Les algorithmes SLIQ et SSDM

Guillaume CALAS et Henri-François CHADEISSON
guillaume.calas@gmail.com et chadei_h@epita.fr

Spécialisation *Sciences Cognitives et Informatique Avancée*

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

14-16 rue Voltaire,
94270 Le Kremlin-Bicêtre,
France

Mots clés: *data mining*, SLIQ, SSDM, logique floue, classfication.

Table des matières

Chapitre 1

Introduction

Avant de voir plus en détails deux algorithmes de *data mining*, quelques petits rappels s'imposent.

1.1 Le *date mining*

1.1.1 Génèse

Avec l'avénement de l'informatique dans le monde, et à plus forte raison dans les entreprises, les sociétés engrangent des quantités gigantesques d'informations que ce soit à propos de leurs activités, de leurs produits ou de leurs clients. Ces données représentent une importe quantité d'information valorisable à condition de savoir les analyser correctement et agir en conséquence. Mais pour faire face aux milliards d'informations diverses contenues dans les entrepôts de données (*data warehouse*) d'une entreprise, une armée d'analystes ne suffit pas, et c'est dans ce contexte orienté *business intelligence* qu'est né le *data mining* (littéralement, en français, exploration de données).

1.1.2 Définition

Le *Data mining* est un domaine pluridisciplinaire permettant, à partir d'une trés importante quantité de données brutes, d'en extraire de façon automatique ou semi-automatique des informations cachées, pertinentes et inconnues auparavant en vue d'une utilisation industrielle ou opérationnelle de ce savoir. Il peut également mettre en avant les associations et les tendances et donc servir d'outil de prévisions au service de l'organe décisionnel.

On distingue le *data mining* supervisé qui sert essentiellement à la classification des données et le *data mining* non supervisé qui est utilisé dans la recherche d'associations ou les groupes d'individus.

Le champ d'action du *data mining* s'étend du chargement et du nettoyage des données (ETL[1]) dans les bases de données, à la mise en forme des résultats, en passant par le plus important : la classification et la mise en relation de différentes données, ce que nous étudierons dans le présent document.

[1] *Extract, Transform and Load*

1.1.3 Applications

De part sa spécificité et la diversité des technologies qui le compose, le *data mining* est utilisé dans domaines divers et variés :

- *marketing/stratégie* : prévisions de ventes, ciblage de clientéle, des besoins, des relations entres les différents produits, etc. ;
- *relations clients* : évaluer les risques, anticiper les attentes futures, etc. ;
- *organisation* : optimisation des chaînes de production, mise en relation des personnes, classification automatique des documents non structurés, classification des employés, etc..

Bien entendu, cette liste est loin d'être exhaustive et de nouvelles applications sont découvertes tous les jours.

1.1.4 Fonctionnement général

Le terme de *data mining* englobe souvent plusieurs processus, qui, même s'ils sont nécécessaire au bon fonctionnement de la fouille dans son ensemble, ne font pas partie du *data mining* à proprement parler. Parmi ces différents processus, on trouve :

a) *la sélection de données*, qui consiste à sélectionner quelles sont les données qui sont les plus pertinentes face à un domaine de recherche.

b) *le nettoyage de données*, qui consiste à nettoyer les données de toutes erreurs (fautes d'orthographe, bruit, erreur de champ, etc.) et de les normaliser (dates, heures, adresses, suppression des abréviations et des sigles) ;

c) *la transformation de données*, qui consiste à transformer les données dans un format, dans une structure, qui les rendra plus facilement exploitable ;

d) *l'exploration de données*, qui va classifier les données et faire apparaître les associations cachées ;

e) *l'évaluation des résultats* ;

f) *la présentation des résultats* sous forme humainement compréhensible ;

g) *l'apprentissage incrémental* qui permet d'intégrer les résultats valides afin de faire naître de nouvelles associations ou d'améliorer la précision ou la rapidité d'exécution.

1.2 Les arbres de décision

1.2.1 Définition

Les arbres de décision sont des structures permettant de trouver la solution d'un problème par l'évaluation successive de tests. Chaque nœud est un test unitaire portant sur une variable discriminante du problème et qui permet d'orienter le cheminement de la résolution.

On démarre l'évaluation à la racine de l'arbre et on termine sur une feuille : la solution.

Dans un arbre de décision, la solution correspond à une classification. On peut d'ailleurs décomposer un arbre de décision en une suite finie de règles permettant de classifier des données.

