ÉPREUVE COMMUNE DE TIPE 2009 - Partie D

TITRE :

Etude d'un algorithme de *Clustering*

Temps de préparation : ……..……2 h 15 minutes
Temps de présentation devant le jury : …….10 minutes
Entretien avec le jury : ……..10 minutes

GUIDE POUR LE CANDIDAT :

Le dossier ci-joint comporte au total : 14 pages
Document principal (12 pages, dont celle-ci)
Annexe : glossaire de 2 pages expliquant les termes notés * dans le dossier

Travail **suggéré** au candidat :

- Illustrer le fonctionnement de l'algorithme sur un exemple
- Avoir une réflexion sur l'étude de la complexité de cet algorithme
- Montrer les similarités et les différences entre le clustering géographique et le clustering de mots.
- Vous pourrez regarder comment vont être regroupés des documents contenant les mots clés de la Section IV en faisant varier les paramètres minPts et ε.
- Vérifier que les mesures de similarité respectent bien les intuitions de la Section IV, et donner des exemples sur la similarité entre documents (Jaccard et Halkidi).
- Vous pourrez discuter de l'amélioration de la similarité entre documents apportée par la mesure de Wu et Palmer par rapport à celle de Jaccard, en prenant les couples de documents (2, 3), (2, 5) et (3, 5).

Attention : s*i le candidat préfère effectuer un autre travail sur le dossier, il lui est* ***expressément recommandé*** *d'en informer le jury avant de commencer l'exposé.*

CONSEILS GENERAUX POUR LA PREPARATION DE L'EPREUVE :

* Lisez le dossier en entier dans un temps raisonnable.
* Réservez du temps pour préparer l'exposé devant le jury.

- Vous pouvez écrire sur le présent dossier, le surligner, le découper … mais tout sera à remettre au jury en fin d'oral.
- En fin de préparation, rassemblez et ordonnez soigneusement TOUS les documents (transparents, *etc.*) dont vous comptez vous servir pendant l'oral, ainsi que le dossier, les transparents et les brouillons utilisés pendant la préparation. En entrant dans la salle d'oral, vous devez être prêts à débuter votre exposé.
- A la fin de l'oral, vous devez remettre au jury le présent dossier, les transparents et les brouillons utilisés pour cette partie de l'oral, ainsi que TOUS les transparents et autres documents présentés pendant votre prestation.

Etude d'un algorithme de clustering

Introduction

Illustration 1: La carte de John Snow de 1854, 1 yard = 0.9144 m

Ce dossier présente un algorithme classique très utilisé dans le domaine de l'*Analyse de Données Spatiales* (ADS) et son application dans un contexte légèrement différent : la *Recherche d'Information* (RI). L'ADS est l'analyse de données spatiales quantitatives (ou géographiques) pour essayer d'en tirer de l'information pertinente. L'origine de cette discipline remonte au XIXe siècle avec les travaux de John Snow, sur des épidémies de choléra à Londres, comme indiqué dans l'illustration 1.

Les points sur la carte représentent des décès dûs au choléra, et les croix représentent des pompes à eau. L'objectif de son étude était de valider son hypothèse selon laquelle la dissémination du choléra se faisait par l'intermédiaire de la distribution d'eau, ce qui a été démontré en laboratoire en 1883 par Robert Koch. Pour ce faire, il a donc dressé une carte avec les décès dûs au choléra localisés spatialement. L'observation de la *densité* de décès l'a conduit à la construction d'une sorte de *cluster* (groupe), qui était corrélé avec la présence de pompes à eau contaminées. Cette expérience peut être considérée comme l'une des premières tentatives d'ADS quantitatives.

Nous allons donc étudier du point de vue informatique un algorithme de regroupement par densité en présence de bruit, appelé DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) inventé par Ester, Kriegel, Sander et Xui en 1996.

