Indexation de documents par concepts de mots *

Damien Nouvel, damien.nouvel@laposte.net

Université Pierre et Marie Curie
Laboratoire d'informatique de Paris VI
8, rue du Capitaine Scott - 75015 Paris - France
http://www.lip6.fr

Abstract. Ce rapport présente l'étude, le test et l'évaluation de techniques de représentation de textes, dans le cadre de la gestion de bases de documents soumises à une classification, à l'aide d'un apprentissage de concepts de mots. Ces derniers sont déterminés selon une technique de classification bayésienne, dans le but d'indexer et de classer les documents de la base plus efficacement. Des modèles de classes de documents sont calculés, à partir d'un ensemble d'exemples. Ils prédisent la classe la plus probable pour chaque document, selon les composantes qui le caractérisent. La représentation par concepts de mots a été comparée à deux autres techniques traditionnelles, la première donnant tous les termes pour chaque document, la seconde donnant les termes qui, statistiquement, apportent le plus d'information pour chaque document. La construction des concepts de mots et l'espace de représentation sont décrits, les modèles de classes sont évalués en termes de précision et de rappel et l'ensemble de l'approche est discutée du point de vue des statistiques, des structures de données et de la linguistique.

* Merci à Massih-Réza Amini, Jean-Luc Minel, à toute l'équipe du laboratoire LIP6 pour leurs conseils dans la réalisation de ce stage

Introduction

Le stage dont ce rapport est l'objet s'est déroulé sur une période de trois mois au LIP6, sous la direction de M. Amini, de M. Gallinari et de M. Minel. Il vise à expérimenter une représentation de documents à l'aide de concepts de mots, élaborés par apprentissage. Le but de cette expérimentation est d'améliorer la classification et la recherche de documents au sein d'une base. L'apprentissage qui détermine les concepts étant bayésien, il suppose l'indépendance entre les termes présents dans la base et ne prend pas en compte la position des termes au sein de chaque document ni la polysémie de certains termes du langage.

Les documents de la base sont indexés de trois manières, en fonction de tous les termes présents dans les documents, en fonction de termes sélectionnés dans les documents et en fonction de concepts présents dans les documents. Des modèles des classes de documents sont ensuite appris à partir de ces trois représentations, qui doivent être précis et permettre de prédire correctement la classe quelque soit le document considéré. Les modèles sont donc appris, puis testés, sur deux ensembles de documents disjoints. Nous verrons comment la représentation par concepts de mots permet d'améliorer considérablement la classification automatique des documents.

Nous donnons en première partie le protocole d'expérimentation et les structures utilisées pour représenter les documents. En seconde partie, nous décrivons la base d'expérimentation et présentons les résultats obtenus pour chaque représentation. Nous discutons en dernière partie les avantages et les inconvénients de notre approche et indiquons les directions susceptibles d'améliorer notre système.

1 Protocole d'expérimentation et structures de données

1.1 Base d'indexation et base de recherche

Toute base de documents nécessite un ensemble de routines pour l'analyse et l'indexation, puis la recherche et le retrait de documents. Le but de ces techniques est donc de construire une représentation des documents, de telle manière à ce que les documents puissent être retrouvés aussi rapidement et précisément que possible. Dans le cadre de notre travail, nous appellerons l'ensemble des documents représentés une base de recherche ou d'indexation. Chacune dispose d'algorithmes associés. Nous garderons à l'esprit que le contenu de ces deux bases est strictement identique, la dénommination permettant uniquement de différencier l'utilisation que l'on souhaite en faire.

Les bases que nous considérons ici doivent alors être conçues à la lumière des algorithmes qui vont y être associés, d'une part pour l'indexation d'un document, d'autre part pour la recherche d'un document. Dans l'absolu, ces deux algorithmes sont supposés être l'inverse l'un de l'autre. La représentation associée à un document le différencie dans la base d'indexation et chaque représentation contenue dans la base de recherche mène à un document. Cette symétrie n'est pas explicite dans l'implémentation, les algorithmes n'ayant pas les même types de paramètres en entrée et ne renvoyant pas les mêmes données.

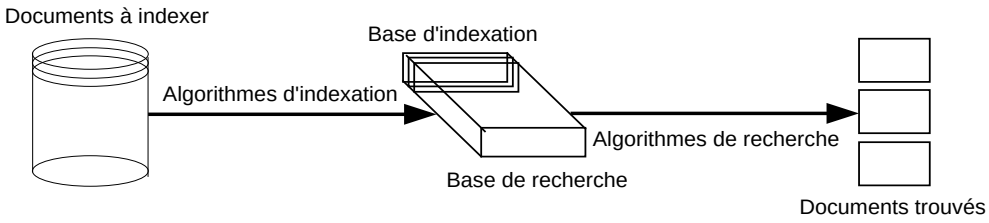

Fig. 1. Illustration de la base d'indexation et de recherche

Nous concentrons notre travail sur l'indexation, de laquelle découle en général les performances obtenues par la suite pour les recherches. Nous ne ferons pas de mesure concernant la complexité et le temps requis pour indexer les documents, nous n'évaluerons que la précision et le rappel des modèles de classes. De fait, si la rapidité est un critère d'importance pour les recherches, il est plus intéressant de garantir la précision pendant l'indexation et nous avons orienté notre travail et notre évaluation en conséquence.

1.2 Analyse de textes et structures de données

Comme à l'accoutumée, les documents sont représentés à partir du nombre d'occurence des termes qu'ils comportent. Nous n'utilisons alors que des données

statistiques pour caractériser les documents de la base. Cette forte contrainte exclut d'emblée la prise en compte de données concernant les lemmes, les structures grammaticales ou la ponctuation. Nous avons effectivement opté pour une analyse syntaxique simplifiée, les termes sont extraits comme suites de caractères, seuls les caractères de l'alphabet sont pris en compte, ils sont tous transformés en minuscules, les accents et les tremas sont supprimés. Tous les autres caractères sont considérés comme des séparateurs. Néanmoins, le contenu des balises hypertexte est évité et une liste d'arrêt de huit cent mots anglais est utilisée pour exclure les mots du langage courant.

Les documents sont donc représentés comme vecteurs de termes, selon une approche maintenant traditionnelle en traitement du langage. La base de documents est une matrice donnant pour chaque document le nombre de termes qu'il contient. Il convient d'analyser chaque document et d'y compter les occurences des termes qui y sont présents. Pendant cette étape, le dictionnaire des mots et la matrice des documents sont construits simultanément. Des indices sont attribués à chaque document et à chaque terme pour pouvoir les identifier ultérieurement. De cette manière, chaque document est une liste de couples d'entiers, le premier entier donnant l'indice et le second le nombre d'occurences pour chaque terme contenu dans le document. L'utilisation d'une liste à la place d'un vecteur permet de ne pas réserver d'espace mémoire aux termes absents d'un document. Cette méthode est particulièrement intéressante dans notre cas, au vu du nombre de dimensions des matrices et de la grande spécificité de certains mots du langage naturel.

Le dictionnaire va nous permettre d'indexer et de compter les mots dans les documents. Il est construit comme un arbre, chaque noeud contenant un caractère. Un mot est alors donné par un parcours entre la racine et un noeud de l'arbre. A l'intérieur de chaque noeud se trouve un indice qui, s'il n'est pas nul, correspond à un mot, résultant de la concaténation des caractères rencontrés sur le chemin de la racine jusqu'au noeud considéré. A l'image d'un arbre, chaque noeud peut disposer d'un frère, un autre caractère possible pour former le mot et d'un enfant, le caractère suivant pour former le mot. Avec un telle structure il devient facile d'indexer un terme puisqu'à chaque caractère rencontré dans le document il suffit de trouver ou de créer l'enfant, ou le frère de l'enfant, du noeud précédent, jusqu'à atteindre la fin du terme considéré. Cette structure n'est cependant pas adaptée pour trouver un mot à partir de son indice, fonctionnalité dont nous n'aurons pas besoin pour notre expérimentation.

