Introduction à la fouille de textes
université de Paris 3 - Sorbonne Nouvelle

I. Tellier

Table des matières

Chapitre 1

Introduction

La fouille de textes (text mining) est l'héritière directe de la fouille de données (data-mining), née dans les années 90. A cette époque, les ordinateurs personnels se généralisent, leur capacité de calcul et de mémorisation atteignent des seuils tels qu'ils commencent à pouvoir traiter de grandes quantités d'informations. La fouille de données vise à tirer le meilleur profit possible de cette situation inédite (hors contexte militaire !) pour créer des programmes capables de prendre des décisions pertinentes. Elle naît dans différents environnements qui ont l'habitude de gérer des bases de données conséquentes. C'est notamment le cas des banques et des assurances (pour décider de l'attribution d'un crédit, par exemple), de la médecine (pour effectuer un diagnostic ou évaluer l'efficacité d'un médicament) ou encore de la vente et du marketing (pour cibler les publicités aux clients) : autant de domaines où l'efficacité est directement monétisable !

Jusqu'alors, pour automatiser des traitements requérant une certaine expertise spécifique, l'informatique s'était focalisée sur les techniques de l'intelligence artificielle (IA) dite "classique", qui reposent sur une modélisation symbolique explicite de la situation, et sur des raisonnements logiques. La fouille de données adopte un parti pris inverse. Son mot d'ordre est en quelque sorte de faire "feu de tout bois" pour obtenir les meilleurs résultats possibles, et ceci en faisant plus confiance à l'information implicite contenue dans de grandes quantités de données qu'aux compétences générales d'un expert. Contrairement à l'IA traditionnelle, la fouille de données privilégie une démarche *inductive* plutôt que *déductive*, et *numérique* plutôt que *symbolique*. Cette mutation est considérable, à partir des années 90 tous les domaines de l'informatique en sont impactés.

C'est le cas du TAL (traitement automatique des langues), qui vise à écrire des programmes capables de comprendre les langues "naturelles", celles que les humains utilisent entre eux. Les outils traditionnels du TAL proviennent de l'informatique théorique et de l'IA classique : automates, grammaires formelles, représentations logiques... Ils sont malheureusement coûteux en développement et en temps de calcul, et de ce fait peu adaptés au traitement de grandes quantités de textes "tout venant", c'est-à-dire ne respectant pas nécessairement les règles de bonnes constructions syntaxiques.

Or le développement d'Internet, et singulièrement du Web 2.0, a rendu accessible une énorme quantité de tels textes, souvent mal rédigés et pourtant potentiellement

riches d'énormément d'informations utiles. Dans la perspective qui nous intéresse ici, il n'est ni possible ni nécessaire d'analyser en profondeur ces corpus pour les exploiter. Ils présentent d'ailleurs une si grande variété de styles et de genres qu'aucun outil générique de TAL n'est capable de les manipuler de façon homogène. Le pari de la fouille de textes est donc plus modeste : elle ne cherche pas à comprendre le sens profond des grandes quantités de textes auxquelles elle s'attaque mais à traiter efficacement certaines *tâches* précises bien délimitées.

C'est donc tout naturellement ces *tâches* qui serviront à structurer ce document, plus que les niveaux d'analyse linguistique habituellement mis en avant quand il s'agit de présenter le TAL. Certaines sont nées de la nature même des corpus manipulés (la recherche d'information), d'autres sont des transpositions directes de celles gérées par la fouille de données (la classification), d'autres enfin sont plus spécifiques des données textuelles (l'annotation, l'extraction d'information).

Le premier chapitre de ce document présente un panorama d'ensemble des principales tâches relevant de la fouille de textes, en évoquant leurs propriétés, leurs points communs et leurs différences. Il se concentre sur un petit nombre d'entre elles, qualifiées d'"élémentaires", en montrant que d'autres tâches plus complexes peuvent être traitées comme une combinaison de ces tâches élémentaires. Chacun des chapitres suivants est consacré à une de ces tâches élémentaires, et présente les principales techniques informatiques permettant de la réaliser. Une liste de logiciels et de sites Web qui les mettent en œuvre clôt chaque chapitre.

La fouille de textes peut être assez frustrante pour ceux que la nature linguistique des données manipulées intéresse en priorité. Bien souvent en effet, on pourrait croire qu'il vaut mieux être mathématicien que linguiste pour comprendre les méthodes employées. Mais l'objectif de ce document n'est pas seulement d'apporter des éléments de vulgarisation aux linguistes. Il vise à montrer que la fouille de textes est un domaine en pleine mutation et qu'il ne peut que bénéficier d'une plus grande hybridation entre techniques issues de la fouille de données et connaissances linguistiques. Il se veut donc aussi une invitation à la recherche de nouvelles pistes pour cette hybridation.

Chapitre 2

Les tâches élémentaires de la fouille de textes

1 Introduction

Une tâche, au sens informatique, est la spécification d'un programme qui mime une compétence précise d'un être humain. Cette unité de découpage est traditionnelle en intelligence artificielle, qui se scinde en sous-domaines très spécialisés, visant par exemple à écrire des programmes capables de jouer aux échecs, ou de conduire une voiture, ou encore de reconnaître le contenu d'une image... La particularité d'une tâche de fouille de textes est de faire intervenir des données textuelles, généralement en grandes quantités. Tenir une conversation écrite ou traduire un texte d'une langue dans une autre pourraient être des exemples de telles tâches, mais elle sont encore trop complexes. Le terme de fouille de textes sera plutôt réservé à des programmes ayant des objectifs plus simples. Pourtant, aussi "élémentaires" que soient les tâches évoquées dans ce document, elles restent difficiles pour les ordinateurs, qui sont encore loin d'avoir atteint le niveau de compétence linguistique d'un enfant de cinq ans. Aussi, la notion d'évaluation quantitative joue-t-il un rôle fondamental dans le domaine. En fouille de textes comme en fouille de données, tout est quantifiable, les différentes solutions envisagées peuvent être évaluées et comparées. Comme nous le verrons, la qualité d'un programme se mesurera à sa capacité à s'approcher le plus possible d'une solution de référence validée par un humain.

Mettre en avant la notion de tâche permet de distinguer la fouille de textes de disciplines comme la lexicométrie ou la textométrie, qui sont l'application de méthodes issues de la statistique descriptive à des données textuelles, et font aussi appel à des évaluations quantitatives. La fouille de textes peut, parfois, exploiter les statistiques, mais la caractérisation des propriétés d'un texte ou d'un corpus n'est pas sa finalité dernière. Elle a toujours en vue un autre but, formulé dans cette notion de tâche. Certaines tâches élémentaires joueront en outre le rôle d'unités de base en fouille de textes, ce sont celles sur lesquelles nous nous attarderons le plus.

Chaque tâche a une visée applicative précise et autonome et peut être spécifiée par ses entrées, ses sorties et les ressources externes auxquelles elle peut éventuellement faire appel. Adopter un tel point de vue, pour lequel les tâches sont des "boîtes noires", permet d'éviter dans un premier temps de préciser les niveaux d'analyse

linguistique requis pour leur réalisation : la fouille de textes est en quelque sorte "linguistiquement agnostique", au sens où tout y est bon du moment que ça marche (Whatever Works) ! Le défi qui intéresse les chercheurs consiste, bien sûr, à traduire en programmes efficaces ces spécifications. Mais, dans ce chapitre préliminaire, on se contentera d'une description externe des tâches considérées ; les techniques utilisées pour les implémenter dans des programmes ne seront explicitées que dans les chapitres suivants.

Dans celui-ci, on commencera donc par analyser les composantes qui rentrent dans la définition d'une tâche de fouille de textes quelconque à l'aide d'un schéma général. On passera ensuite en revue les principales tâches "élémentaires" qui seront détaillées par la suite, en montrant comment elles rentrent dans ce schéma général. On montrera aussi que, via recodage de leurs données ou reformulation de leur objectif, ces tâches élémentaires sont en fait très liées les unes aux autres, par exemple que certaines d'entre elles permettent d'en *simuler* certaines autres. On expliquera aussi comment, en en combinant plusieurs, on peut parvenir à en réaliser d'autres plus complexes. Enfin, on s'attardera sur un pré-traitement des textes qui s'avère indispensable pour plusieurs des tâches élémentaires abordées ici, qui consiste à les transformer en un tableau de nombres. Cette étape préliminaire permet d'appliquer sur les textes les techniques directement issues de la fouille de données, qui s'est spécialisée dans la manipulation de tels tableaux.

2 La notion de tâche et ses composantes

2.1 Schéma général d'une tâche

Dans la suite de ce document, nous nous efforcerons de garder le même mode de description pour chacune des tâches abordées. Ce mode est synthétisé dans le schéma très simple de la figure 2.1.

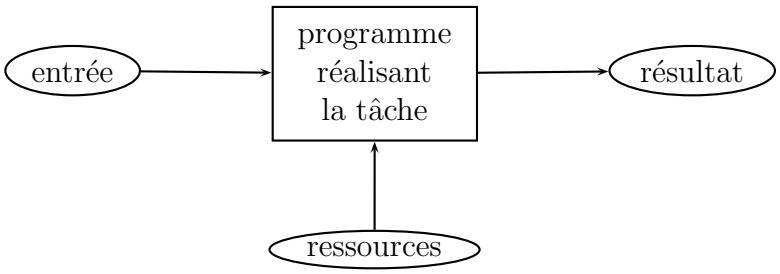


FIGURE 2.1 – Schéma général d'une tâche de fouille de textes

Dans ce schéma, les données figurent dans des ovales tandis que le programme réalisant la tâche est matérialisé par un rectangle. C'est bien sûr dans les différentes données que la spécificité de la fouille de textes se manifestera : tout ou parties d'entre elles seront de nature textuelle, ou en découleront après un pré-traitement. Ce schéma est très simple, mais nous verrons qu'il oblige tout de même à se poser quelques bonnes questions. Par exemple, il n'est pas toujours facile de distinguer ce qui joue le rôle de données d'entrée ou de ressources dans la définition d'une tâche.

Un bon critère serait le suivant : une ressource est une donnée stable, qui n'est pas modifiée d'une exécution du programme à une autre, alors que la donnée d'entrée, elle, change chaque fois. Certaines ressources sont obligatoires dans la définition de certaines tâches, d'autres facultatives. C'est souvent par ce biais que des connaissances externes et générales peuvent être intégrées au processus de réalisation de la tâche. Les ressources sont donc un des principaux leviers pour faire rentrer un peu de linguistique dans le domaine de la fouille de textes. C'est le cheval de Troie des linguistes... Nous détaillons dans la suite de cette partie chacun des composants de ce schéma.

2.2 Les données d'entrées

Les données qui font l'objet de tâches de fouilles se présentent suivant différents formats. Nous en distinguerons trois principaux : les tableaux utilisés en fouille de données, les textes bruts et les documents semi-structurés. Nous présentons simplement ici leurs principales propriétés, et détaillerons plus loin, en partie 5, comment transformer des textes bruts ou des documents semi-structurés en données tabulaires.

Données tabulaires

Commençons donc par les tableaux exploités en fouille de données. Comme évoqué en introduction, cette discipline est née notamment dans les milieux des banques, des assurances et de la médecine, domaines qui ont intégré depuis longtemps l'usage des bases de données informatiques. Une "donnée" peut, dans ce contexte, correspondre à un client ou à un patient mais aussi à un produit, un procédé, un médicament... Dans un tableau de données, chaque instance est décrite par un certain nombre d'*attributs typés* (ou de champs, dans le vocabulaire des bases données). Les différents types possibles des attributs sont les types élémentaires traditionnels de l'informatique : booléen, caractère, nombre (entier ou réel), chaîne de caractères, valeur prise dans une liste finie (on parlera alors d'attribut "nominal")... La valeur prise par un attribut peut être obligatoire (par exemple une date de naissance) ou facultative (par exemple une date de mort...). Les algorithmes ne sont pas tous égaux devant les données : certains requièrent des tableaux entièrement remplis, d'autres s'arrangent très bien de valeurs manquantes. Certains -et ce sont en général les plus efficaces- ne savent manipuler que des tableaux complets de nombres. Une donnée uniquement décrite par une liste de nombres peut en effet facilement être assimilée à un point dans un espace vectoriel ou, ce qui revient au même, à un vecteur dont on fournit les coordonnées. Cette reformulation change tout, car elle permet de bénéficier de l'arsenal mathématique développé dans le cadre de ce type d'espace (nous y reviendrons, bien sûr...). Traditionnellement, les données sont disposées en lignes (une donnée par ligne), les attributs en colonnes. L'ordre des lignes et des colonnes n'a aucune importance, au sens où en changer ne modifiera en rien le résultat des algorithmes de fouille qui y seront appliqués. Seule la dernière colonne joue, pour certaines tâches, un rôle particulier (nous y reviendrons aussi). La figure 2.2 montre

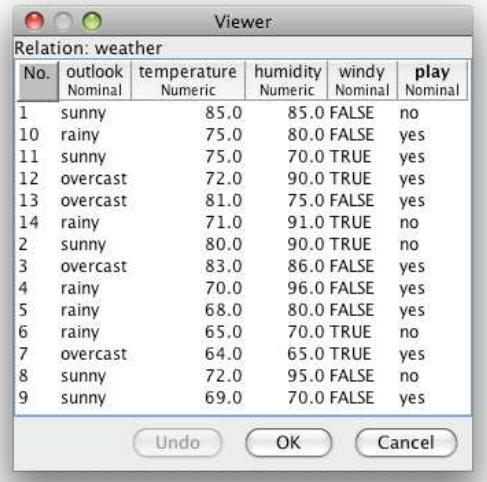


FIGURE 2.2 – Copie d'écran d'un tableau de données du logiciel Weka

un tableau issu du logiciel libre et gratuit Weka[1], qui implémente les principaux algorithmes de fouille de données. Chaque donnée est la description d'une situation météorologique (caractérisée par des attributs de différents types) associée au fait qu'elle permet ou non (dernière colonne) de jouer au tennis. Cet exemple illustre que la fouille de données peut s'appliquer aussi aux sujets les plus futiles...

Textes bruts

Les textes, même numérisés, ne présentent pas du tout les mêmes propriétés que les tableaux de données. En termes de structures, ils semblent même situés à l'opposé du "spectre" : autant les tableaux ont un haut degré d'organisation, autant les textes sont apparemment faiblement structurés. Et ceci d'autant plus qu'en fouille de textes, on ne s'intéressera principalement qu'à des *textes bruts*, c'est-à-dire de simples *séquences de caractères* d'où toute mise en formes est absente. Tout ce qui ne vise qu'à la visualisation (police et taille des caractères, mises en gras ou en italique, alignement de la page, sauts de lignes, etc.) ou à la structuration d'un document (en parties, sous-parties et paragraphes, en listes et énumérations etc.) et constitue la raison d'être des traitements de textes est en effet dans ce cas complètement ignoré. Un texte brut est un simple fichier au format ".txt", uniquement constitué de caractères pris parmi un ensemble fini, codés suivant une certaine norme. Les caractères sont les atomes indivisibles du fichier ; ils sont dits alphanumériques car ils intègrent aussi bien les lettres de l'alphabet (de n'importe quel alphabet, en préservant tout de même la distinction majuscule/minuscule car les codes associés diffèrent) et les symboles numériques et mathématiques que tous les ceux pouvant être tapés sur un clavier d'ordinateur (ponctuations, symboles monétaires, etc.). Toutes les unités d'écriture des langues non alphabétiques (idéogrammes par exemple) sont aussi considérées

1. http ://www.cs.waikato.ac.nz/ml/weka/

comme des caractères indivisibles, si le codage adopté les accepte comme tels. Ainsi, dans un texte brut, la seule structure présente est l'ordre linéaire dans lequel les caractères apparaissent. En revanche, les notions de mots, de phrases, de paragraphe... n'y ont *a priori* pas de sens, sauf à réaliser un pré-traitement qui les identifie. Nous en reparlerons plus tard.

Documents semi-structurés

Le troisième format possible pour les données d'entrée d'un programme de fouille de textes est intermédiaire entre les précédents : il est plus structuré qu'un texte brut, mais moins qu'un tableau, et on l'appelle parfois pour cela "semi-structuré" : c'est celui des documents XML. Nous n'allons pas faire ici un cours complet sur XML, juste mettre en avant ce qui distingue ce format des deux précédents. En fait, rien n'empêche de traiter un document en XML exactement de la même façon qu'un texte brut : il suffit pour cela d'admettre que les éléments propres au langage utilisé (principalement les balises ouvrantes et fermantes) soient considérés comme des "caractères" indivisibles supplémentaires, qui s'ajoutent aux autres. La figure 2.3 montre un morceau de code HTML (qui peut être considéré comme un cas particulier de langage XML) tel qu'il apparaît dans un éditeur de texte. Le pré-traitement consistant à identifier les balises est trivial ; ce code peut ainsi être considéré comme un "texte brut" écrit dans un nouvel alphabet : celui contenant tous les caractères alphanumériques ainsi que les balises ¡table¿, ¡/table¿, ¡tr¿, ¡/tr¿, ¡th¿, ¡/th¿, ¡td¿ et ¡/td¿, considérées comme des unités indivisibles.

```
<table>
<tr><th>produit</th><th>marque</th><th>prix en euros</th></tr>
<tr><td>ordinateur portable</td><td>truc</td><td>800</td></tr>
<tr><td>tablette</td><td>machin</td><td>200</td></tr>
</table>
```

FIGURE 2.3 – Code HTML

Mais ce n'est pas tout. La particularité d'un document XML est qu'il autorise d'autres lectures possibles. Les balises, en effet, respectent une syntaxe qui décrit une *structure*, visualisable dans un *arbre*. La figure 2.4 montre l'arbre associé au code HTML précédent : les balises jouent le rôle de nœuds internes de l'arbre, tandis que le reste du texte se trouve disposé dans ses "feuilles".

Un arbre est un objet à deux dimensions, alors qu'un "texte brut" n'en a en quelque sorte qu'une seule. Deux relations d'ordre distinctes peuvent en effet être définies entre les éléments qui constituent un arbre : la relation verticale de "descendance", qui relie entre eux les noeuds appartenant à un même chemin qui mène de la racine à une feuille (on parle de "nœuds père"/"nœuds fils"...) et la relation horizontale de "précédence", qui ordonne les fils successifs d'un même père. Ces deux relations d'ordre sont partielles, au sens où si on prend au hasard deux nœuds dans un même arbre, ils peuvent très bien n'entretenir aucune de ces relations entre eux. Au contraire, la relation d'ordre linéaire présente dans un texte brut était totale : on peut toujours dire de deux caractères quelconques lequel précède l'autre. Nous

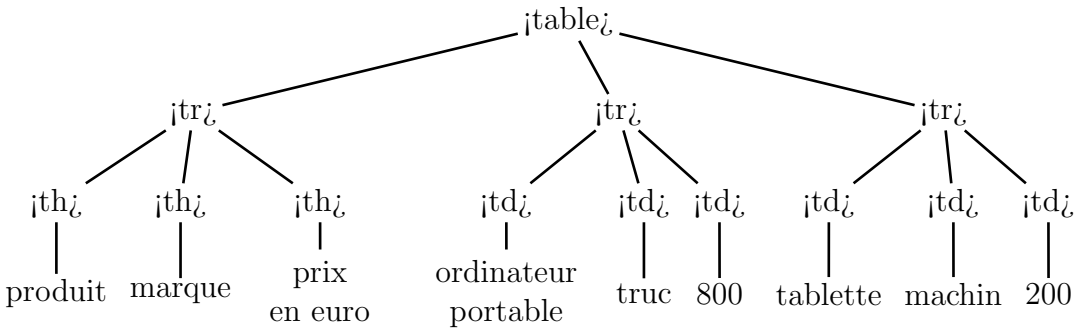


FIGURE 2.4 – Arbre correspondant au code HTML précédent

verrons que ces propriétés auront des incidences sur les programmes de fouille qui auront à manipuler l'un ou l'autre de ces formats. Remarquons pour finir que, dans certains cas, les documents semi-structurés peuvent faire l'objet d'une représentation supplémentaire C'est le cas des pages HTML, destinées à être interprétées et visualisées par un navigateur Web. La figure 2.5 montre le résultat de l'interprétation du code HTML de la figure 2.3 par un navigateur : c'était en fait le code d'un tableau, et on pourrait donc stocker toutes les données tabulaires par des données semi-structurées de ce type. Mais un texte brut ou un document XML constitue une donnée unique, alors que les différentes lignes d'un tableau sont en général des données distinctes, il faudra donc préciser dans ce cas la granularité de la donnée considérée.

produit	**marque**	**prix en euros**
ordinateur portable	truc	800
tablette	machin	200

FIGURE 2.5 – Visualisation du code HTML précédent par un navigateur

Notons que, suivant la représentation que l'on favorise, la notion de "distance" entre les éléments de la donnée (par exemple entre les "mots" "truc" et "machin" dans l'exemple précédent) n'aura pas du tout le même sens : dans le code HTML, elle peut correspondre au nombre de caractères ou de mots qui séparent les deux mots considérés, alors que dans l'arbre associé elle pourrait se calculer comme le nombre d'arcs à parcourir pour aller de l'un à l'autre. Quant au rendu visuel du navigateur, il introduit des relations de voisinage encore nouvelles ("truc" et "machin" y apparaissent dans des cases voisines alors qu'ils étaient assez éloignés l'un de l'autre dans le code HTML et dans l'arbre XML)...

2.3 Les ressources

La notion de ressource est nettement moins bien formalisée que celle de "donnée". En fait, toute donnée ou tout programme disponible (et utile) pendant l'exécution d'une tâche peut être qualifié de ressource. Il n'est pas toujours facile d'isoler la ressource du programme qui l'utilise et il est de toute façon impossible de faire une

recension exhaustive de toutes les formes qu'elle peut prendre. Nous nous contenterons donc ici de citer quelques ressources "classiques", des plus simples aux plus complexes :

- les ressources les plus simples sont des listes, par exemple celle des symboles de ponctuation d'une certaine langue ou de certaines de ses unités multimots, celle de ses "mots vides" (on verra ce qu'ils sont et à quoi ils servent en partie 5) ou encore la recension de noms propres courants (utiles pour la tâche d'extraction d'information). Elles se présentent alors sous la forme de simples fichiers ".txt".
- des collections de textes complets, qu'ils soient dans un format "texte brut" ou "document semi-structuré", peuvent aussi être considérées comme des ressources : le corpus sur lequel porte une tâche de recherche d'information, par exemple. En dernière analyse le Web est en lui-même une ressource, même si son accès est en général médiatisé par un moteur de recherche (peu de personnes peuvent se permettre d'en avoir une copie complète sur leur poste de travail...).
- certaines données ou "bases de connaissances" décrites par des formats textuels, tabulaires ou semi-structurés peuvent aussi jouer ce rôle : lexiques, dictionnaires (qui associent au moins une information à chaque élément qu'ils contiennent), thesaurus et ontologies, en tant qu'ensembles structurés de mots, termes ou concepts...
- côté "programmes", tout logiciel réalisant un pré-traitement élémentaire sur certains formats de données peut être assimilé à une ressource : programme de segmentation d'un texte brut en "mots" ou unités multi-mots, détection des séparateurs de "phrases", etc. Notons qu'il y a de fortes chances que ces programmes fassent eux-mêmes appel aux ressources citées en premier dans cette énumération.
- Enfin, les programmes réalisant des traitements linguistiques plus élaborés, comme les lemmatiseurs, les étiqueteurs morpho-syntaxiques, les analyseurs de surface ou les "parseurs" complets peuvent bien sûr postuler également au statut de ressource. Ils doivent, pour la plupart, faire eux-mêmes aussi appel à d'autres ressources comme des règles morphologiques ou des grammaires. Rappelons toutefois que la fouille de textes manipule de grandes quantités de données, et que la qualité de ces outils ainsi que leur temps d'exécution ne doivent pas être un obstacle à l'obtention rapide d'un résultat.

Dans la description des tâches élémentaires qui suivent, nous ne signaleront en "ressources" que celles qui sont indispensables à leur définition.

2.4 Les programmes

Les programmes réalisant une tâche sont évidemment le cœur de la fouille de textes et leurs principes de fonctionnement feront l'objet des chapitres suivants. Dans cette partie, notre objectif est simplement d'introduire les deux principales approches possibles pour construire un programme de fouille de textes : l'écrire "à la main" ou utiliser des techniques d'apprentissage automatique. Ce sera aussi l'occasion d'introduire les notions d'évaluation et de protocole mis en œuvre dans les compétitions de fouille de textes pour évaluer et comparer la qualité de différents

programmes.

Programmes écrits "à la main"

Ecrire des programmes est la compétence première des informaticiens. Mais écrire un programme de fouille de textes requiert plus que cela : il faut aussi un minimum de *connaissances linguistiques*. Qui peut prétendre implémenter un étiqueteur du chinois s'il ne connaît rien à cette langue ? A défaut de connaissances précises, des *ressources* (au sens de la partie 2.3) sont indispensables. La difficulté de la tâche est multiple : il faut à la fois que le programme prenne en compte les règles de la langue, mais aussi qu'il ne "bloque" pas face à des fautes ou des erreurs, inévitables dans les données réelles. Sans compter qu'une langue n'est pas homogène : elle varie selon la modalité (oral/écrit), le genre du texte, le registre utilisé, le style de l'auteur... Et, bien sûr, chaque programme écrit manuellement est spécifique de la langue pour laquelle il a été écrit : pour chaque langue nouvelle, tout est à refaire.

Apprentissage automatique

L'alternative à l'écriture manuelle de programmes est venue d'un sous-domaine de l'intelligence artificielle qui a connu un très fort développement à partir des années 1990 : l'apprentissage automatique (ou artificiel). L'essor de la fouille de données lui est contemporain et ce n'est pas un hasard. On peut définir l'apprentissage automatique comme "la branche de l'informatique qui étudie les programmes capables de s'améliorer par expérience" (traduction d'une citation de Tom Mitchell, un des pionniers du domaine). Plus simplement, on peut aussi dire que c'est "l'art de transformer des exemples en programme". Concrètement, un programme d'apprentissage automatique est entraîné grâce à des *exemples* de la tâche envisagée. Par différentes techniques que nous présenterons dans les chapitres suivants, ces exemples permettent de fixer la valeur de certains paramètres, et de construire ce que l'on appelle généralement un *modèle*. A son tour, le modèle spécifie (ou définit) un programme capable de réaliser la tâche initiale pour de nouvelles données. Ce processus en deux temps est synthétisé dans le schéma de la figure 2.6. C'est le même programme qui s'instancie dans les deux "rectangles" de cette figure, en deux phases successives : la phase d'apprentissage précède la phase d'utilisation de ce qui a été appris sur de nouvelles données.

La nature des *exemples* fournis au programme d'apprentissage permet de distinguer entre deux familles de méthodes :

- si les exemples sont des couples (entrée, résultat) corrects du programme réalisant la tâche, on dit que l'apprentissage est *supervisé* : il est guidé, dirigé vers l'acquisition d'un programme dont le fonctionnement idéal sur certaines données est fourni explicitement ;
- si les exemples sont simplement des données d'entrées (et éventuellement quelques paramètres supplémentaires) mais sans leur associer un résultat souhaité, on dit que l'apprentissage est *non supervisé*.

Evidemment, l'apprentissage non supervisé est plus difficile et donne généralement de moins bons résultats que l'apprentissage supervisé, mais les données qu'il requiert sont moins couteuses à obtenir. Il existe aussi des situations intermédiaires :

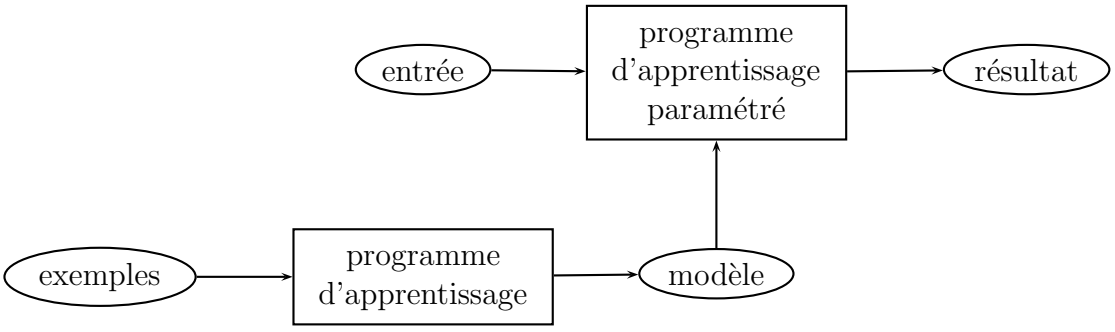


FIGURE 2.6 – Schéma général d'une approche par apprentissage automatique

l'apprentissage *semi-supervisé* s'appuie sur un mélange de données des deux types précédents. Nous verrons plus loin (en particulier dans le chapitre consacré à la classification) dans quels contextes il est fait appel à ces différentes approches. La force d'un programme d'apprentissage automatique est qu'ils *s'adapte aux exemples* qui lui sont fournis. S'ils sont en français, il apprend à traiter la langue française. Mais le même programme, confronté à des exemples en chinois, apprendra à manipuler le chinois. Les enfants humains ne font rien d'autre quand ils acquièrent la langue parlée dans leur environnement : la capacité d'apprentissage du langage est générique, elle s'instancie en fonction des informations disponibles. Les programmes d'apprentissage automatique visent à reproduire cette généricité.

Attention toutefois : il n'y a rien de magique là-dedans. Le résultat d'un programme d'apprentissage automatique dépend beaucoup de la qualité et de la quantité des exemples qui lui ont été présentés. Ces exemples peuvent être difficiles et coûteux (en temps de travail humain notamment) à obtenir, et une expertise linguistique est souvent nécessaire pour les construire. Et les mêmes problèmes se posent que quand tout est écrit manuellement : un programme "appris" risque de ne bien se comporter que sur des données nouvelles qui "ressemblent" (en termes de modalité, de genre, de style...) à celles qui lui ont servi d'exemples.

Protocoles d'évaluation et d'apprentissage

En fouille de textes, on attache une grande importance à l'évaluation quantitative des programmes utilisés. On doit pouvoir comparer objectivement deux programmes distincts censés réaliser la même tâche. Pour cela, il faut disposer de données de référence, pour lesquelles le résultat attendu du programme est connu et validé par des humains. On appelle ce type de ressource un *Gold Standard*. Constituer un tel corpus de données/résultats peut être long et fastidieux (c'est le même travail que pour construire des exemples soumis à un programme d'apprentissage automatique !) mais c'est devenu une pré-requis indispensable du domaine. Et, bien sûr, on doit pouvoir comparer le résultat fourni par un programme avec le résultat de référence attendu, à l'aide de mesures de qualité que nous évoquerons en temps voulu (elles sont en partie spécifiques de chaque tâche).

Mais un point fondamental peut d'ors et déjà être signalé : pour évaluer équitablement un programme, il faut lui présenter des données nouvelles qui ne lui ont pas déjà

été soumises, notamment lors d'une phase d'apprentissage. La notion de *protocole* sert à caractériser le rôle des différentes données qui interviennent dans la constitution/l'évaluation d'un programme. Typiquement, pour une campagne d'évaluation de différents programmes réalisant une certaine tâche de fouille de textes, la procédure est la suivante :

- les organisateurs de la campagne constituent un corpus "Gold Standard" de couples données/résultats corrects pour la tâche considérée. Ce corpus est divisé en deux sous-ensembles disjoints : un "corpus d'entrainement" et un "corpus de test".
- ils diffusent aux participants les couples données/résultats du "corpus d'entrainement" uniquement. Cet échantillon permet aux candidats de comprendre le format des données (et des résultats) utilisé et la nature de la tâche à accomplir. Libre à eux de construire le programme qui réalise cette tâche "à la main" ou par apprentissage automatique en se fondant sur les exemples fournis dans cet ensemble d'entrainement. En pratique, les délais imposés par les campagnes officielles associées à des "challenges scientifiques" permettent rarement de se passer de l'apprentissage automatique ! En général (comme nous l'avons supposé ici), les résultats attendus en font partie, ce qui permet de faire appel aux approches supervisées.
- pour évaluer et comparer la performance des programmes élaborés ou appris par les participants, les organisateurs de la campagne leur soumettent les données (uniquement les données d'entrée bien sûr, sans le résultat associé) du "corpus de test", différentes de celles du "corpus d'entrainement". Les résultats des programmes sur ces données sont évalués en les comparant aux résultats corrects attendus, connus des seuls organisateurs.

Ce protocole garantit une compétition équitable et objective. C'est pourquoi, même sans participer à un "challenge" mettant en concurrence des programmes différents, il est aussi adopté systématiquement pour mesurer la qualité d'un programme, notamment quand celui-ci est construit par apprentissage automatique. Comme nous l'avons vu, ce type de programme requiert un ensemble d'exemples. Il est ainsi désormais d'usage de répartir l'ensemble de tous les exemples disponibles en un ensemble d'entrainement et un ensemble de test disjoints, de faire fonctionner le système d'apprentissage automatique avec l'ensemble d'entrainement uniquement et d'évaluer le programme paramétré obtenu avec l'ensemble de test. Parfois, un troisième ensemble disjoint appelé "de développement" est aussi défini : il est généralement utilisé après l'apprentissage mais avant le test, pour opérer des choix de paramètres, des modifications manuelles ou des tests intermédiaires sur le résultat d'un programme appris, sans pour autant empiéter sur la phase de "test" finale.

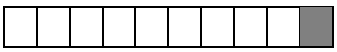
Ce protocole présente toutefois de gros inconvénients : nous l'avons vu, les exemples qui servent à alimenter les programmes d'apprentissage automatique sont souvent rares et précieux. Or, la procédure impose de se passer de certains d'entre eux, pour les réserver à l'évaluation du programme appris. Même si l'ensemble d'entrainement est en général plus grand que l'ensemble de test (une proportion 80%/20% est assez standard), ceci a deux conséquences fâcheuses :

- le programme est appris sur moins d'exemples que ceux réellement disponibles, il risque donc d'être de moins bonne qualité qu'un programme qui serait appris

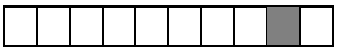
sur la totalité des exemples ;
- la répartition des exemples entre l'ensemble d'entrainement et l'ensemble de test étant arbitraire, il y a le risque d'un "biais" dû à un sort malencontreux : par exemple, certains phénomènes linguistiques peuvent n'être présents que dans un des deux sous-ensembles, ce qui va fausser l'évaluation. Peut-être qu'un autre découpage donnerait des résultats d'évaluation bien différents...