1.2.2 Caractéristiques

Les arbres de décision, qui sont des outils d'analyse, sont très appréciés pour leur lisibilité car cela permet facilement d'expliquer et de jusitifier les résultats aux décideurs. Pour avoir un bon arbre de décision, il est important de surveiller les points suivant :

- sa hauteur ;
- l'homogénéité des groupes générés.

Hauteur de l'arbre de décision

La hauteur de l'arbre de décision est un point à prendre en considération dans la recherche de performance. Plus un arbre à de niveaux, plus il y a de tests à effectuer pour arriver à une classification ce qui pèse sur les performances. On va donc chercher à construire des arbres de décision petits et équilibrés. D'autre part, afin d'améliorer les performances et la qualité de la classification, ainsi que pour éviter les effets d'un sur-apprentissage, la qualité de l'élagage de l'arbre est vitale.

Répartition des classes

L'autre point à prendre en considération est la répartition des individus entre les différentes feuilles de l'arbre. Pour être efficace, un arbre de décision doit être capable de répartir de façon homogène les individus qui lui sont proposés dans les différentes classes (les feuilles) le composant. C'est le rôle de l'algorithme de construction de l'arbre.

1.2.3 Construction de l'arbre

Les arbres de décision sont construits de façon automatiques par des algorithmes tels que ID3, CART, C4.5, C5 et SLIQ qui sera étudié dans les sections suivantes.

On construit un arbre de décision à partir d'un ensemble de données d'entraînement ce qui correspond à un ensemble de couples <donnée, classe>.

Processus

L'algorithme de construction d'un arbre de décision va évaluer l'ensemble des données et essayer d'en dégager les attributs discriminants qui permettent la meilleure séparation des données selon le critère de séparation. L'objectif final étant d'obtenir une répartition homogène et représentative des différentes classes dans l'arbre, ce qui permet d'en limiter la hauteur et donc d'en améliorer les performances.

Critère d'arrêt

L'algorithme arrête la construction de l'arbre lorsqu'il obtient des groupes purs (*id est* dont tous les éléments appartiennent à la même classe).

Élagage

Afin d'améliorer les performances de l'arbre, il est possible d'effectuer un élagage de celui-ci afin de supprimer les branches qui ne seraient pas suffisament représentatives ou résultantes du bruit dans les données de l'ensemble d'entraînement.

Chapitre 2

SLIQ

2.1 Présentation

SLIQ [1], pour *Supervised Learning In Quest*, est une algorithme récursif de classification. Il fonctionne à la fois sur des données continues et nominatives et utilise le principe de Longeur de Description Minimum (MDL) pour l'élagage de l'arbre de décision construit.

La spécificité de cet algorithme est le fait que les attributs sont séparés dans des listes triées (quand c'est possible) ce qui permet de ne pas à avoir à réorganiser tous les éléments d'un nœud. De plus, il maintient un histogramme pour chacun des nœuds afin de pouvoir facilement estimer rapidement la qualité du critère de séparation (cf. **Évaluation du critère de séparation** ci-dessous).

2.2 Algorithme

SLIQ est un algorithme de classification qui se déroule en deux étapes distinctes :

a) construction de l'arbre de décision ;

b) élagage de l'arbre de décision.

2.2.1 Construction de l'arbre de décision

Algorithme 1 MakeTree

ENTRÉES: Dataset T
Partition(T)

Critère d'arrêt

Comme la plupart des algoritmes de construction d'arbre de décision, SLIQ s'arrête lorsqu'il a aboutit à la construction d'une feuille pure.

Évaluation du critère de séparation

Le point critique de l'algorithme est l'évaluation du critère de séparation, qui, de part la nature des données, ne peut se faire sur une simple séparation numéraire. SLIQ utilise le coefficient de GINI qui permet de juger l'inégalité de répartition. Ce coefficient est défini par la relation :

$$gini(T) = 1 - \sum p_c^2$$

où p_c est la fréquence relative de la classe c dans S. Si S est pure, $gini(S) = 0$.

Le coefficient de GINI pour une partition de S en deux ensembles S_T et S_F selon le critère de séparation t est défini par la relation :

$$gini_p(S,t) = \frac{|S_T|}{|S|} gini(S_T) + \frac{|S_F|}{|S|} gini(S_F)$$

où $|S| = Card(S), |S_T| = Card(S_T), |S_F| = Card(S_F)$.

