Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique

Département Informatique

Master 2 — SII — Groupe 2

Module : Data Mining

Rapport du TP :

Partie 2 : Implémentation des techniques de Data Mining

Réalisé par :

BENAMARA SOUFIAN PRZEMYSLAW (201500008989)

DERARDJA MOHAMED ELAMINE (201500008274)

02 / 02 / 2020

1 Introduction et plan de travail

1.1 Data mining

Le monde digital d'aujourd'hui est basé sur l'information. Cette dernière est générée tout le temps (peu importe le moment de la journée), partout (la numérisation touche de plus en plus des dispositifs quotidiens) et en masse (quantité énorme de tout type : texte, image ou vidéo). Ces données, aussi brutes qu'elles puissent paraître, représentent une devise chère : La connaissance. Cependant, l'extraction de cette dernière est une tâche complexe faisant appel aux différentes notions mathématiques (statistiques) et informatiques (bases/entrepôts de données et l'algorithmique) plongeantes toutes dans le domaine de Data Mining qui offre des techniques/outils spécialisés pour fouiller la donnée et d'en récupérer les informations précieuses cachées : Relations de dépendance/causalité entre les différentes composantes [techniquement : attributs] de la donnée ou le groupement des instances en ensembles partageants les mêmes propriétés/caractéristiques.

1.2 Organisation du rapport

Ce rapport présentera les détails relatifs aux quatre algorithmes dans deux différentes techniques du Data Mining (descriptif, implémentation et résultats) : *Apriori* et *Eclat* dans l'extraction des motifs fréquents et règles d'association, ainsi que *K-Medoids* et *CLARANS* dans la classification non supervisée. Tout en illustrant les résultats obtenus directement à partir de l'application conçue pour cela.

2 Extraction des motifs fréquents et règles d'association

2.1 Généralités

L'extraction des motifs fréquents est l'ensemble de méthodes visant à identifier les relations (dans le sens : implications) entre les éléments composites d'un groupe de données (nommé : dataset). Ainsi, les objets apparaissant ensemble dans multiples instances sont susceptibles d'être reliés (l'apparition d'un objet engendre l'apparition de l'autre). La recherche des motifs fréquents joue un rôle principal dans l'identification des corrélations et relations dans le dataset en général, et particulièrement dans la définition des règles d'association.

L'exemple typique d'application de cette technique est l'analyse du panier dans les magasins, cela consiste à étudier les habitudes des consommateurs en trouvant les relations entre les produit achetés (mises dans le panier). La découverte de ces relations peut aider les dirigeants dans le choix de l'emplacement des produits pour maximiser le gain d'un côté, et de l'autre, faciliter la recherche des produits aux acheteurs.

2.2 Discrétisation des attributs

L'application de la technique décrite en haut nécessite un prétraitement du dataset concerné. Il s'agit de discrétiser les attributs numériques continus en utilisant l'*equal-width bining*.

La discrétisation est le processus de transformation d'un ensemble de valeurs continues (grand nombre de valeurs réelles ou entières) en valeurs discrètes (ensemble réduit de valeurs possibles prises par un attribut) dans le but de comprendre mieux la donnée (la réduction permet d'avoir une meilleure vue sur les catégories potentielles de l'attribut), réduire l'impact du bruit (les valeurs des *outliers* seront remplacées par d'autres plus "acceptables") et surtout pour pouvoir interpréter l'attribut (les valeurs continues ont une chance réduite d'être en relation avec d'autres valeurs d'attributs vu l'intervalle énorme des valeurs possibles).

La méthode choisie pour cette procédure est le *"bining"*, une technique de division descendante basée sur un nombre spécifique de *bins*. La variante utilisée est *"equal-width"* avec remplacement par *moyenne* ou *médiane*.

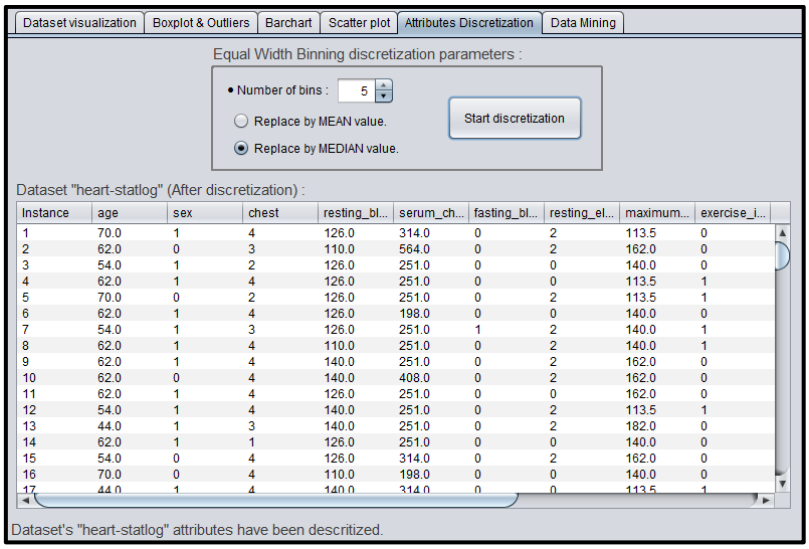


Figure qui montre le processus de discrétisation dans l'application développée

2.3 Algorithme Apriori

2.3.1 Présentation

Apriori est un algorithme classique, proposé par R. Agrawal et R. Sirikant en 1994 pour l'extraction des motifs fréquents et la génération des règles d'association.