I- Survol de l'algorithme

Le pseudo-code de cet algorithme est donné en figure 1. Les paramètres de l'algorithme sont un ensemble de points D, de cardinal `card(D)` = *n*, une distance minimale ε et `minPts`, un nombre minimal de voisins pour qu'un point ne soit pas considéré comme du bruit. Le résultat est un ensemble de clusters (groupes) numérotés contenant des points, plus un ensemble de points appelés BRUIT qui ne font partie d'aucun cluster. La fonction `trouverVoisins(P, ε)` retourne l'ensemble des points voisins de P (et distincts de P), se trouvant à une distance (ici euclidienne) d'au plus ε. La fonction `card(N)` retourne le nombre de points de l'ensemble N. Dans le code on utilise des `{}` pour représenter un ensemble et `[ ]` pour représenter une structure de type produit* (les mots suivis d'un astérisque sont définis en annexe). On peut noter qu'une variation dans le choix du point initial ne changera que la numérotation des clusters et non pas les éléments les composant. La fonction `calculerCluster(M, Cluster)` permet le calcul récursif d'un cluster une fois qu'on a trouvé un premier point permettant de commencer à le construire.

```
Entrées :
            D : {point}
            minPts : entier
            ε : réel
Sortie :
            C : { [points : {point}, id : entier] }
            BRUIT : {point}
Variables locales :
            i : entier
            P : point
            N : {point}
            C_loc : [points : {point}, id : entier]
Début :
i = 0, C = {}
pour chaque point P non VISITE de l'ensemble D faire
        N = trouverVoisins (P, ε)
        si (card(N) < minPts)  alors
              marquer P comme BRUIT
        sinon
              i = i + 1
              créer un nouveau C_loc avec C_loc.id = i
              et ajouter P à C_loc.points
              marquer P comme VISITE
              pour tout Q de N faire
                    calculerCluster (Q, C_id.points)
              ajouter C_loc à C
Fin.

Fonction calculerCluster(M : point, Cluster : {point})
Debut :
        N = trouverVoisins (M, ε)
        si (card(N) < minPts)  alors
              marquer M comme BRUIT
        sinon
              marquer M comme VISITE et ajouter M à Cluster
              pour tout Q non VISITE de N faire
                    calculerCluster (Q, Cluster)
Fin.
```

Figure 1- Algorithme DB-SCAN

II- Exemple

Regardons sur un exemple rapide, figure 2, le fonctionnement de cet algorithme en prenant `minPts=2`, ce qui signifie qu'un point doit avoir au minimum 2 voisins dont le centre est à une distance inférieure à ε pour ne pas être considéré comme étant du bruit.). On voit apparaître deux clusters, C1 et C2, contenant respectivement 3 et 4 points. Les points hachurés horizontalement sont des points de type BRUIT, qui ne font partie d'aucun cluster.

Notons que l'algorithme est déterministe, puisque les résultats ne dépendent pas du choix du point initial. Ce n'est pas le cas de tous les algorithmes de regroupement par densité.