SLIQ recherche le critère de séparation qui **minimise** $gini_p(T,t)$.

Algorithme 2 Partition

ENTRÉES: Dataset S
si (tous les points de S appartiennent à la même classe) **alors**
Sortir de l'algorithme
fin si
Évaluer les partitionnements pour chaque attribut A

Utiliser le meilleur partitionnement de S aboutissant sur la création de S_1 et S_2
Partition(S_1) /* Appel récursif à gauche */
Partition(S_2) /* Appel récursif à droite */

Critère de séparation pour les attributs continus

Le critère de séparation pour un attribut continu A est une relation d'inégalité de la forme $A \leq v$ où v est une valeur continue, généralement arrondie par commodité de lecture, qui permet d'obtenir la meilleure séparation des données selon le coefficient de GINI.

Afin de ne pas à avoir réorganiser les données à chaque test d'un critère de séparation, il est important de trier par ordre croissant les *tuples* (sur la valeur de l'attribut en cours d'évaluation). On considère alors que tous les éléments situés en dessous de l'élément évalué font parti du fils droit du nœud courrant.

En général, on prend pour valeur de séparation v la moyenne entre les valeurs v_i et v_{i+1} où v_i et v_{i+1} sont deux valeurs successives.

Critère de séparation pour les valeurs nominatives

Un attribut B à une valeur nominative si la valeur de cet attribut appartient à un ensemble fini S de possibilités (par opposition à un attribut continu qui peut prendre une infinité de valeurs), *id est* $B \in S$.

On veut séparer le domaine de façon à avoir :

$$\forall B \in S, B \in S_1, B \notin S_2, S_1 \subset S, S_2 \subset S \text{ et } S_1 + S_2 = S$$

Il existe alors $2^{Card(S)}$ possibilités d'agencement entre S_1 et S_2, ce qui peut coûter très cher pour trouver la meilleure séparation. SLIQ utilise une méthode hybride afin de venir à bout de ce problème. Si la cardinalité de l'ensemble à séparer n'excède pas une certaine valeur de seuil, tous les sous-ensembles sont évalués, sinon on utilise un algorithme glouton qui va créer un sous-ensemble par ajout successif de l'élément qui réalise la meilleure séparation dans l'ensemble d'éléments restants.

2.2.2 Évaluation des séparations

L'algorithme d'évaluation des séparations va tester chacune des séparations possibles pour nœud courrant de l'arbre de décision en mettant à jour l'histogramme de ce nœud ce qui pemet de calculer le *gini* au fur et à mesure des tests. Comme les listes des attributs réels sont triées, il n'est pas nécessaire, pour chaque critère de séparation, de tester tous les éléments. Comme les listes sont triées par ordre croissant, tous les éléments qui précèdent l'élément servant de critère de séparation se retrouvent dans le fils gauche et les autres dans le fils droit. Cette technique permet de n'évaluer que $n-1$ valeurs (où n est le nombre d'élément de l'ensemble à séparer) plutôt que de se retrouver dans des complexités de l'ordre de $\sigma(n^2)$, ce qui, sur de grosse bases de données, accroit très sensiblement les performances.

Comme on peut le voir dans l'**algorithme 3**, le cas des attributs discrets est traité en dehors de la boucle par l'algorithme le mieux adapté au problème, ce qui correspond le plus généralement à un algorithme glouton.

Algorithme 3 EvaluateSplits

pour chaque attribut A **faire**
  pour chaque valeur v de la liste d'attribut **faire**
    $l \leftarrow$ feuille contenant l'élément courrant
    mettre à jour l'histogramme de la feuille l
    si A est un attribut continu **alors**
      tester la séparation pour le critère $A \leq v$ de la feuille l
    fin si
  fin pour
  si A est un attribut discret **alors**
    pour chaque feuille de l'arbre **faire**
      trouver le sous-ensemble de A qui réalise la meilleure séparation
    fin pour
  fin si
fin pour

2.2.3 Création des nouveaux nœuds

Une fois le meilleur *gini* trouvé pour le nœud courrant, SLIQ construit les nouveaux nœud avant de se relancer sur les deux nouveaux sous-ensembles.

De par la structure de données utilisée par SLIQ pour construire un arbre de décision, cela revient à :
- créer deux nouveaux histogrammes ;
- mettre à jour le tableau des classes pour chaque élément en les liant aux nouveaux histogrammes.