Le nom de l'algorithme relève son fonctionnement général, en fait, il utilise les connaissances apriori sur les fréquences (nombres d'apparitions) d'un ensemble d'objets dans un dataset en le parcourant d'une manière horizontale (instance par instance, imitant la *"recherche en largeur d'abord"*) entièrement à chaque fois.

2.3.2 Implémentation

2.3.2.1 Structures de données

La mise en pratique de cette approche a nécessité l'utilisation de multiples structures de données, les détails ci-dessous :

- Itemset : Un ensemble d'objets (valeurs d'attributs) implémentant les opérations typiques sur un ensemble, entre autres : ajout d'un élément (s'il n'existe pas auparavant), suppression, union, intersection et soustraction.

- Candidate : Un candidat trouvé (*Itemset* fréquent potentiel), défini par un *Itemset* et *support count* (nombre d'apparitions dans le dataset).

- CandidateItemets : Liste (dynamique) des candidats courants (une représentation de la matrice C_k), permet la mise à jour du *support count* de ses membres (en parcourant le dataset en intégralité).

- FrequentItemsets : Table de hachage (clé : *Itemset*, valeur : *supportCount*) des itemsets fréquents (une représentation de la matrice L_k). La création (construction) d'une telle table est faite à partir du *CandidateIntemsets* en récupérant uniquement les candidats qui satisfassent le minium support count. La table de hachage construite accélérera la génération des règles d'association.

- Rule : Une règle d'association, décrite par des conditions (Itemset), conséquences (Itemset) et un degré de confidence.

- AssociationRules : Liste des règles d'associations, générées à partir de *FrequentItemsets*.

● Apriori : Le point d'entrée de l'algorithme, son fonctionnement est décrit par le pseudo-algorithme suivant :

Algorithme Apriori ;
Input : D : Dataset, positions : attributs choisis, minSup : support minimum, minConf : confidence minimale ;
Output : R : règles d'association ;
Début
$C_1 \leftarrow$ Liste des candidats de taille 1 avec leur *support count*, en récupérant les valeurs d'attributs (pour chaque instance du D) dans positions ;
$L_1 \leftarrow$ candidats_satisfaisant_minSupport(C_1, minSup) ;
$L \leftarrow L \cup L_1$; $L_{k-1} \leftarrow L_1$;
Tant que (L_{k-1} n'est pas vide) **faire**
  $C_k \leftarrow$ *aprioriGen*(L_{k-1}) ;
  Pour chaque (transaction $t \in D$) **faire**
    $C_t \leftarrow$ sous-ensemble(C_k, t) ;
    Pour chaque (candidat $c \in C_t$) **faire**
      c.count++ ;
    Fait ;
    $L_{k-1} \leftarrow \{c \in C_k \mid c.count \geq minSup\}$;
  Fait ;
  $L \leftarrow L \cup L_{k-1}$;
Fait ;
R ← générer les règles d'associations à partir du L (en tenant compte de la *minConf*) ;
retourner(R) ;
Fin ;

2.3.2.2 Complexité

La tâche la plus gourmande (nécessitant le plus de temps de calcul) est la jointure effectuée sur L_{k-1} avec lui-même pour générer les nouveaux candidats potentiels (C_k). Ainsi, la complexité temporelle de ce traitement est dominante (la plus grande), égale à : $\mathbf{O((n * m)^2 * k)}$

Car pour la détermination de l'ensemble courant des candidats : la jointure se fait en $O(|L|^2)$. Mais comme $|L|$ est égale au maximum à $(n * m)$, la complexité de la jointure est $O((n * m)^2)$.

- n → Nombre d'items distincts du dataset.
- m → Nombre de transactions.
- k → Nombre d'itérations (qui correspond aussi au maximum k-itemset).

2.3.3 Expérimentations et résultats

Pour l'évaluation des performances de l'algorithme Apriori, le dataset *"heart-statlog"* a été utilisé, que 4 attributs sélectionnés : *"age"*, *"resting blood pressure"*, *"serum cholestoral"* et *"maxium heart rate achieved"*.

Les deux paramètres (*support minimal* et *confidence minimale*) ont été variés pour voir l'impact d'augmentation (ou diminution) sur le *temps d'exécution global* et le *nombre des règles générées*. Le résumé des tests ci-dessous :

Support minimal (%)	Confidence minimale (%)	Nombre de règles générées	Temps d'exécution (ms)
20	10	**114**	**58**
40	10	**20**	**8**
60	10	**6**	**5**
80	10	**0**	**1**
95	10	**0**	**1**

Tableau montrant l'impact de variation du support minimal sur le nombre des règles générées et le temps d'exécution (pour l'algorithme Apriori)

Les résultats affichés en haut montrent, de façon claire, la relation proportionnelle entre le nombre de règles générées et le temps d'exécution. Ces deux étants en relation inverse avec la valeur du support minimal.