Pour remédier à ces inconvénients, il est recommandé (surtout si le nombre d'exemples disponibles est relativement limité) d'utiliser une variante raffinée du protocole précédent appelée "validation croisée" ("cross-validation" en anglais). Elle consiste à procéder de la façon suivante :

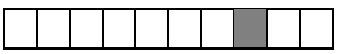
- découper l'ensemble des exemples en n sous-ensembles disjoints (par exemple, $n = 10$). La valeur de n s'appelle le nombre de "plis" (traduction de "folds" en anglais) de la validation croisée.
- réaliser n expériences d'apprentissage automatique/évaluation distinctes : chaque expérience consiste à prendre $n - 1$ des sous-ensembles pour l'entrainement, et le n-ième sous-ensemble restant pour le test. Par exemple, si $n = 10$, la première expérience consiste à prendre les 9 premiers sous-ensembles pour apprendre, le 10ème pour évaluer. Pour la 2ème expérience, on met de côté le 9ème sous-ensemble pour le test et on réalise l'apprentissage avec tous les sous-ensembles sauf ce 9ème, et ainsi de suite pour chacun des sous-ensembles qui va, à son tour, servir d'ensemble de test. On obtient ainsi 10 (n, de manière générale) évaluations distinctes.
- Le programme finalement gardé est celui qui obtient la meilleure des n évaluations (ou alors, le programme obtenu en utilisant *tous* les exemples pour son apprentissage), mais la mesure de qualité retenue est la *moyenne des n évaluations effectuées*.



1ère expérience
(en blanc : ensemble d'apprentissage, en gris : ensemble de test)



2ème expérience



3ème expérience

...

10ème et dernière expérience

FIGURE 2.7 – Etapes d'une validation croisée à 10 plis

Ce protocole, dont les étapes pour $n = 10$ sont visualisées en figure 2.7, remédie bien à nos problèmes : en effet, en réalisant un apprentissage sur 9/10 des exemples,

on se prive de peu de données tout en s'assurant de fournir une évaluation peu "biaisée" car elle est en fait une moyenne de plusieurs expériences. Mais réaliser une "validation croisée" demande plus de travail que lorsqu'il suffisait de répartir les exemples en deux sous-ensembles. La variante la plus extrême de la validation croisée, appelée en anglais "leave one out", consiste même à utiliser en entrainement *toutes les données sauf une, qui servira en test*, et donc à répéter cet apprentissage autant de fois qu'il y a de données dans l'ensemble d'apprentissage. Notons pour finir que les différentes variantes de ces protocoles sont implémentées dans le logiciel Weka, déjà évoqué.

3 Les quatre tâches élémentaires

Il est temps désormais de passer en revue les tâches de fouille de textes que nous qualifions d'"élémentaires", parce qu'elles servent de "briques de base" aux autres tâches plus complexes. Nous en identifions quatre : la recherche d'information, la classification, l'annotation et l'extraction d'information. Nous les présenterons dans cet ordre, de la moins spécifique à la plus spécifique d'un point de vue linguistique. Pour chacune d'entre elles, nous explicitons ici leur nature et leur intérêt applicatif, les données sur lesquelles elle peuvent s'appliquer, les types de ressources qu'elles requièrent ou qui peuvent aider à les effectuer ainsi que les mesures utilisées pour évaluer les programmes qui s'y confrontent. Les chapitres suivants permettront de détailler, pour chacune de ces tâches, les techniques employées pour construire les programmes qui les implémentent.

3.1 La Recherche d'Information (RI)

La tâche

Le schéma de la figure 2.8 instancie celui de la figure 2.1 pour la Recherche d'Information (ou RI, ou IR pour Information Retrieval en anglais). Le but de cette tâche est de retrouver un ou plusieurs document(s) pertinent(s) dans un corpus, à l'aide d'une requête plus ou moins informelle.

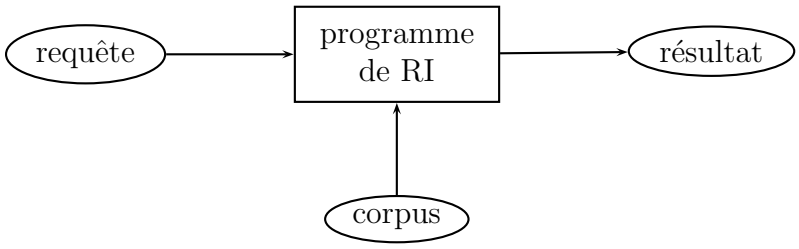


FIGURE 2.8 – Schéma général de la tâche de Recherche d'Information

Détaillons les composantes de ce schéma :
– la ressource "corpus" est fondamentale pour une tâche de RI, qui ne peut exister sans elle. Elle est constituée d'une collection soit de "textes bruts" soit de documents semi-structurés (les formats étant en général homogènes dans

un même corpus). Le Web dans son ensemble, en tant que collection de pages HTML, peut aussi jouer ce rôle.
- dans une tâche de RI, la requête n'est pas formulée dans un langage fortement structuré (de type SQL ou SPARQL), elle prend donc elle aussi la plupart du temps la forme d'un texte brut (qui peut se réduire à quelques mots clés) ou d'un document semi-structuré.
- le résultat du programme peut, lui aussi, prendre différentes formes : un seul ou un ensemble de documents extraits du corpus, soit tous mis "au même niveau" soit présentés en ordre décroissant de pertinence.

La tâche de RI peut aussi s'appliquer à d'autres données que des textes : il existe des systèmes spécialisés dans la recherche d'images, de vidéos ou de morceaux de musique, sur critère de proximité avec une donnée fournie, par exemple ; d'autres encore qui se fondent sur des distances géographiques (pour les téléphones équipés de géolocalisation) ou dans des réseaux sociaux (pour la recherche de connexions possibles), etc. Nous n'aborderons pas ces extensions par la suite, même si les méthodes utilisées sont souvent similaires à celles exploitées pour les textes.

Les domaines d'application

La RI est une tâche très populaire, à laquelle tous les usagers d'Internet font appel quotidiennement dès qu'ils utilisent un *moteur de recherche*. Ceux-ci appartiennent à plusieurs familles : il y a, bien sûr, les moteurs généralistes (Google, Bing, Yahoo !, Baidu en Chine...) qui servent à s'orienter sur l'ensemble du Web, mais la plupart des sites importants (notamment tous les sites marchands ou institutionnels) disposent aussi d'un *moteur interne* permettant de naviguer à l'intérieur de leurs pages. Tout internaute sollicite donc quotidiennement, parfois sans le savoir, plusieurs moteurs de recherche. Des systèmes de recherche sont aussi intégrés au cœur même de chaque ordinateur, pour aider l'utilisateur à fouiller dans son disque dur à la recherche d'un fichier ou d'un mail mal rangé. Enfin, la RI existait déjà avant même l'invention du Web, dans le domaine des "sciences de la documentation". Elle était dans ce cadre cantonnée aux archives et aux bibliothèques, pionnières en matière d'indexation et de requétage de corpus de textes numérisés. Plutôt que de moteurs de recherche, on parlait alors de "logiciels documentaires". Nous verrons que les techniques utilisées pour construire un programme de RI dans ces différents contextes peuvent varier, mais restent assez homogènes.

Les mesures d'évaluation

Pour évaluer la qualité d'un système de RI sur un corpus et une requête donnés, on fait l'hypothèse qu'un humain est toujours capable de dire, pour chaque élément du corpus, s'il est ou non pertinent pour la requête en question. La figure 3.1 représente trois ensembles, caractéristiques de cette situation : l'ensemble D, de *tous les documents du corpus*, l'ensemble P des documents *pertinents pour la requête choisie* et l'ensemble R des documents *retournés par le moteur de recherche* pour cette même requête.

Une autre manière de montrer comment se répartissent l'ensemble des documents de D suivant qu'ils sont pertinents ou non/retournés ou non par le moteur est fournie

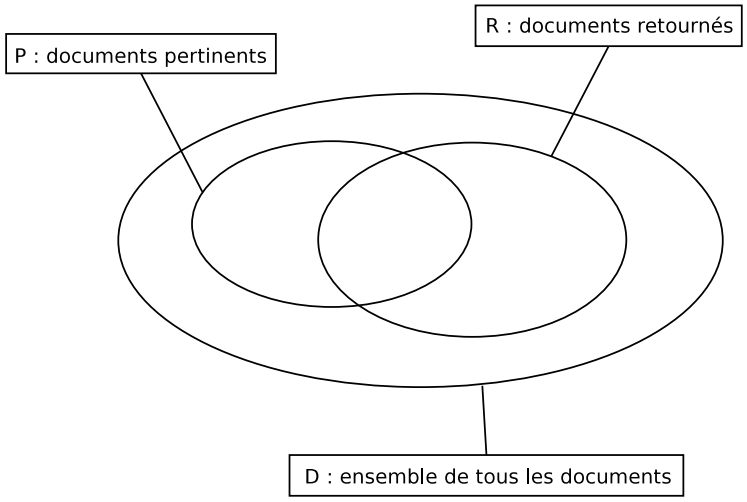


FIGURE 2.9 – Représentation ensembliste des documents pour une requête donnée

par le tableau de la figure 3.1. Dans ce tableau, nous avons utilisé à la fois les notations mathématiques ensemblistes qui font référence à la figure précédente et les termes usuels de la fouille de données :

- le symbole mathématique $\cap$ (respectivement $\cup$) désigne l'usuelle *intersection* (respectivement *union*) ensembliste, tandis que l'opérateur "$-$" est la *différence entre deux ensembles*.
- la terminologie "positifs/négatifs" provient de la fouille de données : dans notre cas, un document est dit "positif" s'il est pertinent, "négatif" sinon. C'est un "vrai positif" s'il a été reconnu à juste titre comme tel par le moteur de recherche", un "faux positif" si le moteur s'est trompé en le désignant comme pertinent (et similairement pour les "vrais négatifs/faux positifs").

	documents retournés	documents non retournés
documents pertinents	vrais positifs : $P \cap R$	faux négatifs : $P - R$
documents non pertinents	faux positifs : $R - P$	vrais négatifs : $D - (P \cup R)$

FIGURE 2.10 – Représentation tabulaire des documents pour une requête donnée

Evidemment, dans une situation idéale (c'est-à-dire avec un moteur parfait), les ensembles P et R devraient coïncider (on aurait alors $P \cap R = P = R = P \cup R$), mais c'est rarement le cas. Les mesures d'évaluation du système visent, précisément, à quantifier cet écart entre P et R. Elles prennent la forme suivante :

- la *précision* p mesure la proportion de documents pertinents parmi ceux qui sont retournés (où nb désigne le nombre d'éléments d'un ensemble, VP le nombre de "vrais positifs" et FP le nombre de "faux positifs", FN celui de "faux négatifs") :
  $p = \frac{nb(P \cap R)}{nb(R)} = \frac{VP}{VP+FP}$
- le *rappel* r mesure la proportion de documents pertinents retournés parmi ceux figurant dans le corpus (mêmes notations que précédemment) :

$r = \frac{nb(P \cap R)}{nb(P)} = \frac{VP}{VP+FN}$

- la F-mesure F est la *moyenne harmonique* de p et r : $F = 2.\frac{p.r}{p+r}$
  On définit parfois une F-mesure plus générale F_β qui pondère différemment p et r avec un paramètre $\beta \in [0, 1]$: $F_\beta = \frac{(1+\beta^2).p.r}{\beta^2.p+r}$. La F-mesure (parfois aussi appelée pour cela F_1-mesure) correspond à la valeur $\beta = 1$.
- pour compléter ces mesures, on peut aussi signaler le silence s et le bruit b, qui se calculent comme suit : $s = \frac{nb(P-R)}{nb(P)} = \frac{FN}{VP+FN}$, $b = \frac{nb(R-P)}{nb(R)} = \frac{FP}{VP+FP}$.
  On a les relations élémentaires suivantes : $p + b = 1$ et $r + s = 1$.

Plusieurs remarques s'imposent pour bien comprendre l'intérêt de ces différentes mesures :

- elles sont toutes comprises entre 0 et 1. La moyenne harmonique a l'avantage de mettre la F-mesure à 0 dès que soit la précision soit le rappel s'annule, obligeant ainsi à ne négliger aucune de ces deux mesures. Pour que la F-mesure soit égale à 1, il faut que $p = r = 1$. Si $p = r$, la formule devient $2.\frac{p^2}{2p} = p$: la moyenne harmonique de deux valeurs identiques se confond avec cette valeur.
- face au résultat d'un moteur de recherche, un utilisateur n'a accès qu'à l'ensemble R et ne peut donc évaluer que la précision du moteur. En effet, pour calculer le rappel, il faudrait connaître l'existence dans le corpus des documents pertinents oubliés par le moteur, ce qui est en général difficile...
- En fait, il n'est pas difficile de construire un moteur de recherche qui se focaliserait uniquement sur la précision ou sur le rappel, en négligeant l'autre mesure. Le vrai challenge consiste donc à obtenir une bonne performance pour ces *deux mesures* simultanément, ce qui revient à se concentrer sur la F-mesure.

Nous nous restreignons ici à supposer que, pour une requête donnée, chaque document est soit pertinent soit non pertinent. En réalité, la notion de pertinence est graduelle et on attend plus d'un moteur de recherche qu'il *ordonne du plus au moins pertinent les documents* plutôt qu'il les classe suivant un simple critère binaire. C'est particulièrement sensible pour les corpus volumineux (notamment le Web !), où seuls les premiers résultats proposés sont réellement consultés par les utilisateurs. Dans ce cas, le résultat attendu d'un système de recherche d'information est un *classement*, qui doit être comparé à un classement de référence. Des mesures d'évaluation spécifiques sont alors requises, que nous ne détaillerons pas ici. Cela pose de nouveaux problèmes difficiles, car un classement de référence est évidemment difficile à constituer. De manière générale, l'évaluation de ces systèmes reste un thème de recherche actif.

3.2 La Classification

La tâche

La classification est la tâche phare de la fouille de données, pour laquelle une multitude de programmes sont implémentés dans le logiciel Weka. Elle consiste à associer une "classe" à chaque donnée d'entrée, comme l'illustre la figure 2.11.

Détaillons encore ces composantes :

- la donnée à classer est en principe de type "texte brut" ou "document semi-structuré". Toutefois, comme nous le verrons lors de la présentation des pro-

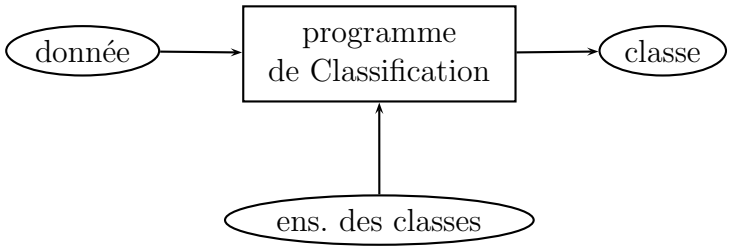


FIGURE 2.11 – Schéma général de la tâche de Classification

grammes existants, cette tâche a été abondamment étudiée pour les données tabulaires de la fouille de données, et la démarche employée pour la résoudre consistera presque systématiquement à transformer les données textuelles en tableaux. Cette transformation fera l'objet de la partie 5.

- l'ensemble des classes possibles est fini et connu au moment où le programme de classification est sollicité, c'est pourquoi nous le faisons figurer en tant que ressource. Toutefois, si le programme en question est issu d'un apprentissage automatique (cf. schéma de la figure 2.6), plusieurs situations sont possibles suivant que les classes sont définies à l'avance et présentes (ou non) dans les exemples d'apprentissage. Nous y reviendrons...
- la classe résultat permet de caractériser la donnée à laquelle elle est associée, à la ranger dans un ensemble existant. Cette classe est en général unique, le programme effectue donc une *partition* de l'ensemble des données possibles. Dans le cas où seules deux classes sont possibles, on parle d'une classification binaire.

Comme la recherche d'information, la classification peut s'appliquer à toutes sortes de données, et pas seulement aux textes : la classification des images, des vidéos, des musiques... de toute donnée, de manière générale, qu'il est possible de décrire à l'aide d'attributs, donne lieu à de multiples et florissantes applications. Nous avons évoqué en introduction que la fouille de données était née dans les domaines des banques, des assurances, du marketing et de la médecine, pour aider à déterminer automatiquement la solvabilité d'un client, l'adéquation d'un produit ou encore l'efficacité d'un médicament... Tous ces objectifs peuvent être reformulés comme des tâches de classification. Le tableau de la figure 2.2 illustre aussi une telle tâche : les cinq premières colonnes servent à décrire une situation météorologique, la dernière ("play", qui vaut "yes" ou "no") est l'étiquette (la classe) à prédire par le programme.

Les domaines d'application

La classification est une tâche qui donne lieu à une multitude d'applications. L'une d'elles est présente dans la plupart des gestionnaires de courriers électroniques : c'est la reconnaissance automatique des "spams", ces messages indésirables qui encombrent toutes les boîtes aux lettres. Cette fonctionnalité est généralement implémentée en mode "apprentissage automatique", l'utilisateur devant, au début, signaler ce qu'il considère comme indésirable afin que le programme apprenne progressivement à les reconnaître lui-même. Des gestionnaires d'e-mails "intelligents" proposent même de

"deviner" le dossier de rangement d'un nouveau courrier, en se fondant aussi sur les exemples déjà triés. La liste des classes, dans ce cas, coïncide avec celle des dossiers de rangement possibles.

La "fouille d'opinion" est un un autre domaine d'application en plein essor. Elle vise à identifier les polarités (positives ou négatives) véhiculées par les textes (par exemple les commentaires d'internautes sur les sites marchands ou de loisir), généralement à des fins marketing, pour mesurer la "e-réputation" d'une société, d'une personne, d'une marque, d'un produit... Cet objectif oblige en général à procéder à plusieurs étapes de classification : d'abord pour séparer les textes qui se veulent "objectifs" ou "neutres" de ceux qui sont porteurs d'opinion, ensuite pour classer ces derniers en "positif" ou "négatif", ou suivant une échelle plus fine.

De nombreux autres exemples peuvent être évoqués. Il ne fait aucun doute que les organismes nationaux de surveillance des échanges (voir les scandales récents autour de "PRISM") appliquent des méthodes de classification pour identifier les messages potentiellement indicateurs de menaces. Plus pacifiquement, des challenges nationaux ou internationaux portent souvent sur des tâches de classification. Le "Défi Fouille de Textes" [2] est ainsi une compétition annuelle organisée conjointement avec la conférence francophone de traitement automatique des langues naturelles (TALN), qui en a proposé plusieurs variantes : la reconnaissance de l'auteur d'un texte politique (2005), de la rubrique dont relève un article de journaux (2008), de sa date de publication (2010), de la variante linguistique dont il est issu (2011), de sa qualité littéraire (2014)...

Les mesures d'évaluation

Comme pour la recherche d'information, l'évaluation d'un programme de classification se fait toujours sur un certain nombre de données pour lesquelles la classe "correcte" est supposée connue. Pour mesurer l'écart entre le résultat du programme et la bonne réponse, on utilise un outil clé appelé "matrice de confusion", qui comptabilise, pour chaque classe, toutes les données bien ou mal rangées. La figure 2.12 montre un exemple de telle matrice, pour un problème à trois classes possibles notées a, b etc.

classé en \ vraie classe	a	b	c
a	16	0	0
b	0	19	1
c	0	2	15

FIGURE 2.12 – Une matrice de confusion pour un problème à trois classes

On lit dans ce tableau que l'expérience a porté sur 53 données en tout (somme des valeurs de toutes les cases), parmi lesquelles 16 étaient de la classe a (somme des valeurs de la première ligne), 20 de la classe b (somme des valeurs de la deuxième

2. DEFT : http ://deft.limsi.fr

ligne)et 17 de la classe c (somme des valeurs de la troisième ligne). Le programme, lui, a classé 16 données en a (somme des valeurs de la première colonne), 21 en b (somme des valeurs de la deuxième colonne) et 16 en c (somme des valeurs de la troisième colonne). Les cases sur la diagonale allant d'en haut à gauche au bas à droite comptabilisent le nombre de fois où la vraie classe coïncide avec la sortie du programme, les cases hors diagonale sont des erreurs. Dans notre exemple, le programme n'a fait aucune erreur sur la classe a, mais a parfois confondu les classes b et c. Les couleurs du tableau montrent comment retrouver les mesures de précision p et rappel r (et donc F-mesure) introduites en partie 3.1 pour cette classe a :

- les "vrais positifs" VP pour a sont comptés dans la case verte ;
- les "vrais négatifs" VN pour a sont comptés dans les cases bleues ;
- les "faux négatifs" FN pour a sont comptés dans les cases jaunes ;
- les "faux positifs" FP pour a sont comptés dans les cases orange.

Les formules $p = \frac{VP}{VP+FP}$ et $r = \frac{VP}{VP+FN}$ s'appliquent alors identiquement, et leur moyenne harmonique F-mesure également. On appelle aussi parfois le rappel le "taux de vrais positifs" ou la sensibilité, et symétriquement on introduit le "taux de faux positifs" $fp = \frac{FP}{FP+VN}$. La spécificité s est $s = \frac{VN}{FP+VN} = 1 - fp$. Enfin, on appelle "exactitude" e (ou "accuracy", en gardant le terme anglais) la proportion de bons classements relativement à a : $e = \frac{VP+VN}{VP+VN+FP+FN}$. Bien sûr, les définitions des ensembles VP, VN, FN et FP intervenant dans ces mesures doivent être adaptées pour chaque classe, dont la précision, le rappel (donc la F-mesure) et l'exactitude se calculent indépendamment.

On peut aussi définir des mesures globales d'évaluation. Ainsi, l'exactitude globale du programme est la proportion de données bien classées, qui se calcule en divisant la somme des contenus de la diagonale par le nombre total de données. Pour moyenner les précision, rappel et F-mesure des différentes classes, il y a deux façons de procéder :

- soit on calcule la moyenne simple des différentes classes, sans pondération particulière : on obtient ainsi la macro-average ;
- soit on pondère les évaluations de chaque classe par la proportion de données qui appartiennent à cette classe : on obtient alors la micro-average.

La micro-average tient compte de la répartition des données, alors que la macro-average donne autant d'importance à chaque classe, indépendamment de ses effectifs.

3.3 L'Annotation

La tâche

L'annotation (ou l'étiquetage), telle qu'elle sera définie ici, est une tâche plus spécifiquement *linguistique* que les précédentes, au sens où elle ne s'applique pas aux données tabulaires et ne relève donc pas de la fouille de données. La figure 2.13 la présente globalement.

Pour bien comprendre en quoi elle se distingue de la tâche de classification, il convient de préciser les points suivants :

- la donnée est exclusivement un texte brut ou un document semi-structuré non transformé en tableau : elle est donc composée d'unités respectant au moins une *relation d'ordre*.

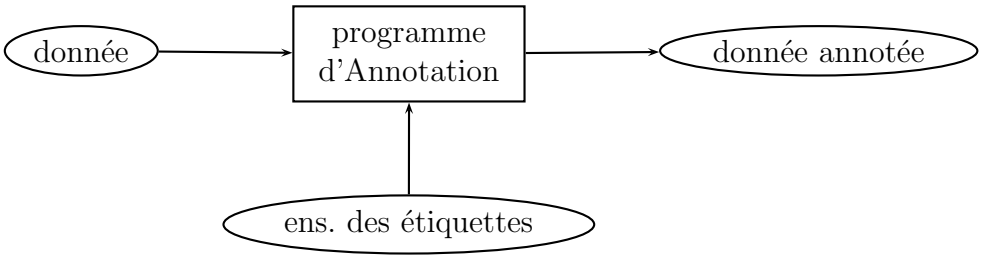


FIGURE 2.13 – Schéma général de la tâche d'Annotation

- l'ensemble des étiquettes possibles est fini et connu à l'avance au moment où le programme est appelé.
- le résultat est la donnée initiale dans laquelle *chaque unité* est associée à une étiquette prise dans l'ensemble des étiquettes possibles (et non une seule étiquette pour l'ensemble de la donnée comme en classification). La relation d'ordre entre les unités d'origine se propage donc en quelque sorte sur les étiquettes figurant dans le résultat du programme (nous en verrons plusieurs exemples ci-dessous).

L'annotation peut aussi s'appliquer à d'autres données structurées que les textes : on peut ainsi annoter des séquences audio ou vidéo, ou des bases de données XML par exemple. On parlera d'annotation quand la structure de la donnée d'origine se trouve "reproduite" sur les étiquettes ajoutées par le programme.

Les domaines d'application

L'annotation est une tâche très courante en linguistique. Mais, au lieu d'opérer sur des *textes bruts* (au sens de séquences de caractères), elle s'applique généralement à des *textes segmentés* en unités plus grandes. Le découpage le plus courant est celui dans lequel les unités de base sont des *tokens* (mots, chiffres ou ponctuations). Ceci requiert bien sûr un pré-traitement qui assure cette segmentation, nous y reviendrons en partie 5. Un texte est alors une *séquence de tokens* qui peut être annotée par une *séquence d'étiquettes*. Annoter une telle séquence revient à "stabiloter" chacune de ses unités dans une certaine couleur, chaque couleur possible correspondant à une étiquette distincte.

Les étiquettes les plus traditionnelles pour annoter un texte brut sont appelées "parties du discours" ("part of speech" abrégé en POS en anglais) : elles caractérisent la nature morpho-syntaxique de chaque token. Par exemple, la phrase "Le petit chat est mort." est constitué de 6 tokens et une séquence d'annotations possible est : DET ADJ NC V ADJ PONCT (où DET désigne les déterminants, ADJ les adjectifs, NC les noms communs, V les verbes et PONCT les ponctuations). Bien entendu, l'annotation "correcte" dépend de l'ensemble des étiquettes autorisées. D'autres découpages linguistiques peuvent être traités comme des annotations : ainsi, la segmentation en "chunks" (constituants non récursifs, c'est-à-dire non emboîtés les uns dans les autres) d'une phrase peut s'interpréter comme l'annotation d'une séquence de *séparateurs de tokens* (c'est-à-dire de ce qui sépare deux tokens consécutifs). La phrase précédente se segmente en chunks de la façon suivante : (Le petit chat) (est)

(mort). Or, un tel parenthésage correspond exactement à étiqueter les "espaces entre les tokens" (en en comptant un au début et un à la fin de chaque séquence) soit avec "(", soit avec ")", soit avec ")(" soit enfin avec une étiquette "vierge" signifiant "aucune frontière de chunk". Segmenter une séquence de tokens en chunks revient, dans ce cas, à annoter la séquence des séparateurs correspondante.

La traduction automatique, telle qu'elle est réalisée actuellement, est un autre domaine où l'annotation est souvent mise à contribution. Une des étapes fondamentales d'un programme de traduction automatique statistique est en effet *l'alignement de séquences*. La figure 2.14 montre un tableau d'alignement entre deux séquences qui sont les traductions l'une de l'autre entre le français et l'anglais. Plutôt que de reconstituer un tel tableau, les programmes d'alignement cherchent à annoter chacune des séquences avec les *positions des traductions des mots dans l'autre séquence*, tel que montré sous le tableau. Les deux séquences annotées visualisent en quelque sorte les *projections* des cases cochées du tableau suivant ses deux dimensions (horizontale et verticale).

	J'	aime	le	chocolat
I	X			
like		X		
chocolate				X

J'_1	$aime_2$	le_3	$chocolat_4$	I_1	$like_2$	$chocolate_3$
1	2	-	3	1	2	4

FIGURE 2.14 – Un alignement bilingue et les deux annotations correspondantes

Une autre application intéressante est la division d'un texte long en *zones thématiques*. Dans ce cas, les unités du texte sont non plus ses tokens mais ses *phrases*. On suppose que chaque phrase ne peut appartenir qu'à une certaine classe parmi un ensemble pré-défini (par exemple : introduction, paragraphe, conclusion...) et on cherche à annoter la séquence des phrases par une séquence de telles étiquettes. Le même genre de traitement peut s'appliquer à une page HTML considérée elle aussi comme une *séquence d'unités*, soit zones de textes soit balises. L'annotation de la page Web peut être destinée par exemple à distinguer ce qui, dans cette page, donne lieu à un titre, un menu de navigation, un en-tête, un pied-de-page, une image, une zone de texte, etc. pour en extraire le vrai contenu informationnel tout en écartant ses éléments parasites (publicités, etc.).

L'annotation d'un document HTML ou XML peut aussi s'appuyer sur sa structure arborescente : dans ce cas, le résultat du processus est également un arbre. La figure 3.3 montre un arbre d'analyse syntaxique qui a été enrichi par l'annotation des fonctions de ses constituants (PRED pour "prédicat", SUJ pour "sujet", OBJ pour "objet", MOD pour "modifieur"). A condition d'ajouter une étiquette "neutre" aux nœuds non annotés, les étiquettes en rouge constituent un arbre de même forme que l'arbre initial.

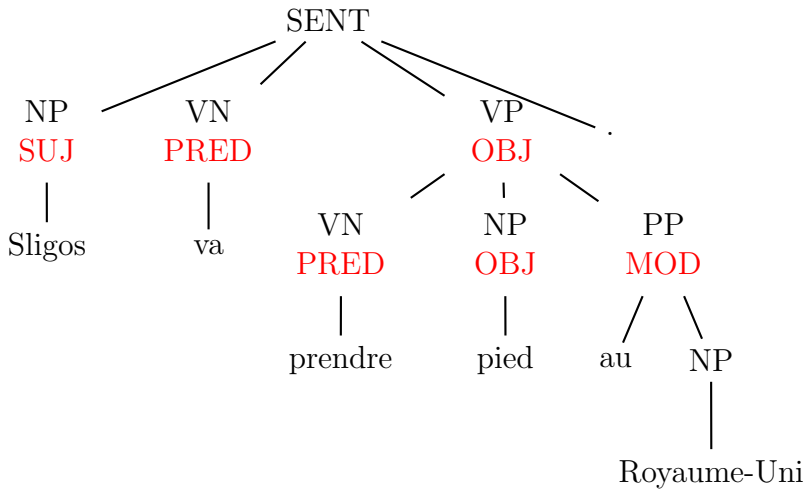


FIGURE 2.15 – Etiquetage fonctionnel (en rouge) d'un arbre d'analyse syntaxique

Les mesures d'évaluation

L'annotation s'évalue avec les mêmes mesures que la classification (cf. partie 3.2). Toutefois, comme cette tâche consiste à associer un *ensemble d'étiquettes* à un *ensemble structuré d'unités*, on a tendance à y privilégier les mesure globales. Ce sera l'occasion d'illustrer par un petit calcul une propriété intéressante : quand chaque unité reçoit exactement une étiquette, alors les micro-averages de la précision, du rappel et de la F-mesure de l'ensemble de l'étiquetage sont toujours égales et coïncident avec son exactitude. Illustrons ce calcul simple par l'exemple d'une annotation avec trois étiquettes a, b et c et reprenons en figure 2.16 la matrice de confusion de la figure 2.12 en y nommant les cases :

classé en \ vraie classe	a	b	c
a	aa	ab	ac
b	ba	bb	bc
c	ca	cb	cc

FIGURE 2.16 – Une matrice de confusion pour une annotation à trois étiquettes

Notons $l_a = aa + ab + ac$ la somme des éléments de la première ligne (et similairement l_b et l_c pour les deuxième et troisième lignes), $c_a = aa + ba + ca$ la somme des éléments de la première colonne (et similairement pour c_b et c_c) et S la somme totale des éléments du tableau ($S = l_a + l_b + l_c = c_a + c_b + c_c$). Les précisions (respectivement p_a, p_b et p_c) et rappels (respectivement r_a, r_b et r_c) des trois étiquettes a, b et c se calculent alors comme suit :
$p_a = \frac{aa}{c_a}$, $p_b = \frac{bb}{c_b}$, $p_c = \frac{cc}{c_c}$, $r_a = \frac{aa}{l_a}$, $r_b = \frac{bb}{l_b}$ et $r_c = \frac{cc}{l_c}$.
Pour obtenir la moyenne moy(p), pondérée par les proportions de données étiquetées dans chacune des classes, des précisions p_a, p_b et p_c, on a donc :

$$
\begin{aligned}
moy(p) &= \frac{p_a.c_a+p_b.c_b+p_c.c_c}{c_a+c_b+c_c} \\
&= \frac{p_a.c_a+p_b.c_b+p_c.c_c}{S} \\
&= p_a.\frac{c_a}{S} + p_b.\frac{c_b}{S} + p_c.\frac{p_c}{S} \\
&= \frac{aa}{c_a}.\frac{c_a}{S} + \frac{bb}{c_b}.\frac{c_b}{S} + \frac{cc}{c_c}.\frac{c_c}{S} \\
&= \frac{aa+bb+cc}{S}
\end{aligned}
$$

Similairement, pour la moyenne moy(r), pondérée par les proportions des données réellement présentes dans chacune des classes, des rappels r_a, r_b et r_c, on a :

$$
\begin{aligned}
moy(r) &= \frac{r_a.l_a+r_b.l_b+r_c.l_c}{l_a+l_b+l_c} \\
&= \frac{r_a.l_a+r_b.l_b+r_c.l_c}{S} \\
&= r_a.\frac{l_a}{S} + r_b.\frac{l_b}{S} + r_c.\frac{l_c}{S} \\
&= \frac{aa}{l_a}.\frac{l_a}{S} + \frac{bb}{l_b}.\frac{l_b}{S} + = \frac{cc}{l_c}.\frac{l_c}{S} \\
&= \frac{aa+bb+cc}{S}
\end{aligned}
$$

Dans les deux cas, on obtient finalement la valeur de l'exactitude globale ! Et la F-mesure de deux valeurs identiques redonne cette même valeur, et aboutit donc au même résultat. Ce calcul se généralise bien sûr à un nombre quelconque d'étiquettes. Cette propriété est en fait compréhensible : en effet, dès qu'une donnée est mal étiquetée (par exemple, un b est mis à la place d'un a), cela constitue à la fois une erreur de précision et de rappel (dans notre exemple, l'étiquette erronée pénalise à la fois la précision de b et le rappel de a). Les moyennes pondérées des rappels et des précisons comptent donc en fait la même chose : la proportion d'étiquetage correct, c'est-à-dire l'exactitude. C'est souvent cette unique valeur qui est fournie pour évaluer la qualité d'un étiquetage.

3.4 L'Extraction d'Information (EI)

La tâche

L'Extraction d'Information (EI ou Information Extraction en anglais, abrégé en IE) est décrite par le schéma de la figure 2.17. Le but de cette tâche, qui relève de l'ingénierie linguistique, est d'extraire automatiquement de documents textuels des informations factuelles servant à remplir les champs d'un formulaire pré-défini.

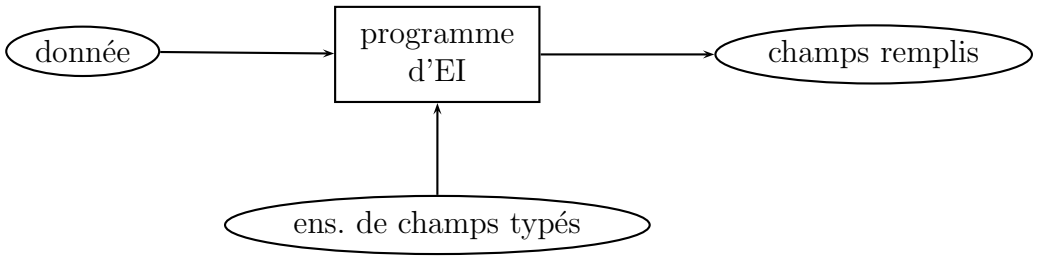


FIGURE 2.17 – Schéma général de la tâche d'Extraction d'Information

Détaillons les composantes de cette tâche :

– la donnée est exclusivement un texte brut ou un document semi-structuré non transformé en tableau. L'objectif de la tâche est précisément de transformer cette donnée en une sorte de tableau (ou une base de données) contenant des éléments factuels précis reflétant son contenu sémantique.