Algorithme 4 UpdateLabels

pour chaque attribut A utiliser comme critère de séparation **faire**
  création des nouveaux nœuds et histogrammes associés
  pour chaque valeur v de la liste de l'attribut A **faire**
    soit e la classe correspondant à l'élément courrant
    trouver la nouvelle classe c de A correspondant à la nouvelle séparation
    modifier la référence de classe de l'élément de e vers c
  fin pour
fin pour

2.2.4 Élagage de l'arbre de décision

L'élagage de l'arbre consiste à supprimer les branches de l'arbre de décision dont le taux d'erreur estimé est le plus

élévé. Cette technique a pour effet :
- de diminuer l'influence du bruit dans l'ensemble d'apprentissage ;
- d'augmenter l'erreur sur l'ensemble d'apprentissage ;
- d'améliorer le pouvoir de génaralisation de l'arbre de décision qui sont par nature sujet au sur-apprentissage avec de grosses bases de données.

Coût de codage

SLIQ utiliser le principe de Longueur de Description Minimale (MDL) qui consiste à choisir le modèle qui permet de codifier les données de la façon la plus compacte possible (données et modèle compris). Cette technique s'appuie sur les régularités des données.

Appliqué à notre problème, le principe de MDL recherche l'arbre de décision (le modèle) qui décrit les données les plus compactes possibles.

Le coût du codage des données correspond à la somme des erreurs de classifications durant la phase de construction de l'arbre.

Le coût de codage du modèle correspond au coût de codage de l'arbre de décision et du test de séparation pour chaque nœud. Le codage de l'arbre dépend de la structure de chaque nœud qui peuvent avoir, il y a trois possibilités :
- $Code_1$: un nœud peut avoir 0 ou 2 fils, soit 2 possibilités ce qui se code sur un seul bit.
- $Code_2$: un nœud peut avoir 0 fils, 1 fils gauche, 1 fils droit, ou 2 fils, ce qui se code sur 2 bits.
- $Code_3$: on examine que les nœuds internes, et dans ce cas, un nœud peut avoir 1 fils gauche, 1 fils droit, ou 2 fils, ce qui se code également sur 2 bits.

Le coût de codage du critère de séparation de chaque nœud dépend de la nature de l'attribut utilisé. S'il s'agit d'une valeur réelles, alors le coût de codage du critère correspond au coût de codage de la valeur. Par contre, si l'attribut est une valeur nominative, son coût de codage correspond à $\ln(n_{A_i})$ où n_{A_i} est le nombre de fois où l'attribut A_i est utilisé comme critère de séparation dans l'arbre de décision.

Le processus d'élagage évalue la longueur du code à chaque nœud et décide, selon la stratégie adoptée, quelle action entreprendre : ne rien faire, supprimer le fils gauche, supprimer le fils droit ou supprimer les deux fils.

La longueur de codage $C(n)$ du nœud n étant définie, selon les options, par :

$$\begin{aligned} C_{feuille}(t) &= L(t) + Erreurs_t \\ C_{total}(t) &= L(t) + L_{test} + C(t_1) + C(t_2) \\ C_{gauche}(t) &= L(t) + L_{test} + C(t_1) + C'(t_2) \\ C_{droite}(t) &= L(t) + L_{test} + C'(t_1) + C(t_2) \end{aligned}$$

où L_{test} correspond au coût de codage d'un critère de séparation et où $C'(t)$ représente le coût de codage des exemples qui appartiennent à la branche coupée en fonction des statistiques des parents du nœud.

Stratégies d'élagage

Les stratégies d'élagage sont les suivantes :

a) *Complète* : cette stratégie utilise le $Code_1$ pour coder les nœud et ne considère que les options (2.3) et (2.4). Si $C_{feuille}(t) < C_{total}(t)$ pour un nœud t alors le nœud est transformé en feuille.

b) *Partielle* : cette stratégie utilise le $Code_2$ pour coder les nœuds. Elle considère les 4 options définies ci-dessus et choisi celle qui à le code le plus court.

c) *Hybride* : premier passage avec la stratégie *Totale* puis un second passage en considérant les options (2.4), (2.5) et (2.6).

Performances

Au niveau des performances, la stratégie *Partielle* est la moins performante des trois mais génère les arbres les plus compacts.

La stratégie *Totale* génère les arbres de décision les plus performants, mais les abres sont plus gros qu'en utilisant les autres stratégies.

