Data Preprocessing : Accuracy : errors in recorded values, outlier values (unexpected) ex : enormous price for a cheap item once (introduced 0s by mistake) **Completeness** : lacking attribute values, only aggregate data, ... ex : sometimes type of payment (card/cash) not recorded **Consistency** : discrepancies in values used ex : some department codes with leading 0, some without **Timeliness** : data missing or incomplete for some time, but eventually complete ex : data recorded by person too late for monthly analysis, but available afterwards **Believability** : Not trusted by users ex : due to a previous bug (now corrected), people distrust system and use manual alternatives **Interpretability** : ambiguous data ex : adjustment orders (not real but to correct mistake/complaint) can be confused as real orders.
Tasks : **Data cleaning** (fill missing value, smoothing, outliers, inconsistencies), **Data integration** (multiple database,...), **Data reduction** (Dimensionality reduction,Numerosity reduction, Data compression), **Data transformation** : Smoothing, Attribute/feature construction, Aggregation. **Data transformation** : convertir les données des sources vers un modèle unifié.
Données incomplètes : ignorer tuple, remplir à la main, automatiquement (mean,std,...). **Clustering** : detect and remove outliers. Coefficient de corrélations : $p'_k = (p_k - \bar{p}/\sigma_p), q'_k = (q_k - \bar{q}/\sigma_q), corr(p,q) = p' \cdot q'$. **Mode of data** : valeur qui apparait le plus, **midrange of data** : $min + max/2$, $\sigma_p = \sqrt{v(p)}, v(p) = \sum(p_i - \bar{p})^2$.
Data reduction : **Dimensionality reduction** (attribut inutiles), techniques : Wavelet transforms, Principal Component Analysis, Supervised and nonlinear techniques. **Numerosity reduction** : régression, histogramme, data cube. **Data compression** : Audio/video compression, string compression.
Data Transformation : Normalization : ***min-max*** : $v' = \frac{v - min_A}{max_A - min_A} \cdot (new_{max_A} - new_{min_A}) + new_{min_A}$. ***z-score*** : $v' = \frac{v - \bar{A}}{\sigma_A}$. ***decimal-scaling*** : $v' = \frac{v}{10^j}$ j is the smallest integer such that $|max(v)| < 1$. **Binning method for data smoothing** : D'abord, séparer les données en n groupe de même taille. Pour chaque groupe, remplacer par la moyenne (***smoothing by mean***) / le min ou max le plus proche (***smoothing by boundaries***) / la médiane ***smoothing by median***.
Market-Basket analysis : permet de trouver les groupes d'articles qui ont tendance à apparaitre ensemble. On dispose des transactions. Soit I un ensemble d'article, D un ensemble de transactions, chaque transaction T_j est un ensemble d'article. Règle de dissociation (comme association mais peut avoir *non*). $Support(X => Y) = P(XetY)$. $Confiance(X => Y) = P(Y|X) = P(XetY)/P(X)$. $Lift(X => Y) = Confiance/Support$. **Reduire D : *Agrégation*** : agréger produits très similaires en produit moins spécifique. ***Taxonomie*** : Généralisation par catégorisation (p.ex : "Vêtement"). Á partir de seuil définis par nous min_{sup}, min_{conf}, trouver les règles intéressantes (juste regarde si ok ou non).
Algorithme d'extraction de règles : Phase 1 : $L1$ = items individuels. Calculer le support de chaque item, supprimer de L_1 quand $<$ que min_{sup}, C_2 : toutes les combinaisons 2 à 2 d'éléments. L_2 : enlever quand support $< min_{sup}$. A partir de $K = 3$ combiner en C_k seulement les éléments avec le même (K-1) départ, enlever de C les éléments dont au moins un des sous-ensembles $K-1$ n'est pas présent dans L_{K-1}, calculer L_K comme étant C_K sans les éléments avec support dans D $< min_{sup}$. Arrêter quand L_K est vide. Prendre pour phase 2 tous les sets non-traçés. **Phase 2 :** Pour chaque set qui n'est pas de longueur 1, calculer toutes les règles $conf(X => Y)/conf(X)$. Garder règles intéressantes : $conf \geq min_conf$.
Partitionnement : Diviser D en p partitions, appliquer $apriori()$ sur chacune et prendre C = union des L_p. Faire L sur D complet. Avantages : s'adapte à la mémoire dispo, facilement //. Défaut : on peut avoir bcq de candidat au deuxième passage.