– l'ensemble des champs typés apparaissant en ressource spécifie la nature des informations qui doivent être extraites de la donnée (nous en donnerons des exemples par la suite). On peut aussi le voir comme la définition d'un formulaire avec "cases vides à remplir".
– le résultat du programme est une instance remplie de l'ensemble des champs typés (ou encore une version "cases remplies" du formulaire).
– Les anglo-saxons utilisent parfois le terme de "wrapper" (du verbe anglais "to wrapp" : envelopper) pour désigner un programme d'extraction d'information, notamment (mais pas exclusivement) quand il opère sur des données semi-structurées.

Cette définition cache en réalité plusieurs variantes possibles : en effet, suivant les cas, les champs à remplir sont obligatoires ou facultatifs, ils peuvent recevoir une unique valeur pour chaque donnée ou plusieurs, leur type peut être strict ou relativement relâché (il y a par exemple plusieurs manières distinctes de donner une "date"), etc. S'il y a n champs à remplir, on parle d'extraction n-naire. Identifier tous les noms propres de personnes figurant dans un texte ou une page Web est ainsi un problème unaire multiple (un seul champs, de multiples instances possibles) tandis que remplir une et une seule fois les n champs d'un formulaire traduisant une petite annonce de vente de voiture (marque, couleur, âge, kilométrage, prix, etc.) est un problème n-aire unique. Certains problèmes sont hybrides, au sens où la multiplicité des informations à extraire varie d'une donnée à une autre (le nombre d'auteurs d'un article scientifique dépend de chaque article par exemple). Cette variabilité entraine des difficultés dans l'évaluation.

Les domaines d'application

L'extraction d'information est née d'un challenge organisé lors des conférences MUC ("Message Understanding Conference") qui se sont déroulées entre 1987 et 1998 aux Etats Unis, sous l'impulsion de la Darpa [3], l'agence de recherche du département de la Défense américain. Les participants se voyaient confier des corpus et leurs programmes étaient comparés en fonction de leur capacité à remplir à partir de chaque texte les champs d'un formulaire prédéfini. Par exemple, en 1992, il s'agissait d'extraire de dépêches d'agences de presse décrivant des attentats des informations telles que : date, lieu, auteur présumé ou revendiqué, nombre de victimes, etc. On mesure aisément l'intérêt stratégique de ce genre d'applications... Les conférences MUC ont disparu en laissant la place à d'autres, mais l'intérêt porté à la tâche d'extraction d'information n'a depuis lors cessé de grandir.

Une de ses applications phare actuelle est la reconnaissance des *entités nommées*, ces mots ou groupes de mots qui identifient soit des noms propres (désignant des personnes, des lieux ou des organisations) soit des quantités mesurables (exprimant notamment des dates, des valeurs numériques ou monétaires). Leur particularité est de référer directement à des "entités" du monde et de ne pas être présentes dans les dictionnaires de la langue commune. Ce sont pourtant elles qui véhiculent l'essentiel du contenu informationnel de certains textes : c'est le cas pour les "petites annonces" ou les dépêches d'agences de presse évoquées précédemment, et de manière générale

3. Defense Advanced research Projects Agency

pour la majorité des articles de journaux. Les fameux "cinq W" du journalisme anglo-saxon ("who did what, where and when, and why", c'est-à-dire "qui a fait quoi, où, quand et pourquoi" en français) attendent, pour la plupart, une réponse en forme d'entité nommée. La démarche de veille informationnelle, dans quelque domaine que ce soit, ou encore celle d'anonymisation de documents, passent également par la reconnaissance des noms propres et des dates présents dans les textes ou les pages HTML ou XML. L'analyse automatique de CV, ou de sites marchands pour faire de la comparaison de prix, sont encore d'autre applications potentiellement très utiles de l'extraction d'information.

On peut citer aussi le domaine de la bibliométrie, qui passe notamment par l'analyse automatique de la partie bibliographique des articles scientifiques, en particulier pour comptabiliser leurs citations. Ce service, initialement proposé par le site web CiteSeer (spécialisé dans le référencement des articles d'informatique, non maintenu à ce jour), est actuellement assuré par Google Scholar [4]. C'est devenu un enjeu important des politiques de recherche publique et privée, puisque tous les chercheurs sont désormais évalués selon des indicateurs fournis par ce genre de programmes.

L'extraction d'information, on le voit, est une tâche potentiellement très riche et très utile. Elle vise en quelque sorte à combler le fossé qui sépare la façon dont les humains appréhendent et intègrent l'information, idéalement présentée pour eux sous forme de textes, et celle dont les ordinateurs la traitent, c'est-à-dire ramenée à des valeurs dans des cases, dans des bases des données. McCallum (un des chercheurs de référence du domaine) parle à son sujet d'une forme de "distillation de l'information". C'est une tâche éminemment difficile, elle résume presque à elle seule la problématique de l'ingénierie linguistique et est certainement, de ce fait, promise à de nombreux futurs travaux et développements.

Les mesures d'évaluation

Comme précédemment, les résultats de programmes d'extraction d'information sont comparés à des résultats de référence validés manuellement. Les différents champs à remplir par le programme peuvent être de nature différente et sont donc en général évalués indépendamment les uns des autres. La précision, le rappel et la F-mesure, introduits en partie 3.1 sont les plus utilisés pour cela, en remplaçant bien sûr les documents par les "valeurs extraites" pour un champ donné. Par exemple, si la tâche consiste à extraire tous les noms propres d'un texte, alors pour un programme donné :

- sa précision s'obtient en divisant le nombre de noms propres corrects trouvés par le nombre total de noms propres extraits ;
- son rappel s'obtient en divisant le nombre de noms propres corrects trouvés par le nombre total de noms propres qui auraient dû être extraits.

La nature factuelle des informations à extraire rend toutefois cette évaluation parfois délicate. Dans le cas des entités nommées, il y a en effet souvent plusieurs manières possibles de désigner une même entité. Une extraction doit-elle être correcte au caractère près ou est-ce l'identité sémantique qui doit prévaloir (et dans ce cas, comment la mesurer automatiquement ?). Par exemple, un nom propre de personne

4. http ://scholar.google.fr

précédé d'un "M." ou "Mme", un nom de pays introduit par un article ("la France") est-il incorrect si la valeur de référence n'inclut pas cette particule ? En cas de citations multiples dans un document d'une même entité, doit-on imposer de les trouver toutes ou une seule occurrence suffit-elle ? Une valeur de champ vide doit-elle être systématiquement considérée comme fausse ? Nous ne trancherons pas ces questions ici, elles font encore l'objet de débats dans la communauté scientifique, et ne peuvent être traitées qu'au cas par cas, en fonction de la tâche spécifique considérée.

4 Relations entre tâches

Il est important de distinguer les tâches les unes des autres, parce que les programmes qui seront décrits dans les chapitres suivants sont spécialisés dans la réalisation d'une et une seule d'entre elles. Pour autant, les quatre tâches élémentaires que nous venons de présenter ne sont pas complètement indépendantes les unes des autres. Tout d'abord, il est souvent possible, via une reformulation du problème ou un "codage" astucieux des données, d'en transformer une en une autre, et de permettre par la même occasion d'employer un programme prévu pour accomplir une certaine tâche dans un autre but. C'est ce que nous évoquons dans la première section de cette partie. Dans un deuxième temps, nous montrons que, pour réaliser des traitements moins "élémentaires" que ceux cités jusqu'à présent, il peut suffire de les décomposer en sous-problèmes correspondant à nos quatre tâches de référence, et d'utiliser des programmes les résolvant les uns après les autres. Jouer avec les entrées/sorties d'une tâche, les reformuler et les enchaîner, font partie des compétences indispensables aux usagers de la fouille de textes.

4.1 Reformulations entre tâches

La RI comme une suite de classifications

Une première reformulation possible simple entre tâches consiste à considérer la recherche d'information (RI) comme une suite de classifications. En effet, sélectionner un ensemble de documents parmi ceux du corpus en fonction d'une requête revient bien à les *classer* soit comme *pertinent* soit comme *non pertinent* relativement à cette requête. Cela ne rend pas nécessairement le problème plus facile à résoudre parce que chaque requête nouvelle oblige à créer un nouveau classifieur, et à l'utiliser sur intégralité des documents du corpus pour déterminer ceux qui la satisfont. Comme l'illustre la figure 2.18, l'idée sous-jacente à cette reformulation est en quelque sorte d'inverser les rôles de la donnée et de la ressource entre les deux tâches : la requête, qui est la donnée de la tâche de RI, devient la ressource de la tâche de classification (puisque c'est elle qui sert de critère pour caractériser les deux classes "pertinent/non pertinent") tandis que les éléments du corpus, qui sont les ressources de la tâche de RI, deviennent les données sur lesquelles opère le programme de classification. Notons que, pour obtenir *une* réponse du programme de RI, il faut appliquer autant de fois le programme de classification qu'il y a d'éléments dans le corpus. Remarquons aussi que l'ordre dans lequel les documents sont successivement classés n'a

aucune importance et que la réponse obtenue ainsi est un ensemble *non ordonné* de documents, puisqu'on s'est ramené à une classification *binaire*.

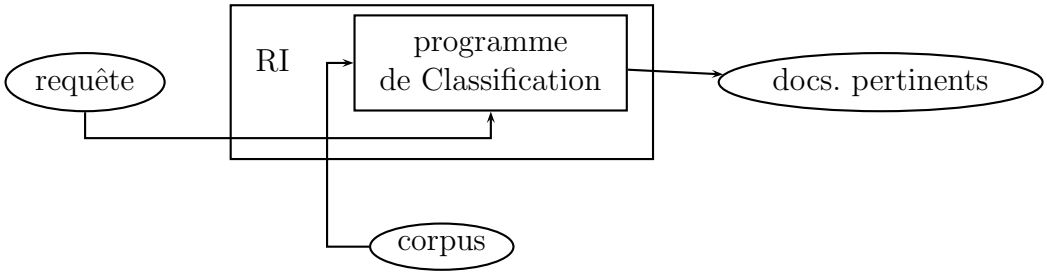


FIGURE 2.18 – Comment la tâche de RI est transformée en une tâche de Classification

L'annotation comme une séquence de classifications

Un autre lien relativement "évident" entre les tâches consiste à voir une annotation comme une succession (éventuellement ordonnée) de classifications, chacune portant non pas sur l'ensemble du document à annoter mais sur une de ses unités qui doivent recevoir une étiquette. Par exemple, associer à chaque "token" d'un texte brut une catégorie morpho-syntaxique (parmi PONCT, NC, ADJ, DET, V... comme illustré en partie 3.3) revient bien à *classer* chacun d'entre eux. Or, dans le cadre d'une tâche d'annotation, rappelons que la donnée d'entrée est un texte brut ou un document semi-structuré, c'est-à-dire un ensemble d'unités doté d'au moins une relation d'ordre. Il est donc simple et naturel d'associer un *sens de parcours* à ces éléments : le sens de lecture pour les textes bruts, un certain type de chemin dans les arbres (en général de la racine aux feuilles, de gauche à droite, en profondeur d'abord) pour les documents semi-structurés. L'intérêt de ce sens de parcours est que les étiquettes associées aux unités déjà parcourues précédemment peuvent être prises en compte dans la classification d'une nouveau unité. Contrairement à la partie précédente, où la classification de chaque document en pertinent/non pertinent se faisait dans un ordre quelconque, les classifications successives d'unités ne sont donc pas nécessairement indépendantes les unes des autres : chaque résultat obtenu peut aider à obtenir les suivants (c'est en tout cas une hypothèse linguistiquement intéressante dans le cas de l'annotation morpho-syntaxique). La figure 2.19 illustre comment une séquence de classifications ordonnée par le sens de lecture annote progressivement la phrase de l'exemple 3.3.

Le tableau de la figure 2.20 explicite, lui, le format des informations qui peuvent être fournies à un programme d'apprentissage automatique de classification (du type de ceux de Weka, cf. figure 2.2) que l'on souhaite utiliser pour réaliser cette annotation, en tenant compte du sens de parcours du texte. Dans ce tableau, le symbole "-" signifie "valeur manquante" et la dernière colonne est celle du résultat attendu.

Il est important de comprendre qu'un classifieur appris de façon supervisée sur ce genre de données sera applicable à un nouveau texte si on l'utilise dans le même sens de parcours que celui qui a permis de collecter les données d'apprentissage (ici, de gauche à droite). Sur une nouvelle phrase, en effet, l'ensemble des tokens est disponible (on peut donc remplir automatiquement les colonnes "token précédent" et "to-

phrase initiale	Le	petit	chat	est	mort	.
1ère classification	DET					
2ème classification	DET	ADJ				
3ème classification	DET	ADJ	NC			
4ème classification	DET	ADJ	NC	V		
5ème classification	DET	ADJ	NC	V	ADJ	
6ème classification	DET	ADJ	NC	V	ADJ	PONCT

FIGURE 2.19 – Comment une annotation se ramène à une succession de classifications

token	position	token précédent	token suivant	étiq. précédente	étiquette
Le	1	-	petit	-	DET
petit	2	Le	chat	DET	ADJ
chat	3	petit	est	ADJ	NC
est	4	chat	mort	NC	V
mort	5	est	.	V	ADJ
.	6	mort	-	ADJ	PONCT

FIGURE 2.20 – Tableau de données/résultats pour la classification précédente

ken suivant") et l'étiquette précédente l'est aussi puisque, en suivant le sens de parcours, on vient juste avant d'y appliquer le classifieur. Son résultat sur un token est donc immédiatement intégré dans la donnée d'entrée pour la recherche de l'étiquette du token suivant. On pourrait d'ailleurs aussi prendre en compte l'étiquette "deux positions avant" celle à trouver, ou n'importe quelle valeur d'étiquette, du moment qu'elle est associée à un token qui précède celui en train d'être traité. En revanche, on ne peut pas avoir de colonne "étiquette suivante" car elle n'a pas encore été trouvée par le classifieur. Cette stratégie du "sens de parcours" permet de réaliser une annotation en général plus fiable que celle consistant à classer chaque token indépendamment les uns des autres, mais elle prend aussi le risque, en cas de mauvais diagnostic à une certaine étape, de propager des erreurs d'étiquetage de token en token.

L'EI comme une annotation

La façon actuellement la plus courante et la plus efficace d'aborder la tâche d'extraction d'information (EI) est de la traiter comme une tâche d'annotation. En effet, extraire des informations factuelles de textes (ou de documents semi-structurés) peut se ramener facilement à annoter dans ces textes les positions des unités porteuses de cette information. Prenons l'exemple d'articles de journaux dont on souhaite extraire des informations telles que : nature de l'événement évoqué, date, lieu... et d'une phrase comme "En 2016, les Jeux Olympiques auront lieu à Rio de Janeiro". On peut désigner simplement la position et le type des informations factuelles à extraire en annotant la phrase de la manière suivante :

En	2016	,	les	Jeux	Olympiques	auront	lieu	à	Rio	de	Janeiro	.
O	D	O	O	E	E	O	O	O	L	L	L	O

Dans cet exemple, l'étiquette D signifie "date", E "événement" et L "lieu", tandis que O (pour "out") est assignée aux tokens non pertinents pour l'extraction. Ce type d'étiquetage présente toutefois un inconvénient : il ne permet pas de trouver l'éventuelle frontière passant entre des extractions différentes de même type portées par des tokens consécutifs. On ne peut ainsi pas distinguer avec une telle annotation entre "Jean Paul" correspondant à un unique prénom composé ou à deux personnes différentes. Pour éviter ce problème, on utilise habituellement le codage dit BIO (Begin/In/out) consistant à ajouter la lettre B à l'étiquette associée au premier élément d'une extraction et I aux éléments internes. L'étiquetage précédent devient alors :

En	2016	,	les	Jeux	Olympiques	auront	lieu	à	Rio	de	Janeiro	.
O	D-B	O	O	E-B	E-I	O	O	O	L-B	L-I	L-I	O

Avec ce codage, l'annotation de "Jean Paul" en "B I" signifie qu'il s'agit d'une seule et même personne, tandis que son annotation en "B B" veut dire que ce sont deux individus différents. Outre sa fonction de désambiguisation des frontières de zones à extraire, le codage BIO est également efficace parce que les propriétés du premier token d'une extraction (celui qui recevra une étiquette comprenant un B) sont souvent différentes de celles des tokens internes, et justifient donc un traitement spécifique. Les noms propres, par exemple, commencent en français par une majuscule mais ce n'est pas nécessairement le cas de tous leurs tokens internes (comme dans "Rio de Janeiro"). Notons que ce codage BIO permet aussi de représenter simplement des segmentations : le découpage en chunks de la phrase "le petit chat est mort" (cf. 3.3) peut ainsi s'annoter comme :

Le	petit	chat	est	mort	.
B	I	I	B	B	O

où chaque étiquette B marque le début d'un parenthésage nouveau (et par la même occasion, le cas échéant, la fermeture du parenthésage précédent).

D'autres codages sont possibles, qui poussent plus loin encore la logique de dissocier les propriétés des débuts et des fins de zones à annoter : certains chercheurs militent ainsi pour le codage BILOU, extension du codage BIO, où l'étiquette L (pour "last") est attribuée aux derniers tokens d'une zone à extraire, et U ("unique") aux tokens qui constituent à eux seuls une donnée à extraire. Dans cette nouvelle norme, notre exemple précédent devient :

En	2016	,	les	Jeux	Olympiques	auront	lieu	à	Rio	de	Janeiro	.
O	D-U	O	O	E-B	E-L	O	O	O	L-B	L-I	L-L	O

Comme on a vu précédemment que la tâche d'annotation pouvait elle-même se traiter comme une séquence de classifications, c'est donc aussi le cas de la tâche d'EI. Ainsi, avec des classifieurs, on peut tout faire ! Cette place pivot de la classification en fouille de textes est le prolongement de son rôle historique prédominant en fouille de données.

4.2 Décompositions en tâches élémentaires

Un autre type de relations existant entre les tâches est la possibilité de les *combiner* afin de réaliser des traitements plus complexes, ou plutôt de décomposer des tâches plus complexes à l'aide de nos quatre tâches élémentaires (et éventuellement d'autres non développées ici).

Les systèmes Question/Réponse (systèmes Q/R)

Un exemple typique de décomposition en sous-tâches est fourni par les "systèmes Question/Réponse" (ou systèmes Q/R, ou "Question Answering Systems" en anglais) dont la spécification générale est donnée par la figure 2.21.

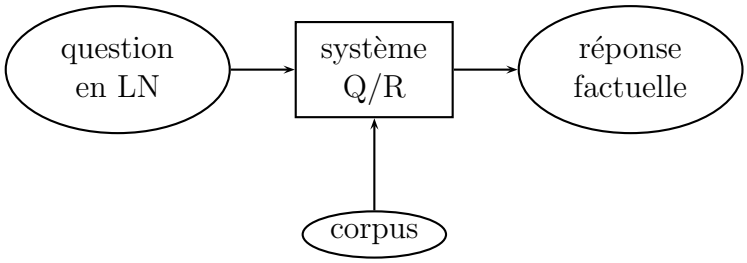


FIGURE 2.21 – Schéma général de la tâche de Question/Réponse

Ce type de système se présente comme un moteur de recherche avancé : il permet à l'utilisateur de poser une question en "langue naturelle" (LN sur la figure), c'est-à-dire formulée comme on le ferait à un interlocuteur humain (par exemple "Quand est né Mozart ?"). Il a par ailleurs accès (en ressource) à un corpus suffisamment conséquent pour y contenir la bonne réponse (par exemple le Web, ou au minimum les pages de Wikipedia [5]) et est censé la fournir en sortie (dans notre exemple, ce serait "1756"). Le résultat d'un système Q/R, contrairement à celui des systèmes de RI, est une réponse (ou un ensemble de réponses) précise(s) et non un document (ou une collection de documents) pertinent(s). Evidemment, de tels systèmes ne sont capables de répondre qu'à des questions factuelles élémentaires et ne prétendent pas donner d'explications élaborées ; ils ne savent pas traiter les questions commençant par "pourquoi", par exemple.

Il existe deux grandes familles de systèmes Q/R, qui se distinguent par la stratégie adoptée pour les décomposer en sous-tâches. Nous les exposons brièvement toutes les deux en montrant que chacune d'elles, à des degrés divers, fait appel à certaines des quatre tâches élémentaires que nous avons détaillées précédemment.

Première stratégie : se ramener à de la RI

La première stratégie est illustrée par la figure 2.22.

Elle consiste à décomposer le problème en une séquence de tâches simples, parmi lesquelles figurent plusieurs de nos tâches élémentaires (en gras dans la figure) :

5. http ://fr.wikipedia.org/wiki/Wikipédia :Accueil_principal

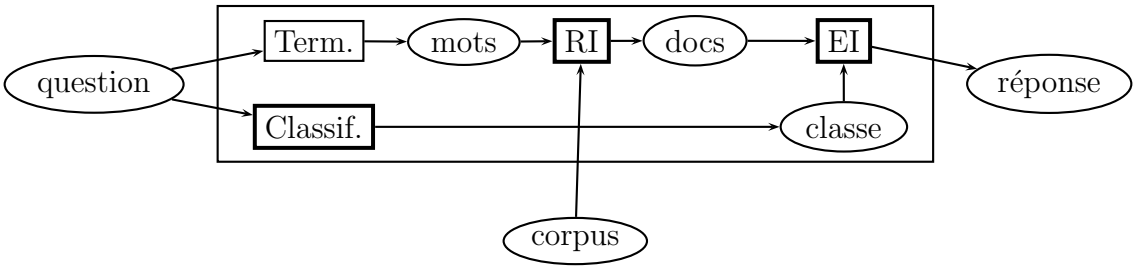


FIGURE 2.22 – 1ère décomposition possible de la tâche Q/R

- la question posée en LN est traitée doublement : d'une part, elle est soumise à un programme de terminologie (Term. dans la figure) qui en extrait les mots clés pertinents ("Mozart" serait le principal dans notre exemple), d'autre part elle est envoyée à un programme de classification chargé de déterminer le "type" de sa réponse attendue, relativement à une taxonomie spécifique de chaque système (on peut imaginer un système qui disposerait d'un type "date de naissance") ;
- le (ou les) mot(s) clé(s) extrait(s) de la question sont transmis à un moteur de recherche standard, qui a lui-même accès au corpus du système : il fournit donc en sortie un ensemble de documents pertinents pour ce (ou ces) mot(s) clé(s) (une collection de textes parlant de Mozart, dans notre cas) ;
- chaque classe (ou type) possible du programme de classification est associée à un programme d'extraction d'information spécialisé dans la recherche de la réponse à ce type de question (la classe "date de naissance" correspond bien à un champ possible de formulaire). Les documents sélectionnés précédemment dans le corpus sont utilisés comme données d'entrée à ce programme : la (ou les) réponse(s) sélectionnée(s) est celle de l'ensemble du système.

Deuxième stratégie : se ramener à une requête dans un langage formel

L'autre stratégie possible pour aborder les systèmes Q/R est de se ramener à une requête exprimée dans un langage formel de type SQL ou, plus récemment, SPARQL. Elle est illustrée par la figure 2.23, dans laquelle LF est l'abréviation de "langage formel" et BC signifie "base de connaissances".

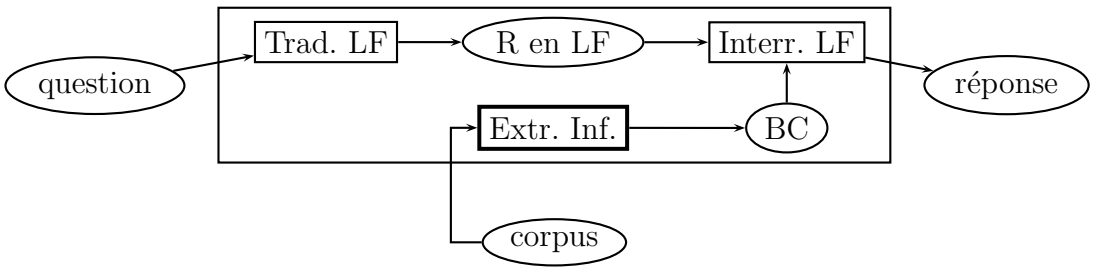


FIGURE 2.23 – 2ème décomposition possible de la tâche Q/R

Dans ce cas, les étapes essentielles du système (parmi lesquelles figure une phase

d'EI, en gras) sont les suivantes :

- la question en langue naturelle est traduite (Trad. LF) en une requête en langage formel (R en LF), elle-même soumise à un système d'interrogation en LF (Interr. en LF) ;
- ce système doit avoir accès à une base de connaissances (BC) formalisée, interrogeable dans le LF choisi. Cette étape requiert d'avoir préalablement transformé le corpus en une telle base de connaissances : c'est le rôle de l'EI dans cette stratégie.

Longtemps, seule la première de ces deux stratégies a été possible à grande échelle, faute de disposer de bases de connaissances suffisamment complètes et fiables, ou de manières efficaces de les construire. Avec l'émergence progressive du Web sémantique et de l'EI, de telles bases commencent maintenant à exister et à être disponibles (c'est le cas par exemple de DBPedia [6], la traduction en RDF des informations factuelles de Wikipedia) et les systèmes Q/R fondés sur cette deuxième stratégie arrivent, dans certains cas, à être compétitifs.

En France, plusieurs équipes travaillent sur ces systèmes, en se rattachant à l'une de ces deux approches : par exemple, l'équipe ILES du LIMSI [7] a adopté la première, tandis que le portail d'Orange [8] fonctionne en suivant la deuxième. Le système question/réponse en ligne le plus avancé s'intitule Wolframaplpha [9] : il permet le traitement de questions en anglais sur de nombreux sujets (voir les exemples fournis).

5 Représentation des données

Nous avons vu que les tâches étaient liées les unes aux autres, et que la classification jouait un rôle pivot. Nous montrons pour finir que les divers types de données que nous avons évoqués ne sont pas non plus si différents qu'ils en ont l'air, et que le type "données tabulaires" occupe une place particulière. C'est en effet le type privilégié en fouille de données, surtout si les champs du tableau sont à valeurs *numériques*. Il est donc apparu assez naturel aux pionniers de la fouille de textes d'essayer de transformer les autres types de données qu'ils avaient à traiter en tableaux de nombres, afin d'y appliquer les méthodes et algorithmes qui avaient fait leur preuve en fouille de données. Cette stratégie s'est avérée payante et est encore largement utilisée de nos jours, surtout pour les tâches de recherche d'information et de classification, qui peuvent se permettre de négliger la structure interne (les relations d'ordre entre éléments constitutifs) des données textuelles. L'objet de cette partie est donc d'expliquer les différentes manières possibles de transformer un texte brut ou un document semi-structuré en un tableau de nombres (ou un vecteur, puisque nous verrons que c'est équivalent). Cette transformation est souvent considérée comme un *prétraitement* à appliquer aux données textuelles, visant à les *représenter* dans un format compatible avec certains algorithmes. Nous montrerons qu'elle peut, dans certains cas, faire appel à des ressources linguistiques plus

6. http ://fr.dbpedia.org
7. http ://www.limsi.fr
8. http ://www.orange.fr
9. http ://www.wolframalpha.com/

ou moins complexes. Avant de passer en revue ces méthodes, nous nous attardons sur quelques spécificités des textes qu'il sera utile de prendre en compte pour les transformer en tableaux sans (trop) dénaturer leur contenu informationnel.

5.1 Spécificités statistiques des données textuelles

L'analyse statistique des textes n'a pas attendu l'invention des ordinateurs pour commencer. On doit ainsi à Georges Zipf (1902-1950) une étude empirique célèbre de la répartition des mots dans le roman *Ulysse* de James Joyce, dans laquelle il remarque que le mot le plus courant revient 8000 fois, le dixième mot le plus courant 800 fois, le centième 80 fois et le millième 8 fois. Ces résultats (presque trop beaux pour être vrais) se généralisent suivant ce qui est désormais connu comme la "loi de Zipf", qui s'énonce de la façon suivante : si les mots d'un texte (ou d'un corpus) sont rangés du plus courant au plus rare et que l'on note $f(n)$ le nombre d'occurrences du mot de rang n, alors on a la relation $f(n) * n = k$ ou encore $f(n) = \frac{k}{n}$ où k est une constante qui ne dépend que de la langue du texte (ou du corpus). Dans *Ulysse*, k vaudrait apparemment 8000. La courbe correspondante (où le rang n est décliné suivant l'axe des abscisses, et $f(n)$ en ordonnées) a alors l'allure de la figure 2.24. Cette loi a connu divers variantes et affinements que nous ne détaillerons pas ici. Elle n'est évidemment pas toujours vraie à l'unité près mais c'est une approximation qui a été largement validée sur un très grand nombre d'exemples. Elle signifie intuitivement que, dans un corpus, il y a un petit nombre de mots très fréquents (ceux classés dans les premiers rangs) et un très grand nombre de mots très peu fréquents (ceux classés en queue de peloton), car la courbe se rapproche très rapidement de l'axe des abscisses.

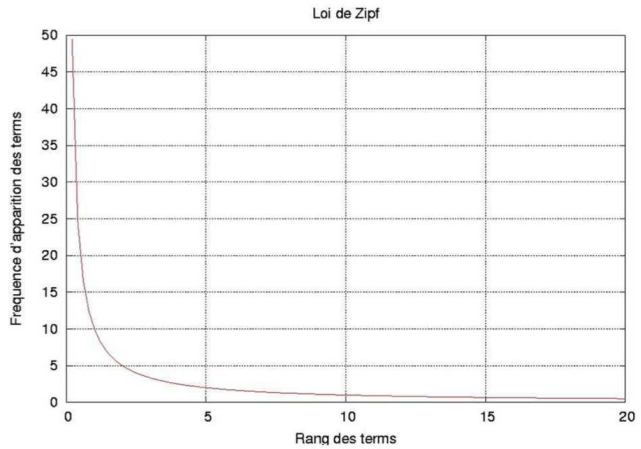


FIGURE 2.24 – Courbe d'une loi de Zipf (d'après M-R. Amini)

Ce type de répartition très inégalitaire des fréquences des unités constituant un ensemble de données est aussi appelée "loi de puissance" ou, dans d'autres contextes, "loi de Pareto". Elle se retrouve dans de nombreux autres domaines, par exemple la fréquentation des sites Web (un petit nombre de sites cumulent beaucoup de visites, un grand nombre en attirent peu) ou encore la vente de produits (peu de produits

sont très vendus, un grand nombre le sont très peu), etc. Cette dernière constatation a d'ailleurs donné lieu à la théorie de la "longue traine", suivant laquelle il peut être rentable (pour des sites relevant de l'économie numérique notamment), de mettre en vente des produits peu demandés mais qui intéressent globalement un grand nombre de personnes constituant autant de "clientèles de niche". Plus étonnant encore, La loi de Zipf se vérifie aussi quand, au lieu de compter les mots d'un texte, on comptabilise par exemple les catégories grammaticales qu'ils représentent, ou les règles de grammaires (au sens des grammaires formelles) qu'il faut utiliser pour analyser les phrases qu'il contient. Et il en est encore de même quand on compte le nombre d'occurrences des balises d'un documents semi-supervisé.

La loi de Heaps, moins connue, caractérise elle la variabilité du vocabulaire d'un corpus. Elle stipule que la taille du vocabulaire V d'un texte ou d'un corpus (c'est-à-dire le nombre d'unités distinctes qui y figurent) croît exponentiellement (mais avec une valeur d'exposant inférieure à 1) en fonction du nombre de mots M présents dans ce texte ou ce corpus. On a ainsi la relation $V = K * M^{\beta}$ où K (distinct du k précédent de la loi de Zipf) et β sont des paramètres dépendants du texte ou du corpus (en anglais, $K \in [30, 100]$ et $\beta \approx 0,5$). On a alors une courbe du genre de celle de la figure 2.25. Cette loi signifie que prendre en compte de nouveaux textes dans un corpus (ou de nouvelles phrases dans un texte) a toujours pour conséquence d'ajouter de nouvelles unités qui n'étaient pas déjà présentes avant : on n'a jamais de description exhaustive d'une langue. Contrairement à la loi de Zipf, cette dernière loi n'est, elle, valable que pour les mots car les catégories grammaticales ou les balises constituent des ensembles finis assez limités : à partir d'une certaine taille de la collection de textes ou de documents analysés, elles ont toutes été rencontrées au moins une fois et la courbe correspondante devient donc plate.

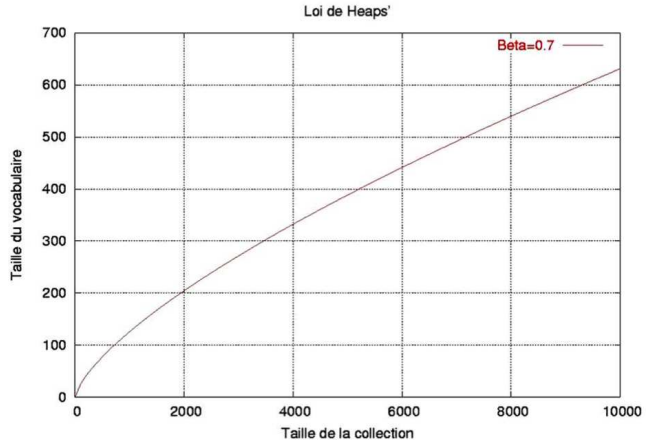


FIGURE 2.25 – Courbe d'une loi de Heaps (d'après M-R. Amini)

5.2 Choix des attributs

On vise donc à transformer des collections de textes ou de documents en tableaux de telle sorte que chacune de leur ligne corresponde à une donnée distincte, comme

dans la figure 2.2, c'est-à-dire à un texte spécifique du corpus. Pour cela, il faut commencer par se demander ce qui jouera le rôle des *champs* ou *attributs* (c'est-à-dire des colonnes) de ces tableaux. Une première idée serait d'exploiter, quand elles sont disponibles, des méta-données du genre : auteur du texte, date d'écriture ou de parution, genre, thèmes traités, etc. C'est ce que font les logiciels documentaires, qui servent à indexer les livres dans les bibliothèques -mais ce n'est pas du tout l'esprit de la fouille de textes ! Les méta-données sont en effet en général absentes des textes eux-mêmes, et leur recueil nécessite un travail manuel. C'est inenvisageable en fouille de textes, où tout doit être automatisé et réalisable efficacement par des programmes. Par ailleurs, nous voulons autant que possible privilégier des *tableaux de nombres*, parce que ce sont ceux qui sont les mieux traités par les algorithmes de fouille de données que nous allons exploiter. Les méta-données se présentent en général comme des informations symboliques, elles ne se prêteront pas bien à ces programmes. Les seuls champs possibles sont donc des attributs élémentaires des textes ou des documents, qu'il sera possible de *comptabiliser*. Nous les passons maintenant en revue, du plus simple au plus complexe.