Support minimal (%)	Confidence minimale (%)	Nombre de règles générées	Temps d'exécution (ms)
10	20	**221**	**32**
10	40	**103**	**23**
10	60	**23**	**21**
10	80	**1**	**21**
10	95	**0**	**18**

Tableau montrant l'impact de variation de la confidence minimale sur le nombre des règles générées et le temps d'exécution (pour l'algorithme Apriori)

L'analyse des résultats obtenus conclue l'indépendance de la variation de la *confidence minimale* et le *temps d'exécution*. Par contre, le nombre de règles générées est affecté par l'augmentation de la *confidence minimale* (tout comme pour le *support minimal*).

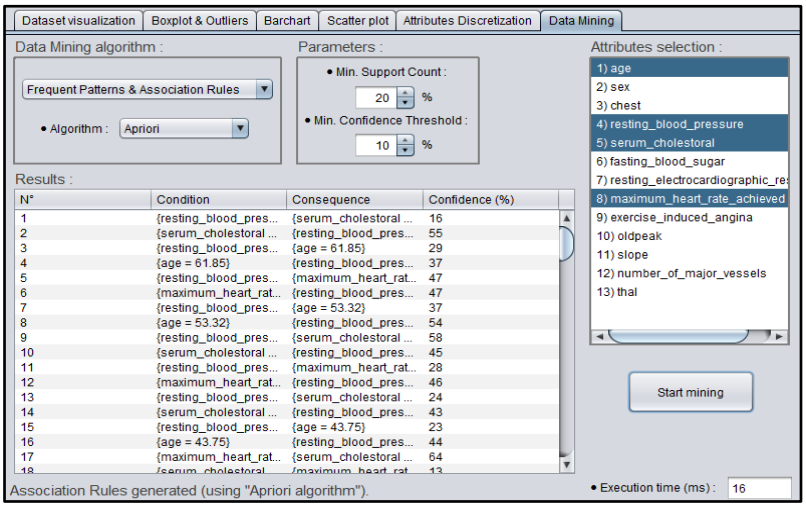


Figure qui montre l'utilisation de l'algorithme Apriori à travers l'application développée

2.4 Algorithme Eclat

2.4.1 Présentation

Eclat est une alternative à Apriori, fameux lui-aussi, proposé par Zaki en 2001. Il parcourt le dataset verticalement (imitant la *"recherche en profondeur"*) en transformant ce dernier en une structure adéquate à cela. Une telle approche rend Eclat plus efficace (rapide, dans la plupart des cas) qu'Apriori.

2.4.2 Implémentation

2.4.2.1 Structures de données

Pour mettre en œuvre l'algorithme Eclat, quelques structures de données ont été créées, avec la conservation de celles utilisées pour Apriori (puisqu'il s'agit des mêmes traitements). Les structures ajoutées sont :

● VerticalInstanceSet : Le dataset transformé en format vertical, les objets (valeurs d'attributs) sont mis dans une table de hachage (clé : itemset [l'ensemble des objets], valeur : Candidate [liste des numéros d'instances avec leur support]).

● Eclat : Le point d'entrée de l'algorithme, son fonctionnement est similaire à Apriori, sauf que cette fois-ci le format vertical est exploité au lieu de l'horizontal.

2.4.2.2 Complexité

Tout comme pour Apriori, la tâche la plus gourmande est la jointure effectuée sur L_{k-1} avec lui-même pour générer les nouveaux candidats potentiels (C_k). Ainsi, la complexité temporelle est égale à : $\mathbf{O((n * m)^2 * k)}$

Car pour la détermination de l'ensemble courant des candidats : la jointure se fait en $O(|L|^2)$. Mais comme $|L|$ est égale au maximum à $(n * m)$, la complexité de la jointure est $O((n * m)^2)$.

- n → Nombre d'items distincts du dataset.
- m → Nombre de transactions.
- k → Nombre d'itérations (qui correspond aussi au maximum k-itemset).

2.4.3 Expérimentations et résultats

De même que pour Apriori, le dataset "heart-statlog" a été utilisé avec les 4 attributs. Les résultats des tests ci-dessous :

Support minimal (%)	Confidence minimale (%)	Nombre de règles générées	Temps d'exécution (ms)
20	10	**114**	**25**
40	10	**20**	**6**
60	10	**6**	**4**
80	10	**0**	**2**
95	10	**0**	**1**

Tableau montrant l'impact de variation du support minimal sur le nombre des règles générées et le temps d'exécution (pour l'algorithme Eclat)

Les résultats montrent la relation proportionnelle entre le nombre de règles générées et le temps d'exécution. Ces deux étants en relation inverse avec la valeur du support minimal.