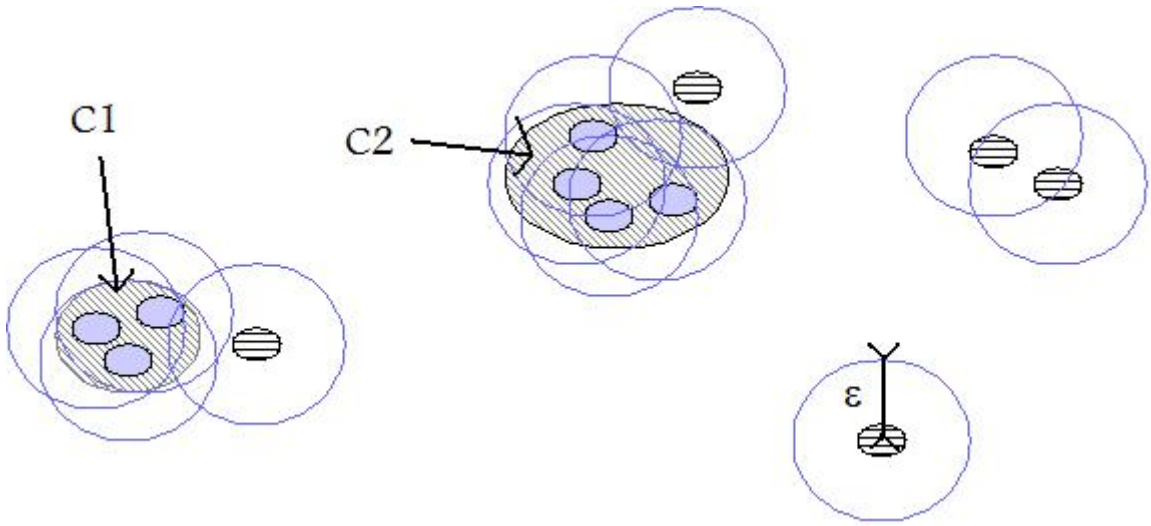


Figure 2: Exemple de fonctionnement de DB-SCAN

III- Etude de complexité

Une rapide analyse de complexité de l'algorithme de la figure 1 nous permet de voir que si toutes les opérations appelées ici se font en temps O(1), alors nous allons effectuer un nombre borné d'opérations O(1) pour chaque point de l'ensemble D, et donc que nous aurons une complexité finale qui sera *linéaire* selon le paramètre n. Poussons cette analyse davantage, car il n'est pas évident que la fonction `trouverVoisins` soit O(1).

Prenons un algorithme et des structures naïfs permettant de réaliser les fonctions `trouverVoisins(point, réel)` et `card({point})`. L'algorithme de cette fonction, donné en Figure 3, permet d'obtenir l'ensemble des voisins d'un point donné en paramètre, et le cardinal de cet ensemble en faisant appel à une fonction `distance(point, point)` qui retourne la distance entre deux points en temps O(1).

On constate que cela nous coûte un temps O(*n*) de calculer les plus proches voisins, ainsi que la taille de l'ensemble. Le temps d'exécution de l'algorithme n'est donc pas linéaire en *n* comme annoncé précédemment mais quadratique !

Mais quelle est en réalité la manière dont un utilisateur va se servir de l'algorithme? Quelles tâches va-t-il devoir effectuer de nombreuses fois, et quelles tâches pourrait-il n'exécuter qu'une seule fois ? Lors de l'utilisation d'un algorithme tel que DBSCAN qui possède des paramètres d'ajustement (`minPts` et ε), il est courant de l'exécuter plusieurs fois pour trouver des paramètres pertinents, ou bien pour réaliser un clustering à plusieurs granularités, comme montré dans la figure 4. En effectuant une fois pour toutes le calcul des plus proches voisins de chaque point, nous allons pouvoir améliorer la complexité de l'algorithme lors d'exécutions successives : si on sauvegarde pour chaque point dans un tableau `tabVoisins`, à *n* lignes et *n* colonnes, la liste ordonnée de ses plus proches voisins avec la distance les séparant de ce point, alors il est immédiat de voir qu'on pourra déterminer en un temps O(1) si ses `minPts` plus proches voisins sont à une distance de ε ou pas : il suffit de regarder la valeur de distance de l'élément `minPts` de ce tableau.

```
Entrées :
            P : point
            ε : réel
Sortie :
            V : {point}
            cardV : entier
Variables globales :
            D : {point} est l'ensemble des points
            traités dans l'algorithme
Variables locales :
            M : point
Début :
      V = {}
      cardV = 0
      pour chaque point M de D faire
            si (distance(P, M) < ε)alors
                  ajouter M à V
                  cardV = cardV + 1
 Fin.
```

Figure 3- Fonction trouverVoisins

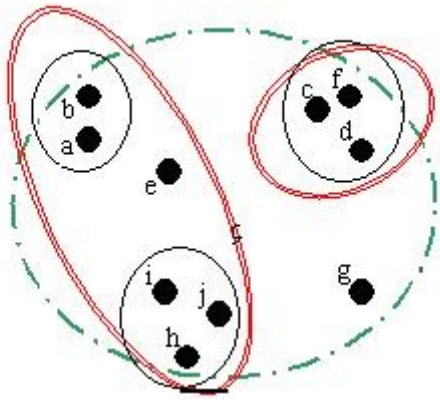


Figure 4: Exécutions successives de DBSCAN avec plusieurs valeurs de ε et `minPts = 2`