Caractères, n-grammes de caractères

Puisque les textes bruts (cf. partie 2.2) ne sont rien d'autres que des *séquences de caractères*, utiliser l'ensemble des caractères possibles comme attributs et compter leur nombre d'occurrences dans un texte est une manière simple de transformer un corpus en un tableau de nombres. Combien d'attributs cela fait-il ? Cela dépend bien sûr de l'éventuelle normalisation initiale que l'on fait subir au texte : distingue-t-on les caractères majuscules des minuscules, prend-on en compte les caractères spéciaux (voyelles accentuées, "ç" français, symboles monétaires, etc.), les caractères blancs, les signes de ponctuation, les chiffres ? Si l'on s'en tient aux lettres de l'alphabet latin minuscules, on se ramène à seulement 26 attributs (colonnes) ; à quelques centaines au maximum si on garde tous les caractères alphanumériques distincts possibles. Cela fait des tableaux de taille très raisonnable, qu'il est très facile de remplir par programmes. Sont-ils pour autant suffisants pour représenter les textes initiaux pour les tâches élémentaires que nous avons évoquées ? C'est moins sûr... Peut-on en effet espérer retrouver un document traitant d'un thème donné dans une collection (tâche de recherche d'information) sur la seule base du nombre de chacune des lettres de l'alphabet qu'il contient ? Sera-t-il possible de distinguer par exemple un spam d'un mail intéressant, ou une critique littéraire positive d'une négative (ce qui relève de la tâche de classification) sur ce seul critère ? C'est évidemment douteux. La fréquence relative des différentes lettres de l'alphabet dans un texte est représentative de la langue dans laquelle il est écrit, mais ne dit absolument rien de son contenu. A la limite, si l'objectif visé est simplement la reconnaissance automatique de la langue d'écriture d'un texte (dans une tâche de classification), cela peut suffire, mais c'est le seul cas de figure pour lequel une représentation aussi élémentaire est envisageable.

Si l'on veut faire mieux à moindre frais, c'est-à-dire sans s'engager dans des traitements informatiques ou linguistiques complexes, il est toutefois possible de se concentrer non pas sur les seuls caractères isolés mais sur les *n-grammes de caractères*. Un n-gramme (où n est un nombre entier supérieur ou égal à 1) de caractères

est une sous-séquence de n caractères consécutifs à l'intérieur d'une séquence plus grande. Prenons par exemple la phrase "Le petit chat est mort." et comptons les n-grammes de caractères qui y figurent pour différentes valeurs de n :

- si $n = 1$, on se ramène au compte des différents caractères du textes (appelés aussi dans ce cas "unigrammes"), à savoir ici : "L", "e", " ", "p", "t", etc. Dans cette phrase, la lettre "t" est présente 5 fois, le caractère blanc " " 4 fois, tandis que "e" (qui est normalement la lettre la plus fréquente en français) apparaît 3 fois.
- pour $n = 2$, on parle de bigrammes. Ceux présents dans notre phrase sont "Le", "e ", " p", "pe", "et", etc. Seul le bigramme "t " apparaît plusieurs fois dans la phrase (3 fois, ici).
- les trigrammes sont les n-grammes pour $n = 3$; ceux de notre phrase sont : "Le ", "e p", " pe", "pet", "eti", etc., chacun n'apparaît ici au plus qu'une seule fois.
- on peut compter les nombres d'occurrences des n-grammes de notre exemple pour toutes les valeurs de n comprises entre 1 et 23, la phrase complète étant elle-même un n-gramme de 23 caractères.

Quels sont les coûts et les gains de cette variante ? Comptons d'abord le nombre de colonnes ainsi définies. Si on suppose qu'il y a environ 100 caractères distincts, c'est-à-dire 100 colonnes ou attributs quand on se contente de prendre en compte les unigrammes, alors on doit prévoir 100^n colonnes possibles pour les autres valeurs de n. Pour $n = 3$ on aboutit déjà à $100^3 = 1\ 000\ 000$ attributs différents. En fait, beaucoup de ces trigrammes "théoriques" ne sont *a priori* présents dans aucune langue humaine (par exemple "wzk", "qbq") et de nombreuses colonnes seront ainsi évitées. Mais l'évolution des usages est imprévisible : des trigrammes apparemment improbables ont récemment fait leur apparition dans certains corpus (par exemple " :-)" ou encore "ooo" dans un message comme "looool !"). Aussi surprenant que cela puisse paraître, la représentation des textes en trigrammes de caractères s'est avérée efficace pour de nombreuses applications. Il semble en effet que certains trigrammes, à défaut d'être porteurs de sens, sont suffisamment représentatifs de certains mots pour permettre la réalisation de quelques unes des tâches que nous avons citées. La valeur $n = 3$ est un compromis intéressant car pour $n < 3$ la représentation est trop élémentaire, et pour $n > 3$ elle génère trop de colonnes distinctes. Le principal intérêt de la représentation en n-grammes de caractères est la simplicité du programme à mettre en œuvre pour transformer un corpus en un tableau de nombres. Et comme aucune ressource externe n'est nécessaire pour cela, cette approche est applicable sur toutes les langues écrites (même quand les caractères de base ne sont pas ceux de l'alphabet latin) et garantit une totale neutralité linguistique.

Mots, lemmes, racines, versions n-grammes, unités multi-mots

Plutôt que de compter les caractères ou les n-grammes de caractères d'un texte brut, il est évidemment naturel et tentant de compter ses *mots*. Mais les "mots" ne sont pas des unités linguistiques, leur définition est problématique. Pour cette raison, les spécialistes de la fouille de textes parlent plus volontiers de "tokens". Un token est une unité purement formelle définie comme une séquence de caractères comprise

entre deux séparateurs, les séparateurs étant les blancs, les signes de ponctuation et certains autres caractères comme les guillemets ou les parenthèses (ces caractères spéciaux, sauf le blanc, constituent eux-mêmes en général des tokens autonomes). Pour y avoir accès, il faut donc disposer d'un *segmenteur*, c'est-à-dire d'un programme capable de découper la suite de caractères qui constituent le texte brut en une séquence de tokens. Ce programme est une ressource linguistique pas toujours aussi élémentaire qu'il y paraît : par exemple, les symboles "." ou "-" sont parfois de vrais, parfois de faux (dans "M." ou "méta-linguistique") séparateurs. Il y a plusieurs inconvénients à prendre les tokens comme unités de comptage pour représenter les textes. Tout d'abord, pour que les différents textes d'une même collection puissent être représentés dans un même tableau, il faut que les colonnes de ce tableau soient les mêmes quels que soient les textes en question. L'ensemble des tokens à prendre en compte pour représenter un texte sera donc en fait tous ceux qui apparaissent dans le *corpus* dont ce texte fait partie. Ainsi, la représentation d'un texte dépend de la collection dont il est membre, et pas uniquement de son contenu propre. Or, par ailleurs, il est pratiquement impossible de faire un inventaire exhaustif de tous les tokens possibles, chaque langue en compte plusieurs centaines de milliers. Comme le montre la loi de Heaps (cf. partie 5.1), plus le corpus est volumineux, plus le tableau risque de contenir un grand nombre d'attributs.

Pour réduire la taille de l'espace de représentation (c'est-à-dire l'ensemble des colonnes ou des attributs) ainsi défini, plusieurs stratégies sont possibles. Une première solution est d'éliminer les mots (ou tokens) dont on soupçonne à l'avance qu'ils n'influeront en rien sur le résultat d'une tâche de recherche d'information ou de classification. C'est le cas des mots dits "vides" parce que non porteurs d'une valeur sémantique forte. On range traditionnellement dans cette catégorie les mots grammaticaux (déterminants, prépositions, conjonctions...) mais aussi les auxiliaires (être et avoir en français) et les verbes supports (comme "faire", "prendre"...) tellement courants que leur sens spécifique est très faible. Des listes de mots vides (avec toutes leurs variantes flexionnelles) dans diverses langues sont disponibles sur Internet et faciles à intégrer à un programme chargé de représenter une collection de textes dans un tableau [10]. Elles constituent un autre exemple de contribution (minimale) de la linguistique à la fouille de textes. La loi de Zipf (cf. partie 5.1) sert aussi à justifier l'élimination d'un certain nombre de mots ou tokens : typiquement, ceux qui sont soit très fréquents (et qui coïncident d'ailleurs en général avec les "mots vides" précédemment cités) soit très rares (au point qu'ils ne sont souvent présents qu'une seule fois dans un seul texte du corpus) ne permettent pas de comparer les textes entre eux efficacement. La figure 2.26 montre la zone, comprise entre deux seuils, des mots à conserver pour servir d'attributs : ce sont les mots les plus *représentatifs* des textes où ils sont présents. Les seuils sont à déterminer en fonction du contexte.

Pour certaines applications, on peut avoir des *connaissances a priori* sur les types de mots (ou tokens) qui sont discriminants pour la tâche : en classification d'opinion, par exemple, il est courant de se concentrer sur les verbes et les adjectifs, car c'est principalement sur eux que repose le caractère subjectif des textes. Les adverbes servent, pour leur part, à intensifier ou atténuer la force de l'opinion exprimée, ils

10. on en trouvera par exemple une pour le français à l'adresse suivante : http ://snowball.tartarus.org/algorithms/french/stop.txt

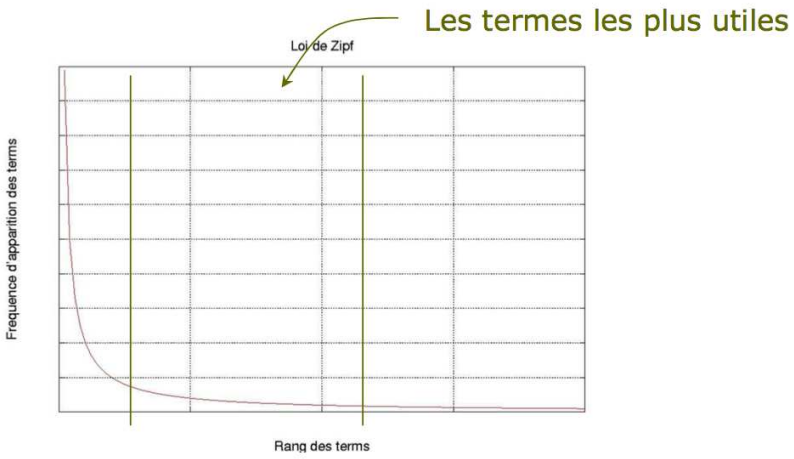


FIGURE 2.26 – Portion "utile" des mots suivant une loi de Zipf (d'après M-R. Amini)

sont donc également utiles. Mais sélectionner ces types de mots requiert bien sûr de disposer d'un étiqueteur morpho-syntaxique.

Pour diminuer encore le nombre de colonnes, on peut faire appel à d'autres ressources linguistiques qui permettent de regrouper certains tokens entre eux :

- si on dispose d'un *lemmatiseur*, c'est-à-dire d'un programme qui associe à chaque token la forme canonique qui le représente dans un dictionnaire (l'infinitif pour les verbes, la forme masculin singulier pour les noms et les adjectifs, etc.), alors on peut s'en servir pour ramener l'ensemble des tokens distincts à l'ensemble des lemmes distincts.
- à défaut de lemmatiseur, qui est une ressource complexe, il existe des *raciniseurs* plus rudimentaires : un tel programme se contente de supprimer les lettres habituellement porteuses des flexions dans une langue donnée (en français, ce sont au maximum les 2 ou 3 dernières lettres d'un mot), afin de ramener les tokens à leur "racine". Il y a évidemment moins de racines distinctes que de tokens distincts.

Pour chacune de ces unités (tokens, lemmes, racines), il est évidemment possible de définir les attributs de notre tableau non pas à partir de leur seule présence, mais en considérant des *n-grammes* (de mots, lemmes, racines, ou d'un mélange entre eux...). Cette solution a du sens quand elle vise à capturer des unités poly-lexicales ou (multi-mots) comme "pomme de terre" qui sont constituées de plusieurs tokens successifs. Mais en procédant aveuglément, on prend le risque de définir un très grand nombre d'attributs différents, pas toujours pertinents. La solution idéale est de disposer de segmenteurs en unités multi-mots, c'est-à-dire de programmes capables de découper un texte en de telles unités. Là encore, la linguistique est mise à contribution. Certains auteurs ont aussi proposé des attributs prenant la forme de *sous-séquences* (c'est-à-dire de suites de mots extraits du texte, dans l'ordre où ils y apparaissent) pas nécessairement continues (c'est-à-dire en laissant la possibilité de "sauter" certains mots). Par exemple, l'attribut "ne aime pas", en tant que sous-séquence (présente par exemple dans "ce film, je ne l'aime absolument pas") peut être intéressant, notamment en classification d'opinion où la prise en compte

des négations est une problématique importante, car elles changent la polarité des mots qu'elles qualifient. Mais l'ensemble de toutes les sous-séquences possibles d'une phrase est volumineux, et toutes ne présentent pas le même intérêt.

Catégories, concepts et autres attributs linguistiques

Il est possible d'aller encore plus loin dans le pré-traitement des textes, afin d'en extraire des attributs plus spécifiques, dont certaines tâches pourraient bénéficier. Par exemple, si l'objectif est de distinguer entre plusieurs styles d'écriture (pour une tâche de classification d'auteurs ou de reconnaissance de registres de langue, par exemple), indépendamment des sujets traités, alors il peut être intéressant de se concentrer non pas sur les mots employés mais sur les *catégories morpho-syntaxiques (ou grammaticales)* dont relèvent ces mots. Cela suppose bien sûr, comme évoqué précédemment, de disposer d'un étiqueteur morpho-syntaxique capable d'associer la bonne catégorie en contexte à chaque token. Si c'est l'enchaînement des catégories qui est supposé pertinent, alors il faut prendre comme attributs les n-grammes (ou les sous-séquences) de ces catégories. Cela reste dans les limites du raisonnable car, même en y intégrant les propriétés flexionnelles des mots (genre et nombre pour les noms et les adjectifs, conjugaison et personne pour les verbes, etc.), on n'en recense généralement pas plus d'une centaine distinctes pour une langue donnée.

Si on est en mesure de passer un analyseur syntaxique produisant des arbres (de dépendances ou de constituants) sur les phrases des textes, alors on peut aussi mettre en attributs certaines de leurs propriétés : comptage des symboles figurant dans les nœuds (pour les constituants) ou sur les arcs (pour les dépendances), énumération des règles utilisées et/ou des branches (couples de nœuds reliés par une relation père-fils) ou des "fourches" (triplets de nœuds constitués d'un père et de deux fils successifs) et/ou des chemins ou sous-chemins, ou sous-arbres représentés dans les analyses, etc.

Si, enfin, la représentation des documents vise plutôt à privilégier leur contenu sémantique (c'est en général le cas pour la recherche d'information), il faut chercher à rattacher les tokens à des *concepts*. Les analyseurs sémantiques sont hélas encore rares mais certains programmes sont capables de situer les noms présents dans un texte dans une ontologie, et donc de reconnaître les concepts dont ils dépendent (ceux dont ils héritent dans la hiérarchie). Ces programmes doivent résoudre des problèmes d'ambiguité difficiles, aucun n'est exempt d'erreurs. Si on les estime suffisamment fiables, utiliser les concepts ainsi identifiés comme attributs et compter combien de mots de chaque texte les représentent est alors une solution potentiellement originale et intéressante.

On l'a vu : plus la représentation est élaborée, plus la (ou les) ressource(s) linguistique(s) nécessaire(s) pour la produire sont complexes. Le rapport entre l'effort à fournir pour produire la représentation souhaitée et le gain de performance ainsi permis est à évaluer pour chaque tâche. Il est parfois hélas décevant.

Attributs pour les documents semi-structurés

Les attributs précédemment cités concernent surtout les textes bruts. Ils restent toutefois pertinents pour les documents semi-structurés, à condition de considérer

ces derniers comme des textes particuliers, dont les balises sont des tokens parmi d'autres. La vision arborescente de ces documents suggère quant à elle l'utilisation des attributs évoqués dans le cas où une analyse syntaxique du texte est disponible : on peut en effet dans ce cas isoler les nœuds, les branches , les "fourches", les chemins, les sous-chemins, les sous-arbres, etc. Ces attributs caractérisent la *structure* du document plus que son contenu, qui en général n'est exprimé que dans les feuilles. Certaines tâches de fouille de textes (classification de pages web par exemple) ont sans doute beaucoup à gagner à la prise en compte de cette structure.

5.3 Choix des valeurs : des sacs de mots aux vecteurs

Nous avons implicitement suggéré dans la partie précédente qu'une fois les attributs définis, il n'y avait plus qu'à compter le nombre de fois où ils sont présents dans le texte ou le document pour déterminer la valeur de cet attribut pour ce texte. En fait, d'autres choix plus judicieux sont possibles. Pour formaliser plus aisément les calculs qui suivent, notons $a_1, a_2, ..., a_n$ les attributs (ou colonnes) définis et $t_1, t_2, ...t_m$ la liste des textes ou des documents (constituant autant de lignes dans le tableau). A l'intersection de la ligne t_i et de la colonne a_j, on veut fixer la valeur $v_{i,j}$. On peut choisir par exemple :

- $v_{i,j}$ est un simple booléen qui vaut 1 si a_j est présent dans t_i (quel que soit le nombre de fois où il figure), 0 sinon ;
- $v_{i,j}$ est le nombre d'occurrences de a_j dans t_i, qu'on note $n_{i,j}$;
- pour limiter l'impact de la taille des documents, on normalise la valeur précédente en comptant la *proportion* de chaque attribut $v_{i,j} = \frac{n_{i,j}}{\sum_j n_{i,j}}$: dans ce cas, la somme des valeurs $v_{i,j}$ sur la ligne i vaut toujours 1 ;
- on normalise pour se ramener à un vecteur de norme 1 : $v_{i,j} = \frac{n_{i,j}}{\sqrt{\sum_j n_{i,j}^2}}$: dans ce cas, c'est le vecteur constitué des valeurs $v_{i,j}$ qui a pour norme 1 (on rappelle le calcul de cette norme plus loin) ;

Ces calculs ne visent pour l'instant qu'à mesurer l'importance d'un attribut a_j dans un texte t_i. Mais nous avons vu que la représentation d'un texte dépend de l'ensemble de la collection dont il fait partie, et pas uniquement de son seul contenu. Or, l'importance d'un attribut dans un texte sera d'autant plus grande que celui-ci apparaît beaucoup dans ce texte mais peu dans les autres. C'est cette combinaison particulière que cherche à capturer la pondération dite TF.IDF (pour "term frequency" et "inverse document frequency"), extrêmement populaire en fouille de textes. Dans ce cas, la valeur $v_{i,j} = TF(i,j).IDF(j)$ se calcule de la façon suivante :

- la valeur $TF(i,j) = n_{i,j}$ ou bien $TF(i,j) = \frac{n_{i,j}}{\sum_j v_{i,j}}$: attention, en français, on réserve d'habitude le mot "fréquence" pour désigner un *rapport* entre deux valeurs (comme dans les valeurs normalisées) mais les anglo-saxons utilisent le mot "frequency" pour compter un simple nombre d'occurrences, d'où une certaine ambiguité parfois pour définir cette valeur. On verra plus loin pourquoi ces deux calculs sont possibles.
- $IDF(j) = ln(\frac{m}{|\{t_i|a_j \in t_i\}|})$, où m est le nombre de textes de la collection et $|\{t_i|a_j \in t_i\}|$ le nombre de textes qui contiennent au moins une fois l'attribut a_j. Comme son nom l'indique, IDF mesure l'inverse de la fréquence (au sens

de rapport cette fois !) d'un attribut dans l'ensemble des documents (ou des textes) : on prend le logarithme de ce rapport (toujours supérieur ou égal à 1, ce qui assure que le logarithme est positif) pour atténuer sa valeur. $IDF(j)$ mesure en quelque sorte la *rareté* de l'attribut a_j dans la collection. En effet, si l'attribut apparaît partout, alors il ne permet pas de distinguer un texte d'un autre, il est donc neutralisé (sa rareté est nulle) : $IDF(j) = ln(\frac{m}{m}) = ln(1) = 0$. Si au contraire il est très rare en n'étant présent que dans un seul texte (c'est le minimum, sinon l'attribut n'aurait aucune raison d'avoir été conservé), alors il vaut $IDF(j) = ln(\frac{m}{1}) = ln(m)$, sa valeur maximale. On utilise parfois la mesure TF.IDF pour éliminer les attributs non pertinents (ceux ayant une valeur TF.IDF faible), plutôt que d'utiliser des seuils de nombre d'occurrences comme dans la figure 2.26.

Beaucoup d'autres pondérations sont possibles, pour renforcer l'importance de certaines propriétés. Par exemple, si on fait l'hypothèse que plus les attributs apparaissent tôt dans un texte, plus ils sont importants, alors on peut moduler les valeurs en fonction des positions par rapport au début du texte. Nous ne détaillerons pas plus ces variantes ici.

De manière générale, quelles que soient les options privilégiées en matière d'attributs et de valeurs associées, on désigne l'ensemble des représentations décrites ici comme de type "sac de mots" ("bag of words" en anglais) -même si, on l'a vu, ce ne sont pas nécessairement les "mots" eux-mêmes qui servent d'attributs. Ce terme générique restitue assez bien ce à quoi est "réduit" un texte transformé en un tableau de nombres suivant ces directives : il est découpé en segments indépendants qui sont comme "jetés en vrac" dans un grand sac, puisque seul leur nombre d'occurrences compte. Certaines des propriétés fondamentales des textes initiaux, notamment *l'ordre des mots* sont ainsi complètement oubliées. Rappelons en effet que, dans les tableaux ainsi construits, l'ordre des colonnes est totalement arbitraire (en général, on prend l'ordre alphabétique mais tout autre conviendrait aussi bien). Avec le même "sac de mots", il est évidemment possible de rédiger des textes très différents ("Jean tue Marie" et "Marie tue Jean" ne veulent pas tout à fait dire la même chose !), de même qu'on peut exprimer le même contenu sémantique avec des "sacs de mots" complètement distincts. D'un point de vue linguistique, ramener un texte à la simple comptabilité de ses unités est donc frustrant. Mais la fouille de textes n'a pas ces scrupules : du moment que le résultat de la tâche est satisfaisant...

En fait, l'hypothèse sous-jacente à la plupart des programmes auxquels seront soumis nos tableaux est encore plus radicale : elle consiste à supposer que l'ensemble des attributs constituent les dimensions d'un *espace vectoriel euclidien* et que la suite de nombres qui caractérisent un texte sont les coordonnées dans cet espace d'un *vecteur*. Cela signifie en particulier que les attributs sont non seulement indépendants mais "orthogonaux" les uns aux autres (au sens où la présence d'aucun d'entre eux n'est prévisible à partir de la présence des autres), et qu'un texte n'est qu'une combinaison linéaire (c'est-à-dire une somme pondérée) des mots (ou unités tenant lieu d'attributs) qui y apparaissent. Cette ré-interprétation des tableaux permettra de bénéficier d'un attirail mathématique considérable, rendant par exemple très simples les calculs de distances entre deux textes. Les espaces ainsi définis peuvent avoir des milliers de dimensions (autant que d'attributs distincts),

mais ce n'est pas un problème pour les ordinateurs.

5.4 Mesures de distances et de similarité

Le principal intérêt de ramener les données textuelles à des tableaux de nombres est de faciliter le calcul de *distances* entre textes. Etre en mesure d'évaluer des proximités entre données est en effet un pré-requis fondamental de beaucoup de programmes de fouille de textes, notamment pour les tâches de RI et de classification. Il est possible aussi de définir des distances entre des chaînes de caractères ou des valeurs symboliques, mais nous nous restreignons ici aux mesures entre données représentées par *des suites de booléens ou de nombres*. Remarquons d'ailleurs pour commencer que les calculs proposés en partie précédente aboutissent toujours à des valeurs positives ou nulles. En termes d'interprétation vectorielle, cela signifie que tout se passera dans une sous-partie restreinte de l'espace euclidien, correspondant en dimension deux au quart "en haut à droite" d'un repère cartésien. Pour illustrer nos formules, nous les appliquerons systématiquement au cas des deux vecteurs en dimension deux dont les coordonnées sont dans le tableau de la figure 2.27

attributs \ textes	a_1	a_2
t1	3	1
t2	1	2

FIGURE 2.27 – Coordonnées de deux "textes" dans un espace à deux dimensions

Cette représentation est bien sûr une simplification extrême (il est rare de n'avoir que deux attributs !) mais elle permettra de se faire une intuition de la signification des formules. On peut en particulier visualiser les vecteurs sur un plan, comme dans la figure 2.28. Dans le cas général on note, comme dans la partie précédente $a_1, a_2, ..., a_n$ les attributs et $t_1 = (v_{1,1}, v_{1,2}, ..., v_{1,n})$ et $t_2 = (v_{2,1}, v_{2,2}, ..., v_{2,n})$ les textes à comparer ($v_{i,j}$ est la valeur de l'attribut a_j pour la donnée t_i). Suivant les cas, on va calculer une *distance* ou une *similarité* entre deux textes : la similarité est d'autant plus grande que les textes sont proches, tandis que la distance, à l'inverse, diminue avec la proximité. Quand la similarité est comprise entre 0 et 1 (si ce n'est pas le cas, on normalise pour que ça le devienne), il suffit de prendre 1-similarité comme mesure de distance.

Commençons par les mesures de distance vectorielles :

– la distance de Manhattan : $|t_1 - t_2| = \sum_{k=1}^{n} |v_{1,k} - v_{2,k}|$. Avec les vecteurs de la figure 2.28, on obtient : $|t_1 - t_2| = |3-1| + |1-2| = 2+1 = 3$. Cette distance s'interprète comme l'espace minimum à parcourir pour rejoindre l'extrémité d'un vecteur à un autre (quand ils ont tous les deux la même origine) en ne s'autorisant que des déplacements horizontaux et verticaux (c'est-à-dire, de manière générale, parallèles aux axes) ;

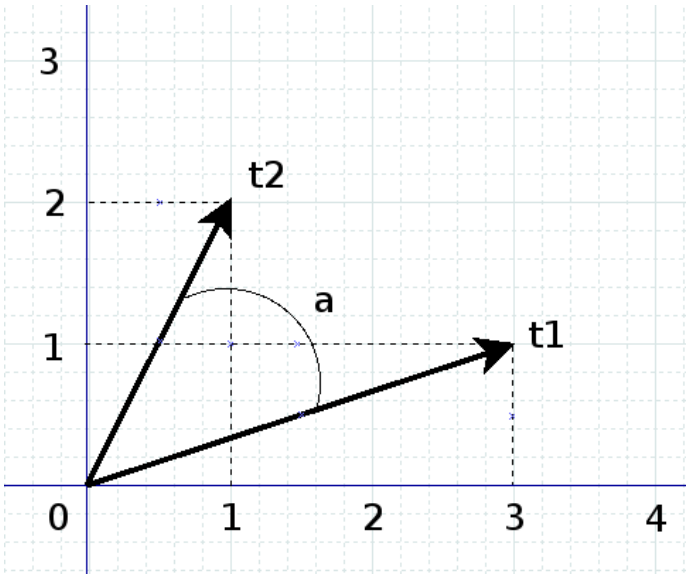


FIGURE 2.28 – Deux vecteurs dans un espace à deux dimensions

- la distance euclidienne : $||t_1 - t_2|| = \sqrt{\sum_{k=1}^{n}(v_{1,k} - v_{2,k})^2}$. Par application du théorème de Pythagore, cette distance coïncide avec la mesure de l'écart entre les deux extrémités des vecteurs t_1 et t_2, (autrement dit à la distance mesurable avec une règle entre les deux points de coordonnées les valeurs de t_1 et t_2). Dans notre exemple, cela donne :
  $||t_1 - t_2|| = \sqrt{\sum_{k=1}^{n}(3-1)^2 + (1-2)^2} = \sqrt{2^2 + 1^2} = \sqrt{5}$
- la distance de Minkowski pour $p > 2$, qui généralise la formule précédente (correspondant à $p = 2$) à d'autres valeurs de la puissance p-ème et de la racine p-ème : $||t_1 - t_2||_p = \sqrt[p]{\sum_{k=1}^{n}|v_{1,k} - v_{2,k}|^p}$. Cette formule a pour effet d'exagérer les écarts entre les valeurs d'un même attribut (s'ils sont déjà > 1), de leur donner en quelque sorte encore plus d'importance. Par exemple, pour $p = 4$, notre exemple devient :
  $||t_1 - t_2||_4 = \sqrt[4]{\sum_{k=1}^{n}|3-1|^4 + |1-2|^4} = \sqrt[4]{2^4 + 1^4} = \sqrt[4]{17}$
- la distance de Tchebychev : si on fait tendre p vers l'infini dans la distance de Minkowski, c'est l'écart de valeurs (pour un même attribut) le plus important qui sera prédominant et on peut se contenter de ne prendre que lui. C'est ce que propose cette distance valant : $\lim_{p\to\infty} \sqrt[p]{\sum_{k=1}^{n}|v_{1,k} - v_{2,k}|^p} = max_k|v_{1,k} - v_{2,k}|$. Dans notre cas, elle vaut $max(2, 1) = 2$.

Pour définir les mesures de *similarité*, d'autant plus grandes que les données sont similaires ou proches, nous aurons besoin du *produit scalaire* entre deux vecteurs, défini par $t_1.t_2 = \sum_{k=1}^{n}(v_{1,k} * v_{2,k})$. On peut montrer que ce produit scalaire correspond également au calcul suivant : $t_1.t_2 = ||t_1|| * ||t_2|| * cos(a)$, où $||t_1|| = \sqrt{\sum_{k=1}^{n} v_{1,k}^2}$ est la norme de t_1, c'est-à-dire la distance euclidienne entre son point de départ (le point de coordonnées $(0, 0)$ ici) et son point d'arrivée (et similairement pour $||t_2||$), et a est l'angle entre les deux vecteurs, comme dans la figure 2.28. Le produit scalaire entre les deux vecteurs de cette figure vaut : $t_1.t_2 = (3 * 1) + (1 * 2) = 5$, et les normes respectives des vecteurs : $||t_1|| = \sqrt{3^2 + 1^2} = \sqrt{10}$ et $||t_2|| = \sqrt{1^2 + 2^2} = \sqrt{5}$.

Nous pouvons maintenant donner les formules de différentes mesures de similarité classiques :

- mesure de Dice : $\frac{t_1.t_2}{||t_1||+||t_2||}$. Pour nos vecteurs, cela donne : $\frac{5}{\sqrt{10}+\sqrt{5}}$. Quand les suites de nombres sont en fait des booléens, on se ramène à une définition plus simple qui prend la forme suivante : $\frac{|t_1 \cap t_2|}{|t_2|+|t_2|}$ où $|t_1|$ (respectivement $|t_2|$) compte le nombre de 1 dans t_1 (respectivement t_2), c'est-à-dire le nombre d'attributs présents au moins une fois dans t_1 (ou t_2), tandis que $|t_1 \cap t_2|$ recense les attributs communs à t_1 et t_2.
- mesure de Jaccard : $\frac{t_1.t_2}{||t_1+t_2||}$. Les coordonnées de la somme de deux vecteurs (qui est elle-même un vecteur) s'obtiennent en additionnant les valeurs de chaque coordonnée, ce qui mène pour notre exemple à $\frac{5}{\sqrt{(3+1)^2+(1+2)^2}} = \frac{5}{\sqrt{25}} = 1$. Comme précédemment, cette mesure est plutôt utilisée dans le cas de valeurs booléennes, sous la forme simplifiée suivante : $\frac{|t_1 \cap t_2|}{|t_1 \cup t_2|}$ où $[t_1 \cup t_2|$ comptabilise les attributs présents soit dans t_1 soit dans t_2.
- le coefficient ovelap est une légère variante des mesures précédentes : $\frac{t_1.t_2}{min(||t_1||,||t_2||)}$, soit, pour nos exemples : $\frac{5}{min(\sqrt{10},\sqrt{5})} = \frac{5}{\sqrt{5}} = \sqrt{5}$. La version booléenne s'écrit, quant à elle : $\frac{|t_1 \cap t_2|}{min(|t_1|,|t_2|)}$.
- enfin, une des mesures les plus populaires est le cosinus de l'angle entre t_1 et t_2 qui, d'après la deuxième expression du produit scalaire, s'obtient facilement par : $\frac{t_1.t_2}{||t_1||*||t_2||}$. Dans notre exemple, cela fait : $\frac{5}{\sqrt{5}*\sqrt{10}} = \frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}$. Cette mesure ne dépend que de *l'orientation* des facteurs (l'angle qu'ils font entre eux) et pas du tout de leur taille. Dans ce cas, les variantes normalisées des vecteurs (la différence entre les deux calculs possibles de TF, par exemple), qui ne modifient précisément que leur dimension et pas leur direction dans l'espace, donnent exactement les mêmes mesures de distances entre eux suivant le cosinus.

Pour les mesures de Dice, de Jaccard et d'overlap, seules les variantes booléennes sont nécessairement dans $[0, 1]$. La valeur du cosinus est, elle aussi, pour des vecteurs de coordonnées positives, nécessairement dans $[0, 1]$. Le cosinus est particulièrement intéressant : plus l'angle est petit, plus celui-ci est proche de 1. Or, un petit angle signifie que les différents vecteurs ont des *proportions similaires* des différents attributs. Par exemple si, pour différents vecteurs, l'attribut a_2 est toujours deux fois plus grand que l'attribut a_1 (quelles que soient les valeurs en question, donc quelle que soit la taille des vecteurs), alors ils feront entre eux un angle nul et auront donc une similarité maximale de 1.

5.5 Un exemple récapitulatif

Pour illustrer les notions introduites dans les deux parties précédentes avec des données "quasi-réelles" mais très simples, nous proposons un mini-corpus, qui servira également dans les chapitres suivants :

texte 1	"Le cinéma est un art, c'est aussi une industrie." (phrase célèbre d'André Malraux)
texte 2	"Personne, quand il est petit, ne veut être critique de cinéma. Mais ensuite, en France, tout le monde a un deuxième métier : critique de cinéma !" (citation approximative de deux phrases de François Truffaut)
texte 3	"Tout le monde a des rêves de Hollywood."
texte 4	"C'est la crise, l'économie de la France est menacée par la mondialisation."
texte 5	"En temps de crise, reconstruire l'industrie : tout un art !"
texte 6	"Quand une usine ferme, c'est que l'économie va mal."

Pour un locuteur du français, il est évident que les trois premiers textes évoquent (plus ou moins) le cinéma, tandis que les trois suivants parlent (de manière caricaturale) de crise économique. Ils appartiendraient respectivement aux rubriques "culture" et "société" d'un journal généraliste. C'est en tout cas l'hypothèse que nous ferons pour une tâche de classification. Voyons comment ces données vont être transformées en tableaux de nombres.