Support minimal (%)	Confidence minimale (%)	Nombre de règles générées	Temps d'exécution (ms)
10	20	**221**	**38**
10	40	**103**	**22**
10	60	**23**	**25**
10	80	**1**	**21**
10	95	**0**	**23**

Tableau montrant l'impact de variation de la confidence minimale sur le nombre des règles générées et le temps d'exécution (pour l'algorithme Eclat)

L'analyse des résultats obtenus conclue l'indépendance de la variation de la *confidence minimale* et le *temps d'exécution*. Par contre, le nombre de règles générées est affecté par l'augmentation de la *confidence minimale* (tout comme pour le *support minimal*).

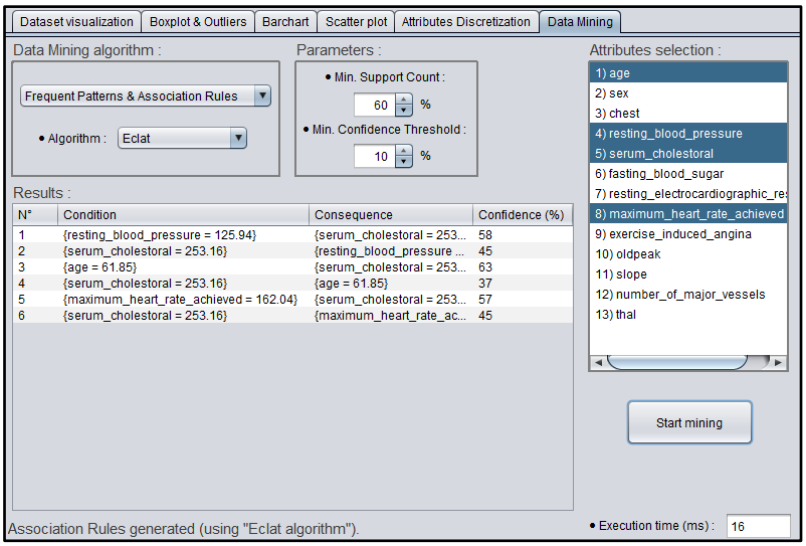
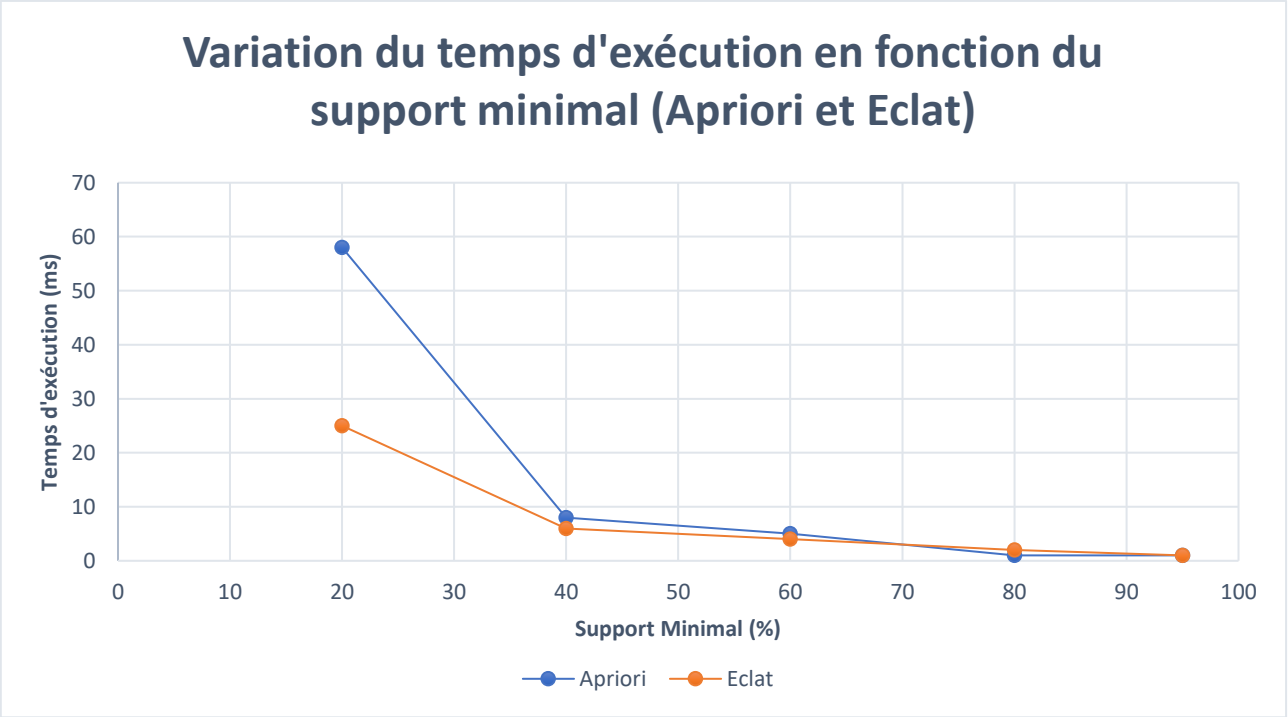