Le coût de construction du tableau des voisins est O(n^2 ln$n$) si on utilise un algorithme naïf comme celui de la Figure 5, et le temps d'exécution de DBSCAN est donc O(n). On suppose ici qu'on se donne un indiçage arbitraire des points : P_1, P_2, ..., P_n

```
Entrées :
Sortie :
                tabVoisins : tableau de n, n : [point, réel]
Variables globales :
                D : {point} est l'ensemble des points
    traités dans l'algorithme
Variables locales :
                i, j : entier
Début :
          pour i allant de 1 à n faire
                pour j allant de 1 à n faire
                      tabVoisins[i, j] = (Pj , distance(Pi,Pj))
          pour i allant de 1 à n faire
                trier par ordre croissant la ie ligne de tabVoisins
                en utilisant le 2e champ (c'est-à-dire la distance)
Fin.
```

Figure 5- Algorithme de construction de tabVoisins

Ainsi, si on prend `tabVoisins[3,5]` on connaîtra l'indice j et la distance du cinquième point le plus proche du point P_3, ce qui nous coûte O(1).

IV- Clustering de mots

La classification automatique de documents Web est un autre exemple où le clustering de données est appliqué, et où on peut réutiliser ces algorithmes pratiquement tels quels.

L'objectif de cette classification est de regrouper des documents traitant des mêmes sujets. Ce genre de classification existe depuis longtemps dans les bibliothèques : un ouvrage sera identifié comme un livre d'informatique, peut-être avec une spécialité de Recherche d'Information, selon la granularité de la classification. Dans les bibliothèques, cette classification est manuelle : il existe une liste de classes et chaque livre doit être rangé dans la bonne catégorie.

Lorsqu'on est sur Internet, et face au volume de données à traiter, il est impossible d'espérer un jour que tous les documents (sites Web) soient classés à la main. Il se présente deux problèmes :

(a) Obtenir pour chaque document une liste de mots-clés le représentant

(b) Définir une distance entre des mots-clés, pour pouvoir appliquer un algorithme de clustering.

Nous ne considérerons pas ici le problème soulevé par (a), qui peut être résolu par des techniques de *data mining*[*] textuel, permettant d'extraire un petit ensemble de mots pertinents caractérisant le document. Nous allons supposer connu cet ensemble, et nous concentrer sur le problème (b). Dans ce domaine, il est fréquent de parler de la similarité (un réel entre 0 et 1) entre documents plutôt que de leur distance (un réel entre 0 et +∞). Il est courant de passer de la distance $d(a, b)$ à la similarité $S(a, b)$ par la fonction suivante :

$$S(a,b) = \frac{2 \times \arctan\left(\frac{1}{d(a,b)}\right)}{\pi} \quad (1)$$

Voici quelques propriétés intuitives qui devront être respectées par une mesure de similarité. Pour deux ensembles A et B:

- La similarité entre A et B est fonction de ce qu'ils ont en commun. Plus ils ont d'éléments en commun, plus leur similarité sera élevée. (P1)
- La similarité entre A et B est fonction de leurs différences. Plus ils ont de différences, plus leur similarité sera faible. (P2)
- La valeur de similarité maximale est obtenue lorsque A et B sont identiques, quel que soit leur nombre d'éléments communs. (P3)

La mesure de similarité la plus courante utilisée en recherche d'information est appelée coefficient de Jaccard, qui se calcule selon la formule :

$$S(A,B) = \frac{\text{card}(A \cap B)}{\text{card}(A \cup B)} \quad (2)$$

Il est facile de vérifier que cette mesure de similarité respecte les propriétés P1, P2 et P3. Une fois que l'on dispose de cette mesure de similarité, on peut exécuter l'algorithme de clustering pour obtenir des clusters de documents similaires. On voit également apparaître la problématique du paramétrage de l'algorithme de clustering ! En effet, quelle signification faut-il donner à une similarité de 0,5 entre deux documents ? Une similarité de 0,8 ? La validation des clusters se faisant par échantillonnage et vérification manuelle, on voit qu'il est capital de pouvoir lancer de nombreuses fois l'algorithme de clustering, avec de petites variations dans les paramètres, sans pour autant recalculer toutes les similarités entre les documents. Nous sommes bien dans le cas de figure décrit dans la partie III.