Nous n'utiliserons pas ici de représentation fondée sur les seuls caractères, qui sont trop rudimentaires. En l'absence de ressource capable de reconnaître les unités poly-lexicales (il n'y en a pas vraiment dans les exemples, sauf peut-être "critique de cinéma"), nos unités de base, les tokens, coïncident donc avec les mots des textes. En guise de pré-traitements, les signes ponctuations sont supprimés et les mots mis en minuscule. Pour obtenir le moins possible d'attributs, nous considérons comme mots vides les déterminants (y compris les déterminants généralisés comme "personne"), les prépositions, les conjonctions, les pronoms, les verbes auxiliaires ("être"), modaux ("vouloir") et support ("aller"), ainsi que les adverbes ("aussi", "ensuite", "tout", "quand", "mal") et même certains adjectifs ("deuxième"). Ne restent alors que 18 tokens distincts, qui constituent notre ensemble d'attributs, ou espace de représentation : "art", "cinéma", "crise", critique", "économie", "ferme", "france", "hollywood", "industrie", "menacée", "métier", "monde", "mondialisation", petit", "reconstruire", "rêves", "temps", "usine" (par ordre alphabétique). La représentation en nombre d'occurrences de chacun des textes dans cet espace est donnée par la Figure 2.29 (copie d'écran du logiciel Weka, dans lequel on a ajouté un attribut de classe instancié par une valeur "culture" ou "société" suivant les textes).

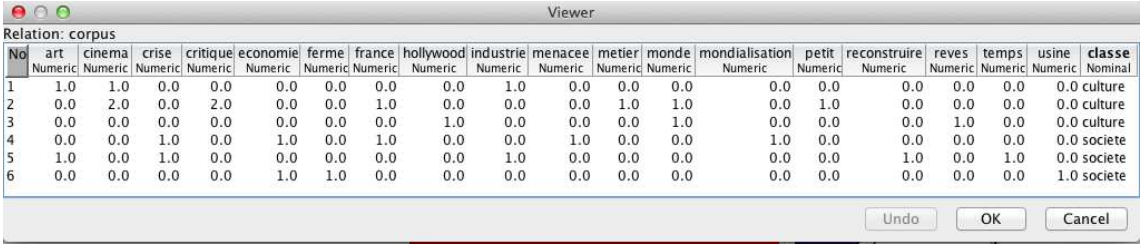


FIGURE 2.29 – Représentation en nombre d'occurrences des textes dans Weka

Une représentation booléenne ne différerait de celle-ci que pour les attributs "cinéma" et "critique", dont les valeurs 2 pour le texte 2 seraient remplacées par 1. L'application d'un lemmatiseur ne réduirait pas l'espace, seul peut-être "monde" et "mondialisation" pourraient être fusionnés (ou plutôt ramenés tous les deux à "mond") par l'utilisation d'un raciniseur particulièrement radical. Les nombres présents dans le tableau de la Figure 2.29 correspondent au calcul de $TF(i, j)$ pour le mot j dans le texte i. Les valeurs de $IDF(j)$, quant à elles, valent :

- pour les mots j présents dans un seul des textes du corpus (à savoir "critique", "ferme", "hollywood", "menacée", "métier", "mondialisation", "petit", "reconstruire", "rêves", "temps" et "usine") : $IDF(j) = ln(\frac{6}{1}) = ln(6)$
- pour les mots j présents dans deux des textes du corpus (tous les autres !) : $IDF(j) = ln(\frac{6}{2}) = ln(3)$.

Pour obtenir la représentation TF.IDF du corpus, il faut appliquer le coefficient multiplicatif $IDF(j)$ sur toutes les valeurs de la colonne j dans le tableau de la Figure 2.29. Les nouveaux nombres obtenus varieront entre 0 (quand le nombre d'occurrences $TF(i, j)$ vaut 0 dans le tableau initial) à $2 * ln(6)$, valeur maximale atteinte pour l'attribut "critique" du texte 2, qui est à la fois très présent (2 fois) dans ce texte et rare (absent) dans tous les autres.

Si maintenant on dispose d'une ontologie permettant de lier certains des attributs aux mots génériques "cinéma" et "économie", alors on peut limiter la représentation à un espace à deux dimensions, ce qui aura l'avantage d'être représentable dans un plan. Supposons donc que "art", "cinéma", "critique" et "hollywood" héritent plus ou moins directement du domaine "cinéma" tandis que "crise", "économie", "industrie", "mondialisation" et "usine" se rattachent à "économie" (les autres mots étant négligés). Alors, les coordonnées des six textes dans le nouvel espace à deux dimensions ainsi défini sont données par le tableaux de la Figure 2.30 (on revient ici à des nombres d'occurrences pour garder des nombres entiers) :

attributs \ textes	cinéma	économie
texte 1	2	1
texte 2	4	0
texte 3	1	0
texte 4	0	3
texte 5	1	2
texte 6	0	2

FIGURE 2.30 – Coordonnées des textes dans un espace à deux dimensions

Ils sont donc représentables par des points (ou des vecteurs) dans un espace cartésien de dimension deux, comme le montre la Figure 2.31. Dans les chapitres suivants (principalement ceux qui abordent la recherche d'information et la classification), nous reviendrons sur cet exemple pour illustrer sur des données simples les différents algorithmes que nous évoquerons.

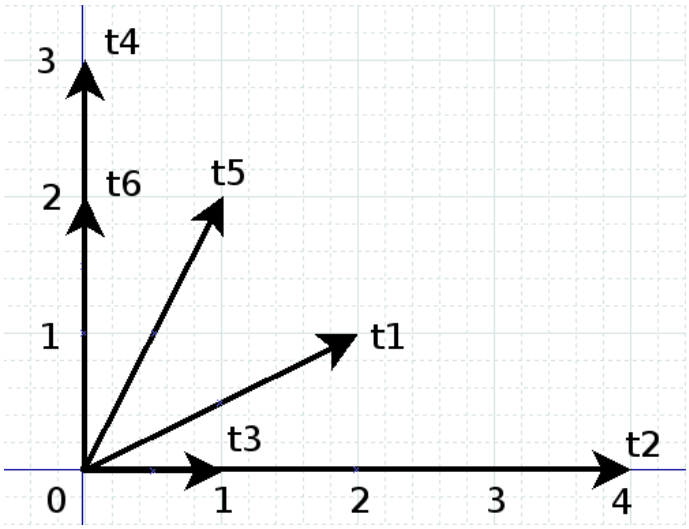


FIGURE 2.31 – Vecteurs représentant les textes dans l'espace simplifié

6 Conclusion

Nous avons voulu montrer, dans ce chapitre introductif, que la fouille de textes est un domaine à la fois vaste et homogène. Il est vaste parce qu'il couvre des types d'applications -caractérisés par des tâches- variés et apparemment sans grands rapports les uns avec les autres. Nous verrons aussi par la suite que les méthodes employées pour réaliser les tâches en question sont elles-mêmes très diverses, que certaines héritent de l'IA symbolique tandis que d'autres trouvent leurs fondements dans les probabilités, les statistiques ou encore l'optimisation. Pourtant, derrière cette apparente disparité, se cachent aussi beaucoup de points communs, qui rendent le domaine beaucoup plus homogène qu'il n'y paraît au départ. La nature textuelle des données est le premier et le plus évident d'entre eux, mais ce n'est pas le seul. Les protocoles d'acquisition et d'évaluation des programmes, la reformulation des tâches les unes dans les autres et la représentation vectorielle de la plupart des informations manipulées attestent tout autant d'un cadre commun largement partagé. Ce cadre, pour l'essentiel, est l'héritier de la fouille de données. C'est à elle et à sa maturité que l'on doit la prédominance, largement illustrée ici, de la tâche de classification et de la représentation des données en "sacs de mots". La prise en compte des spécificités linguistiques des textes est un souci plus récent qui tend à hybrider ces approches et ces méthodes avec celles issues du TAL. Les défis à relever pour approfondir cette hybridation sont encore nombreux, et les linguistes ont encore beaucoup à dire dans cette aventure. Mais pour cela, il faut absolument qu'ils comprennent et maitrisent les techniques à l'œuvre dans les programmes d'ors et déjà utilisés pour les applications citées ici. C'est pourquoi les chapitres suivants vont s'attacher à décrire ces techniques, en essayant d'insister plus sur les intuitions qui les motivent que sur l'appareillage mathématique sur lesquelles elles s'appuient.

Chapitre 3

La Recherche d'Information (RI)

1 Introduction

On l'a déjà remarqué : la Recherche d'Information (ou RI) fait partie, via les moteurs de recherche, du quotidien de tous les internautes, et est sans doute de ce fait la tâche de fouille de textes la plus populaire. Elle n'a pourtant pas attendu Internet pour exister, puisqu'elle trouve son origine dans les "logiciels documentaires", ces programmes de gestion de bibliothèques apparus dès les années 60. Le terme lui-même de "recherche d'information" est dû à l'américain Calvin Mooers et remonte à 1950, aux tout débuts de l'informatique. Le domaine n'a depuis cessé d'évoluer, et continue de faire l'objet de recherches actives. En France, la conférence annuelle Coria, organisée par l'association ARIA [1] lui est entièrement consacrée, et les compétitions et conférences internationales sur ce thème ne manquent pas.

Un système de RI, on l'a vu, a accès à un corpus stable, et se voit soumettre des requêtes auxquelles il doit répondre en y sélectionnant des documents pertinents. Pour ce faire, il doit nécessairement passer par trois étapes fondamentales :

- indexation du corpus (réalisée une seule fois, indépendamment des requêtes)
- traitement de la requête
- appariement requête/corpus

Pour chacune de ces étapes, différents choix sont possibles qui donnent lieu à différentes approches. A l'heure actuelle, trois principaux modèles coexistent pour aborder une tâche de RI : ils sont dits booléen, vectoriel et probabiliste. Nous consacrerons successivement une partie aux deux premiers d'entre eux, en les illustrant sur le corpus rudimentaire introduit dans le chapitre précédent, en section 5.5. Ces deux approches présentent la particularité de ne pas faire appel à de l'apprentissage automatique. Nous négligerons en revanche les modèles probabilistes, qui eux y ont recours, et sont de ce fait plus complexes. Nous souhaitons en effet réserver dans ce document la présentation de l'apprentissage automatique aux autres tâches (le lecteur intéressé par les modèles probabilistes pourra se reporter au livre consacré entièrement à la Recherche d'Information, cité en section bibliographie).

Mais nous ne nous en tiendrons pas pour autant à cette présentation "classique" et élémentaire de la RI. La structure d'Internet a en effet suscité des problématiques

1. www.asso-aria.org

nouvelles qui ont profondément renouvelé le domaine. L'algorithme du PageRank, en particulier, à l'origine du succès de Google, mérite d'être connu et compris car il s'est révélé pertinent bien au-delà de son domaine d'application initial. Une partie lui sera donc consacrée. Nous évoquerons aussi rapidement pour finir d'autres extensions et évolutions contemporaines de ce domaine de recherche en pleine évolution.

2 RI booléenne

La RI booléenne (parfois aussi appelée, nous verrons pourquoi, RI ensembliste) est, historiquement, la première apparue. Elle est à la base de la plupart des "logiciels documentaires" encore en usage actuellement. Son utilisation efficace requiert toutefois une relative expertise pour formuler des requêtes, qui reste la plupart du temps de la seule compétence de bibliothécaires professionnels.

2.1 Indexation par fichier inverse

La phase d'indexation, en RI booléenne, passe en général par la constitution d'un *fichier inverse*. Celui-ci indique, pour chaque terme d'indexation, les documents du corpus où il apparaît. Dans les logiciels documentaires, les "termes d'indexation" intègrent souvent les valeurs des méta-données associées au contenu de la bibliothèque à indexer (nom et prénom du ou des auteur(s) de chaque livre, date de parution, éditeur, etc.). Mais, nous l'avons vu, la fouille de textes se passe de telles informations pour se concentrer sur le seul contenu des documents eux-mêmes. Aussi, pour les textes de notre corpus d'exemples, les termes d'indexation sont simplement les tokens sélectionnés après les pré-traitements d'usage, et figurant dans le tableau de la figure 2.29. A partir de ce tableau, il est très simple de construire le fichier inverse suivant (où les numéros sont les identifiants des textes où les mots sont présents) :

```
art : 1 , 5
cinéma : 1 , 2
crise : 4 , 5
critique : 2
économie : 4 , 6
ferme : 6
france : 2 , 4
hollywood : 3
industrie : 1 , 5
menacée : 4
métier : 2
monde : 2 , 3
mondialisation : 4
petit : 2
reconstruire : 5
rêves : 3
temps : 5
```

```
usine : 6
```

On parle de "fichier inverse" parce que le rapport contenu/contenant (matérialisé par la présentation en lignes/colonnes) est en quelque sorte interverti : au lieu de présenter, comme dans le tableau 2.29, la liste des mots de chaque texte, ce fichier associe à chaque mot *l'ensemble des documents* où il figure. Notons que, dans ce fichier, les nombres d'occurrences sont ignorés : qu'un mot apparaisse une ou plusieurs fois dans un même texte ne change rien à son indexation, c'est sa seule présence qui compte.

2.2 Algèbre booléenne

La particularité de la RI booléenne est qu'elle requiert des requêtes formulées en suivant des règles particulières, issues de la *logique booléenne* et de la théorie des ensembles. Ces requêtes peuvent prendre la forme suivante :

- soit un unique terme d'indexation
- soit plusieurs termes d'indexation combinés avec les opérateurs ET, OU ou SAUF

Quand la requête est réduite à un seul terme, le fichier inverse fournit directement l'ensemble des réponses. Les opérateurs ET, OU et SAUF s'interprètent quant à eux comme des manières de combiner des ensembles entre eux, comme le montrent les dessins de la figure 3.1. ET correspond à l'*intersection* $\cap$ entre deux ensembles, OU à leur *union* $\cup$ et SAUF à la *différence* entre un ensemble et un autre. Dans ces dessins, il faut imaginer que chacun des deux ensembles est associé à un terme d'indexation, et que la zone en rouge est celle sélectionnée par la requête.

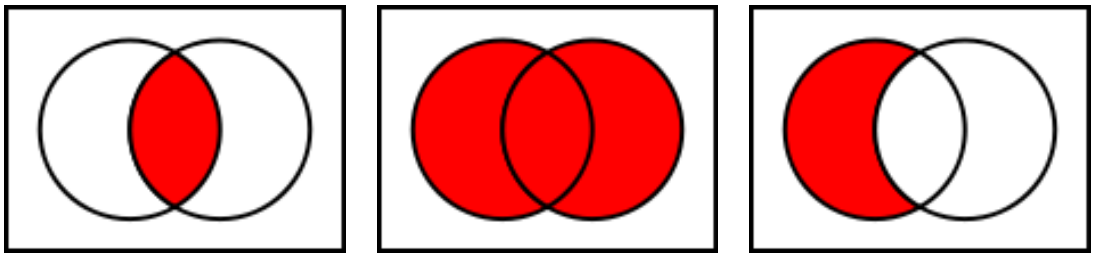


FIGURE 3.1 – Représentations ensemblistes du ET, du OU et du SAUF

Illustrons le fonctionnement de requêtes booléennes simples sur le corpus précédemment indexé :

- "crise ET industrie" : pour traiter cette requête, on doit réaliser l'intersection entre l'ensemble $\{4, 5\}$ (associé à "crise") et l'ensemble $\{1, 5\}$ (associé à "industrie"). Le seul élément présent dans chacun des deux ensembles (c'est-à-dire dans la zone rouge du ET) est le document 5, qui sera donc l'unique réponse proposée par le système de RI : $\{4, 5\} \cap \{1, 5\} = \{5\}$.
- "cinéma OU hollywood" : pour traiter cette requête, on doit réaliser l'union entre l'ensemble $\{1, 2\}$ (associé à "cinéma") et $\{3\}$ (associé à "hollywood"). La zone sélectionnée couvre l'intégralité des éléments des deux ensembles, et

la réponse fournie par le système de RI est donc l'ensemble des documents 1, 2 et 3 : $\{1,2\} \cup \{3\} = \{1,2,3\}$.

– "cinema SAUF france" : pour traiter cette requête, on doit réaliser la différence entre l'ensemble $\{1,2\}$ (associé à "cinéma") et $\{2,4\}$ (associé à "france"). La réponse obtenue est constituée des éléments du premier ensemble qui ne sont pas aussi dans le deuxième, c'est le cas uniquement du document 1 : $\{1,2\}$ SAUF $\{2,4\} = \{1\}$.

Il est bien sûr possible de combiner plusieurs critères simultanément pour exprimer des requêtes plus complexes, mais il convient dans ce cas de fixer l'ordre des opérateurs par des parenthèses, au risque sinon de ne pas maîtriser le résultat obtenu. Illustrons ce phénomène sur un exemple :

– "(économie OU monde) SAUF france" se ramène au calcul suivant :
$(\{4,6\} \cup \{2,3\})$ SAUF $\{2,4\} = \{2,3,4,6\}$ SAUF $\{2,4\} = \{3,6\}$

– "économie OU (monde SAUF france)" se ramène au calcul suivant :
$\{4,6\} \cup (\{2,3\}$ SAUF $\{2,4\}) = \{4,6\} \cup \{3\} = \{3,4,6\}$

On voit sur cet exemple que, suivant la position des parenthèses, on exprime des requêtes différentes qui ne donnent pas le même résultat.

$\cap$ (correspondant à ET), $\cup$ (correspondant à OU) et SAUF sont des opérateurs qui agissent sur des ensembles, au même titre que +, -, * et / agissent sur des nombres, mais nous ne détaillerons pas ici leurs propriétés mathématiques. On peut aussi les présenter comme des opérateurs logiques agissant sur des booléens (valant soit 0 soit 1). Dans ce cas, on les définit par la "table de vérité" de la figure 3.2 (le terme "table de vérité" provient du fait que 0/1 s'interprète aussi comme faux/vrai) .

A	B	A ET B	A OU B	A SAUF B
0	0	0	0	0
0	1	0	1	0
1	0	0	1	1
1	1	1	1	0

FIGURE 3.2 – Table de vérité des opérateurs booléens ET, OU et SAUF

Pour retrouver les mêmes résultats que précédemment sur nos exemples en utilisant cette table, il suffit de se rapporter à une version booléenne (c'est-à-dire où la valeur 2 est simplement remplacée par 1) du tableau de la figure 2.29 et de procéder comme suit :

– chercher dans le tableau de la figure 2.29 les colonnes correspondant aux termes d'indexation A et B ;

– en fonction des valeurs affectées à A et B pour chaque texte (c'est-à-dire chaque ligne du tableau), sélectionner la ligne correspondante dans la table de vérité ;

– la valeur booléenne de l'opérateur dans la table de vérité indique si le texte en question est sélectionné (si elle vaut 1) ou pas (si elle vaut 0).

Avec cette formulation booléenne, on peut même se passer de construire le fichier inverse : le tableau initial suffit. Les formulations booléenne ou ensembliste de ce système sont bien sûr équivalentes, c'est pourquoi on parle indifféremment de RI

ensembliste ou booléenne. Remarquons que, dans ce type de moteur de recherche, les étapes de traitement de la requête et de son appariement avec le corpus indexé sont en quelque sorte simultanées : la requête n'a pas vraiment à être "traitée" car elle se présente directement de façon structurée. Les formules booléennes se rapprochent en ce sens des requêtes de type SQL ou SPAQL, encore plus syntaxiquement structurées.

2.3 Intérêts et limites

Ce modèle s'avère à la fois très simple à programmer et très efficace, au moins en temps de calcul nécessaire pour obtenir un résultat. Son principal défaut est qu'il requiert de la part de son utilisateur une connaissance minimale du corpus à interroger, afin d'utiliser les bons termes d'indexation, et une certaine expertise pour la formulation de ses requêtes. Il n'autorise pas (comme ce sera le cas pour le modèle vectoriel) à se servir d'un texte comme requête. Si ces conditions d'utilisation sont réunies, le système peut permettre d'atteindre une bonne précision, même si le rappel risque d'être plus faible. Il est en effet difficile d'imaginer *a priori* toutes les combinaisons pertinentes possibles de termes caractérisant une demande.

Pour remédier à ce défaut, les "sciences de l'information et de la documentation" ont développé différents dispositifs d'extensions de requêtes via des *thesaurus*. Un thesaurus est une hiérarchie de termes structurée par des relations sémantiques, permettant de faire le lien entre ceux utilisés en indexation et ceux utilisés dans une requête. Mais constituer et exploiter intelligemment un thesaurus est délicat, son usage est encore réservé aux spécialistes. Et personne n'envisage d'en construire un qui serait exploitable par un moteur de recherche généraliste : les seuls thesaurus disponibles sont restreints à des champs terminologiques précis, caractérisant des domaine spécialisés. La RI booléenne reste de ce fait confinée à l'interrogation de bases documentaires homogènes structurées en méta-données, dont certaines peuvent donner lieu à un thesaurus.

L'autre handicap des systèmes de RI booléens, notamment pour une exploitation sur de grandes bases de textes, est que la réponse qu'ils fournissent à une requête est... booléenne ! Pour eux, les documents sont soit pertinents soit non pertinents, sans ordre et sans nuance possible entre les deux. Sur certaines requêtes, les ex-aequo vont être nombreux...

3 RI vectorielle

Les systèmes de RI vectoriels sont actuellement les plus répandus. Leur principe est de transformer le corpus et les requêtes en vecteurs dans un même espace, et d'utiliser une mesure de proximité pour trouver les textes les plus "proches" de celui servant de question. Tous les outils techniques nécessaires à ce processus ont déjà été présentés en partie 5 du chapitre précédent, nous nous contenterons donc ici de l'illustrer sur quelques exemples.

3.1 Principe et illustration

Reprenons les étapes évoquées en introduction dans le cas des moteurs de recherche vectoriels :

- indexation : le corpus, une fois pré-traité (segmentation, élimination des mots vides ou inutiles, normalisation, éventuellement lemmatisation, etc.) permet de définir un espace vectoriel (c'est-à-dire un ensemble d'attributs supposés indépendants les uns des autres, chacun constituant une dimension de cet espace). Chaque texte du corpus est représenté dans cet espace par un vecteur (nous avons vu aussi les différents choix possibles pour cela : en booléens, en nombre d'occurrences, en TF.IDF, etc.).
- traitement de la requête : chaque requête est ensuite pré-traitée de la même façon que les textes du corpus, et elle est représentée suivant la même convention *dans l'espace précédent*. Cela signifie en particulier que si des mots (ou tokens) absents des textes du corpus y figurent, ils ne sont pas pris en compte dans sa représentation car aucune dimension de l'espace ne leur est consacrée. Comme ces unités sont absentes du corpus initial, elles ne servent à rien pour trouver les textes de ce corpus qui répondent à la requête.
- appariement requête/corpus : le vecteur représentant la requête est comparé systématiquement à chacun des vecteurs représentant le corpus, selon une des mesures de proximité/distance donnée précédemment : les textes les plus "proches" suivant cette mesure sont proposés par le moteur de recherche.

Prenons comme exemple le corpus de la section 5.5 du chapitre précédent, et la requête suivante : "Pendant la crise, l'usine à rêves Hollywood critique le cynisme de l'industrie." Une fois pré-traitée, cette requête est ramenée à 1 occurrence de chacun des termes suivants : "crise", "critique", "hollywood", "industrie", "rêves" et "usine" (et aucune occurrence pour les autres attributs du corpus). Notons que sans étiquetage POS, on ne peut distinguer le terme "critique" en tant que nom commun (dans le corpus) et en tant que verbe conjugué (dans la requête), ils ne font qu'un seul et même attribut ambigu. En absence de répétition d'aucun terme d'indexation dans la requête, sa représentation booléenne et sa représentation en nombre d'occurrences sont identiques. La figure 3.3 donne les mesures de proximité de cette requête avec chacun des vecteurs qui représentent le corpus, dans deux cas différents.

texte	booléen + Dice	nb. occ. + cosinus
texte 1	$\frac{1}{3+6} = 0,1111$	$\frac{1}{\sqrt{3}\sqrt{6}} = 0,2357$
texte 2	$\frac{1}{6+6} = 0,0833$	$\frac{2}{\sqrt{12}\sqrt{6}} = \frac{1}{\sqrt{3}\sqrt{6}} = 0,2357$
texte 3	$\frac{2}{3+6} = 0,2222$	$\frac{2}{\sqrt{3}\sqrt{6}} = 0,4714$
texte 4	$\frac{1}{5+6} = 0,0909$	$\frac{1}{\sqrt{5}\sqrt{6}} = 0,1826$
texte 5	$\frac{2}{5+6} = 0,1818$	$\frac{2}{\sqrt{5}\sqrt{6}} = 0,3651$
texte 6	$\frac{1}{3+6} = 0,1111$	$\frac{1}{\sqrt{3}\sqrt{6}} = 0,2357$

FIGURE 3.3 – calculs de proximités entre la requête et les textes

On constate sur cet exemple que, dans les deux cas, le texte considéré comme le plus proche de la requête est le texte 3, suivi du texte 5, puis ex-aequo le 1 et le 6 (et le 2 dans le deuxième cas uniquement). Le texte 4 se retrouve classé soit en 5ème position (devant le 2), soit en 6ème et dernière position. Avec d'autres valeurs, les positions relatives des textes auraient pu être d'avantage modifiées suivant la représentation et la mesure utilisées.

Si, maintenant, on fait subir au vecteur représentant la requête le même traitement à base d'une ontologie que celui ayant donné lieu aux coordonnées de la figure 2.30 et au dessin de la figure 2.31, alors on le ramène lui aussi dans cet espace à deux dimensions (la dimension "cinéma" et la dimension "économie"). Dans cet espace, il a pour coordonnées (2,3) car deux de ses mots relèvent du domaine "cinéma" ("Hollywood" et "critique", même si c'est pour de mauvaises raisons !), et trois de l'"économie" ("crise", "usine" et "industrie"), le mot "cynisme", absent du corpus initial, n'est pas pris en compte. La figure 3.4 reprend la figure 2.31 en y ajoutant (en rouge) le vecteur représentant la requête. Cette fois, la mesure "cosinus" y est évaluable à l'œil nu via les angles. Les textes sont alors ordonnés comme suit, du plus proche au plus lointain : 5, 1 (de très peu !), 4 et 6 ex-aequo, 2 et 3 ex-aequo. Cet exemple rudimentaire montre bien que, en fonction des choix effectués au niveau de la représentation des textes en vecteurs et des calculs de proximité, le comportement d'un moteur de recherche vectoriel peut changer assez considérablement.

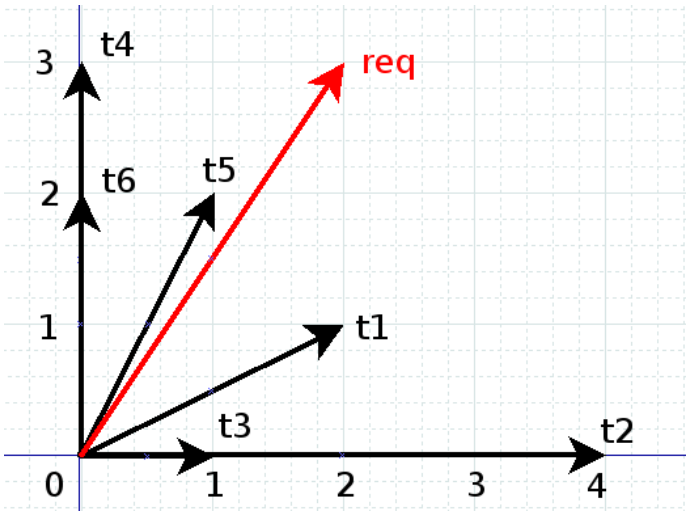


FIGURE 3.4 – Vecteurs représentant les textes et la requête dans l'espace simplifié

3.2 Intérêts et limites

Les moteurs de recherche vectoriels présentent plusieurs avantages, notamment vis-à-vis de moteurs booléens. Dans ces systèmes, en effet, les textes et les requêtes sont traités exactement de la même manière : aucune expertise n'est requise pour formuler les requêtes, et un texte complet peut parfaitement jouer cet ce rôle. L'utilisateur n'a pas à savoir comment le corpus est indexé pour l'interroger. Ces systèmes sont par ailleurs tout aussi faciles à programmer que les systèmes booléens, et tout

aussi efficaces en temps de calculs. Les résultats qu'ils produisent sont aussi plus intéressants, parce que plus souples et plus nuancés que dans les systèmes booléens : les diverses mesures de proximité possibles prennent une plage de valeurs qui n'est pas réduite à 0/1, ce qui permet d'ordonner beaucoup plus finement les textes, du plus proche au plus lointain de la requête. Nous avons vu et illustré également que, via des pré-traitements plus ou moins radicaux, des ressources linguistiques de différentes natures pouvaient facilement être intégrées et prises en compte dans le processus d'indexation et de traitement de la requête. Cela laisse la place à de nombreuses expériences possibles, où la fouille de textes et la linguistique peuvent interagir...

4 L'algorithme PageRank

L'algorithme PageRank a fait la notoriété de Google. Les techniques spécifiques utilisées par ce moteur de recherche ne se limitent plus depuis longtemps à ce seul paramètre, mais le PageRank a trouvé de nombreux autres domaines d'applications, et mérite donc pour cela d'être connu.

4.1 L'objectif du PageRank

L'objectif du PageRank n'est pas du tout d'apparier une requête avec un document, mais d'ordonner par ordre d'importance (ou de notoriété) les éléments d'un réseau, représenté par un *graphe*. Les graphes sont des objets mathématiques très simples et très utiles, composés de *nœuds* et d'*arcs* (ou flèches) qui les relient. Ils permettent de modéliser de nombreux phénomènes de la vie courante, que ce soit les réseaux routiers (ou ferroviaires ou aériens...), les liens entre personnes ou... le Web ! Pour considérer le Web comme un graphe, il suffit en effet de voir chacune des pages HTML qui le constituent comme autant de nœuds, tandis que les liens hypertextes qui renvoient d'une page à une autre jouent le rôle des arcs.

Le schéma de la Figure 3.5 montre un graphe comprenant quatre nœuds numérotés de 1 à 4 reliés entre eux : il peut correspondre à un petit morceau du Web contenant quatre pages HTML et autant de liens hypertextes entre ces pages qu'il y a d'arcs dans le graphe. Dans le cadre d'un système de recherche d'information, on peut imaginer que parmi ces pages Web, certaines sont des réponses pertinentes à une requête. L'objectif du PageRank est de fixer l'ordre d'importance de ces différentes réponses possibles sur la base non pas de leur contenu (le contenu a déjà servi à les considérer comme pertinentes), mais de leur crédibilité, évaluée en tenant compte de leurs positions relatives dans le graphe. L'idée de base est simple : quand une page A contient un lien hypertexte vers une page B, cela signifie que A donne un certain crédit au contenu de B, elle lui accorde une certaine valeur, ou crédibilité. C'est comme si A "votait" pour B. Mais ce vote a d'autant plus de poids que la page A est elle-même considérée comme crédible par les autres membres du réseau, c'est-à-dire qu'elle-même reçoit des liens en provenance des autres nœuds. On voit que la crédibilité de chacun des nœuds dépend de celle de chacun des autres, ce qui en fait une notion récursive. Comment, donc, mesurer précisément cette crédibilité d'un nœud dans un graphe ? C'est exactement ce que permet la formule du PageRank.

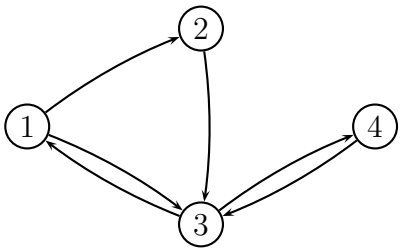


FIGURE 3.5 – graphe modélisant un petit morceau de Web

Notons au passage que le nom de cet algorithme entretient savamment l'ambiguité : il permet de "ranger" (rank : ordonner en anglais) les pages en fonction de leur importance, mais il fait aussi référence à Larry Page, son auteur, co-fondateur de Google avec Sergei Brin...

La formule permettant de calculer $PR(A)$, le PageRank du nœud A dans un graphe, en fonction des PageRank $PR(E_i)$ $(1 \leq i \leq n)$ des n autres nœuds E_1, E_2... E_n du graphe qui sont "entrants" par rapport à A, c'est-à-dire qui envoient au moins un lien hypertexte aboutissant à A, est la suivante :

$$\begin{aligned} PR(A) &= \tfrac{1-d}{n+1} + d\sum_{i=1}^{n} \tfrac{PR(E_i)}{l_i} \\ &= \tfrac{1-d}{n+1} + \tfrac{d}{l_1}PR(E_1) + \tfrac{d}{l_2}PR(E_2) + ... + \tfrac{d}{l_n}PR(E_n) \end{aligned}$$

où on a :

- $n+1$ est le nombre total de nœuds du graphe (les n nœuds E_i plus A) ;
- pour chaque nœud E_i $(1 \leq i \leq n)$, l_i est le nombre de *liens sortants* (flèches qui partent) du nœud E_i ;
- d est un *facteur d'amortissement* qui permet de normaliser les valeurs (pour qu'elles somment à 1) tout en évitant certains pièges.

Nous expliquons dans la suite le sens des différents éléments qui interviennent dans ce calcul, et comment il est réalisé en pratique.

4.2 Calculs et signification

La partie la plus facile à expliquer de la formule du PageRank est celle qui fait intervenir la somme, sans tenir compte de d et n. Pour bien la comprendre, imaginons une version simplifiée de la formule permettant de calculer la crédibilité c d'un nœud d'un graphe en fonction des crédibilités c_i des nœuds dont il reçoit des liens :

$$\begin{aligned} c &= \sum_{i=1}^{n} \tfrac{c_i}{l_i} \\ &= \tfrac{c_1}{l_1} + \tfrac{c_2}{l_2} + ... + \tfrac{c_n}{l_n} \end{aligned}$$

Tout se passe dans ce cas comme si chaque nœud contenant un lien vers notre nœud initial répartissait équitablement sa propre crédibilité c_i entre tous les nœuds vers lesquels il pointe (il y en a l_i), pour accorder à chacun d'eux un "vote" valant $\frac{c_i}{l_i}$. La crédibilité du nœud récepteur initial se calcule alors en sommant tous les "morceaux de crédibilité" qu'il reçoit de ses liens rentrants. Dans notre exemple de la Figure 3.5, on aurait donc la série d'équations donnée dans la Figure 3.6.

$$\begin{cases} c_1 = \frac{1}{2}c_3 \\ c_2 = \frac{1}{2}c_1 \\ c_3 = \frac{1}{2}c_1 + c_2 + c_4 \\ c_4 = \frac{1}{2}c_3 \end{cases}$$

FIGURE 3.6 – équations de "crédibilité" du graphe de la Figure 3.5

	nœud 1	nœud 2	nœud 3	nœud 4
$t = 0$	0	1	0	0
$t = 1$	0	0	1	0
$t = 2$	$\frac{1}{2} = 0,5$	0	0	$\frac{1}{2} = 0,5$
$t = 3$	0	$\frac{1}{4} = 0,25$	$\frac{3}{4} = 0,75$	0
...	...	...	...	...
$t = 10$	0,228...	0,105...	0,437...	0,228...
...	...	...	...	...
$t = 20$	0,222...	0,111...	0,444...	0,222...