Nous stucturons également la matrice, qui représente la base de documents, comme un arbre, dont les noeuds sont ordonnés selon les indices des documents. Chaque document est représenté par un arbre, dont les noeuds, également ordonnés par indices, contiennent le nombre d'occurences de chaque terme trouvé dans le document. La base de documents est alors construite comme une liste de listes et implémentée comme un arbre d'arbres. Par souci de rapidité, ces arbres sont structurés comme des arbres équilibrés dont le facteur de branchement est fixe. Tous les noeuds sont complètement remplis, sauf ceux qui contiennent les

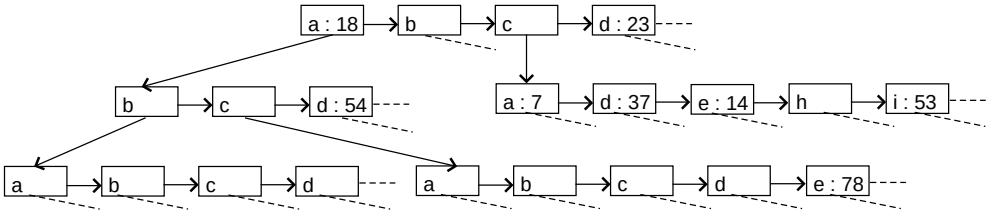

Fig. 2. Exemple de dictionnaire

feuilles de l'arbre. A la différence d'un arbre équilibré, les noeuds autant que les feuilles contiennent des données, les mécanismes d'insertion et de suppression diffèrent en conséquence.

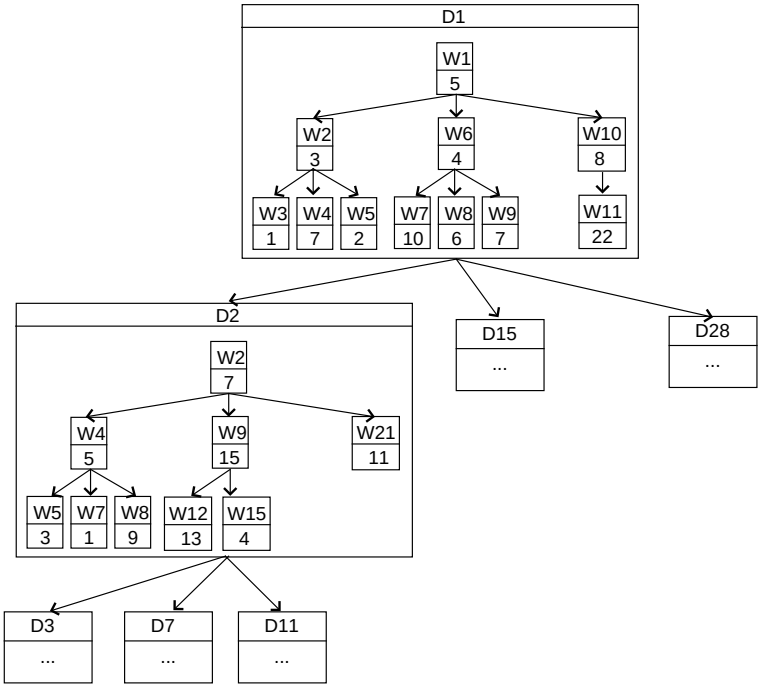

Fig. 3. Exemple de base de documents

1.3 Algorithmes d'apprentissage et de classification

Notre objectif est de représenter les documents selon les concepts de mots, qui peuvent être apparentés à des thématiques. Pour répartir les mots dans ces concepts, plusieurs algorithmes ont été étudiés, le but étant de former une partition

du dictionnaire aussi révélatrice que possible des thèmes sur lesquels portent les documents. A cet effet, les nombres d'occurences des mots dans les documents sont comparés, les mots peuvent ainsi être regroupés selon leurs distances aux concepts, ou selon leurs probabilités d'apparition dans ces concepts. Il faut appliquer un de ces deux formalismes de classification aux vecteurs qui donnent, pour un terme, ses nombres d'occurences dans les documents de la base. Ces derniers sont obtenus par transposition de la matrice originale des termes par les documents et les mêmes structures que précédemment peuvent être utilisées à cet effet.

En premier lieu, nous avons étudié l'algorithme EM (Expectation - Maximisation). Il permet l'apprentissage, à partir d'un ensemble de données, d'une hypothèse expliquant la génération des données selon un maximum de vraisemblance. Il exhibe une formule qui peut servir à déduire des données manquantes, à l'aide de données connues et de variables cachées. Cet algorithme est utilisé lorsque le calcul des paramètres d'une hypothèse est analytiquement trop complexe et peut être simplifié par la vraisemblance, ou lorsqu'un ensemble de données est incomplet et que les données manquantes doivent être déduites grâce à l'exemple des données connues. Dans le cadre de notre travail et au vu de la taille de la base, ce formalisme d'apprentissage est appliquée aux deux méthodes de classification.

Le premier algorithme de classification testé est l'algorithme de classification par moyennes (K-Means), considéré généralement comme une spécialisation de l'algorithme EM, appliqué à un mélange de probabilités suivant une loi log-normale. Les mots sont regroupés selon leurs distances à des centroides. Ces derniers sont eux-mêmes calculés par moyenne des mots qui leur sont associés. La mesure de vraisemblance est évaluée selon une loi hypersphérique gaussienne et donne une idée de la convergence de l'algorithme. L'évaluation selon une norme euclidienne reste désavantageuse, ne permettant pas d'évaluer le poids des documents lors de la catégorisation des termes. Nous avons également étudié l'algorithme X-Means, dont le nombre de centroides n'est pas précisé à l'avance et doît être déterminé dynamiquement. Nous n'utilisons pas cette version de l'algorithme, le nombre de concepts étant alors trop fortement dépendant des données utilisées à l'initialisation de l'apprentissage.

Le second algorithme est un classifieur bayésien probabiliste, qui permet de mesurer les probabilités d'apparition des mots dans les concepts. Il estime les probabilités des mots par concept, les probabilités des concepts à priori et les probabilités des mots qui n'apparaissent pas dans un concept. Cet algorithme s'est révélé tout à fait adapté à notre problématique, étant donné la taille de notre matrice, car il permet de séparer l'analyse pour chaque mot et pour chaque concept. En contrepartie, ce modèle contient un bien plus grand nombre de paramètres, ce qui explique en partie sa plus grande précision. Ayant d'abord été retenu pour apprendre les modèles de classes de documents, il sera par la suite également préféré à l'algorithme K-Means pour l'apprentissage des concepts de mots.

Les concepts de mots sont donc constitués par l'évaluation de variables cachées, qui donnent la probabilité pour chaque mot d'appartenir à un concept, grâce à un algorithme d'apprentissage. Dans le cadre de notre implémentation, les informations de structures dont nous tenons compte sont les mots et les documents. Remarquons que les classes de documents, les chapitres, les paragraphes de chaque document sont des niveaux que nous ne considérons pas dans notre approche. Les concepts peuvent être interprétés comme une représentation intermédiaire de la base, qui se situe entre la représentation par documents et la représentation par mots. Nous reviendrons et discuterons cette problématique concernant la granularité de notre représentation en dernière partie.