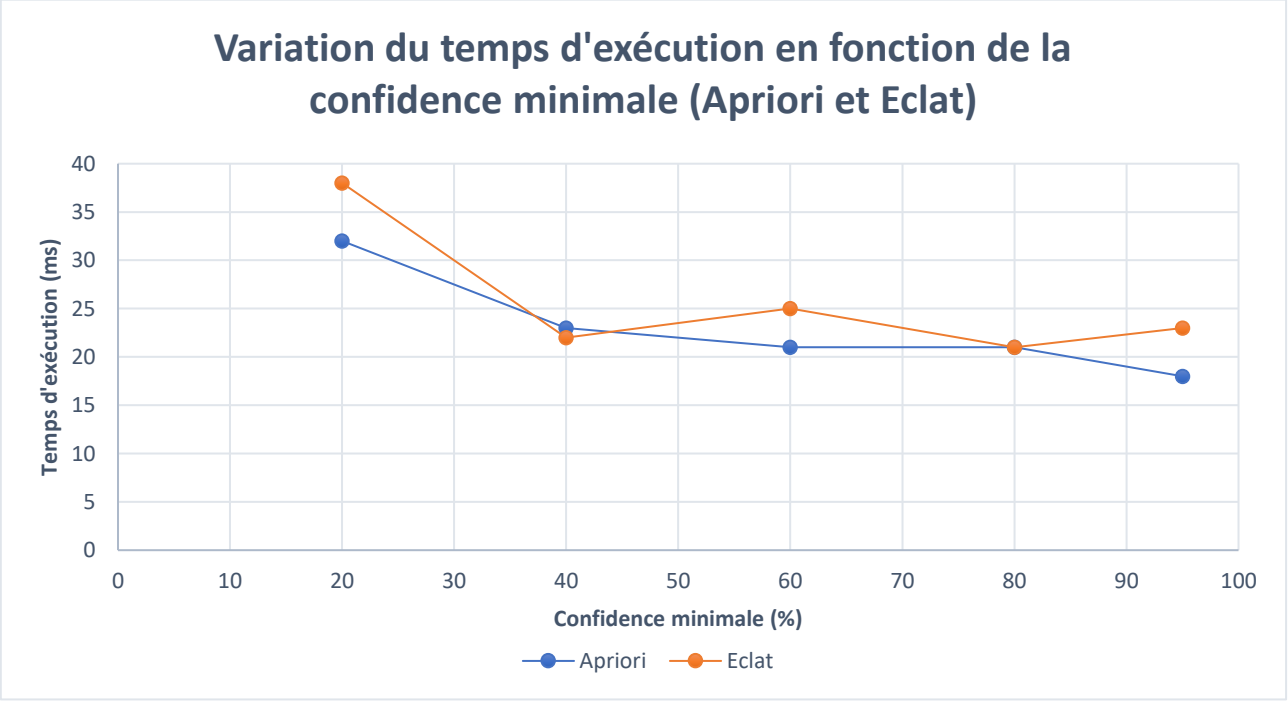
Figure qui montre l'utilisation de l'algorithme Eclat à travers l'application développée

2.5 Comparaison des algorithmes

Puisque le principe des deux algorithmes est le même, ils retournent les mêmes règles d'association. Ainsi, la comparaison d'Apriori et Eclat portera sur le *temps d'exécution* mesuré en variant le *support minimal* et la *confidence minimale*.



Le graphe ci-dessus montre la variation du temps d'exécution en fonction du support minimal pour Apriori et Eclat. En effet, un écart significatif (33 ms) en faveur de l'algorithme Eclat est remarqué dans la première valeur du test (support minimal = 20 %). Cet écart diminue rapidement pour devenir presque nul pour les valeurs de support minimal plus élevées (40, 60, 80 et 95).



Le graphe en haut montre la variation du temps d'exécution en fonction de la confidence minimale pour les deux algorithmes étudiés. En effet, une stabilité du temps d'exécution est remarquée pour les différentes valeurs de la confidence minimale (20, 40, 60, 80 et 95), et cela pour Apriori et Eclat.

3 Classification non supervisée

3.1 Généralités

Différemment de la classification supervisée, où les classes d'objets sont connues, la classification non supervisée (ou le *Clustering*) n'a pas d'information préliminaire sur les classes pouvant être assignées aux objets ou bien celles où les objets seront assignés. Ainsi, la classification non supervisée permet de classer (ou de grouper) les objets tout en se basant sur la similarité entre les objets.

Le Clustering est un processus de groupement des objets en groupes ou "clusters" de tel sorte que les objets d'un cluster donné soient plus similaires l'un de l'autre que d'autres objets d'autres clusters.

Plusieurs techniques ont été proposées pour cela, elles peuvent être divisées en 4 catégories : Les méthodes basées sur le partitionnement, les méthodes basées sur la hiérarchie, les méthodes basées sur la densité et les méthodes basées sur le *Grid*.