FIGURE 3.7 – valeurs successives de la probabilité de présence

On obtient ainsi 4 équations linéaires à 4 inconnues : c'est solvable mais plus il y aura de nœuds (et le nombre de pages du Web se compte en milliards...), plus ce sera difficile ! Dans cet exemple, si on met la valeur 0 à tous les c_i, l'ensemble des équations est satisfaite. Mais il y a une autre solution bien plus intéressante, qui consiste à donner les valeurs suivantes aux inconnues : $(2, 1, 4, 2)$, c'est-à-dire $c_1 = 2 = c_4, c_2 = 1, c_3 = 4$. Cette solution a l'avantage de bien montrer comment se répartit la crédibilité dans notre graphe : le nœud 3, qui est le seul à recevoir trois liens, obtient la valeur maximale de 4 ; les nœuds 1 et 4, qui reçoivent un lien en provenance de 3, bénéficient de sa notoriété et sont de ce fait à 2, tandis que le nœud 2, un peu isolé, n'atteint que la valeur 1.

Plutôt que de chercher à résoudre l'équation avec des méthodes algébriques, on peut aussi en donner une interprétation en termes de probabilité de présence dans les nœuds du graphe. Imaginons pour cela un internaute en train de visiter à l'instant $t = 0$ une des pages, par exemple la page correspondant au nœud 2. A chaque instant, il clique au hasard sur un des liens hypertextes de la page en question. Comme il n'existe qu'un lien sortant de 2 pour aller en 3, à l'instant $t = 1$, l'internaute sera nécessairement en train de visiter la page 3. Comme il clique au hasard, à $t = 2$ il a 1 chance sur 2 de se trouver en 1, et une chance sur 2 de se trouver en 4. La suite des valeurs des probabilités de présence dans les nœuds 1 à 4 à chaque instant t est donnée dans le tableau de la Figure 3.7.

Pour calculer chaque ligne de ce tableau, il suffit d'utiliser les équations de la Figure 3.6 comme si elles expliquaient comment obtenir la nouvelle valeur de chaque c_i en fonction des valeurs des crédibilités de la ligne précédente (c'est-à-dire de l'instant d'avant). Par exemple, l'équation $c_1 = \frac{1}{2}c_3$ signifie dans ce cas que la nouvelle

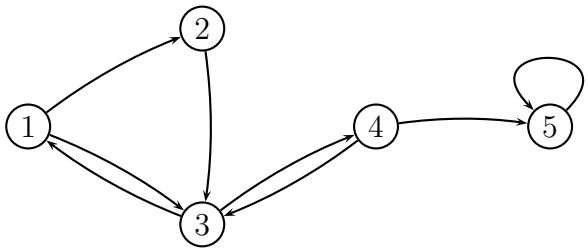


FIGURE 3.8 – graphe modélisant un petit morceau de Web avec une "page piège"

valeur de c_1 vaut la moitié de l'ancienne valeur (à l'instant d'avant) de c_3, et ainsi de suite. On constate en effectuant ces calculs que les valeurs obtenues se rapprochent de plus en plus de $\frac{2}{9} = 0,222...$ pour les nœuds 1 et 4, $\frac{1}{9} = 0,111...$ pour le nœud 2 et $\frac{4}{9} = 0,444...$ pour le nœud 3, c'est-à-dire exactement les valeurs de la solution non nulle du système d'équations que nous avions proposée, normalisées pour que leur somme soit 1 : $(\frac{2}{9}, \frac{1}{9}, \frac{4}{9}, \frac{2}{9})$. On peut vérifier aussi qu'on aboutirait exactement aux mêmes valeurs de "probabilité de présence" en supposant que notre internaute commence sa navigation à partir d'une autre page du réseau initial.

Cette propriété est extrêmement intéressante. Elle montre que la notion de "crédibilité" d'une page Web que nous avons définie avec notre formule simplifiée coïncide (à une normalisation près) avec la probabilité de se trouver sur cette page au bout d'une navigation aléatoire. Cela renforce la signification des valeurs obtenues. Elle nous donne de plus un moyen simple de résoudre le système d'équations permettant le calcul de ces crédibilités.

Pourquoi, alors, cette formule simplifiée ne suffit-elle pas à calculer le PageRank ? La similitude avec le comportement d'un internaute cliquant au hasard va nous aider à le comprendre. Supposons en effet qu'une des pages du réseau ne contienne aucun lien sortant ou, pire, un unique lien bouclant sur la page elle-même, comme le nœud 5 dans le dessin modifié de la Figure 3.8.

Dès que l'internaute aura atteint cette page 5 (et il finira toujours par y arriver, en cliquant au hasard), il en restera prisonnier puisqu'il ne pourra plus en repartir. Si l'on s'en tient à notre mode de calcul précédent, cela signifie que les probabilités de présence vont être "avalées" par la page "piège" dont la probabilité va tendre vers 1, tandis que celles de tous les autres nœuds tendront vers 0. Pourtant, nous n'avons pas envie d'attribuer à cette page piège la "crédibilité" maximale. D'où l'idée que, lors d'une navigation au hasard sur le Web, il est toujours possible de visiter n'importe quelle page à partir de n'importe quelle autre par "téléportation", c'est-à-dire sans nécessairement suivre un lien (avec un navigateur, il suffit de taper directement l'adresse de la destination). Le facteur d intervenant dans la formule du PageRank est destiné précisément à modéliser cette possibilité. Si on fixe $d = 1$ dans la formule du PageRank de la section 4.1, on retombe exactement sur notre formule simplifiée. En prenant une valeur de d comprise entre 0 et 1, cela revient à autoriser avec une probabilité non nulle $1 - d$ à "sauter" d'une page à une autre sans suivre de lien. Typiquement, on prend généralement la valeur $d = 0,85$, ce qui revient à autoriser une téléportation avec une probabilité de $1 - d = 0,15$, soit environ 1 fois

$$\begin{cases} PR(1) = 0,0375 + 0,85 * \frac{1}{2} * PR(3) \\ PR(2) = 0,0375 + 0,85 * \frac{1}{2} * PR(1) \\ PR(3) = 0,0375 + 0,85 * \frac{1}{2} * PR(1) + 0,85 * PR(2) + 0,85 * PR(4) \\ PR(4) = 0,0375 + 0,85 * \frac{1}{2} * PR(3) \end{cases}$$

FIGURE 3.9 – équations de PageRank du graphe de la Figure 3.5

	nœud 1	nœud 2	nœud 3	nœud 4
$t = 0$	0	0	0	0
$t = 1$	0,0375	0,0375	0,0375	0,0375
$t = 2$	0,0534	0,0534	0,1172	0,0534
$t = 3$	0,0873	0,0602	0,1510	0,0873
...	...	...	...	...
$t = 10$	0,1761	0,1091	0,3417	0,1761
...	...	...	...	...
$t = 40$	0,2196	0,1308	0,4285	0,2196

FIGURE 3.10 – valeurs successives du PageRank PR

sur 6, répartie équitablement sur les $n + 1$ nœuds du graphe.

Avec la formule complète du PageRank, on obtient pour le graphe de la Figure 3.5 l'ensemble d'équations de la Figure 3.9 (où $0,0375 = \frac{1-d}{n+1} = \frac{1-0,85}{4}$).

Pour résoudre ce système, on peut procéder comme précédemment :

- à un instant $t = 0$, on initialise les valeurs de PR(1) à PR(4) comme on veut (par exemple : toutes à 0, ou toutes à 1...) ;
- on utilise les équations pour calculer les valeurs des différents PR à l'instant $t + 1$ en fonction de celles disponibles à l'instant t ;
- ces valeurs finissent par converger vers des solutions de l'équation de départ sommant à 1, qui sont les valeurs finales des PR .

Dans notre exemple, en initialisant tous les PR à 0, on obtient la succession de valeurs donnée par la figure 3.10. On estime qu'environ 40 itérations suffisent pour permettre aux valeurs de se stabiliser. On constate qu'avec ce calcul, le nœud 3 continue de recevoir la meilleure valeur de PageRank, suivi des nœuds 1 et 4 ex-aequo, puis du nœud 2.

Des sites Web [2] permettent de définir la structure d'un graphe en cliquant sur des items, et de calculer automatiquement la valeur du PageRank de chacun de ses nœuds. La Figure 3.11 est une copie d'écran du calcul effectué par ce type de site sur le graphe de la Figure 3.5. Les valeurs coïncident avec celles de notre précédent calcul, à une normalisation près (les PageRank de nos formules somment à 1, ceux du site somment au nombre $n + 1$ de nœuds du graphe, 4 ici).

2. par exemple : http ://www.webworkshop.net/pagerank_calculator.php3

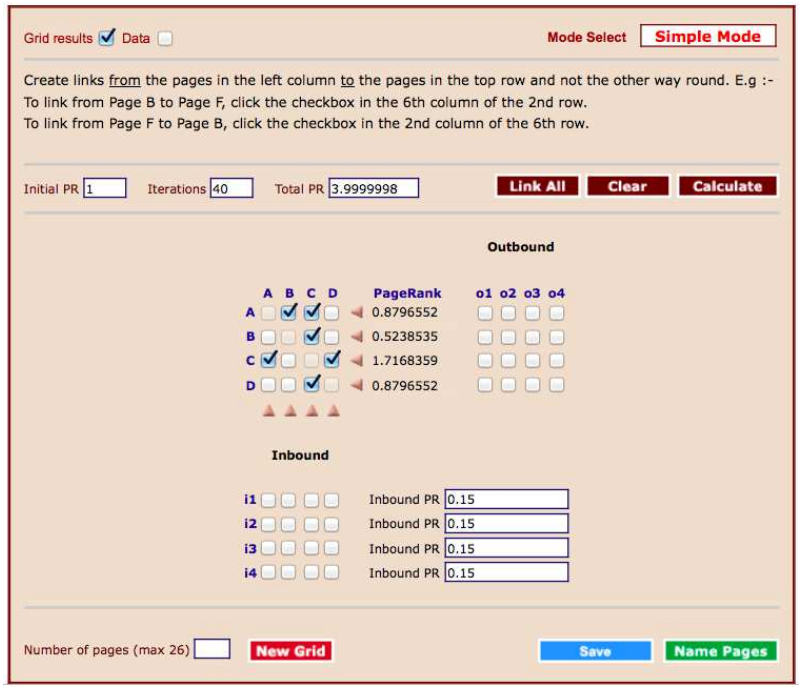


FIGURE 3.11 – Site de calculateur de PageRank paramétré avec le graphe exemple

4.3 Utilisations du PageRank

En fait, la vraie valeur utilisée par Google comme PageRank est un nombre entier entre 0 et 10 obtenu en prenant la partie entière du logarithme (par exemple en base 10, mais d'autres choix sont possibles) de la valeur ainsi calculée. Autrement dit, les valeurs calculées comprises entre 0 et 1 sont ramenées à 0, celles entre 1 et 9 à 1, entre 11 et 99 à 2, etc. Pour gagner 1 point de "vrai" PageRank suivant ce mode de calcul, il faut être 10 fois meilleur qu'avant avec les formules !

A pertinences de contenus égales, les PageRanks des pages Web sont donc utilisés pour définir l'ordre de présentation des réponses à une requête. Cela favorise les sites "de référence", ceux auxquels les autres sites renvoient. En fait, Google ne rend pas public l'ensemble de tous les paramètres qui lui servent à classer les pages qui satisfont une requête. Le PageRank est l'un d'eux, mais ce n'est pas le seul. C'est évidemment un enjeu commercial énorme, car seules les premières réponses d'un moteur de recherche sont en général consultées par les utilisateurs. Les sites marchands utilisent toutes sortes de stratégies pour améliorer leur PageRank, donc leur référencement par Google, qui garde pour cela une partie de ses critères secrets.

Mais on peut aller plus loin. L'intérêt de l'algorithme PageRank est qu'il est indépendant du domaine d'application visé : dès que ce domaine peut être modélisé sous la forme d'un graphe, il est possible de déterminer grâce au PageRank l'importance relative des différents nœuds qui le constituent. Même si nous nous éloignons de la fouille de textes, il nous semble intéressant de signaler ici certaines de ces applications.

Les réseaux d'entités qui se citent les unes les autres ne se limitent pas aux pages Web. On peut aussi faire rentrer dans ce cadre d'autres domaines, par exemple :

– les sections "bibliographie" des articles scientifiques font référence à d'autres articles. Appliquer le PageRank sur un graphe modélisant un réseau d'articles reliés par des relations de citations permet d'identifier les articles de référence, ceux qui ont vraiment influencé les autres ;
– les réseaux constitués par les liens entre blogs, les relations d'amitié dans les réseaux sociaux ou celles des "followers" dans Twitter peuvent aussi se représenter par des graphes. PageRank permet de repérer les acteurs clés de ces systèmes, d'identifier parmi leurs membres les vrais "influenceurs".

Il existe aussi des travaux plus originaux qui ont exploité PageRank dans des contextes plus inattendus. Par exemple, dans un dictionnaire, les mots sont définis par d'autres mots. On peut donc construire un graphe en prenant les mots (non vides) comme nœuds et en les reliant à ceux utilisés dans leur définition par des arcs. En calculant le PageRank des mots ainsi reliés entre eux, on peut associer des poids au lexique, et donc repérer automatiquement les mots importants d'une langue, ceux qui servent le plus souvent à définir les autres. Une autre application originale du PageRank a consisté à modéliser les relations de prédation entre espèces animales d'un même écosystème par un graphe dont les animaux sont les nœuds, reliés entre eux par un arc quand ils jouent les uns pour les autres les rôles de prédateur/proie. En calculant le PageRank des animaux de ce réseau, on repère les espèces "cruciales", celles dont la survie d'autres animaux dépend le plus, et donc ceux dont la disparition causerait le plus de tort à l'ensemble de l'écosystème.

5 Conclusion

La RI est une tâche clé aux très forts enjeux commerciaux. Tout site marchand y est confronté, à la fois en tant que "client", référencé par les moteurs de recherche généralistes du Web, et en tant que "fournisseur", pour la navigation dans ses pages à l'aide d'un moteur interne. Elle est donc au cœur des préoccupations de nombreuses sociétés. Quand Google modifie son algorithme de classement des sites (comme cela arrive régulièrement), certaines d'entre elles, qui dépendent de leur visibilité sur le Web pour prospérer, peuvent se retrouver en difficulté économique...

Nous n'avons présenté ici que les méthodes de base de la recherche d'information. En plus des approches statistiques (évoquées mais pas développées dans ce document), de nombreuses autres techniques sont couramment utilisées à l'heure actuelle pour améliorer les résultats d'une requête. Par exemple :
– l'"expansion de requête" vise à augmenter le rappel en enrichissant une requête avec des termes synonymes ou proches sémantiquement, susceptibles de se trouver dans les textes pertinents. Elle fait en général appel à des ressources linguistiques (dictionnaires, ontologies...) ;
– le "relevance feedback" (parfois traduit "retour de pertinence" en français) s'appuie sur un retour de l'utilisateur chargé de désigner, parmi les résultats présentés, lesquels sont vraiment pertinents pour lui. Ce "retour" sert à affiner la requête et à relancer une nouvelle recherche.

La problématique de l'évaluation des moteurs de recherche est également loin d'être réglée : la simple distinction pertinent/non pertinent n'est en effet pas suffisante

pour qualifier un résultat. Des effets de degré (c'est l'ordre de classement qui compte surtout) ou de contexte (certains résultats sont pertinents pour certains utilisateurs et pas pour d'autres...) doivent aussi être pris en compte.

La recherche d'information, on le voit, est un domaine vaste et quasi autonome par rapport à la fouille de textes. Elle ne concerne d'ailleurs pas que les textes : des moteurs de recherche spécialisés dans les images ou les morceaux musicaux existent aussi. Quand les données à explorer présentent de nouvelles propriétés, on a vu que la RI gagne à être combinée à des techniques qui exploitent ces propriétés. Ainsi, le PageRank, basé sur la structure en graphe du Web, est très efficace pour quantifier l'importance relative des pages qui figurent dans le réseau. La RI s'applique maintenant dans les réseaux sociaux, où elle gagne aussi à exploiter les relations d'amitié déclarées présentes dans ces réseaux. Des problématiques nouvelles naissent ainsi dès que les données à traiter s'enrichissent et se diversifient.

De nombreux auteurs associent aussi RI et classification : la classification de documents (que l'on traite dans le chapitre suivant) peut en effet souvent être utilisée comme un préalable à leur indexation. La RI peut aussi interagir avec la traduction automatique, quand les documents qu'elle doit manipuler sont dans plusieurs langues distinctes. Elle entretient également des liens forts avec les systèmes question/réponse, qui peuvent apparaître comme des extensions des moteurs de recherche (cf. section 4.2 du chapitre précédent), et aussi avec les systèmes de recommandation (que nous n'aborderons pas ici), qui visent à suggérer des éléments d'informations à un utilisateur sans même qu'il les ait demandés.

Pour toutes ces raisons, la recherche d'information est un domaine très actif qui mobilise l'attention de nombreux chercheurs académiques ou industriels. Des compétitions internationales (TREC, NTCIR, CLEF, INEX...) sont régulièrement organisées pour comparer les performances de différents moteurs sur des données communes. C'est un domaine encore certainement appelé à de grands développements dans les années qui viennent.

Pages Web

Pour approfondir les sujets abordés ici (en plus des livres cités en section bibliographie), voici quelques pages Web pertinentes :

- le contenu du livre de référence sur la RI en anglais est disponible gratuitement sur Internet : http ://www-nlp.stanford.edu/IR-book/
- des transparents présentés lors d'une "école d'automne en RI" en 2012 : http ://www.asso-aria.org/earia2012/programme
  et en 2014 : http ://www.asso-aria.org/earia2014/programmeearia2014
- la partie sur PageRank est inspirée de sites disponibles sur Internet :
  - un article sur un (excellent) site de vulgarisation de l'informatique : https ://interstices.info/jcms/c_47076/comment-google-classe-les-pages-web
  - un article sur un site plus commercial (orienté "référencement"), mais assez pédagogique et avec des exemples à reproduire : http ://www.webmaster-hub.com/publication/L-algorithme-du-PageRank-explique.html

Chapitre 4

La Classification

1 Introduction

Comme on l'a vu, notamment en section 4.1, la tâche de classification est centrale en fouille de textes, parce que toutes les autres tâches évoquées peuvent se ramener à elle. Par sa simplicité (associer un résultat factuel unique à une donnée, à choisir parmi un ensemble fini de réponses possibles), elle constitue en quelque sorte la pierre angulaire de nombreux traitements, et donne lieu à de très nombreuses applications. Rappelons les exemples des mails à classer en spam/non spam, la reconnaissance du caractère positif/négatif d'un texte d'opinion ou encore l'identification de tout ce qui peut correspondre à une méta-donnée associée à un document (son auteur, sa date d'écriture, son domaine, son genre littéraire, sa variante linguistique...) du moment que les valeurs possibles du résultat attendu soient en nombre fini.

Dans ces exemples, la donnée est toujours un texte complet. Mais la classification peut aussi s'appliquer à des données tabulaires, comme dans le tableau de la figure 2.20 où il s'agit d'associer à chaque mot d'un texte son étiquette morpho-syntaxique, en tenant compte de propriétés intrinsèques (le fait de commencer par une majuscule, de contenir des chiffres, etc.) ou contextuelles (sa position dans la phrase, les mots précédents ou suivants, etc.). Dans tous les cas, nous verrons que, dès que de l'apprentissage automatique -supervisé ou non- est utilisé, la tâche de classification se ramène à trouver la valeur d'un champ symbolique dans un tableau de données : typiquement, la dernière colonne d'un tableau dont les autres colonnes sont connues (voir aussi le tableau extrait du logiciel Weka, Figure 2.2).

Comme les données tabulaires peuvent provenir de quantités de domaines divers, la classification n'est en rien une tâche spécifiquement linguistique. Elle intéresse aussi les banquiers, les médecins, les professionnels du marketing ou de l'assurance (qui, tous, traitent des données numériques décrivant des individus), mais aussi les spécialistes de l'analyse des images (qui cherchent à reconnaître le contenu d'images pixellisées), par exemple. De ce fait, cette tâche est aussi celle qui, historiquement, a suscité le plus de travaux en fouille de données, et pour laquelle existent donc un très grand nombre d'algorithmes et de programmes. Ce sont les mêmes programmes qui sont utilisés, quelle que soit la nature des données stockées dans les tableaux en question. .

Terminons par un point de vocabulaire : suivant la communauté d'origine (infor-

matique, statistique...), le terme de "classification" n'est pas toujours utilisé de la même façon. Nous l'utilisons ici pour désigner une tâche générique, indépendamment de la façon de l'implémenter dans un programme. Les statisticiens ont tendance à assimiler "classification" et "clustering" (ce que nous nommerons "classification par apprentissage non supervisé"), tandis que les informaticiens la ramènent plutôt à l'application de méthodes d'apprentissage automatique supervisé. Nous décrirons l'ensemble de ces techniques dans la suite de ce document, ce qui contribuera à mieux en faire comprendre les différences. Mais le début de ce chapitre sera tout d'abord consacré à l'écriture "manuelle" de programmes de classification.

2 Classification par programme "manuel"

Rien n'empêche, même si c'est devenu de plus en plus rare, de construire un programme de classification "à la main" pour un texte. Dans ce cas, c'est donc un humain qui définit à l'avance les critères qui permettront d'associer à chaque donnée possible la classe qui lui correspond. Une telle stratégie n'est applicable que dans des domaines bien définis pour lesquels le programmeur dispose d'une bonne expertise, ou d'un ensemble de ressources aussi exhaustives que possible. Elle ne nécessite pas de pré-traiter le texte en le transformant en tableau, comme ce sera le cas des techniques d'apprentissage automatique.

Par exemple, s'il s'agit de reconnaître les courriers indésirables à partir de leur seul contenu textuel, on aura envie de constituer une liste des mots apparaissant spécifiquement dans ce type de messages. Dès qu'un mail contiendra un certain nombre de ces mots (on peut fixer un seuil plus ou moins arbitrairement), il sera rangé dans les "spams".

De même, pour créer un classifieur d'opinion, une stratégie pertinente possible consiste à recueillir dans deux listes distinctes les mots porteurs d'opinion "positive" et ceux porteurs d'opinion "négative". Pour cela, il est courant d'initialiser les listes avec un petit nombre de mots (en général, principalement des adjectifs et des verbes) fortement connotés dans un sens ou un autre, puis d'utiliser des lexiques où des relations de synonymie (voire d'antinomie) sont présentes, pour enrichir chacune de ces deux listes : les synonymes enrichissent la liste initiale, les antonymes enrichissent la liste "opposée". Tout cela requiert bien sûr de disposer de tels lexiques, et de vérifier qu'un mot ne figure pas dans les deux listes en même temps. Un texte sera ensuite évalué en "positif" ou "négatif" suivant qu'il contient plus de représentants d'une des listes que de l'autre. Cette stratégie présente l'inconvénient de ne pas traiter spécifiquement les négations (qui ont le pouvoir d'"inverser" les polarités), ni les situations d'ironie par exemple.

Dans les deux cas, on le voit, la qualité du programme ainsi construit dépendra fortement de la qualité des listes sur lesquelles il est fondé, ou à défaut des critères que le programmeur y aura inséré grâce à son expertise du domaine. De tels programmes sont évidemment très dépendants de ce domaine, ainsi que de la langue des textes traités, et sont fixes dans le temps, alors que de nombreux domaines présentent un vocabulaire très évolutif. Les techniques d'apprentissage automatique, que nous détaillons maintenant, permettent de remédier (au moins en partie) à ce problème.

3 Généralités sur l'apprentissage automatique

L'apprentissage automatique est un très vaste domaine qu'il ne sera pas possible de présenter exhaustivement ici. C'est un champ de recherche à part entière qui occupe encore à l'heure actuelle un grand nombre de mathématiciens, statisticiens, spécialistes de l'intelligence artificielle... En regard de tout ce qui existe dans ce domaine, nous nous contenterons donc de donner un très bref aperçu très simplifié de quelques unes de ses propriétés générales. Nous nous concentrerons ensuite, dans les sections suivantes, sur les techniques les plus courantes issues de l'apprentissage automatique supervisé et non supervisé qui s'appliquent à la tâche de classification. Ces techniques ne traitent que des données tabulaires. Elle ne sont donc utilisées pour la classification de textes qu'après transformation de ces derniers en tableaux (via les approches de type "sacs de mots"). D'autres techniques d'apprentissage automatique capables d'aborder la tâche d'annotation (et donc de tenir compte de la structure séquentielle des données) seront, quant à elles, présentées dans le chapitre consacré à cette tâche.

3.1 Exemple introductif

L'apprentissage "automatique" (ou artificiel), nous allons le voir, est beaucoup plus restreint que ce que le terme d'"apprentissage" désigne habituellement pour les humains. Personne ne prétend que les machines sont capables d'apprendre tout ce que les humains apprennent au cours de leur vie, ni qu'elles procèdent pour cela de la même façon qu'eux. Pour introduire certains concepts clés de l'apprentissage automatique, nous allons partir d'un problème qui ne relève pas de la classification mais plutôt de la *régression mathématique*. Ce problème consiste à chercher, à partir d'exemples de couples de nombres (x, y) à valeurs réelles, une fonction f capable de calculer y à partir de x : $y = f(x)$. Sur le dessin de la Figure 4.1, chaque couple de nombres est représenté par une croix : en abscisse (valeur de x sur l'axe horizontal), on mesure la taille (superficie) d'une maison ou d'un appartement, en ordonnées (valeur de y sur l'axe vertical) le prix de vente de cette maison. On peut espérer, à partir de ces exemples, "apprendre" la fonction f qui relie ces deux valeurs et ainsi *prédire* le prix de nouvelles maisons. Quelle est cette fonction ? A quelle courbe reliant les points correspond-elle ? En fait, on peut imaginer une infinité de façons différentes de relier les points, ou au moins de s'en approcher. La Figure 4.1 montre plusieurs solutions possibles pour un même ensemble de points (il y en a beaucoup d'autres !) [1]. Chacune, à sa manière, est la meilleure possible, et pourtant elles sont très différentes.

Qu'est-ce qui les différencie ? C'est *l'espace de recherche* dans lequel chacune a été choisie. Détaillons comment elles ont été construites.

La première courbe, à gauche, est une droite : c'est la meilleure droite possible, celle qui "s'approche le plus" des différents points. L'espace de recherche dans lequel elle a été sélectionnée est donc l'ensemble de toutes les droites possibles, qui correspond aussi à l'ensemble des fonctions f de la forme : $f(x) = y = ax + b$ avec a et b

1. ces courbes sont extraites de l'article en ligne `https://medium.com/@nomadic_mind/new-to-machine-learning-avoid-these-three-mistakes-73258b3848a4`

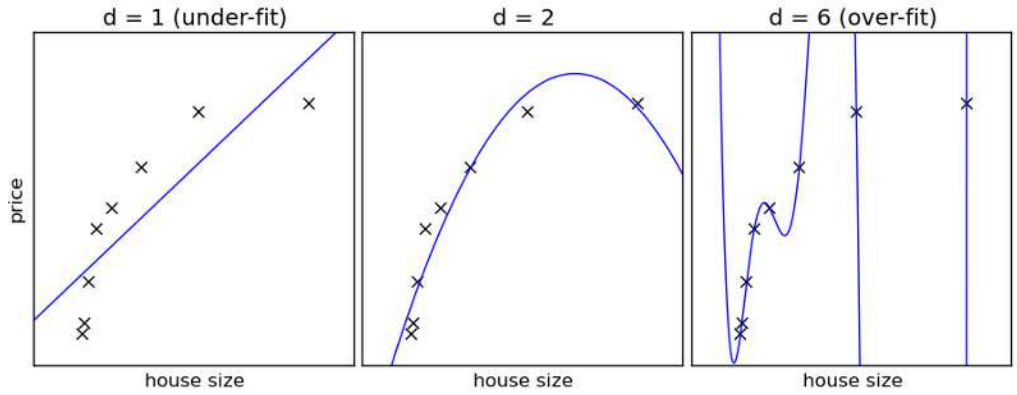


FIGURE 4.1 – Trois courbes possibles pour un même ensemble de points

des nombres quelconques (on dit aussi : fonctions affines ou polynômes de degré 1). Apprendre la "meilleure droite possible" a donc consisté à chercher les valeurs de a et b de telle sorte que la droite s'approche le plus possible des points.

La deuxième, au milieu, est la "meilleure parabole" possible. Les paraboles sont les représentations graphiques des fonctions f de la forme : $f(x) = y = ax^2 + bx + c$ (on dit aussi : polynômes de degré 2) pour des valeurs de a, b et c quelconques. Notez que si $a = 0$, la parabole se transforme en droite : les droites sont des cas particuliers de paraboles ! Cette courbe est donc la "meilleure" (au sens de "plus proche de tous les points") parmi l'ensemble de toutes les droites et de toutes les paraboles possibles. Sa forme semble mieux rendre compte des légers infléchissements qui se produisent pour les données extrêmes (pour les maisons les plus petites et pour les plus grandes), mais peut-être les exagère-t-elle aussi. En tout cas, c'est cette solution qui apparaît comme la plus satisfaisante des trois.

La dernière, à droite, est la représentation graphique d'une fonction de la forme : $f(x) = y = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g$ (on dit aussi un polynôme de degré 6) avec des valeurs sélectionnées de a, b,..., g pour qu'elle soit la plus proche possibles des points initiaux. Comme précédemment, l'ensemble de ces fonctions inclut les précédents (l'ensemble des paraboles et celui des droites). L'espace de recherche est donc plus vaste. De fait, la courbe obtenue passe exactement par les points fixés initialement, elle devrait donc en théorie être très "bonne". Pourtant, elle a l'air très irrégulière et pas très satisfaisante. Que s'est-il passé ? On appelle ce phénomène le "sur-apprentissage" (over-fitting) : à force de vouloir s'approcher plus près des données du problème, on finit par trouver une solution aberrante. Pour apprendre certaines lois, surtout dans le cas de problèmes issus de la "vie réelle" (le prix des maisons, par exemple !) qui ne suivent aucune loi parfaitement, il faut savoir s'éloigner des valeurs précises.

Une solution "approximative" pourrait donc être meilleure qu'une solution parfaite ! Comment est-ce possible ? Quel critère adopter pour trouver la vraie "meilleure" solution ? Cet exemple nous montre que le vrai critère pour savoir si une solution est bonne n'est pas qu'elle "colle" bien aux données qui servent d'exemples pour apprendre mais qu'elle *généralise* correctement sur des *données nouvelles*. Pour prédire le prix de nouvelles maisons, quelle courbe vaut-il mieux suivre ? Sans doute la

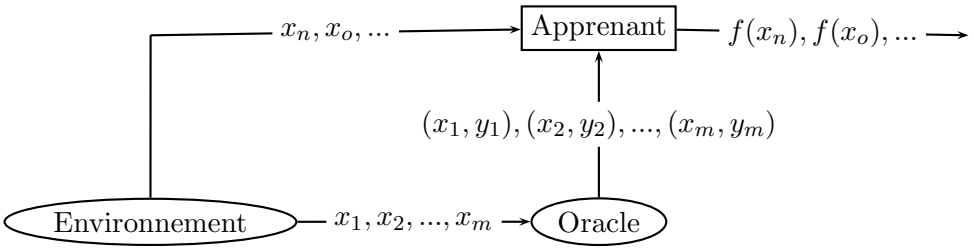


FIGURE 4.2 – Schéma général de l'apprentissage automatique supervisé

deuxième ! C'est celle dont la forme montre mieux les "tendances" que doit suivre l'évolution des prix en fonction de la superficie. Repensons à l'exemple des textes qu'il faut classer en "spam"/"pas spam". Evidemment, il n'existe aucun programme idéal qui réalise cette opération, tant la notion de spam/pas spam est mouvante et dépendante de chaque utilisateur. La seule façon de savoir si un détecteur de spams est "bon", c'est de lui soumettre des textes *nouveaux* et de comparer son jugement à celui d'un humain.

C'est pourquoi, lors d'expériences d'apprentissage automatique (en particulier *supervisé*) il est absolument fondamental pour mesurer la qualité d'une solution de l'évaluer sur des *données nouvelles* x qui n'ont pas servi lors de la phase d'apprentissage. Le vrai critère de qualité est l'écart entre ce que le modèle issu de l'apprentissage prédit sur ces données (la valeur $f(x)$) et la vraie valeur (y) associée à x. D'où la nécessité de disposer d'exemples de référence réservés à l'évaluation et d'où aussi l'importance des "protocoles" évoqués en section 2.4 du chapitre 2. La seule évaluation disponible en apprentissage automatique est empirique : c'est uniquement en mettant à l'épreuve un programme appris sur des données inédites que l'on évalue sa qualité.

3.2 Qu'est-ce qu'apprendre (pour une machine) ?

Apprendre, pour une machine, signifie donc transformer des données isolées (les exemples (x, y)) en *une règle* (une fonction f). Cette règle *généralise*, puisqu'elle permet d'évaluer $f(x)$ pour des valeurs de x qui ne figuraient pas dans les exemples initiaux. Le schéma de la Figure 4.2, inspiré du livre de Miclet et Cornuejols (voir en bibliographie), synthétise ce dispositif. L'apprenant est un programme qui reçoit des données x associées dans le cas supervisé par un "oracle" (ou un professeur !) à des valeurs y, et il doit s'en servir pour être capable d'associer de lui-même un résultat $f(x)$ à de nouvelles données.

Comment cette fonction f est-elle trouvée ? L'exemple de la section précédente fournit quelques enseignements fondamentaux.

- avant d'essayer "d'apprendre" quoi que ce soit, il faut se fixer un espace de recherche, c'est-à-dire une *famille de solutions possibles* à laquelle la fonction recherchée appartient (dans l'exemple : les polynômes d'un certain degré) ;
- une fois choisie cette famille, apprendre revient à trouver la valeur d'un certain nombre de *paramètres* inconnus (dans notre exemple : les valeurs des nombres a, b...). Pour cela, on cherche en général à *minimiser l'écart* entre $f(x)$ (valeur

prévue la règle) par y (valeur observée) sur les exemples (x, y) disponibles. Ce qu'on appelle "modèle" est la valeur d'un ensemble de tels paramètres. Si l'on sait que la fonction est un polynôme de degré 2 et qu'on connaît la valeur des nombres a et b et c (les paramètres), alors on dispose d'une définition complète de la fonction f.

– les meilleures solutions ne sont pas nécessairement celles qui "collent" le plus aux données initiales : ce sont celles qui généralisent le mieux sur de nouvelles données. Pour éviter le phénomène de "sur-apprentissage" illustré dans l'exemple précédent, on ajoute des "régularisations" dans le critère que les programmes cherchent à minimiser. Ces régularisations visent à favoriser les fonctions les plus *simples* : la courbe de degré 2 est ainsi meilleure que celle de degré 6, bien que son "erreur" (l'écart entre la courbe et les points) soit plus grande, parce qu'elle est plus simple et généralise mieux.

En fait, on pourrait résumer la situation en disant qu'apprendre, c'est *généraliser mais pas trop !* Généraliser est indispensable car "apprendre par cœur" n'a jamais été une bonne méthode d'apprentissage ! Et "coller aux données" non plus. Mais *trop généraliser*, par exemple en utilisant une règle trop simple, est également un défaut : c'est ce qu'on peut craindre pour la première courbe de notre exemple... En principe, plus on a au départ d'exemples d'apprentissage (ici, de points), meilleur sera le résultat. Mais on observe aussi parfois des phénomènes de "plafonnement" voire de "régression" (résultats qui baissent). L'apprentissage automatique repose donc sur un très subtil et très délicat équilibre.