Decision Tree : Accuracy rate : $\frac{n_{pred}}{n_{true}}$, évaluation du modèle : accuracy, speed, robustness, scalability. **Basic algorithm** : Tree is constructed in a top-down recursive divide-and-conquer manner, At start, all the training examples are at the root, Attributes are categorical, Examples are partitioned recursively based on selected attributes. **Condition d'arrêt** : no more sample / no more attribute. All samples for a given node belong to the same class. **Sélection d'attribut :** plus gros gain de performance : $Gain(A) = Info(D) - Info_A(D), Info(D) = -\sum_{i=1}^{m} p_i \cdot \log_2(p_i), Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times I(D_j)$. $SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|})$. $GainRatio = Gain(A) - SplitInfo(A)$. $gini(D) = 1 - \sum_{j=1}^{n} p_j^2$, p_j est la fréquence relative $\frac{n_j}{n_{tot}}$. $gini_A(D) = \frac{D_1}{D} gini(D_1) + \frac{D_2}{D} gini(D_2)$. $\Delta gini(A) = gini(D) - gini_A(D)$. **Information gain** : biaisé au travers des attributs multi-valués. **Gain ratio** : tendance à choisir des séparations mal balancée quand des partitions sont plus petite que d'autre. **Gini index** : biaisé au travers des attributs multi-valués, difficultés avec beaucoup de classe, à tendance à favoriser les tests qui donnent des partitions et des "purity" équilibrés. **Overfitting :** preprunming (ne split pas un noeud si le gain est en dessous des certains threshold), postprumning (supprime les branches qui minimisent une certaine erreur).
Clustering : hiérarchique (analyse détaillée) ou non (gd ensembles de données). **HAC** : démarre avec 1 cluster par élément, groupe les clusters les plus similaire 2 par 2 jusqu'à atteindre k clusters voulus. **Saut minimum** : valeur max. de $Sim(v_{C1}, v_{C2})$ pour tous les couples. **Diamètre** : valeur min. de de $Sim(v_{C1}, v_{C2})$ pour tous les couples. **Moyenne** : moyenne de Sim pour toutes les combinaisons (aussi intra-cluster). **K-Means** : On démarre avec k centroid, lors de chaque itérations on fait : attribuer chaque point au centroid le plus proche, recalculer le centre des centroids (vecteurs moyen des distances aux centroid). Faire jusqu'à convergence (pas de centroid changé) ou bien un nombre d'itérations fixé. Distances : Euclidienne, Manhattan, similarité par le cosinus $1 - \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$. Complexité : $O(iknm)$ iteration, taille vecteur, nombre cluster, nombre sample. **Buckshot** : faire HAC sur $\sqrt{n}$ samples et utiliser comme base pour K-mean. En O(n) et évite mauvais centroids. **Qualité du clustering** : $Purity(\omega_i) = \frac{1}{n} max_j(n_{ij}), j \in C$.
Data warehouse Type de schéma : **Star** une fact table relié à des dimensions table, **SnowFlake** idem mais avec des dimensions table qui peuvent être normalisé. **Fact Constellation** comme star mais avec plusieurs fact table. Utilisation des schémas : datawarehouse plutôt constellation et datamart plutôt star ou snowflake. Type de warehouse : **enterprise warehouse** sujet qui concerne toute l'entreprise (données détaillées et summarisé), **datamart** un groupe spécifique d'utilisateur (département) les données sont souvent summarisé (par mois p.example). **Virtual warehouse** un ensemble de vue sur une base de données opérationnelle. Une **fact table** est composé de clé (vers les autres **tables de dimensions** (information sur différents objets)) et de mesure. **Star net query model** Groupe de ligne qui émanent d'un point central, chaque ligne est une dimension et chaque point de la ligne est un niveau de la dimensions. **Pourquoi DW** competitive advantage, business productivity, customer relationship management, cost reduction. **OLTP vs OLAP** label : users, function, DB design, data, usage, access, unit of work, records accessed, users, metric, DB size. **Architecture** data sources (extract,transform,load,refresh) -> data storage (serve) -> OLAP engine -> Front-end tools. **Opérations sur un cube** : ***Roll-up*** -> summarize data (ville -> pays), ***Drill-down*** -> inverse of roll-up, ***Slice*** -> where en SQL, ***dice*** -> select en SQL, ***Pivot*** -> réoriente cube. $N_{cuboïdes}$ = produit du nombre de niveaux par dimensions. Si 4 dim. ont 5 niv. chaqu. -> 5^4 cuboïdes. **Apex cuboïd** : cuboïde avec 0 dimensions (max summarization). **Base cuboïd** : cuboïde de base avec toutes les dimensions. **Catégorie de mesure de cube** : Algebric, Holistic, Distributive.