Comme, il semble clair, la similarité joue un rôle très important, elle permet de mesurer la distance séparant un objet d'un autre en basant sur leurs attributs.

Dans ce qui suit, nous allons présenter la manière utilisée pour mesurer la similarité entre deux objets (des instances), ensuite nous présentons la mesure utilisée pour évaluer l'erreur absolue d'une configuration donnée (clustering). En fin, nous nous occuperons de K-Medoids et CLARANS.

3.2 Distance entre les instances

Comme il y a plusieurs types d'attributs définissant une instance donnée, nous allons utiliser la mesure suivante :

$$d(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}}$$

Où l'indicateur $\delta_{ij}^{(f)} = 0$ si soit x_{if}ou x_{jf}sont manquantes ou bien $x_{if} = x_{jf} = 0$ et l'attribut "f" est un attribut asymétrique binaire. A part ça, $\delta_{ij}^{(f)} = 1$.

La dissimilarité entre les différents types d'attributs est comme suit :

- Si f est numérique : $d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_h x_{hf} - min_h x_{hf}}$
- Si f est nominal ou binaire : $d_{ij}^{(f)} = 0$ si $x_{if} = x_{jf}$, sinon $d_{ij}^{(f)} = 1$.
- Si f est ordinal : $d_{ij}^{(f)} = |\frac{x_{if}-1}{M_f-1} - \frac{x_{jf}-1}{M_f-1}|$ où $M_f$représente le nombre de valeurs que peut prendre l'attribut f.

3.3 L'erreur absolue

Une manière pour distinguer deux clusterings consiste à utiliser la somme des erreurs absolues qui est la somme de toutes les dissimilarités entre un objet p et l'objet le plus proche parmi les objets représentatifs (médoïdes), ou en d'autres termes, c'est la somme de toutes les dissimilarités (erreurs) entre un objet p et l'objet représentatif O_i de son cluster. Tant que cette somme est petite tant que le clustering est meilleur.

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist\,(p, o_i)$$

3.4 Algorithme K-Medoids

3.4.1 Présentation

```
Algorithme K-Medoids ;
  Entrée : D : un dataset contenant N objets (instances), K : le nombre de clusters ;
  Sortie : K clusters ;
Début ;
    Choisir aléatoirement k objets (instances) de D comme current_medoids ;
    Calculer l'erreur absolue AE_CM de current_medoids ;
    Répéter
        changement ← Faux ;
        Pour chaque medoid Oi dans current_medoids faire
            Pour chaque objets Oh (non déjà sélectionné) dans les (N-K) objets faire
                new_medoids ← current_medoids + remplacement de Oi par Oh ;
                Calculer l'erreur absolue AE_NM de new_medoids ;
                Si AE_NM < AE_CM alors
                    Oi ← Oh ;
                    AE_CM ← AE_NM ;
                    changement ← Vrai ;
                Fsi ;
            Fait ;
        Fait ;
    Jusqu'à changement = Vrai ;
    Remplir les K clusters où les medoids sont best_medoids ;
    retouner les K clusters ;
Fin ;
```

3.4.2 Implémentation

3.4.2.1 Structures de données

Pour la représentation des variables suivants : current_medoids, new_medoids et best_medoids, nous avons utilisé les listes chainées.

3.4.2.2 Complexité

La complexité à chaque itération est en $O(K * (N - K)^2)$.

Soit M le nombre d'itérations faites par l'algorithme, la complexité de l'algorithme est donc $O(M * K * (N - K)^2)$.

3.5 Algorithme CLARANS

3.5.1 Présentation

Algorithme CLARANS ;
Entrée : D : un dataset contenant N objets (instances).
K : le nombre de clusters.
numLocal : le nombre de régions à explorer.
maxNeighbor : le nombre de fois qu'il n'y a aucun changement lorsqu'on essaye de remplacer les représentants courants (medoids) par un voisin.
Sortie : K clusters

```
Début ;
    Initialiser min_cost à une valeur très grande, et best_medoids à nul ;
    Pour chaque région [1 .. numLocal] faire
        Choisir aléatoirement k objets (instances) de D comme current_medoids ;
        Calculer l'erreur absolue AE_CM de current_medoids ;
        J ← 1 ;
        Tant que Vrai faire
            Considérer un voisin aléatoire (random_neighbor) de current_medoids ;
            Calculer l'erreur absolue AE_RN de random_neighbor ;
            Si AE_RN < AE_CM alors
                J ← 1 ;
                current_medoids ← random_neighbor ;
                AE_CM ← AE_RN ;
            Sinon
                J ← J+1 ;
                Si J > maxNeighbor alors
                    Sortir de la boucle Tant que ;
            Fsi ;
        Fait ;
    Fait ;
        Si AE_CM < min_cost alors
            best_medoids ← current_medoids ;
            min_cost ← AE_CM ;
        Fsi ;
    Remplir les K clusters où les medoids sont best_medoids ;
    retouner les K clusters ;
Fin ;
```

3.5.2 Implémentation

3.5.2.1 Structures de données

Pour la représentation des variables suivants : current_medoids, random_neighbor et best_medoids, nous avons utilisé les listes chainées.