C'est donc la stratégie d'élagage *Hybride* qui permet d'avoir le meilleur compromis et génère des arbres dont les performances sont très proches de la stratégie *Totale* et la taille des arbres est très proche de ceux obtenus avec la stratégie *Partielle*.

Chapitre 3

SSDM

3.1 Introduction

SSDM [5] (*Semantically Similar Data Miner*) est un algorithme de découverte d'association au sein d'un large ensemble de données.

Le principe de base de cet algorithme est d'utiliser la logique floue afin de mettre en évidence des liens syntaxiques entre différents éléments.

Cet algorithme est base sur la génération de règles associatives entre deux *itemsets* extraits de la base de données. $X \Longrightarrow Y$ est une association où X et Y sont deux *itemsets*.

On définit les deux variables suivantes :
- *support* : pourcentage de transactions qui contiennent X et Y ;
- *confidence* : pourcentage de transactions contenant X qui contiennent aussi Y.

Soit le jeu de transactions suivant :

Attribut 1	*Attribut 2*
chair	table
sofa	desk
chair	desk
chair	table

$$chair \Longrightarrow table(support = 50\%, confidence = 67\%)$$
$$sofa \Longrightarrow desk(support = 25\%, confidence = 100\%)$$
$$chair \Longrightarrow desk(support = 25\%, confidence = 33\%)$$

Si on definit support ($X \Longrightarrow Y$) doit être supérieur à 50% et confidence de ($X \Longrightarrow Y$) doit être supérieur 60% alors on ne retient ici que la règle $chair \Longrightarrow table$.

En tant que chaîne de caractères, *Table* est différent de *Desk*. Cependant, ces deux attributs sont conceptuellement très proches. Le but de l'algorithme SSDM est donc de mettre en évidence des relations conceptuelles par une approche sémantique, afin de permettre de trouver de nouvelles relations.

3.2 Théorie des ensembles flous et *data mining*

La théorie des ensembles flous est une théorie mathématique du domaine de l'algèbre abstraite. Elle a été dévéloppée en 1965 par Lotfi Asker ZADEH [13] afin de représenter mathématiquement l'imprécision relative à certaines classes d'objets, et sert de fondement à la logique floue.

Les ensembles flous (ou parties floues) ont été introduits afin de modéliser la représentation humaine des connaissances, et ainsi améliorer les performances des systèmes de décision qui utilisent cette modélisation. Les ensembles flous sont utilisés soit pour modéliser l'incertitude et l'imprecision, soit pour représenter des informations précises sous forme lexicale assimilable par un système expert.

Cette théorie a beaucoup été utilisée dans différentes techniques de *data mining*, notamment dans la génération de règles associatives.

En effet, la qualité des règles associatives générées influent directement sur le rendement de l'algorithme de *data mining* en lui même.

L'utilisation des ensembles flous permet donc de créer des familles d'éléments différents combinables simultanément dans des règles.

De nombreux algorithmes utilisent la théorie des ensembles flous, tels que :
- *Lee and Lee-Kwang's algorithm* [10] ;
- *Kuok, Fu and Wong's algorithm* [9] ;
- *F-APACS* [4] ;
- *FARM* [3] ;
- *FTDA* [8].

Tous ces algorithmes utilisent la théorie des ensembles flous pour travailler sur des données de type quantitatives, et non qualitatives.

3.3 L'algorithme SSDM

3.3.1 Similarité sémantique

L'objectif du *data mining* est de découvrir un savoir dans des données. Ce savoir est extrait de deux types de données (quantitatives et qualitatives). Dans SSDM, on utilise la logique floue pour mettre des concepts en relation, à partir de données qualitatives.

La plupart des algorithmes de *data mining* travaillent directement par comparaison de chaînes de caractères des données qualitatives.

Cette méthode de comparaison est très restrictive quant au sens des données étudiées (pour ce qui est des données qualitatives).

En effet, des chaînes de caractères différentes peuvent parfois avoir un sens très proche, voire identique (exemple : *Table*/*Desk*).

On peut aussi faire des associations sémantiques sur des notions similaires, quoique différentes. Pour exemple, "Break" et "Coupe" désignent 2 types de voitures différentes.

Le but de l'algorithme SSDM est donc de réaliser une analyse sémantique sur les données qualitatives étudiées.

Lors de l'éxécution de l'algorithme, on définit ainsi des liens sémantiques entre différents termes si leur similarité est supérieure à une certaine valeur définie par l'utilisateur.