Ces propriétés générales expliquent aussi pourquoi il n'existe pas *une* mais *de nombreuses méthodes différentes* en apprentissage automatique. Plusieurs critères permettent de classer les différentes familles de solutions. Nous les passons en revue dans ce qui suit.

Tout d'abord, la nature des données elles-mêmes et de la réponse attendue de la part du programme conditionne beaucoup les solutions possibles. On essaiera toujours de se ramener à un problème où les données sont de la forme (x, y), mais x et y peuvent être très différents d'un problème à un autre. Nous avons déjà évoqué la distinction fondamentale entre les méthodes d'apprentissage automatique supervisées et non supervisées (suivant que des valeurs de y sont disponibles ou pas lors de la phase d'apprentissage).

D'autres distinctions sont aussi à prendre en compte. Dans l'exemple précédent, x et y étaient tous les deux des nombres réels. En classification de textes, x est un texte (ou sa représentation en "sac de mots") et y est une étiquette à choisir parmi un ensemble fini. En annotation, x est une donnée avec une relation d'ordre (une séquence de mots par exemple) et y est une donnée de même structure que x mais constituée d'autres éléments (une séquence d'étiquettes dans ce cas). Ce n'est pas du tout la même chose ! Quand chaque x est en fait une ligne dans un tableau (comme pour un texte après transformation en "sac de mots", mais aussi comme pour toutes les données d'exemples fournies avec le programme Weka, cf. chapitre 2, Figure 2.2), plusieurs situations sont possibles : toutes les cases du tableau sont-elles de même nature (symbolique/numérique) ? Toutes sont-elles nécessairement remplies ou certaines peuvent elles rester vides ? Différentes situations peuvent survenir...

Pour traiter un même ensemble d'exemples (x, y), il reste encore de nombreux

choix à opérer. Le plus fondamental est *l'espace de recherche*, autrement dit la "forme" de la fonction recherchée. Est-ce un polynôme d'un certain degré connu à l'avance, comme précédemment ? Sinon, quoi ? Ce choix conditionne bien sûr énormément la suite des opérations à effectuer. Les espaces possibles sont plus ou moins grands, plus ou moins "structurés" (au sens de : une solution y appartenant est "plus générale" ou "plus spécifique" qu'une autre, par exemple une droite est un cas particulier de parabole...). Rechercher la "meilleure" fonction (au sens de la distance avec les données fournies) appartenant à un certain espace est un problème que les mathématiciens appellent *optimisation*. Chaque courbe de la Figure 4.1 est la solution d'un tel problème d'optimisation. L'ensemble des exemples est le même mais l'espace de recherche a changé d'une courbe à une autre, ce qui explique que les solutions soient différentes. La "régularisation" utilisée pour éviter le sur-apprentissage peut aussi faire varier la solution du problème.

Le défi initial était de donner à un programme la capacité d' "apprendre" et on s'est progressivement ramené à la résolution d'un problème mathématique. Cette simplification est typique d'une démarche scientifique, qu'on désigne aussi avec le terme de "modélisation". Il est essentiel de bien la comprendre, pour être capable de prendre du recul sur la (ou plutôt les !) solution(s) proposée(s). Il y a même des raisons théoriques fondamentales qui justifient l'existence de plusieurs méthodes d'apprentissage automatique. Un *théorème mathématique* appelé "no free lunch" (pas de repas gratuit !) prouve (en très gros) qu'il ne peut pas exister de méthode meilleure que toutes les autres sur tous les problèmes d'apprentissage automatique supervisé possibles !

Dans ces conditions, quelle méthode choisir, quel espace de recherche privilégier ? Là aussi, plusieurs critères de choix sont possibles. Nous avons déjà évoqué ceux qui relèvent de la *performance* du programme en prédiction, c'est-à-dire sur des nouvelles données n'ayant pas servi à l'apprentissage. Les mesures de précision/rappel/F-mesure, déjà largement présentées, sont les plus usuelles, mais ce ne sont pas les seules ! Pour certaines applications, on peut préférer d'autres critères, comme par exemple :

- le temps de calcul requis pour trouver la meilleure solution dans l'espace de recherche : certains espaces sont tellement grands qu'il est très long pour un programme de les "parcourir". Rechercher les 7 paramètres d'un polynôme de degré 6 prend plus de temps que trouver les 2 paramètres qui définissent une droite. C'est parfois bien pire : certaines solutions ne peuvent être qu'approchée, évaluées par approximation, sans garantie d'exactitude. Disposer rapidement d'une solution imparfaite est parfois préférable à obtenir une solution meilleure mais nécessitant beaucoup plus de temps de calcul. Une fois la fonction apprise (phase d'apprentissage), il reste aussi le temps de calcul qui lui est nécessaire pour associer un étiquette à une nouvelle donnée (phase de classification). Ce dernier est toutefois en général moins long que le temps requis par la phase d'apprentissage.
- la lisibilité du résultat : certaines fonctions sont très facilement interprétables par les humaines (les règles symboliques, notamment), d'autres le sont nettement moins (les calculs statistiques, par exemple). Pour résoudre une tâche, un utilisateur peut préférer une solution dont il est capable de comprendre le fonc-

tionnement à une solution qui prend ses décisions de façon incompréhensible pour un humain.
- incrémentalité : une méthode d'apprentissage automatique est *incrémentale* s'il n'est pas nécessaire de refaire tous les calculs à partir de zéro quand on dispose de nouveaux exemples. Dans certains problèmes, en effet, les exemples servant de données d'apprentissage arrivent en "flux continu" plutôt que tous en même temps. C'est le cas des mails à classer en spam/non spams, par exemple. Pouvoir les exploiter un par un ou au contraire attendre d'en avoir un grand nombre avant de lancer le moindre calcul change beaucoup la stratégie d'apprentissage.
- certaines fonctions sont plus ou moins *robustes* à la variété des données sur lesquelles elles s'appliquent. Un programme qui a appris à distinguer les spams/non spams à partir des mails d'un certain utilisateur sera-t-il efficace pour un autre utilisateur, ayant des habitudes et des contacts différents ? La capacité à généraliser peut opérer à plusieurs niveaux : utilisateur, domaine, genre des textes, langue utilisée... Certaines approches peuvent bien généraliser pour un même utilisateur, mais être moins robustes suivant les autres niveaux, c'est-à-dire les autres facteurs de variabilité des données.

Tous ces choix montrent qu'utiliser l'apprentissage, ce n'est en rien laisser à l'ordinateur la main sur le travail à réaliser. Cela ne dispense pas l'humain de réfléchir !

4 Classification par apprentissage supervisé

Nous allons maintenant présenter les techniques les plus classiques d'apprentissage automatique supervisé pour la tâche de classification. Les fonctions f recherchées prennent en entrée une donnée x tabulaire (dont les valeurs sont des booléens ou des nombres), et fournissent en résultat le nom d'une classe y (c'est-à-dire une valeur symbolique à prendre parmi un nombre fini de valeurs possibles). Nous illustrerons de façon systématique le fonctionnement des différentes approches en prenant comme ensemble d'apprentissage les six textes de l'exemple récapitulatif du chapitre 2, section 5.5, appartenant à deux classes distinctes : "culture" et "société". On y ajoute la phrase qui jouait le rôle de requête dans le chapitre 3, section 3, considérée comme appartenant à la classe "culture". Les représentations en termes de nombres d'occurrences dans le logiciel Weka (attention, comme on a ajouté un texte dans la classe "culture", les textes ont été renumérotés par rapport à ceux du chapitre 2) et en termes de simples points de coordonnées (x_1, x_2) dans "l'espace simplifié" de dimension 2 dont la couleur indiquera la classe y (rouge pour les points de la classe "culture", bleu pour ceux de la classe" société") sont repris dans les Figures 4.3 et 4.4 .

Compte tenu des propriétés générales de l'apprentissage automatique que nous venons d'exposer, cette présentation suivra chaque fois une structure commune :
- *espace de recherche* ("forme" de la fonction de classification recherchée)
- *technique utilisée* par le programme pour trouver la meilleure fonction de cet espace, et pour associer une étiquette à une nouvelle donnée
- *propriétés de la méthode et de la fonction trouvée* (performance, temps de cal-

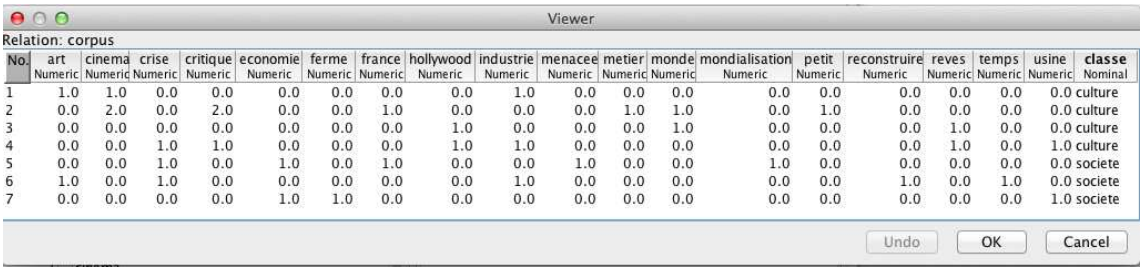


FIGURE 4.3 – Représentation en nombre d'occurrences des textes dans Weka

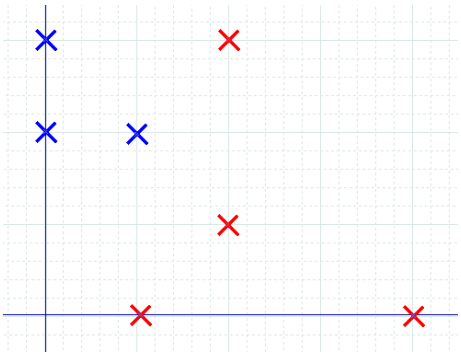


FIGURE 4.4 – Représentation des 7 textes dans l'espace simplifié

cul, lisibilité, incrémentalité...) et, éventuellement, domaines qui les utilisent.

4.1 Classe majoritaire

L'algorithme de la "classe majoritaire" est appelé ZeroR dans Weka, où il joue le rôle de programme "par défaut" de la rubrique "Classify". C'est une technique extrêmement rudimentaire. Elle ne sert habituellement que comme "baseline", c'est-à-dire stratégie de référence minimale qu'il faut battre en utilisant une autre méthode.

Espace de recherche

La fonction recherchée dans le cadre de cette approche est très très simple : c'est une fonction *constante*, c'est-à-dire qui fournit toujours la même valeur en résultat quelle que soit la donnée d'entrée. Il y a donc exactement autant de fonctions dans l'espace de recherche que de valeurs de résultat possibles, c'est-à-dire de classes distinctes dans le problème : 2, pour nous !

Technique utilisée

Quelle fonction choisir ? Le choix est limité ! On a évidemment intérêt à prendre celle qui est la plus représentée dans l'ensemble d'apprentissage, en espérant que c'est parce qu'elle est la plus courante pour l'ensemble de toutes les autres données. En cas d'égalité, on tire au sort... Dans notre exemple, comme on dispose de 4 exemples de

la classe "culture" et de 3 exemples de la classe "société", c'est "culture" qui gagne. La fonction sélectionnée est donc celle qui, pour tout x, donne $f(x) =$ "*culture*". Pour cette sélection, on n'a même pas eu besoin de regarder le contenu des données x, les deux représentations sont donc traitées de la même manière : peu importe x, seule compte la classe.

Propriétés générales

Cet algorithme est très rapide, aussi bien en temps de calcul qu'en temps nécessaire pour donner une valeur à une nouvelle donnée. Son résultat est aisément interprétable et il est facile à mettre à jour en cas d'ajout de nouvelles données étiquetées. Mais, évidemment, il n'est pas très efficace en termes de performance ! Tout ce qu'on peut espérer, c'est qu'il fasse "un peu mieux que le hasard" si les différentes classes du problème sont effectivement déséquilibrées. Appliqué au problème de la reconnaissance des spams/pas spams, en supposant que les exemples fournis sont majoritairement des spams, alors le programme ainsi appris classerait systématiquement tous les nouveaux mails reçus en spams ! Sa robustesse et sa capacité d'adaptation à de nouveaux domaines sont donc très mauvaises.

4.2 k-plus proches voisins

L'algorithme des "k-plus proches voisins" est simple dans son principe et assez efficace dans certains contextes. Il figure dans la rubrique "Classify" de Weka suivant diverses variantes (dossier des algorithmes "lazy" pour "paresseux"). La valeur de k qui lui est associée est un nombre entier qui n'est pas un multiple du nombre de classes (on verra pourquoi plus loin). Il est choisi une fois pour toute avant tout calcul.

Espace de recherche

Exceptionnellement, il n'y a pas de phase d'apprentissage proprement dite pour cette approche, c'est pourquoi elle est qualifiée de "paresseuse". Des calculs seront nécessaires uniquement lors de l'attribution d'une classe à une nouvelle donnée. La fonction f est complètement définie par les exemples (x, y) de l'ensemble d'apprentissage : *le modèle coincide avec les données étiquetées !* L'"espace de recherche" est en quelque sorte l'ensemble des fonctions définies par un certain nombre de données x associées à une classe y.

Technique utilisée

Tout repose donc ici sur la façon d'associer une classe à une nouvelle donnée. L'idée est de chercher, parmi les exemples disponibles (x, y), les k dont la valeur de x est la plus "proche" (au sens d'une distance prédéfinie) de cette nouvelle donnée et de lui associer la classe y majoritaire *au sein de ces k voisins*. L'algorithme impose donc de calculer la distance entre la nouvelle donnée et toutes celles fournies en exemples et de mémoriser la classe des k plus proches. Les schémas de la figure 4.5 montrent plusieurs cas possibles pour une nouvelle donnée (en vert) dans notre espace simplifié.

Comme notre exemple comporte 2 classes, il faut prendre une valeur de k impaire : chaque donnée ne pouvant être que d'une classe parmi 2 possibles, une majorité se dégagera nécessairement (il n'y aura pas d'ex-aequo). Ainsi, en choisissant dans notre exemple $k = 3$ et la distance euclidienne, le premier point sera affecté à la classe "société" bleue (2 de ses 3 plus proches voisins sur le sont), tandis que les deux suivants recevront l'étiquette "culture" rouge. Notons que le résultat serait ici le même avec $k = 5$. En revanche, avec $k = 7$, on retrouverait l'algorithme de la classe majoritaire précédent qui, lui, associe toujours la valeur rouge !

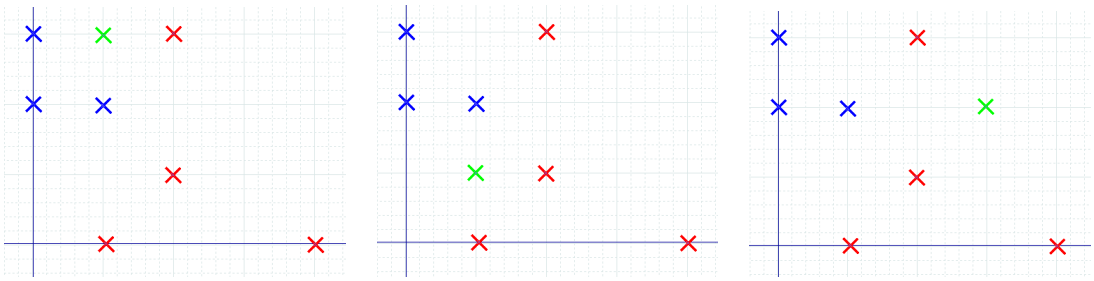


FIGURE 4.5 – Trois configurations pour une nouvelle donnée dans l'espace simplifié

Propriétés générales

Les calculs requis par cette méthode sont raisonnables, similaires à ceux qu'effectue un moteur de recherche : la donnée à classer joue en quelque sorte le rôle de requête, on évalue sa proximité avec toutes les autres données disponibles. Aucun traitement préliminaire n'est nécessaire, tout s'effectue lors de l'étiquetage d'une nouvelle donnée. La mise à jour avec de nouveaux exemples est donc immédiate et le résultat est interprétable (on peut demander à voir les données les plus proches qui décident du résultat). Pourtant, cette approche est peu utilisée pour la classification de textes. Cela tient sans doute à la très grande dimension des données dans ce cas (plusieurs milliers de colonnes pour les "vrais textes"). Or, les "plus proches voisins" ne sont efficaces que si les exemples disponibles "couvrent" le territoire des valeurs possibles dans chacune des dimensions de l'espace. Quand l'espace est très grand, il faut donc disposer d'un très grand nombre d'exemples étiquetés pour espérer trouver des voisins vraiment "proches" pour n'importe quelle nouvelle donnée. Avec peu d'exemples, d'autres méthodes sont plus efficaces.

En revanche, on peut signaler que cette approche est très utilisée dans un autre domaine aux forts enjeux commerciaux, même s'il ne concerne pas directement les textes : celui de la *recommandation*. Un système de recommandation a pour objectif de prévoir les comportements d'individus vis-à-vis de certains produits de consommation, en se fondant sur des comportements précédents. Ils sont particulièrement utilisés pour recommander des produits culturels comme les livres, les films ou les chansons (le fameux "ceux qui ont commandé ce produit ont aussi apprécié..." d'Amazon). Le point de départ d'un tel système est un tableau à double entrée où sont référencés quels individus (en lignes) ont acheté/consulté/apprécié quels produits (en colonnes). Beaucoup de cases sont vides, fautes d'appréciations expli-

cites connues. Pour prédire la valeur (en général : une note) d'une de ces cases vides, des extrapolations fondées sur les plus proches individus et/ou produits voisins sont calculées.

4.3 Arbres de décision

Les arbres de décision sont des objets de nature *symbolique*, faciles à lire et interpréter par les humains. Les premiers algorithmes d'apprentissage automatique d'arbres de décision datent des années 1990. Ils ont tendance à être de moins en moins utilisés de nos jours (d'autres méthodes sont plus efficaces en termes de performance), mais il semble important de les présenter car ils constituent une classe de modèles originale et agréable à manipuler. Ils illustrent aussi le fait que l'apprentissage automatique n'est pas toujours synonyme de calculs statistiques et qu'il peut cibler des objets symboliques. L'algorithme le plus connu et utilisé pour construire des arbres de décision s'appelle C4.5 (et sa version améliorée, mais payante, C5). Dans Weka, il est disponible dans le dossier "trees" sous le nom J48.

Espace de recherche

Un arbre de décision est un arbre qui se lit "de la racine aux feuilles" pour toute donnée x et se compose des éléments suivants :

- chacun de ses nœuds (y compris sa racine) contient un *test* portant sur la valeur *d'un unique attribut* (une colonne) servant à décrire x. Les différentes réponses possibles au test doivent couvrir toutes les valeurs possibles que peut prendre cet attribut et chacune "oriente" la lecture de l'arbre vers une *unique branche* partant de ce nœud.
- chacune des feuilles de l'arbre contient la valeur de la classe y qui sera associée à la donnée x aboutissant à cette feuille.

L'arbre de la Figure 4.6 est par exemple celui trouvé par Weka (algorithme J48) quand on lui fournit en apprentissage les données de la Figure 2.2 qui, rappelons-le, sont censées classer des conditions météorologiques en "yes" si elles sont adaptées pour jouer au tennis dehors et "no" sinon.

Cet arbre est équivalent à une suite de tests emboités, et peut être paraphrasé en Python de la façon suivante :

```
IF outlook == sunny:
       IF humidity <= 75:
              result = yes
       ELSE :
              result = no
ELSIF outlook == overcast:
       result = yes
ELSIF outlook == rainy:
       IF windy == TRUE:
              result = no
       ELSE:
              result = yes
```

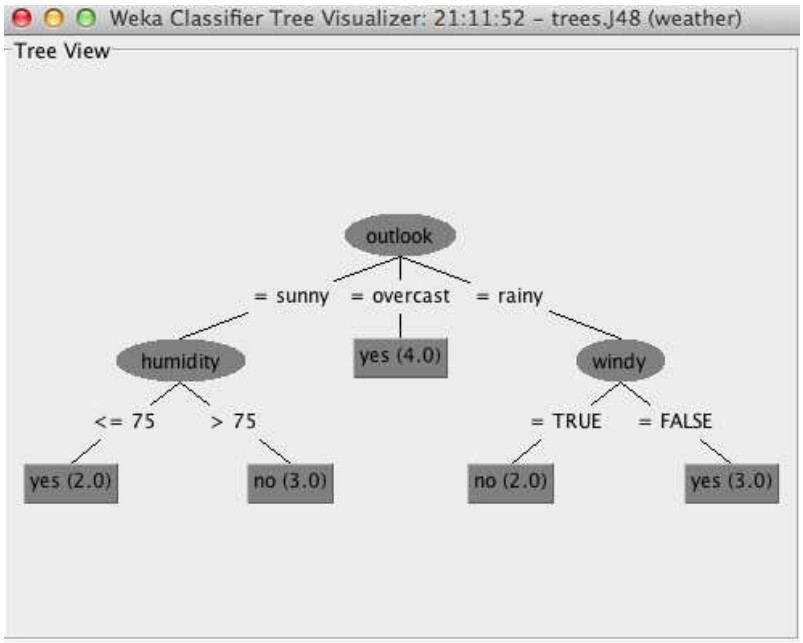


FIGURE 4.6 – Arbre de décision trouvé par Weka pour les données de la Figure 2.2

Les valeurs numériques qui figurent dans les feuilles de l'arbre produit par Weka sont le nombre de données de l'ensemble d'apprentissage qui "aboutissent" à ces feuilles en partant de la racine (leur somme fait bien 14, qui était le nombre d'exemples initiaux).

Pour les données de la Figure 4.3 issues de nos textes en exemple, Weka propose l'arbre de la Figure 4.7. Cet arbre est extrêmement simple et ne rend pas parfaitement compte des données d'apprentissage (appliqué sur ses propres données d'entrainement, il classe 5 textes en "culture" dont 1 à tort, comme cela est signalé dans la feuille correspondante). Nous verrons pourquoi en expliquant comment fonctionne l'algorithme de recherche implémenté dans Weka. Cet algorithme (et ses variantes) a pour espace de recherche l'ensemble de tous les arbres de décision possibles définis à partir des attributs des données.

Technique utilisée

A partir de n'importe quel ensemble de données d'apprentissage, on peut toujours construire très facilement un arbre de décision "parfait" pour ces données : il suffit pour cela d'énumérer les attributs les uns après les autres, de tester toutes les valeurs possibles qu'ils prennent dans les exemples et de mettre finalement la bonne classe dans les feuilles. Mais un arbre construit de la sorte serait beaucoup trop proche des données initiales. Pour éviter le "sur-apprentissage", il faut privilégier les arbres simples, petits, le moins profond possible : ce sont eux qui *généralisent* le mieux. Trouver le plus petit arbre de décision possible compatible avec un ensemble de données est un problème intrinsèquement difficile et les algorithmes employés se contentent en général de solutions approximatives fondées sur des "heuristiques" (méthodes imparfaites mais faciles à appliquer).

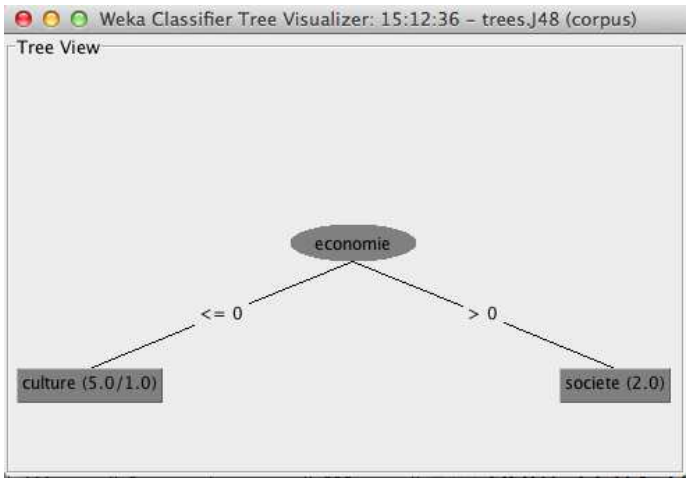


FIGURE 4.7 – Arbre de décision trouvé par Weka pour les données de la Figure 4.3

La construction heuristique de l'arbre se fait comme sa lecture : en partant de la racine, puis en réitérant le processus de façon récursive (un arbre étant par nature un objet récursif !). L'objectif initial est donc de trouver sur quel attribut faire porter le premier test. Différents critères sont possibles pour cela, mais l'intuition qui les sous-tend est toujours la même : il s'agit de trouver un test qui "sépare" le mieux possible les données associées à des classes différentes.

Prenons l'exemple d'un problème de classification à 2 classes appelées c_1 et c_2 (par exemple spam/pas spam ou les yes/no du problème de savoir si on peut jouer au tennis dehors) pour lesquels on dispose d'un ensemble S d'exemples étiquetés. On suppose qu'il y a parmi eux n_1 données recevant l'étiquette c_1 et n_2 recevant l'étiquette c_2. Soit $p_1 = \frac{n_1}{n_1+n_2}$ et $p_2 = \frac{n_2}{n_1+n_2}$ les proportions respectives des représentants de chaque classe dans les exemples. Ces formules se généralisent bien sûr simplement à un nombre quelconque n de classes. On définit les fonctions suivantes :

- $Gini(S) = \sum_{i=1}^{n} p_i(1-p_i)$. Avec deux classes, cette formule devient donc $Gini(S) = p_1(1-p_1) + p_2(1-p_2)$. Or dans ce cas $p_1 + p_2 = 1$ donc la formule se simplifie en $Gini(S) = p_1(1-p_1) + (1-p_1)p_1 = 2p_1(1-p_1)$
- $H(S) = \sum_{i=1}^{n} p_i log_2(p_i)$. $H(S)$ est aussi appelé l'*entropie* de l'ensemble S. Notons que la fonction $log_2(x)$ donne le nombre de bits nécessaires pour coder x en binaire, elle mesure donc traditionnellement la *quantité d'information* contenue dans x. Avec deux classes, on a donc $H(S) = -p_1 log_2(p_1) - p_2 log_2(p_2)$. Avec la même propriété que précédemment, $H(S) = -p_1 log_2(p_1) - (1-p_1) log_2(1-p_1)$.

Ces deux fonctions mesurent la *dispersion* d'un ensemble de données relativement aux classes dans lesquelles elles se répartissent. La Figure 4.8 montre leurs courbes pour des valeurs de $p1$ variant de 0 à 1. Quand $p_1 = 0$, cela signifie que la classe c_1 est vide : toutes les données sont donc dans c_2. Inversement, quand $p_1 = 1$, toutes les données sont dans c_1 (et aucune dans c_2). Dans ces deux cas, les fonctions $Gini$ et H valent 0 : la dispersion est nulle, les données sont parfaitement homogènes en termes de classe, il n'y a aucun mélange. Les fonctions sont à leur maximum ($\frac{1}{2}$ pour $Gini$, 1 pour H) quand les données sont équitablement réparties entre les deux classes (la moitié dans l'une, la moitié dans l'autre) autrement dit quand le mélange

est maximum.

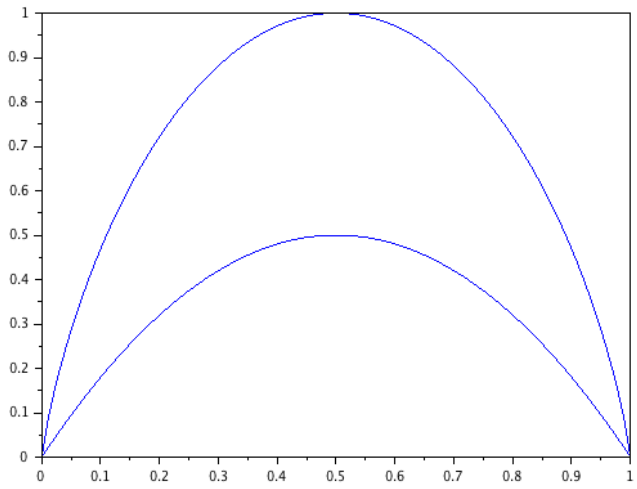


FIGURE 4.8 – Courbes de Gini (en bas) et d'entropie H (en haut) en fonction p_1, la proportion d'une des classes dans un ensemble de données à deux classes

Grâce à ces fonctions, on peut mesurer pour chaque attribut quel *gain en homogénéité* il est susceptible d'apporter. Il faut pour cela calculer la *différence* entre la dispersion (ou le mélange) des données *avant utilisation de l'attribut* et *après utilisation*. Le *gain* de la fonction *Gini* sur l'ensemble S suivant l'attribut a qui peut prendre un nombre fini de valeurs v est ainsi défini par :
$gain(Gini, a) = Gini(S) - \sum_{v \ : \ valeurs \ de \ a} \frac{|S_{a=v}|}{|S|} Gini(S_{a=v})$
où $|S_{a=v}|$ compte le nombre d'éléments de S pour lesquels l'attribut a vaut v, et $Gini(S_{a=v})$ est la valeur de la fonction *Gini* sur ce même sous-ensemble des données. Un calcul similaire peut être effectué pour la fonction H. Nous illustrons ces calculs sur les données météorologiques de la Figure 2.2 (S comporte donc 14 exemples, parmi lesquels 9 "yes" et 5 "no"), en imaginant successivement chacun des attributs en racine de l'arbre :

- la proportion de la classe "yes" est $p_1 = \frac{9}{14}$ donc
  $Gini(S) = 2p_1(1 - p_1) = 2 * \frac{9}{14} * (1 - \frac{9}{14}) = \frac{9}{7} * \frac{5}{14} = \frac{45}{98}$
- si l'attribut "outlook", qui peut prendre 3 valeurs différents possibles, est sélectionné en premier, on construit un arbre qui commence comme dans la Figure 4.9.

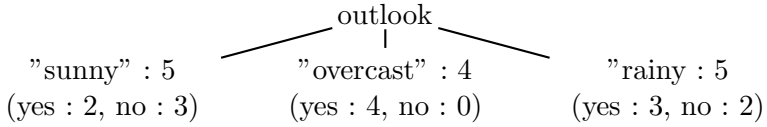


FIGURE 4.9 – Début d'un arbre fondé sur l'attribut outlook

On obtient alors le gain suivant :
$gain(Gini, outlook) = Gini(S) - \sum_{v \ : \ valeurs \ de \ outlook} \frac{|S_{outlook=v}|}{|S|} Gini(S_{outlook=v})$

$= Gini(S) - \frac{|S_{outlook="sunny"}|}{|S|} Gini(S_{outlook="sunny"}) - \frac{|S_{outlook="overcast"}|}{|S|} Gini(S_{outlook="overcast"}) - \frac{|S_{outlook="rainy"}|}{|S|} Gini(S_{outlook="rainy"})$
$= Gini(S) - \frac{5}{14} Gini(S_{outlook="sunny"}) - \frac{4}{14} Gini(S_{outlook="overcast"}) - \frac{5}{14} Gini(S_{outlook="rainy"})$
$= \frac{45}{98} - \frac{5}{14} * 2 * \frac{2}{5} * (1 - \frac{2}{5}) - \frac{4}{14} * 0 - \frac{5}{14} * 2 * \frac{3}{5} * (1 - \frac{3}{5}) = \frac{57}{490} = 0,1163...$

– si l'attribut "windy", qui peut prendre 2 valeurs (True/False) est sélectionné en premier, on construit un arbre qui commence comme dans la Figure 4.10.

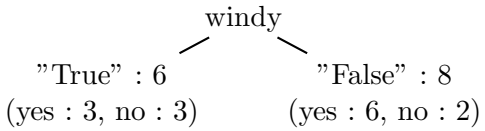


FIGURE 4.10 – Début d'un arbre fondé sur l'attribut windy

On obtient alors le gain suivant :
$gain(Gini, windy) = Gini(S) - \sum_{v \ : \ valeurs \ de \ windy} \frac{|S_{windy=v}|}{|S|} Gini(S_{windy=v})$
$= Gini(S) - \frac{|S_{windy="True"}|}{|S|} Gini(S_{windy="True"}) - \frac{|S_{windy="False"}|}{|S|} Gini(S_{windy="False"})$
$= Gini(S) - \frac{6}{14} Gini(S_{windy="True"}) - \frac{8}{14} Gini(S_{windy="False"})$
$= \frac{45}{98} - \frac{3}{7} * 2 * \frac{1}{2} * (1 - \frac{1}{2}) - \frac{4}{7} * 2 * \frac{3}{4} * (1 - \frac{3}{4}) = \frac{3}{98} = 0,0306...$
Le gain obtenu est moins bon que précédemment.

– pour les attributs restants, qui sont numériques, il faut procéder différemment. On ne peut pas définir un critère qui énumérerait toutes les valeurs possibles d'un tel attribut (puisqu'il peut y en avoir une infinité) ; en revanche, on peut exploiter le fait que ces valeurs sont *ordonnées* et chercher donc un critère du type : "attribut <= seuil" versus "attribut > seuil". Chaque seuil possible opère une division en deux des données et fonctionne donc comme un attribut binaire, sur lequel les mêmes calculs que précédemment peuvent être effectués. La Figure 4.11 montre la répartition des "yes" (en bleu) et des "no" (en rouge) en fonction de la valeur de l'attribut temperature. Nous ne détaillerons pas tous

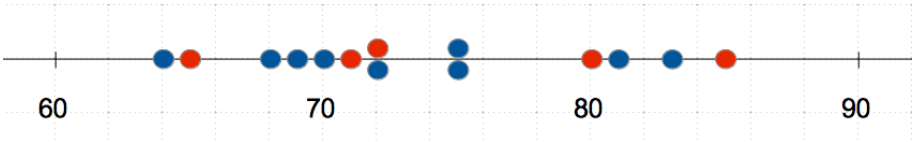


FIGURE 4.11 – Répartitions des "yes" (en bleu) et des "no" (en rouge) en fonction de la temperature

les calculs ici, mais seulement quelques uns. Par exemple, si on positionne le seuil quelque part entre 72 (inclus) et 75 (exclu), on obtient une séparation entre un groupe de 8 points parmi lesquels figurent 5 "yes" et 3 "no", et un autre de 6 points, dont 4 "yes" et 2 "no". On a donc :
$gain(Gini, temp <= 72) = Gini(S) - \frac{|S_{temp<=72}|}{|S|} Gini(S_{temp<=72}) - \frac{|S_{temp>72}|}{|S|} Gini(S_{temp>72})$
$= Gini(S) - \frac{8}{14} Gini(S_{temp<=72}) - \frac{6}{14} Gini(S_{temp>72})$
$= \frac{45}{98} - \frac{8}{14} * 2 * \frac{3}{8} * (1 - \frac{3}{8}) - \frac{6}{14} * 2 * \frac{4}{6} * (1 - \frac{4}{6})$
$= \frac{1}{1176} = 0,00085...$

Ce gain est mineur et peu intéressant. Si on positionne le seuil entre 75 (inclus) et 80 (exclu), on a alors :
$gain(Gini, temp <= 75) = Gini(S) - \frac{|S_{temp<=75}|}{|S|} Gini(S_{temp<=75}) - \frac{|S_{temp>75}|}{|S|} Gini(S_{temp>75})$
$= Gini(S) - \frac{10}{14} Gini(S_{temp<=75}) - \frac{4}{14} Gini(S_{temp>75})$
$= \frac{45}{98} - \frac{5}{7} * 2 * \frac{7}{10} * (1 - \frac{7}{10}) - \frac{2}{7} * 2 * \frac{1}{2} * (1 - \frac{1}{2})$
$= \frac{4}{245} = 0,0163...$
Ce gain est meilleur que le précédent mais toujours inférieur à celui apporté par l'attribut outlook. Il en est de même pour tous les autres gains possibles apportés en faisant varier la valeur du seuil sur l'attribut des températures.