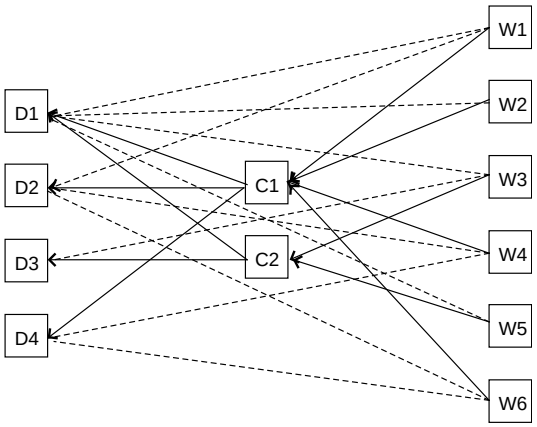

Fig. 4. Illustration de la position des concepts de mots

1.4 Réduction d'information

Pour l'analyse de grandes bases de documents, il est souvent nécessaire de réduire la taille des matrices afin de ne retenir que les composantes essentielles à la tâche considérée. De nombreuses mesures permettent d'évaluer l'importance de chaque composante afin de réduire le nombre de dimensions de la matrice représentant la base de documents. Nous donnons ici quelques techniques issues de la théorie de l'information, dans le but de modéliser les classes de documents plus finement. Mais nous garderons à l'esprit que nombre de ces techniques sont souvent équivalentes entre elles, même si la correspondance semble difficile à exhiber étant donné les grandes différences entre les formalismes utilisés pour chaque méthode.

Les transformations matricielles sont souvent utilisées dans le cadre de la réduction d'information, pour leurs fondements mathématiques solides et parce

que de nombreux algorithmes rapides existent qui facilitent leur implémentation. Les matrices carrées disposent de valeurs et de vecteurs propres qui contiennent les composantes principales de l'information contenue dans la matrice. Dans notre cas, la matrice n'est pas carrée, mais il il existe une méthode similaire, la décomposition en valeurs singulières. Celle-ci permet d'obtenir le même genre de résultats pour des matrices non carrées, en déterminant simultanément les composantes principales des lignes et des colonnes de n'importe quelle matrice rectangulaire. Il est par suite possible d'ordonner ces composantes selon l'importance des informations qu'elles contiennent et de ne conserver que les composantes principales.

La plupart des autres méthodes utilisées sont des mesures statistiques de la quantité d'information apportée par un terme au sein de la base de données. Nous pouvons citer, entre autres, la fréquence des documents par terme, le gain d'information, l'information mutuelle, le test chi-deux et la force des termes. Parmi celles-ci, la force des termes et l'information mutuelle restent assez médiocres, selon les études réalisées. Les autres sont approximativement équivalentes en terme de précision. Nous avons choisi le gain d'information, qui est obtenu à partir d'une mesure statistique de l'information apportée par un terme aux classes dans lesquelles on le retrouve. La fonction G(w) nous permet de déterminer cette valeur pour un mot, C étant l'ensemble des classes, c est une instance de classe et w une instance de mot.

$$\begin{aligned} G(w) = & -\Sigma^{c_k \in C} P(c_k) log(P(c_k)) \\ & + P(w) \Sigma^{c_k \in C} P(c_k|w) log(P(c_{k|w})) \\ & + P(\overline{w}) \Sigma^{c_k \in C} P(c_k|\overline{w}) log(P(c_k|\overline{w})) \end{aligned}$$

Nous voyons ici que le gain d'information nécessite de prendre en compte les classes dans lesquelles sont comptabilisés chaque mot, ce que les concepts de mots ne requièrent pas. Cette contrainte est assez gênante dans la mesure où l'indexation des documents n'est pas supposé être dépendant des classes de ces derniers. De plus, nous verrons que, si cette réduction d'information nous donne une méthode succinte et efficace pour représenter les documents, elle n'aide pas à discriminer les documents selon les thèmes sur lesquels ils portent.

1.5 Apprentissage de modèles de classes

L'objectif de notre étude est de trouver la classe des documents de la base grâce au un modèle construit, qui est fonction des concepts ou des termes présents dans les documents. Nous utilisons pour cela un modèle bayésien, qui exprime la probabilité qu'a un document d'appartenir à une classe, à partir de ses composantes. Comme nous l'avons expliqué plus haut, les modèles bayésiens ont l'avantage de séparer les paramètres d'apprentissage par composantes et ainsi de donner une classification assez fine. Les modèles de classes sont calculés sur un ensemble

d'apprentissage, afin d'être ultérieurement chargés dans un classifieur, qui est lui même évalué sur un ensemble de test. La construction des modèles de classes consiste alors à calculer puis à enregistrer les paramètres de ce classifieur.

Les paramètres enregistrés sont les probabilités des classes à priori, les probabilités des classes pour les composantes rencontrées dans la classe et les probabilités des classes marginales, qui sont fonction de n'importe quelle autre composante. Pour note, cette dernière probabilité est requise, par l'utilisation de valeurs logarithmiques lors de la classification des documents. Il est intéressant de passer par le logarithme afin de pouvoir additionner les logarithmes des probabilités par composantes, plutôt que de devoir multiplier des probabilités de très faible valeur. Pour des questions de précision et de complexité, le logarithme facilite beaucoup les calculs. Ainsi, lorsque la probabilité des termes absents d'une classe de documents est proche de zéro, elle n'est cependant jamais nulle, son logarithme est donc fortement négatif mais il ne peut absolument pas être omis.

Ces modèles sont appris pour la matrice originales des termes par les documents, sur la matrice des termes par documents après réduction d'information et sur la matrice des concepts de mots par documents. Cette derniére est obtenue par projection des documents sur les concepts de mots trouvés précédemment, le nombre d'occurences de chaque concept résultant simplement de la somme des occurences des mots qu'il contient. Ces trois modèles seront ensuite évalués par des mesures de précision et de rappel sur un autre ensemble de documents, ces deux ensembles étant disjoints et tirés au hasard au sein de la base de documents. Ceci garantit l'indépendance des résultats à l'ensemble d'apprentissage, tout en gardant à l'esprit que ces deux ensembles sont issus d'une même base de document et sont donc soumis à la même classification.

2 Processus, formules, bases d'expérimentation et résultats

2.1 Processus de catégorisation, d'apprentissage et d'évaluation

Le processus considéré est implémenté comme une série de programmes, qui prennent des chemins de fichiers ou de répertoires en paramètres afin de charger et d'enregistrer les données. La base de documents est également un ensemble de fichiers, chaque fichier représentant un document et chaque répertoire une classe de documents. Les programmes principaux effectuent l'indexation des textes et la création des matrices, la réduction de l'information dans ces matrices, la détermination des concepts de mots, la construction des modèles de classes. D'autres programmes permettent la transposition ou la projection de matrices, l'évaluation des modèles de classes, le tirage des données, la partition du dictionnaire.

Le processus de catégorisation, d'apprentissage et d'évaluation peut alors être écrit sous forme de script, par sérialisation des programmes qui indexent les documents, divisent la base en deux ensembles, déterminent l'espace des concepts de mots, y projettent les documents, calculent les paramètres du classifieur pour l'ensemble d'apprentissage et évaluent les modèles sur l'ensemble de test. L'ensemble d'apprentissage ne contient que le tiers de la base de documents, le reste étant dédié à l'évaluation. En annexe se trouve un exemple de script et tous les paramètres nécessaires à l'ensemble du processus.

Les algorithmes et les structures utilisés pour créer et manipuler les vecteurs, les matrices et les dictionnaires sont stockées dans une librairie. Les opérations à implémenter dans les programmes sont de plus haut niveau que celles qui concernent les structures de données, ce qui a l'avantage de les rendre plus visibles. En outre, nous pouvons ainsi plus facilement améliorer les structures de données et optimiser les algorithmes correspondants, notamment en ce qui concerne l'insertion et la recherche de valeurs dans les vecteurs et les matrices. De fait, c'est ainsi que les structures de données ont été progressivement améliorées pour aboutir aux arbres que nous avons présentés plus haut.