3.3.2 Structure de l'algorithme

L'algorithme a besoin que l'utilisateur définisse des données par défaut :

- *minsup* : valeur support minimum ;
- *minconf* : valeur confidence minimum ;
- *minsim* : valeur de similarite minimum avant d'établir un lien sémantique entre 2 mots différents.

Tous ces parametres sont définis entre 0 et 1.

Déroulement de l'algorithme :

a) *Data scanning* : identifier les éléments et leurs domaines ;

b) Calcul du degré de similarité entre chaque élément pour chaque domaine ;

c) Identifier les éléments qui sont considérés comme similaires ;

d) Générer les candidats ;

e) Calculer le poids de chaque candidats ;

f) Évaluer chaque candidat ;

g) Générer les règles.

Dans la suite de l'étude de cet algorithme, nous baserons nos exemples sur la base suivante :

id	Domaine 1	Domaine 2	Domaine 3
10	chair	table	wardrobe
20	sofa	desk	cupboard
30	seat	table	wardrobe
40	sofa	desk	cupboard
50	chair	board	wardrobe
60	chair	board	cupboard
70	chari	desk	cupboard
80	seat	board	cabinet
90	chair	desk	cabinet
100	sofa	desk	cupboard

Les paramètres de l'algorithme seront :

- *minimal support* (*minsup*) $= 0.45$
- *minimal confidence* (*minconf*) $= 0.3$
- *minimal similarity* (*minsim*) $= 0.8$

Data scanning

La première étape de cet algorithme est un *datascan*, qui identifie les éléments de la base. SSDM identifie chaque élément et l'associe à un domaine. Ainsi, les comparaisons sémantiques de chaque élément seront faites sur des éléments appartenants au même domaine.

Lorsque les données sont stockées dans des structures relationnelles, les domaines peuvent être définis par les libellés de chaque colonnes.

Voici le résultat du *data scanning* sur le tableau précédent :

Éléments	Domaines
sofa, chair, seat	Domaine 1
board, desk, table	Domaine 2
cabinet, cupboard, wardrobe	Domaine 3

- *Domaine 1* contient les éléments sur lesquels on peut s'asseoir.
- *Domaine 2* contient les éléments sur lesquels on peut poser quelque chose.
- *Domaine 3* contient les éléments qui peuvent contenir quelque chose.

Degrés de similarité

Une fois que les éléments et domaines sont bien établis, on détermine pour chacun les degrés de similarité entre ses éléments.

Cette étape doit être réalisée par un spécialiste pour chaque domaine (en réalité, le plus souvent, l'utilisateur final).

Il est possible d'utiliser des algorithmes pour automatiser cette tâche. Cependant, les résultats ne seront jamais aussi bons que si un être humain le fait lui même.

Les degrés de similarité sont stockés dans une matrice dite de similarité.

Exemples :

Domaine 1	*Sofa*	*Seat*	*Chair*
Sofa	1	0.75	0.7
Seat	0.75	1	0.6
Chair	0.7	0.6	1

Domaine 2	*Desk*	*Table*	*Board*
Desk	1	0.9	0.75
Table	0.9	1	0.7
Board	0.75	0.7	1

Domaine 3	*Cabinet*	*Wardrobe*	*Cupboard*
Cabinet	1	0.9	0.85
Wardrobe	0.9	1	0.8
Cupboard	0.85	0.8	1

Identification des éléments similaires, cycle de similarités

Une fois les matrices de similarités construites, on peut identifier les éléments qui sont similaires entre eux, en fonction de la variable *minsim*.

Dans ce document, on représente deux éléments similaires comme suit : $Item_1 \sim Item_2$. Une telle pair est appellee *fuzzy association* de taille 2.

Ici, avec un *minsim* égal à 0.8, on obtient la table associative suivante :

Domaine	*Valeur*	*Relation*
Domaine 2	0.9	$Desk \sim Table$
Domaine 3	0.9	$Cabinet \sim Wardrobe$
Domaine 3	0.85	$Cabinet \sim Cupboadr$
Domaine 3	0.8	$Cupboard \sim Wardrobe$

Maintenant que l'on dispose de similarités entre des paires d'éléments, on cherche à identifier des cycles entre 3 éléments ou plus (au maximum la taille du domaine).