– Nous ne détaillons pas non plus les calculs (similaires aux précédents) pour évaluer le gain qu'apporterait la prise en compte de l'attribut numérique humidity comme premier attribut. Ils restent inférieurs celui d'outlook, d'où le choix de prendre ce attribut en premier.

Ces calculs ont permis de déterminer qu'outlook est l'attribut le plus "discriminant" parmi ceux disponibles, c'est-à-dire celui qui, pris en compte seul, sépare le mieux les classes "yes" et "no". Ce premier choix permet donc de sélectionner le début de l'arbre de la Figure 4.9. Parmi les branches restantes, nous constatons que celle du milieu (correspondant au test outlook="overcast") est complètement homogène : on peut donc directement lui attribuer la valeur de la classe "yes". Les deux autres branches présentent encore des données mélangées, où les deux classes sont représentées. Il faut réitérer pour chacune d'elles le processus de calcul que nous venons de détailler (en excluant toutefois l'attribut outlook, déjà exploité). Nous ne détaillons pas les calculs, mais on peut se convaincre facilement que c'est bien l'arbre de la Figure 4.6 que l'on va ainsi finalement sélectionner d'autant que, chaque fois, un seul critère sur un attribut permet une séparation parfaite des deux classes (assurant un gain maximal).

Revenons maintenant à l'exemple de nos textes. Parmi les 18 attributs de la représentation vectorielle de la Figure 4.3, les calculs de gain (non détaillés ici) sélectionnent le mot "économie" qui permet de correctement classer 6 des 7 textes, produisant ainsi l'arbre de la Figure 4.7. Il serait bien sûr possible de réitérer la phase de recherche d'un critère optimal sur les données de la branche de gauche, afin d'obtenir un arbre "parfait". Mais l'ajout d'un tel critère ne servirait à distinguer qu'un seul texte (qui n'est peut-être qu'un cas particulier aberrant) parmi 5, et nous avons vu que "coller" trop aux données faisait courir le risque du sur-apprentissage. J48 implémente donc une stratégie d'"élagage" consistant à privilégier les arbres simples même imparfaits par rapport aux arbres parfaits mais trop spécifiques. C'est ce qui explique que Weka se contente de l'arbre de la Figure 4.7.

La représentation des mêmes textes dans l'espace simplifié de dimension 2 (Figure 4.4) nous donne l'occasion d'expliquer le principe des arbres de décision un peu différemment. Puisque les deux dimensions de cet espace sont numériques, on cherche le meilleur critère parmi ceux qui sont de la forme "attribut <= seuil". Or chaque attribut est associé à une dimension distincte de l'espace, et les critères de cette forme correspondent donc à des *droites séparatrices parallèles aux axes*. Le meilleur critère est donc la meilleure de ces droites, celle permettant d'opérer la distinction la plus claire entre les points bleus (classe "culture") et les rouges (classe "société"). La figure 4.12 montre deux de ces droites : si on nomme respectivement m_1 et m_2

les deux dimensions, la première réalise le critère $m_2 <= 1,2$ et la seconde le critère $m_1 <= 1,6$. Ces deux critères sont d'égales qualités, puisque chacun d'eux sépare les données entre d'une part 3 textes "société" correctement regroupés, d'autre part trois textes "culture" mélangés à un seul texte "société".

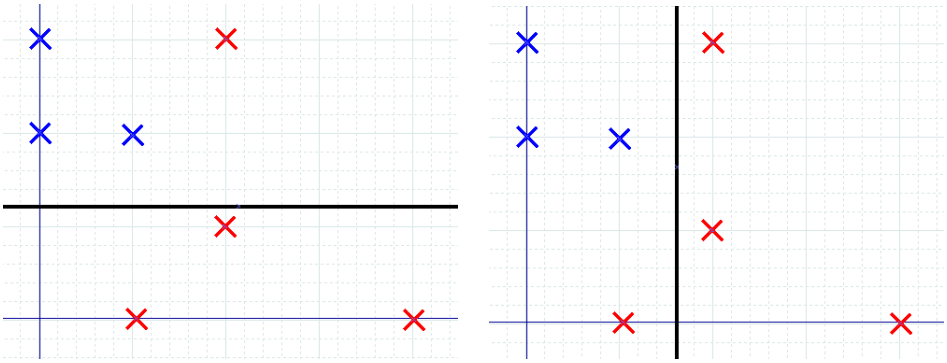


FIGURE 4.12 – Deux droites séparatrices parallèles aux axes optimales dans l'espace simplifiée

Notons que les tests $m_2 < 2$ et $m_1 < 2$ donneraient exactement les mêmes résultats en termes de répartition des points de part et d'autres des droites. Sans avoir besoin de faire les calculs, on peut se convaincre facilement que ces critères sont ceux apportant le meilleur gain parmi tous les critères possibles de la forme "attribut <= seuil". Supposons donc que nous choisissions le premier d'entre eux, représenté à gauche sur la Figure 4.12 et correspondant à $m_2 <= 1,2$. Les données qu'il sépare sont homogènes d'un coté mais pas de l'autre, essayons donc de continuer la séparation sur la zone de l'espace pour lesquels $m_2 > 1,2$. Sur cette zone, une nouvelle droite parallèle aux axes suffit à distinguer les points rouges des points bleus, par exemple en testant si $m_1 <= 1,4$. La composition de ces deux critères, représentée par la Figure 4.13, correspond à l'arbre de la Figure 4.14.

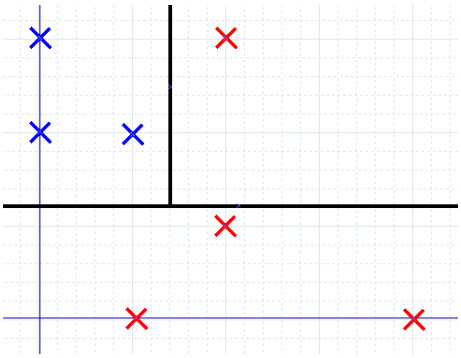


FIGURE 4.13 – Composition de deux critères

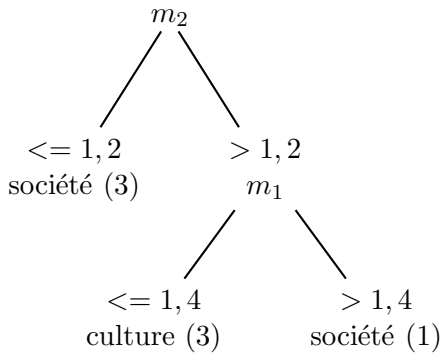


FIGURE 4.14 – Arbre correspondant à la Figure 4.13

Propriétés générales

Les arbres de décision sont surtout appréciés pour leur *lisibilité*, qui les rend en général compréhensibles par un humain (dans les limites d'une certaine taille). Ils sont applicables sur des données de toute nature, que les attributs prennent des valeurs symboliques ou numériques, et pour un nombre quelconque de classes.

Une propriété intéressante de ces arbres est qu'ils ordonnent les attributs en fonction de leur caractère *discriminant*, c'est-à-dire pertinent pour distinguer une classe d'une autre : plus un attribut se trouve proche de la racine dans un arbre de décision, plus il est discriminant. Les arbres de décision peuvent ainsi être utilisés comme une forme de pré-traitement, pour sélectionner certains attributs et en négliger d'autres.

Les programmes qui les construisent sont en général rapides et efficaces, mais pas incrémentaux. Sur les textes, leurs performances (en termes de précision/rappel/F-mesure) sont habituellement moindres que celles obtenues avec d'autres méthodes. Mais Weka intègre aussi de nombreuses extensions de l'algorithme de référence J48 qui, pour certaines dans certains cas, peuvent s'avérer plus performantes, mais que nous ne détaillerons pas ici.

4.4 Naive Bayes

Les algorithmes de type "bayésiens" (dossier "bayes" de Weka) reposent sur le "théorème de Bayes", un résultat de probabilités ancien (Bayes a vécu au XVIIIème siècle, même s'il n'a pas lui-même énoncé clairement le théorème qui porte son nom) et très général, que nous présentons (brièvement) d'abord. Ce théorème a de très nombreuses applications, pas seulement en classification. Le caractère "naif" de la technique ne vient pas de lui mais des approximations qui sont faites pour pouvoir l'utiliser dans le contexte de la fouille de données/textes.

Théorème de Bayes

Le théorème de Bayes énonce une relation fondamentale entre des *probabilités conditionnelles*. Il faut donc tout d'abord définir cette notion. Etant donné deux événements A et B, la probabilité conditionnelle de "A sachant B", qu'on note

$p(A|B)$ se calcule de la façon suivante : $p(A|B) = \frac{p(A \cap B)}{p(B)}$. Pour bien comprendre cette définition, prenons l'exemple suivant :

- soit A l'événement "obtenir un nombre inférieur ou égal à 4 en jouant à un dé (non truqué !)" : $p(A) = \frac{4}{6} = \frac{2}{3}$
- soit B l'événement "obtenir un nombre pair en jouant au dé" : $p(B) = \frac{1}{2}$

Ces deux événements ne sont pas indépendants. L'événement $A \cap B$, à savoir "obtenir un nombre à la fois pair et inférieur ou égal à 4 en jouant au dé" a la probabilité $p(A \cap B) = \frac{2}{6} = \frac{1}{3}$. Examinons maintenant les probabilités conditionnelles faisant intervenir A et B :

- $p(A|B)$ correspond à la probabilité de l'événement "obtenir un nombre inférieur ou égal à 4 au dé, sachant qu'on a obtenu un résultat pair". D'après la formule de la probabilité conditionnelle, on a :
  $p(A|B) = \frac{p(A \cap B)}{p(B)} = \frac{\frac{1}{3}}{\frac{1}{2}} = \frac{1}{3} * 2 = \frac{2}{3}$
- on peut aussi calculer $p(B|A)$ en intervertissant les rôles de A et de B dans la formule. Cette probabilité correspond à l'événement "obtenir un nombre pair sachant qu'on a obtenu un nombre inférieur ou égal à 4" :
  $p(B|A) = \frac{p(B \cap A)}{p(A)} = \frac{p(A \cap B)}{p(A)} = \frac{\frac{1}{3}}{\frac{2}{3}} = \frac{1}{3} * \frac{3}{2} = \frac{1}{2}$

Les résultats obtenus par calculs sont conformes à l'intuition de la probabilité de ces événements. On remarque aussi que dans les deux variantes de la formule initiale utilisées, il y a un élément commun : en effet, on a toujours $p(A \cap B) = p(B \cap A)$. Or, d'après les définitions des probabilités conditionnelles, $p(A \cap B) = p(A|B) * p(B)$ et $p(B \cap A) = p(B|A) * p(A)$. En utilisant l'égalité entre ces deux formules, on obtient : $p(A|B) * p(B) = p(B|A) * p(A)$, ou encore : $p(B|A) = \frac{p(A|B) * p(B)}{p(A)}$. Cette relation *est le théorème de Bayes* !

Dans cette formule, on appelle souvent $p(B)$ la *probabilité a priori* de B (au sens où elle est "antérieure" à toute connaissance sur A), tandis que $p(B|A)$ est la probabilité *a posteriori* de B sachant A (ou encore "sous condition" de A). Ce théorème est simple à énoncer et à démontrer, mais il énonce une relation très intéressante entre les probabilités de deux événements (généralement liés entre eux). Il est pour cela utilisé dans de nombreux domaines. Par exemple, en médecine $p(B|A)$ peut désigner la probabilité de développer une certaine maladie (événement B), sachant (observant) certains symptômes (événement A). Le théorème de Bayes relie cette probabilité à la probabilité $p(A|B)$ d'observer les symptômes en question, sachant qu'on a (à coup sûr) développé la maladie, et aussi aux probabilités $p(A)$ et $p(B)$. Grâce à ce théorème, on intervertit en quelque sorte les rôles de ce qui est connu et de ce qui est inconnu.

Espace de recherche

En classification de textes, on doit affecter une classe c à un document d connu. Une approche probabiliste va naturellement chercher la classe c la "plus probable", c'est-à-dire celle qui rend $p(c|d)$ le plus grand possible. La valeur de cette probabilité n'est pas directement évaluable. Mais, en appliquant le théorème de Bayes, on se ramène au problème de calculer : $\frac{p(d|c) * p(c)}{p(d)}$. En fait, on ne cherche pas tant à obtenir la valeur précise de cette formule qu'à trouver la classe c qui la rend la plus grande

mot	art	ci.	cr.	crit.	éc..	fe.	fr.	hol.	ind.	me.	mét.	mo.	mond.	p.	rec.	rêv.	tps	us.
nb	1	3	1	3	0	0	1	2	2	0	1	2	0	1	0	2	0	1
+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
tot.	2	4	2	4	1	1	2	3	3	1	2	3	1	2	1	3	1	2
nb	1	0	2	0	2	1	1	0	1	1	0	0	1	0	1	0	1	1
+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
tot.	2	1	3	1	3	2	2	1	2	2	1	1	2	1	2	1	2	2

FIGURE 4.15 – sacs de mots cumulés pour les classes "culture" et "société"

possible. Pour cela, il est inutile de connaître la valeur de $p(d)$, qui ne varie pas d'une classe c à une autre. On est donc ramené au problème de trouver c telle que la valeur de $p(d|c) * p(c)$ soit la plus grande possible. L'espace de recherche de l'approche "bayésienne" de la classification est en quelque sorte l'ensemble des moyens de calculer les probabilités inconnues $p(d|c)$ et $p(c)$. C'est à cela que vont servir les exemples fournis lors de la phase d'apprentissage.

Technique utilisée

Evaluer les valeurs de $p(c)$ pour les différentes classes c est la partie la plus simple du problème, car ces classes ne dépendent pas du document d à classer. Il suffit pour les évaluer de calculer la proportion de chaque classe dans l'ensemble des exemples disponibles. Dans notre exemple avec 7 textes, on prend donc $p(culture) = \frac{4}{7}$ et $p(societe) = \frac{3}{7}$.

Evaluer $p(d|c)$ est apparemment une tâche paradoxale. Elle consiste en effet à calculer la probabilité d'un document d connaissant une classe. Qu'est-ce que cela signifie ? C'est ici qu'une hypothèse "naive" va être nécessaire. En réalité, cette hypothèse a déjà faite pour transformer les textes en tableaux de nombres : elle consiste à considérer qu'un texte n'est qu'un *sac de mots* et que chaque mot est *indépendant des autres*. Les exemples étiquetés ont été traités de cette façon. La connaissance qu'ils nous apportent sur chaque classe c se confond ainsi avec l'ensemble des mots présents dans les textes qui y sont rangés. Habituellement, en probabilité, on imagine des problèmes de boules de différentes couleurs piochées dans une urne. Ici, les urnes sont remplacées par les "sacs". Il y a autant de sacs différents que de classes et ce sont les *occurrences de mots* qui jouent le rôle des "boules". Obtenir le texte d connaissant la classe revient alors à tirer au sort indépendamment chacun des mots de d dans les différents "sacs" disponibles. Comme les mots sont indépendant les uns des autres, on a : $p(d|c) = \prod_{mot \in d} p(mot|c)$

Reprenons l'exemple de nos 7 textes dont les représentations en nombres d'occurrences des mots (non vides) sont données dans la Figure 4.3. Ils sont rangés dans deux classes différentes. Les "sacs de mots" correspondant à ces deux classes sont obtenus en *cumulant* les nombres d'occurrences des mots de chacun des textes qui en font partie. Pour tout nouveau document d à classer, on cherche dans quelle classe (donc dans quel sac) la probabilité de ses mots est la plus grande. On ne prend en compte que les mots de d présents dans les exemples. Pour que chacun d'eux ait une probabilité non nulle d'être tiré au sort dans chaque "sac", on ajoute systématiquement dans chacun des sacs un exemplaire de chacun des mots. Sinon, on pourrait avoir $p(mot|c) = 0$ et donc $p(d|c) = 0$. On obtient donc à partir de la Figure 4.3 les deux "sacs" représentés dans le tableau de la figure 4.15.

mot	cinéma	économie
nombre	9	4
+	1	1
total	10	5
nombre	1	7
+	1	1
total	2	8

FIGURE 4.16 – sacs de mots cumulés pour les classes "culture" et "société" dans l'espace simplifié

Imaginons maintenant que le nouveau document d à classer est le suivant : "En temps de crise, un cinéma ne ferme jamais !". Le programme "Naive Bayes" doit décider d'associer à d soit la classe "culture", soit la classe "société". Les seuls mots à prendre en compte dans ce texte sont ceux ayant servi à indexer l'ensemble d'apprentissage, à savoir ici : "temps", "crise", "cinéma", "ferme". On a donc :
$p(d|c) = \prod_{mot \in d} p(mot|c) = p(cinema|c) * p(crise|c) * p(ferme|c) * p(temps|c)$

On évalue la probabilité $p(mot|c)$ d'un mot connaissant une classe en calculant la proportion des occurrences de ce mot dans le "sac" qui représente cette classe. Les calculs deviennent donc :
$p(d|culture) = \frac{4}{38} * \frac{2}{38} * \frac{1}{38} * \frac{1}{38} = \frac{2}{19^2 * 38^2}$
$p(d|societe) = \frac{1}{31} * \frac{3}{31} * \frac{2}{31} * \frac{2}{31} = \frac{12}{31^4}$
N'oublions pas de multiplier ces valeurs par la probabilité des classes, calculée précédemment :
$p(d|culture)p(culture) = \frac{2}{19^2 * 38^2} * \frac{3}{7} \approx 1,64.10^{-6}$
$p(d|societe)p(societe) = \frac{12}{31^4} * \frac{4}{7} \approx 7,42.10^{-6}$
Ce calcul indique qu'il est plus probable de tirer les mots de d dans le sac "société" que dans le sac "culture". Le programme lui associe donc la classe "société".

Nous pouvons aussi bien sûr refaire ces calculs dans le cas de la représentation dans l'espace simplifié à deux dimensions. Dans ce cas, les "sacs de mots" obtenus pour les deux classes à partir des 7 textes initiaux sont donnés par le tableau de la Figure 4.16.

Dans l'espace simplifié, le texte "En temps de crise, un cinéma ne ferme jamais !" se trouve représenté par le vecteur de coordonnées $(1, 1)$ car seuls les mots "crise" et "cinéma" sont pris en compte (le premier est rattaché par une ontologie à la dimension "économie", le deuxième à la dimension "cinéma"). Les calculs de probabilités conditionnelles sont alors les suivants :
$p(d|culture) = \frac{10}{15} * \frac{5}{15} = \frac{2}{3} * \frac{1}{3} = \frac{2}{9}$
$p(d|societe) = \frac{2}{10} * \frac{8}{10} = \frac{1}{5} * \frac{4}{5} = \frac{4}{25}$
En multipliant ces valeurs par la probabilité des classes, calculée précédemment :
$p(d|culture)p(culture) = \frac{2}{9} * \frac{3}{7} = \frac{2}{21} \approx 0,0952$
$p(d|societe)p(societe) = \frac{4}{25} * \frac{4}{7} = \frac{16}{175} \approx 0,0914$
Cette fois, c'est donc dans la classe "culture" que le nouveau texte serait rangé !

Propriétés générales

Les programmes de type "NaiveBayes" mettent en œuvre le théorème de Bayes sur des données quelconques. Il en existe plusieurs variantes, nous n'avons illustré ici que la plus "classique" d'entre elles. Ils sont dits "naifs" parce que les calculs qu'ils réalisent n'ont de sens statistique qu'à condition de faire une hypothèse d'indépendance entre les attributs qui décrivent les données, hypothèse qui est bien sûr abusive (mais que tous les programmes d'apprentissage automatique présentés dans cette section font de toute façon, à des degrés divers).

Ce sont des programmes simples, rapides et relativement efficaces pour les données textuelles. Un de leur principal intérêt est leur caractère quasi-incrémental. Comme le "modèle" sur lequel ils reposent n'est fait que de comptes de nombres d'occurrences (le tableau de la Figure 4.15 dans notre exemple), il est très facile à mettre à jour si de nouveaux exemples classés sont disponibles. C'est probablement pour cela qu'ils sont utilisés pour ranger en "spam" ou "non spam" les mails qui arrivent en flux continus dans les gestionnaires de courriers électroniques (avec l'étiquetage de l'utilisateur, s'il utilise correctement).

Leur "lisibilité est en revanche limitée car le critère utilisé pour déterminer la classe d'une nouvelle donnée est un calcul statistique peu compréhensible par les humains.

4.5 SVM

Les "Support Vector Machines" ("machines à vecteurs supports" ou "séparateurs à vastes marges" suivant les traductions !) sont des méthodes très puissantes issues d'une analyse mathématique précise et avancée du problème de l'apprentissage d'un *séparateur binaire dans un espace vectoriel*. Nous ne pourrons ici qu'en donner une intuition, sa compréhension complète requiert des connaissances mathématiques trop poussées. Jusqu'à récemment, ces méthodes donnaient la plupart du temps les meilleurs résultats.

Espace de recherche

Un SVM est un séparateur binaire, c'est-à-dire qu'il vise à séparer les données étiquetées en deux sous-ensembles disjoints. Nous supposerons donc pour l'instant que nous cherchons à l'appliquer sur un problème à *deux classes*. Les données d'apprentissage sont décrites par des points dans un espace vectoriel : les SVM s'appliquent donc essentiellement à des *données numériques*, comme celles de la Figure 4.4. Dans cet espace, les séparateurs les plus simples sont des *hyperplans*. Qu'est-ce qu'un hyperplan ? C'est très simple : dans un espace de dimension n, c'est un sous-espace de dimension $n-1$. Par exemple :

- si l'espace est de dimension 1 (une droite, sur laquelle figurent des points appartenant aux deux différentes classes), un hyperplan est de dimension 0 (c'est un point de cette droite) : un point sépare bien la droite en deux "demi-droites". Pour caractériser ce point, un seul nombre suffit : sa coordonnée $x = a$ par rapport à l'origine (le point pour lequel $x = 0$) de la droite.

– Si l'espace est de dimension 2 (un plan), un hyperplan est une droite qui coupe bien le plan en deux. Dans un espace plan où les axes s'appellent x et y, les droites ont toutes une équation de la forme : $y = ax + b$. Elles sont donc caractérisées par les deux nombres a et b.
– Pour "couper en deux" un espace de dimension 3 (comme celui dans lequel nous vivons), il faut un *plan* (un "mur infini"), c'est-à-dire un objet de dimension 2. Ils ont une équation de la forme : $z = ax = by + c$, z étant le troisième axe ajouté aux deux autres.
– Il en va de même pour n'importe quel espace de dimension n : il peut toujours être "coupé en deux" par un hyperplan de dimension $n - 1$, dont l'équation s'écrit $x_n = a_1x_1 + a_2x_2 + ... + a_{n-1}xn_1$ où x_1, x_2,. .., x_n sont les n axes de l'espace et a_1, a_2, ..., a_n les "paramètres" qui distinguent un hyperplan d'un autre.

Le but d'un apprentissage par SVM est donc de trouver le "meilleur" hyperplan séparateur rendant compte des données d'apprentissage, en choisissant la valeur des nombres a_1, a_2, ..., a_n. Cela définit l'espace de recherche de la méthode.

Technique utilisée

Suivant les données d'apprentissage, il peut n'exister *aucun hyperplan séparateur* parfait (par exemple, aucun point ne permet de séparer les "points bleus" des "points rouges" dans la Figure 4.11) ou au contraire une infinité de séparateurs différents possibles : les dessins de la Figure 4.17 montrent deux droites possibles pour nos données d'exemple.

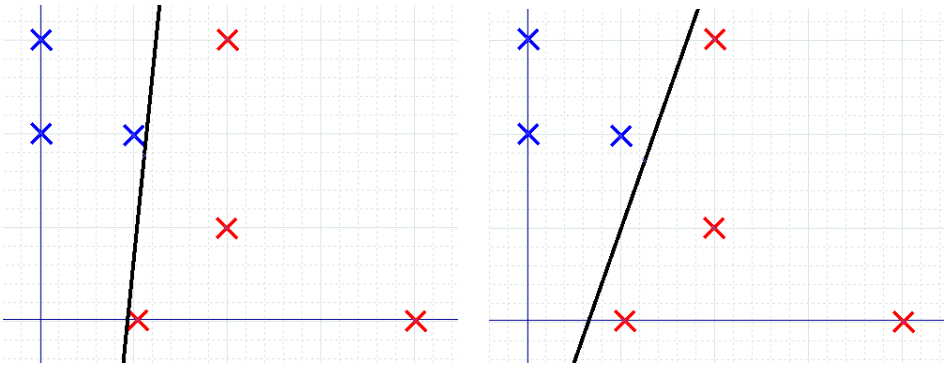


FIGURE 4.17 – Deux droites séparatrices possibles pour notre exemple

Laquelle de ces deux droites préférer ? C'est là que la notion de "marge" va intervenir. La marge d'un hyperplan séparateur est la plus petite distance qui le sépare des points le plus proches. L'algorithme des SVM va privilégier la droite qui assure la plus grande "marge" possible, c'est-à-dire celle qui "colle le moins possible" aux données. Clairement, dans notre exemple, celle de droite satisfait le mieux ce critère et va donc être choisie. On privilégie ainsi la règle qui généralise le mieux les exemples et évite le "sur-apprentissage". Par ailleurs, sur la figure, on voit bien que l'orientation générale de cette droite dépend essentiellement des deux ou trois points qui se retrouvent les plus proches d'elle : ces points sont appelés les "vecteurs

supports". Ce sont eux qui définissent où se positionne la frontière entre les deux classes, ils jouent un rôle discriminant fondamental.

Comment faire quand aucun hyperplan ne permet de séparer complètement les données des différentes classes ? Plusieurs niveaux "d'astuces" sont utilisés. Tout d'abord, il est toujours possible de chercher le meilleur hyperplan possible, si on prend en compte les données d'exemples qu'il classe mal comme des "pénalités". Un point qui se trouve du "mauvais côté" du séparateur induit une erreur qui se mesure comme son écart (sa distance) à ce séparateur. Cette erreur peut être vue comme une "marge négative". Chercher l'hyperplan induisant la plus petite somme des erreurs possible revient au même que chercher l'hyperplan assurant la plus grande marge.

Mais une "astuce de calcul" plus compliquée et plus fondamentale est aussi couramment utilisée avec les SVM, on l'appelle "l'astuce du noyau". Elle part du constat suivant : quand il est impossible de séparer deux classes dans un espace de dimension n, cela peut devenir possible en faisant subir aux données une transformation qui les envoie dans un espace de dimension plus grande. Ce phénomène est illustré dans la Figure 4.18.

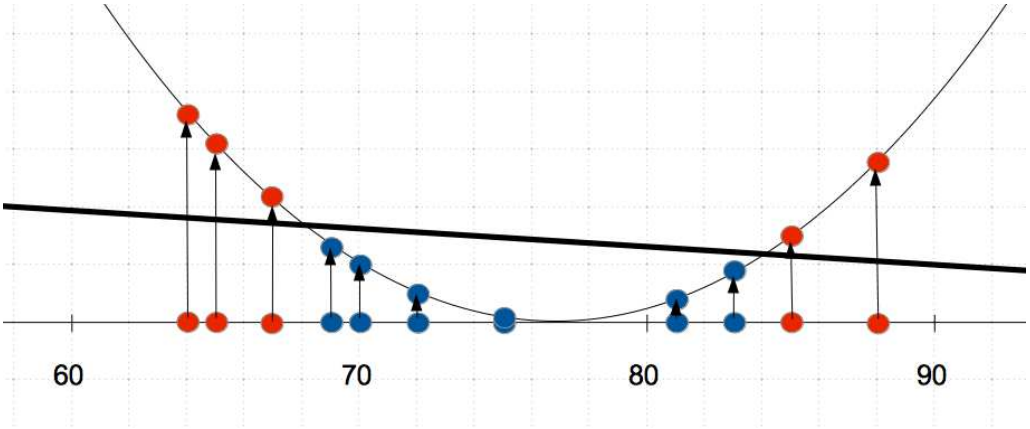


FIGURE 4.18 – Illustration de l'astuce du noyau

Initialement, on dispose de points appartenant à deux classes (couleurs) différentes sur une droite horizontale graduée. Si on reste dans l'espace (de dimension 1) de cette seule droite, il n'est pas possible de positionner un hyperplan (point) séparateur pour ces données. Quelle que soit la position de ce point, en effet, il y aura toujours un mélange de données rouges et bleues au moins d'un des deux côté de la séparation. Mais, en faisant subir une *transformation* aux points (visualisée par les flèches), on peut les positionner sur une courbe dans un espace de dimension 2. Dans ce nouvel espace, plus vaste, il devient possible de trouver un hyperplan séparateur entre les deux classes : il est représenté par la droite en gras. Les points rouges transformés se retrouvent en effet tous "au-dessus" de cette droite, tandis que les bleus restent "en-dessous".

Comment trouver une transformation si avantageuse ? Là réside une bonne part de la difficulté (mathématique) du problème. Il existe évidemment un grand nombre de transformations possibles. Mais ajouter des dimensions à l'espace initial présente l'inconvénient de rendre plus complexes les calculs de recherche de l'hyperplan "op-

timal", d'autant que les nouvelles données qu'il devra classer devront elles aussi subir la même transformation. L'astuce du noyau permet de limiter ce coût calculatoire. En effet, pour trouver les valeurs a_1, a_2, ..., a_n qui caractérisent la position du meilleur hyperplan dans le nouvel espace, il faut essentiellement être capable de calculer des *distances* : celles qui déterminent sa "marge" avec les données, notamment. Or un noyau est un moyen mathématique de calculer une distance entre des *points transformés* sans pour cela avoir besoin de connaître les coordonnées des points transformés. Dans notre exemple, cela signifie qu'on peut calculer l'écart entre les points transformés (sur la courbe) et la droite en gras uniquement en connaissant la position des points initiaux sur la droite horizontale graduée. Et il en va de même des autres données qu'on soumettra au classifieur dans la phase de test. Cette astuce est très avantageuse : cela signifie qu'avec des calculs opérant uniquement dans l'espace initial, on peut tout de même caractériser un séparateur qui n'existe réellement que dans un espace de dimension supérieure, et l'utiliser sur des nouvelles données.

Les logiciels qui implémentent les SVM (SMO dans le dossier "functions" de Weka, par exemple), requièrent toujours implicitement le choix d'un "noyau". Choisir un noyau signifie définir une distance dans un espace éventuellement plus grand que l'espace initial, qui servira de vrai critère pour la définition de la "marge" que le programme d'apprentissage cherche à optimiser. Certains noyaux ont fait leurs preuves, il sont devenus "standards". Mais il est aussi toujours possible d'en changer. De nombreux travaux de recherche dans les années 2000 ont porté sur la définition des noyaux les plus adaptés à certains types de données.

Propriétés générales

Les SVM sont des outils puissants, qui obtiennent souvent les meilleures performances en classification. Il sont difficiles à battre ! Ils sont particulièrement bien adaptés aux problèmes de classification binaire dans des espaces vectoriels de grande dimension, et s'adaptent donc bien aux textes traités en "sacs de mots". Si le problème comporte plus de deux classes, la stratégie habituelle consiste à lancer plusieurs apprentissages indépendants pour chercher à séparer deux des classes entre elles, ou une classe contre toutes les autres, et à combiner ensuite les classifieurs obtenus pour construire une réponse globale sur l'ensemble des classes. Cela fonctionne souvent très bien, le nombre de classes ne constitue donc pas un obstacle à l'application des SVM.

Mais, sur les autres plans de comparaisons entre techniques d'apprentissage automatique, ils ne se situent pas toujours au meilleur niveau. Ils requièrent ainsi en général des calculs importants qui, de plus, doivent être recommencés à zéro dès qu'une nouvelle donnée est ajoutée à l'ensemble d'apprentissage : le calcul du meilleur hyperplan séparateur n'est en effet pas incrémental. Le résultat fourni par le programme n'est pas non plus très lisible. On dispose certes, en général, des paramètres a_1, a_2, ..., a_n de l'hyperplan sélectionné : plus la valeur a_i d'un paramètre est grand, plus l'attribut (la dimension) associé(e) est important pour la tâche de classification. On dispose donc indirectement d'un classement entre les attributs. Mais c'est un indice assez faible qui reste peu exploité. La position des "vecteurs supports" donne aussi des indications sur là où passe la frontière entre les classes.

4.6 Réseaux de neurones

Espace de recherche

Technique utilisée

Propriétés générales

5 Classification par apprentissage non supervisé

L'apprentissage automatique non supervisé (ou "clustering") désigne des méthodes capables de regrouper entre elles dans des "paquets" ("clusters" en anglais) des données, sans autre information que ces données elles-mêmes (et, éventuellement, le nombre de "paquets" souhaités). C'est une tâche plus difficile que l'apprentissage automatique supervisé, car elle s'appuie sur moins d'information. Nous présentons tout d'abord les particularités générales de cette tâche, avant d'exposer trois des principales techniques employées pour l'aborder.

5.1 Spécificités de la tâche

Objectifs généraux

Critères de distinctions des différentes approches

Evaluation

5.2 Clustering hiérarchique

5.3 K-moyennes

5.4 EM

Chapitre 5

L'Annotation

HMM et CRF

Chapitre 6

L'Extraction d'Information (EI)

on se ramène à avant

Chapitre 7

Conclusion

on conclut

Chapitre 8

Bibliographie

- Amini Massih-Reza, Gaussier Eric : *Recherche d'information, Applications, modèles et algorithmes*, Eyrolles, 2013.
- Cornuejols Antoine, Miclet Laurent : *Apprentissage artificiel, Concepts et Algorithmes*, Eyrolles, 2010 (2ème édition révisée).
- Gaussier Eric, Yvon François (Eds) : *Modèles probabilistes pour l'accès à l'information textuelle*, Hermès 2011.
- Ibekwe-SanJuan Fidelia. : *Fouille de textes : méthodes, outils et applications*, Hermès, 2007.
- Preux Philippe : *Fouille de données (notes de cours)*, http ://www.grappa.univ-lille3.fr/ ppreux/Documents/notes-de-cours-de-fouille-de-donnees.pdf, 2011.

Chapitre 9

Annexes

annexes : notions mathématiques de base
- relations d'ordres, relations d'équivalence, partitions
- opérations internes, concaténations...
- espace vectoriel, coordonnées, produit scalaire
- théorème de Bayes ?