Source de conférences : NIPS / ECML / PKDD / KDD / ICDM / ECAI / IJCAI / AAAI / SIAM DM

Data Mining : Pattern mining

Exemple de Carrefour :
Client → achète → BDD local → BDD
On obtient un bout de son intention qui nous permet après recoupement avec plusieurs personnes de prévoir les patterns d'achats.

Exemple sur les étudiants de la classe :

	Riz	Oeufs	Pâtes	Bonbons	Poulet	Yaourt	Céréales	Avoine	Lait de soja
François	x	x	x						
Florence	x	x	x	x					
Marc	x	x	x		x				
Julien		x	x			x	x		
Nadia	x	x	x					x	x
Alexandre				x				x	x

{Riz, Oeufs, Pâtes} : 4/6 → 66% transactions
{Oeufs, Pâtes} : 5/6 → 84% transactions
{Avoine, Lait de soja} : 2/6 → 33% transactions

Fayyad (1996) : but du data mining
- analyser informatiquement de gros volumes de données
- trouver des "patterns"
- patterns doivent être :
    - nouveaux
    - interprétables
    - utiles (si possible)

Le pattern mining permet de découvrir de nouvelles connaissances (exemple des couche et bière de Walmart, étude sur la fainéantise des jeunes pères).

Sur une matrice de données (cf matrice ci-dessus) :
- lignes = transactions
- colonnes = items

itemset = ens. d'items
Support (itemset) = | { ti | itemset inclu dans ti } |
support_relatif(itemset) = support(itemset) / #transactions

ensemble des items I = { i1, … , in }
ensemble des transactions T = { t1, … , tn }

étant donné un seuil de fréquence minimale epsilon,
itemset fréquent si support(itemset) >= **ε**

→ Pb étant donné T, I, **ε**, trouver tous les itemsets fréquents

Générer un itemset
et
Tester sa fréquence

#total d'itemsets : 2^n

Prop d'anti-monotonie :

- (1993) Si itemset X infréquent, alors ∀ X'⊃X, X' infréquent

Algorithme Apriori
Exploration par niveaux de l'espace de recherche

- itemset taille 1
- itemset taille 2
- …

Algo en largeur

Ci : candidats du niveau i Li : fréquents du niveau i

apriori

```
L1 ← itemset fréquent de taille 1
k = 2
tant que Lk != Ø faire
        // Générer
        Ck+1 ← apriori_gen(Lk)
        // Tester
        Forall t ∈ T
                Forall c ∈ Ck+1
                        if c ⊆ t
                                c.count++
                Lk+1 ← { c∈ Ck+1 | c.count >= ε
                k++
        return ∪k (Lk)
```

apriori_gen(Lk)

```
// Génération
Ck+1 ← {
        select p, q
        from Lk
        where p[1] = q[1], … , p[k-1] = q[k-1] and p[k] < q[k]
// ne génère pas de doublons
}
// pruning (élagage)
Forall c ∈ Ck+1
        Forall s ∈ k-subset de C
                if s∉Lk
                        remove c from Ck+1
```