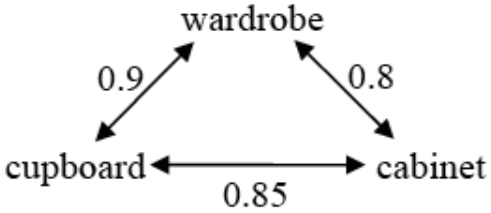

FIG. 3.1 – Cycle de similarité

Algorithme 5 Algorithme pour trouver les cycles de similarité

$A_2 =$ ensemble d'associations floues de taille 2
pour $(k = 3; k < size(D_n); k{+}{+})$ **faire**
  comparer chaque paire d'association floues dans A_{k-1}
  si $(\text{prefixe}(a_i) = \text{prefixe}(a_j), i \neq j)$ **alors**
    // Si l'union des suffixes est suffisament similaire
    si $((\text{suffix}(a_i) \cup \text{suffix}(a_j)) \in A_2)$ **alors**
      $a_k = a_i \cup \text{suffix}(a_j)$
    fin si
  fin si
  fincomparer
  $A_k =$ ensemble de tous les a_k
fin pour
$S_n =$ groupe de tous les A_k

Génération des candidats

La génération d'ensembles de candidats se fait de façon classique (cf. algorithme *Apriori*), couplée à l'utilisation des similarités entre les éléments.

Une fois cette génération faite, on calcule le poids de chaque ensemble.

Calcul du poids de chaque ensemble candidat

Le poids d'un ensemble de candidats correspond au nombre d'occurences trouvées dans la base de donneées.

Dans SSDM, deux cas de figure se présentent pour chaque ensemble :
- L'ensemble est pur (aucun élément n'a été ajouté par similarité).
- L'ensemble flou (*fuzzy set* : présence d'éléments détectés par similarité).

Dans le premier cas, le poids de l'ensemble est calculé de manière classique (on calcule la taille de l'ensemble).

Dans le second cas, on utilise le degré de similarité des éléments afin de pondérer le poids de l'ensemble.

Un élément est choisit comme référence, et sa distance avec les autres éléments est utilisée dans le calcul du poids.

Exemple :

Soit la matrice de similarité suivante :

	$Item_1$	$Item_2$
$Item_1$	1	0.8
$Item_2$	0.8	1

On dispose d'un *dataset* contenant deux fois l'$Item_1$ et une fois l'$Item_2$. L'$Item_1$ a un degré de similarité de 1 avec lui même, et de 0.8 avec l'$Item_2$. Ainsi, le poids obtenu est : $(2 \times 1) + (1 \times 0.8) = 2.8$.

En revanche, si l'on considère le calcul en se basant sur l'$Item_2$, on obtient le résultat suivant : $(2 \times 0.8) + (1 \times 1) = 2.6$.

Selon le référentiel dans lequel on se place, la formule utilisée ne donne pas les mêmes résultats.

Dans le premier cas, la formule du poids est :

$$poids(Item_1) + poids(Item_2) \times sim(Item_1, Item_2)$$

Dans le second cas, on a :

$$poids(Item_1) \times sim(Item_1, Item_2) + poids(Item_2)$$

Afin de palier a ce problème, il a été décidé de calculer la moyenne de ces deux formules :

$$\begin{aligned} poids &= [poids(Item_1) + poids(Item_2)] \\ &\times \frac{[1 + sim(Item_1, Item_2)]}{2} \end{aligned}$$

Cette formule marche bien pour le calcul sur des associations composées de deux éléments. Il est cependant nécessaire de pouvoir généraliser le calcul à des associations de trois (ou plus) éléments.

On utilise un facteur nommé f, qui est le plus petit coefficient de similarité présent dans les associations considérées, et défini par :

$$f = min(sim(Item_1, Item_2), ..., sim(Item_{n-1}, Item_n))$$

Pour obtenir la formule suivante :

$$poids = \left[\sum_{i=1}^{+\infty} poids(Item_i)\right] \left[\frac{1+f}{2}\right]$$

Évaluation des candidats

C'est dans cette phrase de l'algorithme que l'on évalue le *support* de chaque ensemble d'éléments, défini par :

$$Support = \frac{poids(\text{Ensemble d'éléments})}{\text{Nombre d'entrées dans la base de données}}$$

Si le support est supérieur à la valeur de *minsup*, alors l'ensemble est sélectionné. Dans le cas contraire, l'ensemble n'est pas considéré comme contenant des éléments fréquents, et est donc écarté.