2.2 Probabilités bayésiennes pour la classification

Nous avons choisi d'utiliser un classificateur bayésien probabiliste dans le but de répartir les mots entre un nombre de concepts prédéterminé. L'algorithme utilisé est itératif, il converge nécessairement vers un minimum local, dépendant de l'ensemble d'apprentissage et de l'affectation initiale. Nous donnons ici les formules pour un classificateur bayésien. Dans les formules, C est l'ensemble des concept, D l'ensemble des documents, W l'ensemble des mots. Les lettre c, d et w représentent respectivement des instances de concepts, de documents ou de mots. Enfin, la norme permet de donner le nombre d'occurences d'un mot

dans un document, le nombre de mots distincts dans un concept ou le nombre de documents, de concepts, de mots distincts dans la base.

$$P(c_j|d_i) = \frac{1 + \Sigma^{w_t \in c_j} |w_t \in d_i|}{|D| + \Sigma^{d_k \in D} \Sigma^{w_t \in d_k \cap c_j} |w_t \in d_k|}$$

$$P(c_j) = \frac{|w_t \in c_j|}{|W|}$$

$$P(w_t \in c_j) = \frac{P(c_j) \Pi^{d_i \in D} P(c_j|d_i)^{|w_t \in d_i|}}{\Sigma^{c_k \in C} P(c_k) \Pi^{d_i \in D} P(c_k|d_i)^{|w_t \in d_i|}}$$

A l'initialisation, chaque mot est affecté à un concept, de manière pseudo-aléatoire, ce qui permet de calculer les probabilités de chaque concept ou pour un document d'être généré par un concept, qui correspondent aux deux premières formules. Remarquez la complexité de la troisième formule, elle ne sera pas implémentée telle quelle. Effectivement, comme nous l'avons mentionné plus haut, le logarithme va nous permettre de réduire considérablement la complexité des calculs et d'éventuelles erreurs de précision. De plus, nous n'avons besoin de déterminer que le concept le plus probable pour chaque mot. Le dénominateur étant le même pour un mot quel que soit le concept considéré, nous pouvons nous contenter, pour un mot donné, de calculer et comparer les logarithmes des numérateurs entre tous les concepts possibles, que nous notons par la fonction D(w, c) dans les formules suivantes.

$$D(w_t, c_j) = log(P(c_j)) + \Sigma^{d_i \in D} |w_t \in d_i| log(P(c_j|d_i))$$

$$\{w_t \in c_j\} = \{w_t / D(w_t, c_j) = Max^{c_k \in C} D(w_t, c_k)\}$$

Nous pouvons associer une mesure de vraisemblance au modèle que nous construisons, produit des probabilités pour un mot d'appartenir au concept auquel il est affecté. Sous forme logarithmique, c'est une somme qu'il nous est facile de calculer et qui nous permet de vérifier la convergence de notre algorithme. L'algorithme se termine lorsqu'entre deux itérations, aucune affectation n'est modifiée, auquel cas un optimum local aura été trouvé. La nature de l'algorithme EM ne permetttant pas d'atteindre l'optimum global, la solution trouvée sera dépendante de l'affectation initiale et les résultats auront alors, relativement, une qualité variable.

2.3 Concepts de mots, projection des documents et évaluation

Les concepts de mots sont calculés selon l'apprentissage bayésien et sont enregistrés comme vecteurs, dans lesquels chaque indice de mot est associé à un indice de concept. Cette représentation est la plus simple et la plus efficace pour représenter les documents sous forme de concepts de mots et utilise les mêmes

structures de données que pour les vecteurs de mots de la base de documents. Par ailleurs, les concepts de mots peuvent être visualisés grâce à un utilitaire de partitionnement du dictionnaire, qui crée un fichier par concept de mots contenant tous les termes présents dans le concept considéré. Nous présentons ici deux exemples de concepts très bien formés pour donner une idée du résultat auquel on peut s'attendre dans le meilleur des cas.

```
Concept-27 = {archives, bibliotheques, bibliotheque, centrale, disques,
    documentation, documents, fonds, images, manuscrits, ouvrages,
    periodiques, serie, specialises, zoologie}

Concept-53 = {cristaux, geants, gemmes, geologie, mineralogie, mineraux
    , pierres, quartz, roches, salle}
```

Par la suite, les documents sont projettés sur les concepts trouvés. Remarquons que les mots ne sont pas pondérés au sein des concepts, chacun étant représenté pour un document par la somme des occurences des mots présents dans le concept. Nous obtenons à nouveau une représentation de la base de document sous forme de matrice, de concepts par documents. Le même type de modèle de classes peut alors être appris pour la matrice des termes, pour la matrice des termes réduite et pour la matrice des concepts par documents.

Termes	D1	D2	D3	D4	D5
archives	3	0	12	5	8
cristaux	0	4	9	2	7
bibliothèque	13	17	0	0	2
gemmes	0	0	2	1	0
manuscrits	22	6	1	0	9
mineralogie	1	2	7	5	0

>

Concepts	D1	D2	D3	D4	D5
Concept-27	38	23	13	5	19
Concept-53	1	6	18	8	7

Table 1. Exemple de matrice de document projettée

Lorsque les modèles ont été appris, les classes de documents sont prédites sur l'ensemble de test. Des mesures de rappel et de précision nous permettent alors d'évaluer la qualité de nos modèles. Ce sont les deux critères les plus répandus dans l'évaluation de systèmes d'information, nous les rappellons ici.

$$precision = \frac{nombre\ de\ documents\ trouves\ et\ corrects}{nombre\ de\ documents\ trouves}$$

$$rappel = \frac{nombre\ de\ documents\ trouves\ et\ corrects}{nombre\ de\ documents\ corrects}$$

2.4 Bases de documents pour l'expérimentation

Afin d'obtenir des résultats significatifs, il est nécessaire de tester le système sur des bases d'assez grandes tailles, afin d'obtenir des concepts de mots aussi

généraux que possible. Nous avons donc utilisé trois bases de documents en anglais pour expérimenter le système et un site en français qui ne servira qu'à visualiser les concepts, dont sont issus les deux exemples de concepts présentés plus haut. Enfin, les trois bases de données en anglais sont réunies à titre expérimental, afin de mélanger différentes de classifications et d'évaluer la dépendance du système à une base donnée.

La première base de documents, 7Sectors, concerne un ensemble de rapports, classés par secteurs d'activités. Ceux-ci seront évalués par secteurs et par sous-secteurs d'activités puisqu'il y a deux niveaux de classification, le premier niveau contenant sept classes et le second niveau en contenant trente neuf. La seconde base de documents est le contenu de sites internet universitaires, nommée WebKB, qui contient des pages hypertextes sur les cours, les professeurs, les étudiants et les informations administratives de sept universités américaines. La troisième base de documents concerne des dépêches, classées parmi vingt genres de dépêches possibles et nommée 20News. La réunion de ces trois bases de documents est appellée All.

	Taille	Documents	Termes	Concepts	Calculs
7Sectors	27 Mo	4 580	12 322	123	30 mn
WebKB	42 Mo	8 282	19 429	194	1 h
20News	44 Mo	19 997	39 549	395	5 h
All	113 Mo	32 861	51 216	512	9 h

Table 2. Caractéristiques des bases de documents

Le nombre de concepts de mots souhaité est déterminé manuellement, puis passé en paramètre du programme de construction des concepts de mots. Celui-ci est simplement fonction du nombre de mots présents dans chaque base de documentet est rêglé arbitrairement au centième. Notons que si, en moyenne, chaque concept devrait contenir cent mots, il n'y a pas de contrainte à ce sujet lors de la détermination des concepts et le nombre de mots qu'ils contiennent sera donc déterminé par le poids de chaque mot dans les documents. Enfin, les temps de calculs ont été indiqués uniquement pour donner une idée de la complexité des algorithmes utilisés et pour montrer que cette complexité ne semble pas être linéaire avec la taille des données, mais plutôt polynômiale ou exponentielle.

