les systèmes de recommandation reposant sur une modélisation "graphe" et peuvent prolonger la lecture de ce cours.

Références

Remerciements

La plupart des illustrations présentées dans cette page sont tirées des publications de Jean-Loup Guillaume ou de celles des auteurs de Mining of Massive Datasets. Qu'ils soient tous chaleureusement remerciés pour leur travail.

- search
- genindex

[CABB04] Crucianu, M., J.-P. Asselin de Beauville, et R. Boné. Méthodes d'analyse factorielle des données : méthodes linéaires et non linéaires. Hermès, Paris, 2004.

[Sap11] Saporta, G. Probabilités, Analyse des Données et Statistique. Technip, Paris, 2011.

[TAL14] Tang, J., S. Alelyani, et H. Liu. *Feature selection for classification : A review*. Dans *Data Classification : Algorithms and Applications*, pages 37–64. 2014.

[Til01] Tillé, Y. Théorie des sondages. Dunod, Paris, 2001.

[GIM99] Gionis, A., P. Indyk, R. Motwani. *Similarity search in high dimensions via hashing*. Dans *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB'99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[LJW07] Lv, Q., W. Josephson, Z. Wang, M. Charikar, K. Li. *Multi-probe LSH : Efficient indexing for high-dimensional similarity search*. Dans *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB'07, pages 950–961. VLDB Endowment, 2007.

[Sam05] Samet, H. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[BGL15] Beel, J., B. Gipp, S. Langer, C. Breitinger. *Research-paper recommender systems : a literature survey*. *International Journal on Digital Libraries*, pages 1–34, 2015.

[BOH13] Bobadilla, J., F. Ortega, A. Hernando, A. Gutiérez. *Recommender systems survey*. *Knowledge-Based Systems*, 46 :109–132, July 2013.

[KBV09] Koren, Y., R. Bell, C. Volinsky. *Matrix factorization techniques for recommender systems*. *Computer*, 42(8) :30–37, Aug. 2009.

[Pat07] Paterek, A. *Improving regularized singular value decomposition for collaborative filtering*. Dans *Proc. KDD Cup Workshop at SIGKDD'07*, 13th *ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, 2007.

[PCB08] Poullot, S., Crucianu, M., Buisson, O. *Scalable Mining of Large Video Databases Using Copy Detection*, Dans *Proceedings of ACM Multimedia 2008*, Vancouver, Canada, 27-30 octobre 2008, pp. 61-70.

[ZWS08] Zhou, Y., D. Wilkinson, R. Schreiber, R. Pan. *Large-scale parallel collaborative filtering for the net-flix prize*. Dans *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM'08, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.

[AV07] Arthur, D. and S. Vassilvitskii. *K-means++ : The advantages of careful seeding*. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[BMV12] Bahmani, B., B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. *Scalable k-means++*. *Proc. VLDB Endowment*, 5(7) :622–633, 2012.

[BNJ03] Blei, D. M., A. Y. Ng, and M. I. Jordan. *Latent Dirichlet allocation*. *Journal of Machine Learning Research*, 3 :993–1022, Mar. 2003.

[BBS14] Bouveyron, C., C. Brunet-Saumard. *Model-based clustering of high-dimensional data : A review*, *Computational Statistics and Data Analysis*, Volume 71, March 2014, Pages 52-78, ISSN 0167-9473, http://dx.doi.org/10.1016/j.csda.2012.12.008.

[JMF99] Jain, A. K., M. N. Murty, and P. J. Flynn. *Data clustering : a review*. *ACM Comput. Surv.* 31, 3 (September 1999), 264-323. DOI=http ://dx.doi.org/10.1145/331499.331504.

[LC10] Lin, F. and W. W. Cohen. *Power iteration clustering*. In J. Fürnkranz and T. Joachims, *editors, International Conference on Machine Learning*, pages 655–662. Omnipress, 2010.

[BNJ03] Blei, D. M., A. Y. Ng, and M. I. Jordan. *Latent dirichlet allocation*. *Journal of Machine Learning Research*, 3 :993–1022, Mar. 2003.

[DDL90] Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. *Indexing by latent semantic analysis*. JASIS, 41(6) :391–407, 1990.

[GM07] Gabrilovich, E. and S. Markovitch. *Computing semantic relatedness using wikipedia-based explicit semantic analysis*. *In Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[Hof99] Hofmann, T. *Probabilistic latent semantic indexing*. *In Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'99, pages 50–57, New York, NY, USA, 1999. ACM.