Regardons comment peuvent être regroupés des documents contenant les mots-clés suivants :

ID du document	**Mots-Clés**
1	Achat
2	Achat, voiture
3	Vente, voiture
4	Vente
5	Achat, maison
6	Achat, voiture, occasion
7	Achat, voiture, neuve
8	Achat, voiture, familiale
9	Vente, maison
10	Vendre, automobile
11	Schmilblick

Tableau 1- Exemples de documents identifiés par leurs mots-clés

On voit se dessiner deux problèmes, l'un dû à l'utilisation de DBSCAN, l'autre dû à la mesure de similarité utilisée.

(a) *Les clusters n'ont parfois pas de véritable signification*, et peuvent devenir trop gros, ce qui est dû au fait que la similarité est la même entre 2 et 3 qu'entre 2 et 5, sans pour autant qu'il existe un lien sémantique fort entre 3 et 5. Il est pourtant possible qu'ils se retrouvent dans le même cluster.

(b) *La similarité entre 3 et 10 est nulle*, ce qui est dû à notre utilisation de mots-clés, sans essayer d'en comprendre leur sens.

De nombreuses recherches ont été faites pour améliorer les algorithmes de clustering, pour pallier au problème (a), tout en essayant de conserver une complexité linéaire dans le nombre de points, par exemple les travaux de Zamir et Etzioni (1998) qui utilisent des arbres des

suffixes*. Des travaux issus du domaine des ontologies ont permis d'améliorer la qualité des mesures de similarité, et c'est ce que nous allons regarder dans la dernière partie de ce dossier.

V- Distance de mots dans une ontologie

Nous définissons une ontologie T comme un graphe orienté (par exemple un arbre ou une forêt), dont les nœuds représentent des mots, et dont les arcs indiquent une relation sémantique entre le père et le fils, par exemple *être un synonyme* ou bien *être une partie de*. La Figure 7 donne un exemple d'ontologie très simple avec les termes évoqués dans le tableau 1. Nous définissons ensuite une mesure de similarité, appelée similarité de Wu et Palmer entre des nœuds d'une forêt (qui se généralise à un graphe orienté) par la formule suivante, et illustrée dans la figure 6.

- Si a et b sont connectés par un chemin et c est leur ancêtre commun le plus proche alors $S_{W\&P}(a,b) = \dfrac{2 \times \text{Path}(c)}{\text{Path}(a) + \text{Path}(b)}$ (3)
- Sinon $S_{W\&P}(a,b) = 0$

Path(c) représente la distance depuis un nœud origine R vers l'ancêtre commun de a et b, et Path(a) et Path(b) sont les plus courts chemins passant par cet ancêtre. Pour des raisons de calcul, on considère que Path(R) = 1. Ainsi sur l'exemple de la Figure 6, Path(c) = 2 Path(b) = 3 et Path(a) = 4.

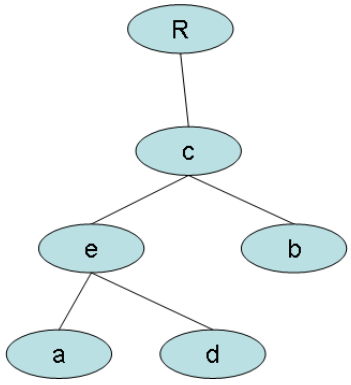


Figure 6: Exemple de distance entre nœuds d'un graphe, par rapport à une racine R