2.5 Résultats expérimentaux

Les résultats que nous présentons ici sur les bases de documents permettent la comparaison de la précision et du rappel des modèles de classes de documents des trois modèles de classes de documents, en fonction des termes (dénommée Termes), des termes avec réduction d'information (dénommée Gain) et des concepts

de mots (dénommée Concepts). Comme indiqué plus haut, nous avons également expérimenté les modèles sur la base de données 7Sectors, avec le second niveau de classes de documents, qui correspond alors à la base 39Sectors. Ces résultats correspondent à une unique application des scripts, ils sont donc dépendants de l'initialisation des concepts, mais d'après notre expérience, ils varient fort peu entre deux applications des scripts.

Base	Méthode	Précision	Rappel
7Sectors	Termes	15.71 %	11.01 %
	Gain	21.84 %	13.79 %
	Concepts	45.16 %	30.49 %
39Sectors	Termes	1.38 %	1.23 %
	Gain	1.83 %	1.69 %
	Concepts	34,00 %	18.96 %
WebKB	Termes	34.09 %	20.56 %
	Gain	33.19 %	18.12 %
	Concepts	40.12 %	34.03 %
20News	Termes	15.05 %	9.61 %
	Gain	14.25 %	8.72 %
	Concepts	78.73 %	51.31 %
All	Termes	16.61 %	9.60 %
	Gain	16.26 %	9.67 %
	Concepts	63.03 %	41.56 %

Table 3. Résultats expérimentaux

Nous remarquons que la réduction d'information, si elle permet un considérable gain en espace, améliore peu les modèles de classes de documents et donne approximativement les mêmes résultats qu'avec la matrice originale. En revanche, les concepts de mots permettent d'améliorer très nettement la représentation des classes de documents, d'autant plus que la base de documents est grande et que les classes de documents sont précises. Notons plus particulièrement l'évaluation de la base 39Sectors, qui donne des résultats impressionnants pour les concepts, comparativement à ceux que donnent les deux autres méthodes.

3 Discussion sur la méthodologie et sur les modèles

3.1 Statistiques et apprentissage automatique

La méthode que nous avons mise en oeuvre pour calculer les concepts et indexer les documents est essentiellement mathématique. Il faut donc être en mesure de faire le lien entre la sémantique des termes et leurs nombres d'occurences. Pour cela, nous discutons en premier lieu de l'utilité de la liste d'arrêt. Cette liste réduit la taille des structures de données et le temps de calcul. Cependant, une bonne construction des concepts permettrait idéalement d'en former un contenant les mots du langage courant. Après expérimentation, ces mots se sont trouvés être dispatchés entre tous les concepts par la méthode bayésienne, ils ne font alors que diminuer la précision des modèles de classes. Il serait intéressant d'expérimenter une mesure tenant également compte de l'entropie des mots, ce qui amènerait à catégoriser un mot non seulement selon son domaine, mais aussi d'après son degré de spécialisation.

Les concepts de mots sont uniquement des ensembles et de ce fait restent des structures à la fois simples et efficaces pour indexer des documents. Lorsque les modèles de classes sont construits sur la matrice des termes, tous les termes sont pondérés dans ces modèles, tandis que lorsqu'ils le sont sur la matrice des concepts, chaque concept est pondéré dans le modèle, mais les mots ne le sont pas au sein de chaque concept. C'est d'ailleurs probablement ici que réside en partie la qualité des résultats, puisque les modèles de classes construits à partir des concepts mettent au même niveau tous les mots qu'ils contiennent, ce qui revient à accentuer la spécificité de chaque mot inclus dans un concept. Cette structure de concept représente à priori un net avantage, mais nous ne doutons pas que des modèles plus évolués puissent encore améliorer la précision des résultats.

Par ailleurs, chaque modèle de classe ne peut correspondre qu'à une classification, qui est forcément une partition des documents. Or il est assez rare de voir les documents classés selon un unique critère, par exemple la base de documents WebKB présente une seconde classification, qui n'est pas hiérarchique à la première, mais constitue un autre espace de classes. Ainsi, lorsque plusieurs classifications doivent être considérées, il faut construire autant de modèles de classes. Alors se pose la problématique des recherches à plusieurs critères, qui demandent à mélanger et éventuellement pondérer plusieurs classifications. Et dans le pire des cas, il peut arriver que les documents appartiennent à plusieurs classes, ce qui, actuellement, ne peut pas être pris en compte.

3.2 Analyse syntaxe et sémantique

Notre approche, orientée par les statistiques et par des méthodes d'apprentissage automatique, n'a pas utilisé de ressource linguistique pour aider à l'analyse syntaxique, à la constitution des concepts de mots et à l'indexation des documents.

Ceci est dû à la courte durée de cette expérimentation et à l'exigence de mener à bien tous les tests pour valider notre méthode quel que soit le corpus utilisé. Il reste évident que les inférences linguistiques spécifiques à une langue seront de la plus grande utilité, nous donnons ici deux problématiques, qui relevent de la syntaxe et de la sémantique et semblent pouvoir apporter beaucoup à notre approche.

L'indexation que nous réalisons est basée sur une analyse syntaxique d'une grande simplicité, les mots dérivés d'un même lemme sont dans le meilleur des cas classés dans les mêmes concepts. Une meilleure analyse syntaxique devrait donc permettre de mieux réunir les probabilités concernant un mot singulier et son pluriel ou les déclinaisons d'un verbe. Intégrer un tel analyseur devrait donc apporter beaucoup de cohérence à la détermination des concepts. Par ailleurs, ceci peut être réalisé assez simplement en amont de notre processus et ne nécessite pas de modifier le reste des programmes. Faute de temps, nous l'avons seulement visualisée sur le site en français, les concepts semblaient alors être plus précis, cela reste un prétraitement relativement aisé à mettre en oeuvre.

Nous l'avons déjà pointé du doigt dans la section précédente, l'apprentissage de modèles bayésiens, sur les concepts de mots ou sur les classes de documents, détermine une partition de ces ensembles. De ce fait, les mots ne peuvent en aucun cas appartenir à deux concepts en même temps. Or cette problématique, bien connue des linguistes, concerne tous les mots polysémiques. C'est ici un grand désavantage de notre méthode d'indexation. Ces mots ont des probabilités partagées entre deux concepts, au mieux ils sont affectés à celui des deux concepts qui lui est le plus probable, dans le pire des cas, il peut être affecté à un troisième concept qui ne lui est pas forcément très proche. Pour remédier à cela, il faudrait une structure de concepts hiérarchique, afin de pouvoir placer certains mots à l'intersection de plusieurs concepts.

3.3 Grammaire, temporalité et structure thématique

A un autre niveau de complexité, notons que les structures grammaticales peuvent également être utiles pour améliorer l'analyse des documents. Par exemple, il peut arriver que certains documents fassent apparaître des termes, mais qu'une structure grammaticale négative indique que le document ne vas pas traiter du sujet évoqué. D'autres structures grammaticales peuvent faire apparaître des exemples, ou encore des citations issues d'autres textes. Toutes ces problématiques sont très complexes et relèvent d'interprétations de haut niveau. Traiter ce type d'informations reste fastidieux, mais pourraît s'avérer intéressant pour certains types de bases de documents.

La temporalité des verbes peut aussi comporter des indications sur l'importance et le poids des propositions. Cette pondération peut éventuellement être prise en compte, par exemple lors de l'utilisation du futur ou du conditionel dont dans propositions dont la véracité pourraît alors être relativisée. Encore une fois, la

difficulté de la tâche semble hors de la portée de notre système et le gain de ce genre de technique ne peut pas vraiment être évalué pour le moment.

La technique que nous utilisons permet de mélanger plusieurs concepts au sein d'un document. Ces concepts peuvent donc être identifiés comme des thèmes, qui apparaissent et sont mélangés tout au long des documents. Une analyse plus fine, au niveau des sections, des paragraphes ou même des propositions pourraît permettre de mieux comprendre les liens qu'il peut y avoir entre divers concepts. La structure thématique d'un document, sur laquelle de nombreuses études ont été réalisées, peut donc donner plus d'information sur la proximité entre les concepts et entre les mots, nous préconisons d'approfondir la prise en compte de ces informations.