Vecteur d'incidence : vecteur binaire qui correspond à si un mot est dans chaque document ou non. OR-query $O(x+y)$. Si on a plusieurs OR avec des AND, il faut faire la somme des *df* et commencer par le plus petit. Three scales of IR systems : Web search Personal, information retrieval Enterprise institutional and domain specific search. **Boolean model** : une matrice d'incidence (binaire) possède un 1 pour chaque intersection terme-document. Pour faire un ET -> bitwise AND, pour un grand nombre de document et de terme, la matrice est immense, on utilise alors un **Inverted index** : pour chaque terme t, on stocke une liste des documents (doc_{id}) où t apparait, on ajoute aussi la term-frequency. 4 phases de text processing : **tokenization** coupe les séquences de caractères en des tokens de mots. **Normalization** transforme texte et query dans une même forme (u.s.a -> usa, enlève accent, umlaut, majuscule). **Stemming** différent mot mais même racine. **Stop words** enlève les mots inutiles (the,a,to,of,...). Pour les query, commencer par faire l'algo sur les postings list les plus petite. Pour des **sentence query**, faire un inverted index de bigram **problème** : peut avoir des faux positifs. **Positional indexes** on stocke la position du terme et dans quel document dans la posting list, la taille de l'index augmente. On peut combiner les deux schémas pour certain requête très spécifique (Michael Jackson,The Who). **Skip pointers** : on place des pointeurs tous les temps d'éléments dans la posting list, avec la valeur de destination, et on se permet de sauter si l'élément ne correspond pas (besoin de trier les listes), on doit également prendre en compte la comparaison avec le skip pointeur (pointeur tous les $\sqrt{length}$). **Jaccard coefficient** : $jaccard(a,b) = \frac{|A \cap B|}{|A \cup B|}$, ne prend pas en compte la tf. **Log frequency weighting** : $w_{t,d} = 1 + \log_{10} tf_{t,d}$, si $tf_{t,d} > 1$, sinon 0. **idf** : $idf_t = \log_{10}(N/df_t)$. **tf-idf** : $w_{t,d} = (1 + \log(tf_{t,d})) \times \log_{10}(N/df_t)$. **Score** : $score(q,d) = \sum_{t \in q \cap d} tf.idf_{t,d}$. **Document et query comme des vecteurs** : utilisé les angles pour mesurer la distance entre deux vecteurs $cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|v|} q_i d_i}{\sqrt{\sum_{i=1}^{|v|} q_i^2}\sqrt{\sum_{i=1}^{|v|} d_i^2}}$. Pour les **vecteurs normalisés**, on utile juste le produit scalaire des vecteurs. **Avantages vecteur** : simple, mathématique, efficace. **Désavantages** : pas de sémantique, pas de syntaxe, ignore les synonymes, moins de contrôle que le modèle booléen.