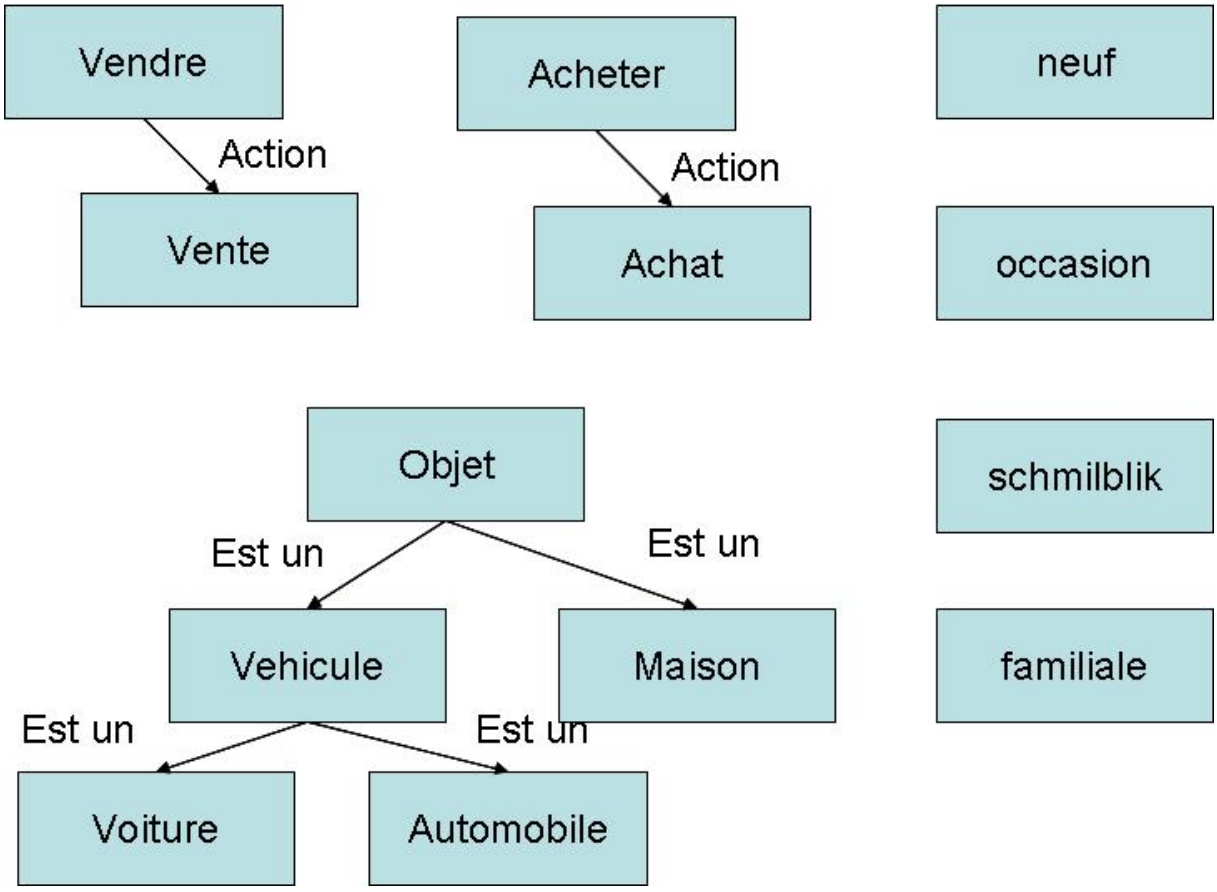
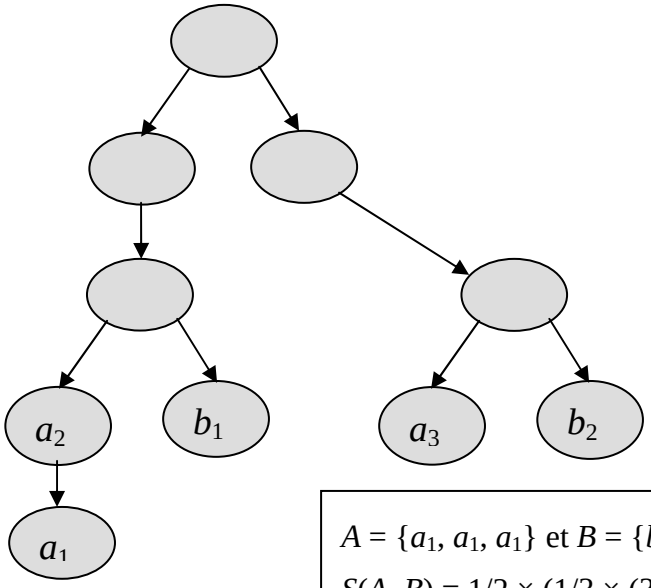


Figure 7- Exemple d'ontologie

Il nous reste à définir une distance entre des ensembles de mots A et B. On peut choisir une généralisation de la mesure de Wu et Palmer, proposée par Halkidi *et al.* en 2003, donnée par la formule suivante :

$$\zeta(A,B) = \frac{1}{2}\left(\frac{1}{\mathrm{card}(A)} \sum_{a \in A} \max_{b \in B} S_{W\&P}(a,b) + \frac{1}{\mathrm{card}(A)} \sum_{b \in B} \max_{a \in A} S_{W\&P}(a,b) \right)$$

On peut vérifier que cette mesure respecte bien les propriétés énoncées au début de la section IV. Un exemple d'application de cette mesure est donné dans la figure 8.