3.4 Mise à jour des concepts de mots et des modèles de classes

Comme souvent pour ce genre de systèmes, une problématique d'importance concerne les mises à jour et l'évolution de la base de documents. Nous n'avons pas orienté notre étude en ce sens, mais avons gardé à l'esprit que, dans l'application, ces mécanismes sont essentiels et doivent absolument être implémentés pour rendre l'indexation pérenne. Il n'est effectivement pas possible, dans la plupart des cas, de compiler à nouveau toute la base de documents pour déterminer quelles sont les modifications au fur et à mesure que les documents sont insérés dans la base.

De fait, tout dépend des données qui sont conservées par le classifieur. Pour les concepts de mots, il faut enregistrer le nombre d'occurences de chaque mot dans chaque document afin de pouvoir déterminer l'impact que provoque l'insertion d'un document et de tous les mots qu'il contient dans la base. Au sujet des modèles de classes, l'insertion d'un document ne nécessite que de projetter le document sur les concepts, mais il faut calculer à nouveau tous les modèle bayésiens de toutes les classe de documents. Formellement, il n'est possible de s'épargner que l'analyse syntaxique, le reste du processus devant être complètement réitéré.

Cependant, la mise à jour d'une base de documents nous amène à considérer les problématiques d'apprentissage semi-supervisé, qui semblent être les méthodes les plus adaptées aux bases de documents dont la taille évolue continuellement. Idéalement, il faudrait être capable de surveiller en temps réel l'ajout de chaque document, dont la classe pourraît être inférée automatiquement par le système, puis validée ou rectifiée par un utilisateur humain. Les concepts de mots et les modèles de classes seraient alors construit incrémentalement, à chaque insertion d'un document dans la base, sans qu'il n'y ait véritablement d'ensemble d'aprentissage ou d'ensemble de test.

3.5 Classification et recherche de documents

Nous l'avons vu au long de cette discussion, les structures que nous utilisons sont trop simples pour permettre une analyse approfondie des structures thématiques des documents ou des structures sémantiques qui peuvent être associées

à un mot. Notre formalisme réalise des partitions, il serait à priori extrêmement intéressant de mettre en oeuvre des hiérarchies, voire des treillis, pour exprimer les relations qui peuvent exister entre les classes, entre les documents, entre les paragraphes, entre les mots. Dans ce cadre, les modèles bayésiens ne sont pas forcément les plus appropriés et leurs probabilités auraient intérêt à être dérivées sur plusieurs niveaux de granularité.

Nous considérons dans notre étude que tous les documents sont du même type. Cependant il arrive souvent que le processus de classification soit utile pour différencier les documents selon leurs cas d'utilisation et non selon leurs thèmes. Dans ce cadre, la structure thématique et l'ordonnancement des thèmes au sein d'un document sont particulièrement importants. Cette problématique d'ordre reste assez complexe; il n'est pas donné qu'un apprentisssage puisse permettre de reconnaître facilement le type d'un document à partir de la structure des thèmes qui le composent.

Notre implémentation permet la classification automatique de documents à partir d'exemples, très largement améliorée grâce à l'indexation des documents par concepts de mots. Ces concepts de mots n'étant pas construits à partir des classes de document, les modèles de classes ne sont qu'une application de la représentation que nous avons mise au point. Il devient alors possible d'imaginer d'autres tâches que la classification de documents. Plus particulièrement, notre approche semble pouvoir être adaptée aux problématiques des moteurs de recherches et pourraît, nous l'espérons, y donner également de bons résultats.

Conclusion

La méthode que nous exposons ici semble tout à fait prometteuse dans le cadre de l'indexation et de la classification de documents, au vu des résultats obtenus en termes de précision et de rappel et de la généralité des techniques que nous avons utilisées. Les concepts de mots peuvent donc être considérés comme une technique à approfondir pour l'indexation de documents au sein d'une base, afin de l'améliorer, mais aussi pour y exhiber des mécanismes de mises à jour robustes.

Dans notre dernière partie, nous avons montré qu'il y avait de très nombreuses directions à explorer afin d'affiner la représentation que nous avons proposée. Parmi celles-ci, nous retenons l'apprentissage des concepts et l'analyse de documents par structures hiérarchiques. Il faut être en mesure de dériver les probabilités bayésiennes que nous avons présentées, afin de les répartir entre tous les niveaux de ce genre de structure.

En généralisant plus encore, remarquons les structures de treillis, qui résolvent en même temps les problématiques de listes d'arrêt, de polysémie et de synonymie d'une part, de structures thématiques et de classes multiples d'autre part. Cette structure nécessite un apprentissage de probabilités bien plus complexe. C'est dans cette direction que la théorie pourraît être approfondie et les techniques très nettement améliorées, afin d'inférer les corrélations qui existent entre les différentes valeurs sémantiques portées les mots et les multiples classifications auxquels sont soumis les documents d'une base.

References

1. Massih-Reza Amini, Patrick Gallinari. Semi-Supervised Learning with Explicit Misclassification Modeling.
2. Michael Berry, Zlatko Drmac, Elizabeth Jessup· Matrices, Vector Spaces, and Information Retrieval.
3. Jeff Bilmes. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report, University of Berkeley, 1997.
4. David Blei, Michael Jordan. Modeling Annotated Data.
5. Marc Caillet, Jean-François Pessiot, Massih-Reza Amini, Patrick Gallinari. Unsupervised Learning with Term Clustering For Thematic Segmentation of Texts. The 7th Proceedings of Rercherche d'Information Assistée par Ordinateur, 2004, pages 1-11.
6. Francesco De Comit, François Denis, Rémi Gilleron, Fabien Letouzey. Positive and Unlabelled Examples Help Learning.
7. Thomes Hofmann. Probabilistic Latent Semantic Indexing. Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval, 1999, pages 50-57.
8. Radford Neal, Geoffrey Hinton. A View of the EM Algorithm that Justifies Incremental, Sparse, and other Variants. Learning in Graphical Models, 1998.
9. Kamal Nigam, Andrew McCallum, Sebastian Thrun, Tom Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. Machine Learning no 39, 1999, pages 103-134.
10. Dan Pelleg, Andrew Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. Proceedings of 17th International Conference on Machine Learning, 2000, pages 727-734.
11. Yiming Yang, Jan Pedersen. A comparative Study on Feature Selection in Text Categorisation. The Fourteenth International Conference on Machine Learning, 1997, pages 412-420.