On peut désormais passer à la génération des règles.

Génération des règles associatives

Une règle est composée de deux éléments : un père et un fils.

Dans cette dernière étape, l'algorithme SSDM génère toutes les combinaisons de règles possibles entre chaque ensemble, et les évalue par le calcul de la *Confidence*. Ce calcul est strictement le même, que les ensembles soient purs ou flous :

$$Confidence = \frac{Support(Fils)}{Support(Pre)}$$

Si la *Confidence* est superieure à la valeur prédéfinie de *minconf*, alors la règle est considérée comme valide. Dans le cas contraire, elle est écartée.

3.4 Conclusion

Cet algorithme est novateur. En effet, c'est la premiere fois qu'on attache de l'importance au sens sémantique des mots dans un algorithme de *data mining*.

Les tests réalisés par les chercheurs qui ont mis au point cet algorithme a révélé de nombreuses règles que d'autres algorithmes ne pouvaient pas trouver.

Ces règles représentent les similarités sémantiques entre les données, ce qui les rend plus riche très proche d'une réflexion humaine (notamment grâce à l'utilisation de concepts de logique floue).

Cette même équipe a publié en 2006 un rapport sur une extention de l'algorithme SSDM [6], dont l'objet est d'aller encore plus loin dans la création de liens sémantiques quantifiés entre différents éléments.

Les objectifs pour le futur sont multiples.

Une première idée serait d'améliorer les performances de cet algorithme (son exécution est très lente) en modifiant les structures de données utilisées.

De plus, il serait très intéressant de chercher des techniques de définition de domaines de données (à ce jour, un domaine est défini par un champs de la base de données).

Enfin, une nouvelle syntaxe d'expression des règles, mettant au mieux en évidence les liens sémantiques mis en œuvre dans les *itemsets* n'est pas à exclure.

Bibliographie

[1] Manish Mehta 0002, Rakesh Agrawal, and Jorma Rissanen. Sliq : A fast scalable classifier for data mining. In Apers et al. [2], pages 18–32.

[2] Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors. Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings, volume 1057 of Lecture Notes in Computer Science. Springer, 1996.

[3] W. Au and K. Chan. Farm : A data mining system for discovering fuzzy association rules. In Proc. of the 8th IEEE Int'l Conf. on Fuzzy Systems, pages 1217–1222, Seoul, Korea, 1999.

[4] Keith C. C. Chan and Wai-Ho Au. An effective algorithm for mining interesting quantitative association rules. In SAC, pages 88–90, 1997.

[5] Eduardo L. G. Escovar, Mauro Biajiz, and Marina Teresa Pires Vieira. Ssdm : A semantically similar data mining algorithm. In Heuser [7], pages 265–279.

[6] Eduardo L. G. Escovar, Cristiane A. Yaguinuma, and Mauro Biajiz. Using fuzzy ontologies to extend semantically similar data mining. In Nascimento [11], pages 16–30.

[7] Carlos A. Heuser, editor. 20 Simpósio Brasileiro de Bancos de Dados, 3-7 de Outubro, 2005, Uberlândia, MG, Brazil, Anais/Proceedings. UFU, 2005.

[8] Tzung-Pei Hong, Chan-Sheng Kuo, Sheng-Chai Chi, and Shyue-Liang Wang. Mining fuzzy rules from quantitative data based on the aprioritid algorithm. In SAC (1), pages 534–536, 2000.

[9] Chan Man Kuok, Ada Wai-Chee Fu, and Man Hon Wong. Mining fuzzy association rules in databases. SIGMOD Record, 27(1) :41–46, 1998.

[10] Jee-Hyong Lee and Hyung Lee-Kwang. Fuzzy identification of unknown systems based on ga. In Yao et al. [12], pages 216–223.

[11] Mario A. Nascimento, editor. XXI Simpósio Brasileiro de Banco de Dados, 16-20 de Outubro, Florianópolis, Santa Catarina, Brasil, Anais/Proceedings. UFSC, 2006.

[12] Xin Yao, Jong-Hwan Kim, and Takeshi Furuhashi, editors. Simulated Evolution and Learning, First Asia-Pacific Conference, SEAL'96, Taejon, Korea, November 9-12, 1996, Selected Papers, volume 1285 of Lecture Notes in Computer Science. Springer, 1997.

[13] Lotfi A. Zadeh. Fuzzy sets. Information and Control, 8(3) :338–353, 1965.