3.5.2.2 Complexité

Soit M = Max(m_i) où mi est le nombre de changement fait lorsqu'on a exploré la région N° i.

La complexité de CLARANS est en $O(numlocal * M * maxNeighbor * (N - K))$.

3.6 Comparaison des algorithmes

Pour comparer les deux algorithmes, nous allons dans un premier lieu présenter les résultats obtenus en appliquant les deux algorithmes à notre dataset, ensuite nous allons donner un tableau comparatif des algos de manière générale.

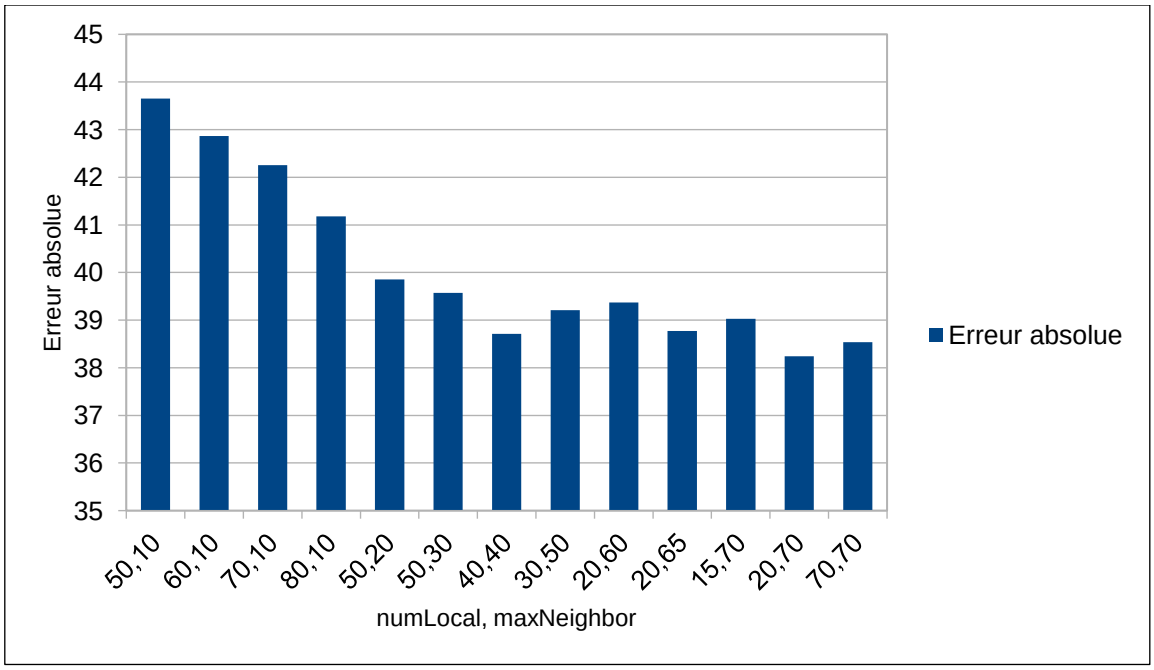
La comparaison entre les algorithmes se fait en voyant l'erreur absolue E et le temps nécessaire (le temps d'exécution).

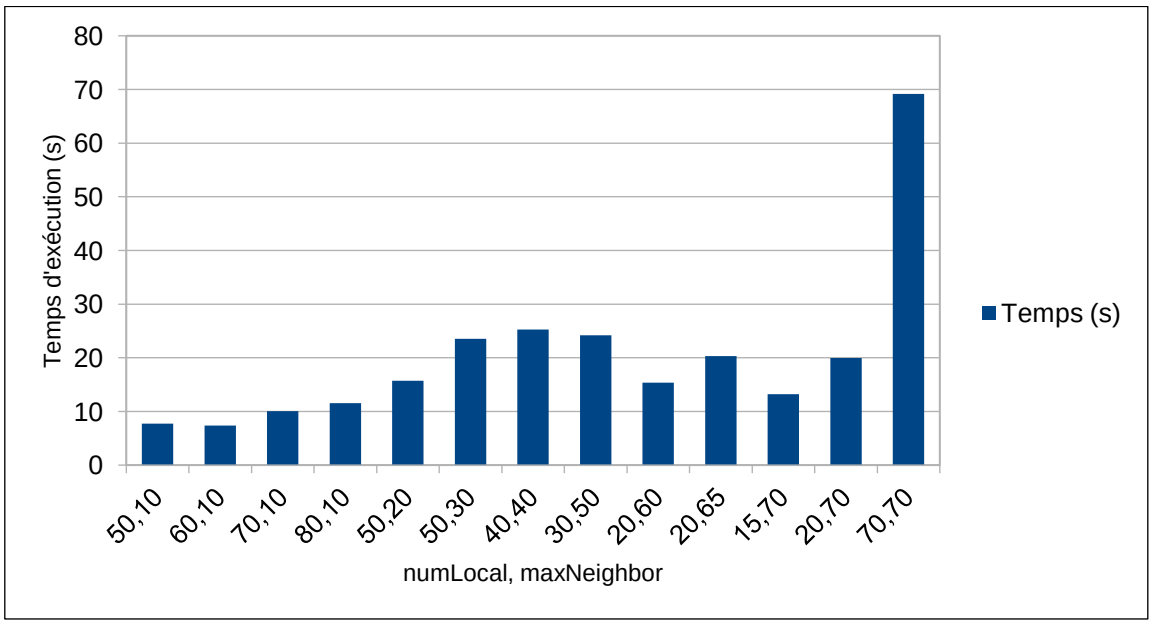
Pour K = 10 :