[MCC13] Mikolov, T., K. Chen, G. Corrado, and J. Dean. *Efficient estimation of word representations in vector space*. CoRR, abs/1301.3781, 2013.

[MSC13] Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. *Distributed representations of words and phrases and their compositionality*. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds, *Advances in Neural Information Processing Systems* 26, pages 3111–3119. Curran Associates, Inc., 2013.

[SMG86] Salton, G. and M. J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

[LRU11] Leskovec, J., Rajaraman, A. and J. D. Ullman. *Mining of Massive Datasets*. *Cambridge University Press*, New York, NY, USA, 2011.

[BBL99] Bousquet, O., S. Boucheron, G. Lugosi. *Introduction to statistical learning theory*. *Neural Information Processing Systems*, 1999.

[BCDW07] Bottou, L., O. Chapelle, D. DeCoste, and J. Weston. *Large-Scale Kernel Machines* (*Neural Information Processing*). MIT Press, 2007.

[CSZ06] Chapelle, O., B. Schölkopf, and A. Zien, eds. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[DT05] Duc Dung Nguyen, Tu Bao Ho. *An efficient method for simplifying support vector machines*. *Proc. 22nd Intl. Conf. on Machine Learning* (ICML'05). ACM, New York, NY, USA, 617-624.

[Gär03] Gärtner, T. *A survey of kernels for structured data*. SIGKDD Explorer Newsletter 5 : 49-58, 2003.

[ScS02] Schölkopf, B and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[Joa99] Joachims, T. *Estimating the generalization performance of an SVM efficiently*. LS VIII Report 25, 1999. Universität Dortmund, Germany.

[Vap98] Vapnik, V. *Statistical Learning Theory*. John Wiley, New York. 1998.

[AB02] Albert, R. and Barabási, A.-L. *Statistical mechanics of complex networks*. Reviews of Modern Physics. 2002. 74 : 47–97.

[G73] Granovetter, M. S. *The Strength of Weak Ties*. American Journal of Sociology, Volume 78, Issue 6 (May, 1973), 1360–1380.

[K00] Kleinberg, J. M. *Navigation in a small world*. Nature. 2000 Aug 24 ;406(6798) :845. http://www.nature.com/nature/journal/v406/n6798/full/406845a0.html#B2

[M67] Travers, J. and Milgram, S. *An experimental study of the small world problem*. Sociometry, Vol. 32, No. 4. (Dec., 1969), pp. 425-443. http://www.jstor.org/stable/2786545

[MABDHLC10] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. *Pregel : A System for Large-Scale Graph Processing*. SIGMOD 2010. https://kowshik.github.io/JPregel/pregel_paper.pdf

[MMDS] Leskovec, J., Rajaraman, A. and J. D. Ullman. *Mining of Massive Datasets*. *Cambridge University Press*, New York, NY, USA, 2011. Accessible en version numérique http://www.mmds.org

[WS98] Watts, D. J. and Strogatz, S. H. *Collective dynamics of 'small-world' networks*. Nature 393, 440-442. http://www.nature.com/nature/journal/v393/n6684/full/393440a0.html

[BGLL08] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E. *Fast unfolding of communities in large networks*. Journal of Statistical Mechanics : Theory and Experiment 2008 (10), P1000. https://arxiv.org/pdf/0803.0476v2.pdf

[MMDS] Leskovec, J., Rajaraman, A. and J. D. Ullman., *Mining of Massive Datasets*. *Cambridge University Press*, New York, NY, USA, 2011. Accessible en version numérique http://www.mmds.org

[SM00] Shi, J. and Malik, J., *Normalized Cuts and Image Segmentation*. IEEE PAMI 2000. https://people.eecs.berkeley.edu/~malik/papers/SM-ncut.pdf

[THL13] Tang, J. and Hu, X. and Liu, H., *Social Recommendation : A Review*.Social Network Analysis and Mining (SNAM), 2013. http://www.jiliang.xyz/publication/socialrecommendationreview.pdf

[BDFFV14] Bernardes, D. and Diaby, M. and Fournier, R. and Fogelman-Soulié, F. and Viennet, E., *A Social Formalism and Survey for Recommender Systems.* SIGKDD Explorations, 16(2), 20–37. http://raphael.fournier-sniehotta.fr/bibliography/files/SocialFormalismSurveyRS-BernardesDiabyFournierViennetFogelman.pdf