Annexes

A1. Script sur la base 7Sectors

```
indexation_documents base_7sectors fichiers_7sectors/stoplist fichiers_7sectors/dictionnaire_termes
     fichiers_7sectors/index_documents_termes 1 fichiers_7sectors/classes_documents
reduction_index fichiers_7sectors/index_documents_termes fichiers_7sectors/classes_documents 2
    fichiers_7sectors/index_documents_termesGI
tirage_index fichiers_7sectors/index_documents_termes 33 fichiers_7sectors/index_app_documents_termes
     fichiers_7sectors/index_test_documents_termes
tirage_index fichiers_7sectors/index_documents_termesGI 33 fichiers_7sectors/
    index_app_documents_termesGI fichiers_7sectors/index_test_documents_termesGI
transposition_index fichiers_7sectors/index_app_documents_termes fichiers_7sectors/
    index_app_termes_documents classification_index fichiers_7sectors/index_app_termes_documents 100
    fichiers_7sectors/concepts_termes
partition_dictionnaire fichiers_7sectors/dictionnaire_termes fichiers_7sectors/concepts_termes
    fichiers_7sectors/concepts/dictionnaire_termes_concept_
projection_index fichiers_7sectors/index_app_documents_termes fichiers_7sectors/concepts_termes
    fichiers_7sectors/index_app_documents_concepts
apprentissage_probabilites fichiers_7sectors/index_app_documents_termes fichiers_7sectors/
    classes_documents fichiers_7sectors/probabilites_termes_classes fichiers_7sectors/
    probabilites_termes_composantes fichiers_7sectors/probabilites_termes_marginales
apprentissage_probabilites fichiers_7sectors/index_app_documents_termesGI fichiers_7sectors/
    classes_documents fichiers_7sectors/probabilites_termesGI_classes fichiers_7sectors/
    probabilites_termesGI_composantes fichiers_7sectors/probabilites_termesGI_marginales
apprentissage_probabilites fichiers_7sectors/index_app_documents_concepts fichiers_7sectors/
    classes_documents fichiers_7sectors/probabilites_concepts_classes fichiers_7sectors/
    probabilites_concepts_composantes fichiers_7sectors/probabilites_concepts_marginales
projection_index fichiers_7sectors/index_test_documents_termes fichiers_7sectors/concepts_termes
    fichiers_7sectors/index_test_documents_concepts
evaluation_modeles fichiers_7sectors/probabilites_termes_classes fichiers_7sectors/
    probabilites_termes_composantes fichiers_7sectors/probabilites_termes_marginales
    fichiers_7sectors/index_test_documents_termes fichiers_7sectors/classes_documents
evaluation_modeles fichiers_7sectors/probabilites_termesGI_classes fichiers_7sectors/
    probabilites_termesGI_composantes fichiers_7sectors/probabilites_termesGI_marginales
    fichiers_7sectors/index_test_documents_termesGI fichiers_7sectors/classes_documents
evaluation_modeles fichiers_7sectors/probabilites_concepts_classes fichiers_7sectors/
    probabilites_concepts_composantes fichiers_7sectors/probabilites_concepts_marginales
    fichiers_7sectors/index_test_documents_concepts fichiers_7sectors/classes_documents
```

A2. Algortihme de classification bayésien

```
Vecteur bayes_index(Matrice donnees, long int nb_clusters){
  // Initialisation des variables
  Matrice *parcours_matrice = NULL, *parcours_matrice2 = NULL, probabilites_clusters_dimensions =
      NULL, logprobabilites_donnees_clusters = NULL, m, n;
  Vecteur *parcours_vecteur = NULL, *parcours_vecteur2 = NULL, dimensions = NULL,
      probabilites_clusters_reste = NULL, tailles_clusters = NULL, clusters = NULL, t, u, v, w;
  double logvraisemblance, x, y;
  long int nb_donnees = 0, nb_dimensions = 0, epoque = 0, stable = 0, i, j, k, l;
  while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i)){
    clusters = ajouter_vecteur_index(clusters, m->indice, nb_donnees++%nb_clusters);
    while(t = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
      dimensions = ajouter_vecteur_index(dimensions, t->indice, t->valeur);
  }
  while(t = parcourir_vecteur_index(dimensions, &parcours_vecteur, &i))
    nb_dimensions++;
  printf("Il y a %d donnees et %d dimensions\n", nb_donnees, nb_dimensions);
  // Iteration jusqu'a stabilisation des clusters
  while(++epoque < epoque_max && !stable){
    printf("Iteration %d\n", epoque);
    stable = 1;
    logvraisemblance = 0;
    // Determination des tailles de clusters
```

```
    tailles_clusters = liberer_vecteur_index(tailles_clusters);
    while(t = parcourir_vecteur_index(clusters, &parcours_vecteur, &i))
      tailles_clusters = ajouter_vecteur_index(tailles_clusters, t->valeur, 1);
    // Probabilites des dimensions pour chaque cluster
    probabilites_clusters_dimensions = liberer_matrice_index(probabilites_clusters_dimensions);
    probabilites_clusters_reste = liberer_vecteur_index(probabilites_clusters_reste);
    for(i = 0; i < nb_clusters; i++)
      probabilites_clusters_dimensions = ajouter_matrice_index(probabilites_clusters_dimensions, i,
          NULL);
    while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i))
      if(t = trouver_vecteur_index(clusters, m->indice))
        if(n = trouver_matrice_index(probabilites_clusters_dimensions, t->valeur))
          while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
            n->vecteur = ajouter_vecteur_index(n->vecteur, u->indice, u->valeur);
    while(m = parcourir_matrice_index(probabilites_clusters_dimensions, &parcours_matrice, &i)){
      x = 0;
      while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
        x += u->valeur;
      x += nb_dimensions;
      while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
        u->valeur = (u->valeur + 1)/x;
      probabilites_clusters_reste = ajouter_vecteur_index(probabilites_clusters_reste, m->indice, 1/x
          );
    }
    // Affectation des donnees aux clusters
    while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i)){
      x = 0;
      l = -1;
      while(t = parcourir_vecteur_index(tailles_clusters, &parcours_vecteur, &j)){
        if((n = trouver_matrice_index(probabilites_clusters_dimensions, t->indice))
            && (u = trouver_vecteur_index(probabilites_clusters_reste, t->indice))){
          y = log(t->valeur/nb_donnees);
          while(v = parcourir_vecteur_index(m->vecteur, &parcours_vecteur2, &k)){
            if(w = trouver_vecteur_index(n->vecteur, v->indice))
              y += v->valeur*log(w->valeur);
            else
              y += v->valeur*log(u->valeur);
          }
        }
        if(y > x || l == -1){
          x = y;
          l = t->indice;
        }
      }
      logvraisemblance += x;
      if((t = trouver_vecteur_index(clusters, m->indice)) && t->valeur != l){
        stable = 0;
        clusters = affecter_vecteur_index(clusters, m->indice, l);
      }
    }
    printf("  vraisemblance : %f\n", logvraisemblance);
  }
  return clusters;
}
```

A3. Algortihme de classification par moyennes

```
Vecteur means_index(Matrice donnees, long int nb_clusters){
  // Initialisation des variables
  Matrice *parcours_matrice = NULL, *parcours_matrice2 = NULL, centroides = NULL, m, n;
  Vecteur *parcours_vecteur = NULL, *parcours_vecteur2 = NULL, dimensions = NULL, tailles_clusters =
      NULL, clusters = NULL, t, u, v, w;
  double distance, x, y, z;
  long int nb_donnees = 0, nb_dimensions = 0, epoque = 0, stable = 0, i, j, k, l;
  while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i)){
    clusters = ajouter_vecteur_index(clusters, m->indice, nb_donnees++%nb_clusters);
```

```
    while(t = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
      dimensions = ajouter_vecteur_index(dimensions, t->indice, t->valeur);
  }
  while(t = parcourir_vecteur_index(dimensions, &parcours_vecteur, &i))
    nb_dimensions++;
  printf("Il y a %d donnees et %d dimensions\n", nb_donnees, nb_dimensions);
  // Iteration jusqu'a stabilisation des clusters
  while(++epoque < epoque_max && !stable){
    printf("Iteration %d\n", epoque);
    stable = 1;
    distance = 0;
    // Calcul des coordonnees des centroides
    tailles_clusters = liberer_vecteur_index(tailles_clusters);
    while(t = parcourir_vecteur_index(clusters, &parcours_vecteur, &i))
      tailles_clusters = ajouter_vecteur_index(tailles_clusters, t->valeur, 1);
    centroides = liberer_matrice_index(centroides);
    while(t = parcourir_vecteur_index(clusters, &parcours_vecteur, &i)){
      while(!(m = trouver_matrice_index(centroides, t->valeur)))
        centroides = ajouter_matrice_index(centroides, t->valeur, NULL);
      if((n = trouver_matrice_index(donnees, t->indice)) && (u = trouver_vecteur_index(
          tailles_clusters, m->indice)))
        while(v = parcourir_vecteur_index(n->vecteur, &parcours_vecteur2, &j))
          m->vecteur = ajouter_vecteur_index(m->vecteur, v->indice, v->valeur/u->valeur);
    }
    // Affectation des donnees aux clusters
    while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i)){
      x = 0;
      l = -1;
      while(n = parcourir_matrice_index(centroides, &parcours_matrice2, &j)){
        y = 0;
        while(t = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &k)){
          if(u = trouver_vecteur_index(n->vecteur, t->indice)){
            z = t->valeur - u->valeur;
            y += z*z;
          }else
            y += t->valeur*t->valeur;
        }
        while(t = parcourir_vecteur_index(n->vecteur, &parcours_vecteur, &k))
          if(!trouver_vecteur_index(m->vecteur, t->indice))
            y += t->valeur*t->valeur;
        if(y < x || l == -1){
          x = y;
          l = n->indice;
        }
      }
      distance += x;
      if((t = trouver_vecteur_index(clusters, m->indice)) && t->valeur != l){
        stable = 0;
        clusters = affecter_vecteur_index(clusters, m->indice, l);
      }
    }
    printf("  distance : %f\n", distance);
  }
  return clusters;
}
```