$A = \{a_1, a_1, a_1\}$ et $B = \{b_1, b_2\}$

$S(A, B) = 1/2 \times (1/3 \times (3/4 + 6/9 + 3/4) + 1/2 \times (3/4 + 3/4))$

$S(A, B) = 0{,}74$

Figure 8: Exemple de calcul de similarité entre ensembles de mots

Disposant désormais d'une mesure de similarité entre ensembles de mots, il est tout à fait possible d'exécuter l'algorithme DBSCAN sur les documents identifiés dans la section IV. Il est important de souligner que la qualité des résultats dépend bien sûr de la précision de l'ontologie utilisée.

Conclusion

L'utilisation d'algorithmes de clustering, dont l'origine vient de la fouille de données spatiale est utilisée aujourd'hui dans des domaines très variés comme nous l'avons montré sur l'exemple du clustering basé sur une mesure de similarité entre documents. D'une manière générale, il est possible de regrouper n'importe quel ensemble d'informations représenté sous la forme de mots-clés, Ainsi dans le domaine du Web la publicité ciblée se base sur ce principe. La difficulté que nous n'avons pas abordée ici, comme dans le cas d'ailleurs des simples documents Web, est de réussir à réduire les documents, ou les utilisateurs, à une liste de mots-clés !

ANNEXES

Glossaire

Type produit

Un type produit est composé d'un ensemble de champs, chacun pouvant être d'un type quelconque, possédant un nom de champ. Par exemple, on peut définir un type produit *Voiture* comme ayant les champs *marque* de type *chaîne de caractères* et *puissance_fiscale* de type *entier*. Dans ce cas, si on définit une variable *v* comme étant de type *Voiture* alors on peut accéder aux champs *marque* et *puissance_fiscale* en utilisant la notation pointée : *v.marque* et *v.puissance_fiscale*. On utilise ensuite *v.marque* exactement comme une variable de type *chaîne* et *v.puissance_fiscale* exactement comme une variable de type *entier* : on pourra écrire *v.marque = "Renault"* et *v.puissance_fiscale* = 7.

Data mining

Le *data mining* (en français *fouille de données* ou encore *extraction de connaissances à partir de données*) est une branche de l'informatique dont le but est l'extraction d'un savoir ou d'une connaissance à partir de grandes quantités de données en utilisant des méthodes automatiques ou semi-automatiques, que nous ne détaillons pas ici.

Le *data mining* est un processus d'analyse dont l'approche est différente de celle utilisée en *statistique*. Cette dernière présuppose en général que l'on se fixe une hypothèse que les données permettent ou non de confirmer. Au contraire, le *data mining* adopte une démarche sans a priori et essaie ainsi de faire émerger, à partir des données brutes, des inférences que l'expérimentateur peut ne pas soupçonner, et dont il aura à valider la pertinence.

Très utilisé dans le monde professionnel, l'exemple le plus connu comme application est l'étude des tickets de caisses de supermarchés, afin de découvrir quels produits sont achetés le plus souvent ensemble. Par exemple : si on achète une sauce bolognaise on achète souvent des pâtes. Plus surprenant, le *data mining* a permis de découvrir que lorsqu'on achetait de la bière on achetait aussi des couches pour bébés !

Fort de cette information, le gérant du supermarché peut choisir de mettre les produits achetés ensemble à côté les uns des autres, ou au contraire loin les uns des autres pour obliger le client à traverser tout le supermarché, et donc voir et être tenté par l'achat d'autres produits.

Un autre exemple souvent cité est le suivant : si on baisse le prix de la boisson XXX de 5%, on va par exemple en augmenter les ventes de 15%, ce que l'on savait sans *data mining*. Mais

le *data mining* révèle l'élément inattendu (bien qu'évident *a posteriori*), à savoir que les ventes des *cacahouètes* vont augmenter dans une proportion voisine (sans doute suite à l'association d'idées : « Puisque j'achète du XXX, il me faut aussi des cacahouètes » ; le *data mining* ne fait pas d'hypothèse sur le sujet). Si la marge sur la boisson XXX est relativement faible, et celle sur les cacahuètes importante, la conclusion s'impose d'elle-même : *baisser le prix de la boisson XXX est un moyen de vendre davantage de cacahouètes*.