K-Medoids :

- Erreur Absolue = **37.4079**
- Temps d'exécution moyen : **16.7775 s**

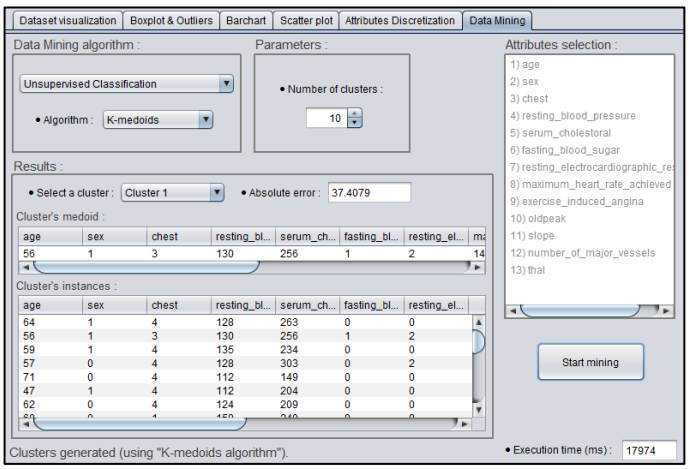
CLARANS :



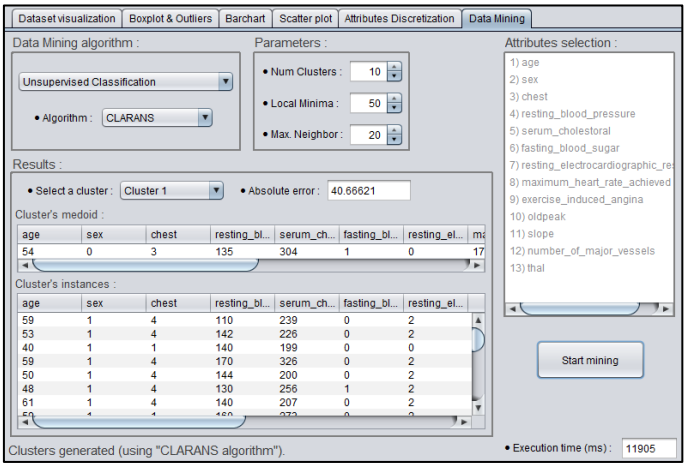


Il est clair que le résultat obtenu par K-Medoids est bien meilleur que celui obtenu par CLARANS dans un intervalle de temps inférieur à 20s (37.4 vs. 38.24). En effet, K-Medoids fonctionne bien pour des datasets de petites, voire de moyennes tailles et donne des résultats raisonnables tout en voyant son temps d'exécution. Le problème majeur avec cet algorithme est qu'il n'y a pas d'amélioration du résultat, contrairement à CLARANS où le résultat peut être amélioré en modifiant les paramètres numlocal et maxNeighbor. L'amélioration du résultat avec CLARANS se fait au détriment du temps, c'est pour cette raison, il ne fonctionne pas bien avec les petites datasets, étant donné qu'il existe K-Medoids qui donne un résultat acceptable dans un temps raisonnable. Néanmoins, lorsque la taille du dataset augmente, là où l'effet de CLARANS devient visible, car avec K-Medoids le temps d'exécution augmente sensiblement au détriment du résultat qui ne soit pas meilleur.

Paramètre	**K-Medoids**	**CLARANS**
Complexité	$O(M * K * (N - K)^2)$	$O(numlocal * M * maxNeighbor * (N - K))$
Efficacité	Efficace pour les datasets de petites / moyennes tailles	Comparativement, meilleur pour des datasets de grandes tailles
Implémentation	Facile	Facile
Optimisé pour	Des datatsets de petites à moyennes tailles	Des datasets de grandes tailles.



K-Medoids dans l'application développée



CLARANS dans l'application développée

4 Conclusion

Cette partie du projet nous a permis de manipuler en pratique les différentes méthodes et approches du data-mining, et plus précisément : L'extraction des motifs fréquents et règles d'association & Le clustering. Ainsi, nous avons établi une étude comparative de ces derniers pour en tirer les points forts/faibles de chacun.

Bibliographie

La référence principale utilisée pour la présentation des techniques implémentées dans ce rapport et leurs pseudo-algorithmes est le livre : *Data mining : concepts and techniques / Jiawei Han, Micheline Kamber, Jian Pei. – 3rd ed.*