A5. Algorithme de construction des modeles de classes

```
void apprentissage_probabilites(char* chemin_donnees, char* chemin_classes, char*
    chemin_probabilites_classes, char* chemin_probabilites_classes_composantes, char*
    chemin_probabilites_marginales){
  // Initialisation des variables
  Matrice *parcours_matrice = NULL, probabilites_classes_composantes = NULL, donnees, m;
  Vecteur *parcours_vecteur = NULL, probabilites_marginales = NULL, tailles_classes = NULL,
      probabilites_classes = NULL, termes = NULL, classes, t, u, v = allouer_vecteur_index();
  double x;
```

```
  long int nb_documents = 0, nb_termes = 0, nb_classes = 0, i, j;
  // Ouverture du fichier de donnees
  printf("Recuperation des donnees sous forme de matrice\n");
  donnees = charger_matrice_index(chemin_donnees);
  // Ouverture du fichier de classes
  printf("Recuperation des classes sous forme de vecteur\n");
  classes = charger_vecteur_index(chemin_classes);
  // Apprentissage du modele de composantes
  printf("Apprentissage des probabilites par composantes et marginales (cette etape peut prendre
      plusieurs minutes)\n");
  while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i))
    if(t = trouver_vecteur_index(classes, m->indice))
      while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j)){
        v->indice = u->indice;
        v->valeur = u->valeur;
        probabilites_classes_composantes = ajouter_matrice_index(probabilites_classes_composantes, t
            ->valeur, v);
        if(!trouver_vecteur_index(termes, u->indice)){
          termes = ajouter_vecteur_index(termes, u->indice, 0);
          nb_termes++;
        }
      }
  while(m = parcourir_matrice_index(probabilites_classes_composantes, &parcours_matrice, &i)){
    x = 0;
    while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
      x += u->valeur;
    x += nb_termes;
    while(u = parcourir_vecteur_index(m->vecteur, &parcours_vecteur, &j))
      u->valeur = (u->valeur + 1)/x;
    probabilites_marginales = ajouter_vecteur_index(probabilites_marginales, m->indice, 1/(double)
        nb_termes);
  }
  // Apprentissage des probabilites des classes a priori
  printf("Apprentissage des probabilites des classes\n");
  while(m = parcourir_matrice_index(probabilites_classes_composantes, &parcours_matrice, &i))
    nb_classes++;
  while(m = parcourir_matrice_index(donnees, &parcours_matrice, &i)){
    nb_documents++;
    if(t = trouver_vecteur_index(classes, m->indice))
      tailles_classes = ajouter_vecteur_index(tailles_classes, t->valeur, 1);
  }
  while(m = parcourir_matrice_index(probabilites_classes_composantes, &parcours_matrice, &i))
    if(t = trouver_vecteur_index(tailles_classes, m->indice))
      probabilites_classes = ajouter_vecteur_index(probabilites_classes, m->indice, (t->valeur + 1)/(
          nb_classes + nb_documents));
  // Enregistrement des probabilites de classes
  printf("Enregistrement des probabilites de classes\n");
  enregistrer_vecteur_index(probabilites_classes, chemin_probabilites_classes);
  // Enregistrement des probabilites de composantes
  printf("Enregistrement des probabilites de composantes\n");
  enregistrer_matrice_index(probabilites_classes_composantes, chemin_probabilites_classes_composantes
      );
  // Enregistrement des probabilites marginales
  printf("Enregistrement des probabilites de classes marginales\n");
  enregistrer_vecteur_index(probabilites_marginales, chemin_probabilites_marginales);
}
```

A6. Entetes des structures de donnees

```
#define pi 3.1415926535
#define string_max 1000
#define largeur_index 7

typedef struct DICTIONNAIRE{
  long int indice;
  char caractere;
```

```
  struct DICTIONNAIRE* frere;
  struct DICTIONNAIRE* enfant;
}*Dictionnaire;

typedef struct VECTEUR{
  long int indice;
  double valeur;
  struct VECTEUR** enfants;
  int profondeur;
}*Vecteur;

typedef struct MATRICE{
  long int indice;
  struct VECTEUR* vecteur;
  struct MATRICE** enfants;
  int profondeur;
}*Matrice;

Dictionnaire allouer_dictionnaire();
Dictionnaire indexer_dictionnaire(Dictionnaire, char);
Dictionnaire trouver_dictionnaire(Dictionnaire, char);
void enregistrer_dictionnaire(Dictionnaire, char*);
void enregistrer_dictionnaire2(Dictionnaire, FILE*, char*, int);
void liberer_dictionnaire(Dictionnaire);

Vecteur allouer_vecteur_index();
void afficher_vecteur_index(Vecteur);
Vecteur parcourir_vecteur_index(Vecteur, Vecteur**, long int*);
Vecteur trouver_vecteur_index(Vecteur, long int);
Vecteur additionner_vecteur_index(Vecteur, Vecteur);
Vecteur affecter_vecteur_index(Vecteur, long int, double);
Vecteur ajouter_vecteur_index(Vecteur, long int, double);
Vecteur insererg_vecteur_index(Vecteur, Vecteur);
Vecteur insererd_vecteur_index(Vecteur, Vecteur);
Vecteur agrandir_vecteur_index(Vecteur);
Vecteur supprimer_vecteur_index(Vecteur, long int);
Vecteur enleverg_vecteur_index(Vecteur, long int);
Vecteur enleverd_vecteur_index(Vecteur, long int);
Vecteur reduire_vecteur_index(Vecteur);
void enregistrer_vecteur_index(Vecteur, char*);
void enregistrer_vecteur_index2(Vecteur, FILE*);
Vecteur charger_vecteur_index(char*);
Vecteur liberer_vecteur_index(Vecteur);

Matrice allouer_matrice_index();
void afficher_matrice_index(Matrice);
Matrice parcourir_matrice_index(Matrice, Matrice**, long int*);
Matrice trouver_matrice_index(Matrice, long int);
Matrice additionner_matrice_index(Matrice, Matrice);
Matrice affecter_matrice_index(Matrice, long int, Vecteur);
Matrice ajouter_matrice_index(Matrice, long int, Vecteur);
Matrice insererg_matrice_index(Matrice, Matrice);
Matrice insererd_matrice_index(Matrice, Matrice);
Matrice agrandir_matrice_index(Matrice);
Matrice supprimer_matrice_index(Matrice, long int);
Matrice enleverg_matrice_index(Matrice, long int);
Matrice enleverd_matrice_index(Matrice, long int);
Matrice reduire_matrice_index(Matrice);
void enregistrer_matrice_index(Matrice, char*);
void enregistrer_matrice_index2(Matrice, FILE*);
Matrice charger_matrice_index(char*);
Matrice liberer_matrice_index(Matrice);
```