Arbre des suffixes

La Figure 9 représente l'exemple d'arbre des suffixes utilisé par Zamir et Etzioni. L'objectif est de voir les similitudes entre trois phrases : "chat mange fromage", "souris mange fromage aussi" et "chat mange souris aussi". Chaque feuille de l'arbre est étiquetée par deux nombres, le premier correspondant au numéro de la phrase et le deuxième correspondant à la position dans la phrase du début de la sous-phrase représentée par le chemin dans l'arbre menant à cette feuille. Par exemple en passant par le nœud *d* on peut avoir soit "souris mange fromage aussi" c'est-à-dire en première position de la phrase 2, soit "souris aussi" c'est-à-dire une sous-phrase commençant à la 3^{e} position de la phrase 3. L'intérêt de cet arbre des suffixes est qu'il se construit en temps linéaire, et que le résultat va permettre de regrouper ensemble des phrases (ou documents) ayant des choses en commun.

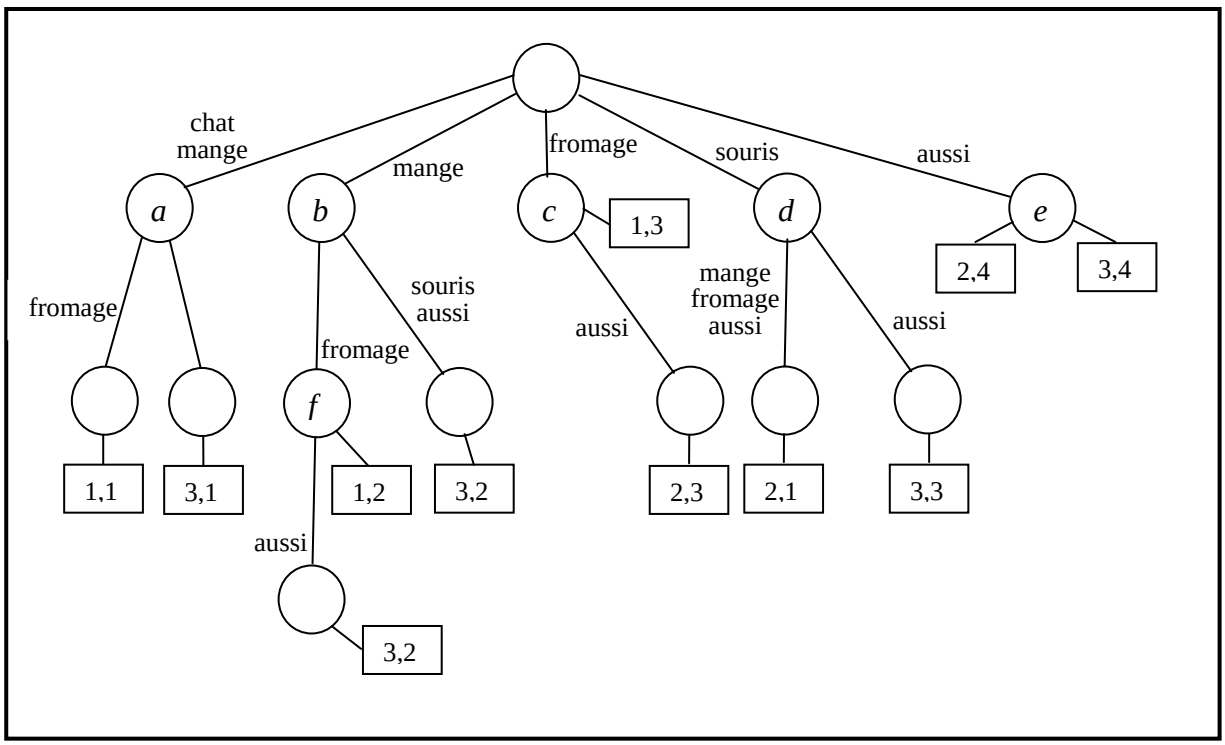


Figure 9 - Arbre des suffixes