Lemmatization : am,are,is -> be. car,cars,car's,cars' -> car. **Stemming** : réduit les mots à leur racine, algorithme : Porters (automation -> automat). **Wildcard query** query qui ressemble à mon* : tous les documents qui contiennent tous les mots qui commences par "mon" -> facile avec les B-Tree pour stocker les termes. Pour les querys "*mon", chercher dans le reverse B-Tree. **Query "pro*cent"** : utilisation du **Permuterm Index** exemple : on prend le mot "hello" et on construit l'ensemble "hello$","ello$h","llo$he","lo$hle","o$hell" et "$hello" qui sont des entrés dans la map/tree qui pointe sur le mot de base. Exemple de requête : "m*n", on met un $ à la fin de la requête et on shift à gauche jusqu'à obtenir l'étoile à la fin du mot. Ensuite on lance simplement la recherche sur le B-Tree / hashmap. Si plusieurs *, séparer les données et effectuer les requêtes séparéments.

Bigram (k-gram) indexes : on énumère toutes les séquences de k caractères, exemple : "hello you" -> "$h","he","el",...,"o$","$l",... On maintient un deuxième inverted index qui fait la relation bigram -> dictionnary term, on peut donc faire des requêtes "mon*" comme des requêtes "$m AND mo AND on".

Evaluation : précision : fraction des documents retrouvés qui sont relevant. recall : fraction des documents relevants qui ont été retrouvé. accuracy : $(tp+tn)/(tp+tn+fn+fp)$ (pas bon dans IR). $F\beta - score$: $F_\beta = (\beta^2+1)RP/(R+\beta^2 P)$ (pour f1-score, $\beta = 1$). **Harmonic mean** : $F = \frac{1}{\frac{1}{2}(\frac{1}{R}+\frac{1}{P})} = \frac{2RP}{R+P}$. Évaluation des documents : recall (à chaque fois que doc est relevant, augmenté le recall par $\frac{1}{n_{relevant}}$) et precision (calculer à chaque pas la précision), pour le ranking, calculer la moyenne des précisions pour chaque étapes du relevant document. Précision R : précision à l'élément R et $R - precision$: précision-R où R vaut le nombre de relevant document.

Finding Near Duplicates : part of text are exactly the same. **Shingling** : divide in k-grams, documents are represented by set of shingles. Choice of k : Small : + better detection of small changes, - a shingle will belong to many documents -> can have too many false positives, Big : + less false positives, - Not detecting small changes -> false negatives, k=5 for short docs, k=10 for long docs. Compress shingles with hashing -> doc represented by set of shingle hashes. **Jaccard** : Docs : 0/1 vectors in space of k-shingles, $sim(D_1,D_2) = |C_1 \cap C_2|/|C_1 \cup C_2|$. **Minhashing** : boolean matrix, Rows = elements (shingles) and columns = sets (documents). Columns similarity is the jaccard's one. Exemple : $sim(C_1,C_3) = 1/4$, $distance(C_1,C_3) = 1 - 1/4 = 3/4$. Idea : instead of permutations, use different hash functions Apply functions h(x),g(x),etc to each row r, (i=number of current row) : 1. compute y = h(i) 2. each doc : if row has a "1" for doc 3. set sig h to y for doc same steps for g(x). **Estimate Jaccard** : Sim(C1, C2) = fraction of columns with all rows agreeing.

LSH : find document with jaccard sim at least s, use a function $f(x,y)$ that tells wheter x and y is candidate pair. Each pair inside the signature matrix M is a candidate pair. **Partition M into b Bands** b : number of bands -> r : rows per band, k : number of buckets (make as large as possible) -> range of hash function. Choice of b : **Large b** (small r) -> > opportunity to go in same bucket, not hard to hash to same bucket once -> good if threshold is low, **Small b** (large r) -> < opportunity to go in same bucket -> good for high threshold.

Algo Sim >= s : **1.** définir s (threshold), b (nb bands) and r (nb rows in band, got from nb rows / b) **2.** Poser Sim(C1, C2) = s **3.** Probabilité C1, C2 identiques sur 1 bande PARTICULIERE : Pidentiques = sr **4.** Probabilité C1, C2 non similaire (faux négatif) : PFN = (1-Pidentiques)b **5.** Taux de paires trouvées : 1-PFN

Algo Sim < s : **1.** Etapes 1 à 3 comme algo ci-dessus **2.** Probabilité identiques sur 1 au moins de b bandes : PFP = b * Pidentiques

S-Curve : Regarless of the chosen constant b and r, the function, $1-(1-t^r)^b$, has the form of an S-curve. The threshold that is the value of similarity s at which the rise becoms steepest, is a function of b and r. Approx threshold : $(1/b)^{1/r}$. For example, with $b = 16$ and $r = 4$, then the threshold is approx. $\frac{1}{2}$ since 4th root of 16 is 2.