Université des Sciences et de la Technologie Houari Boumediene

Data Mining

Rapport du mini projet Implémentation de Techniques de Data-Mining

Rédaction:
MOULAI Hassina Safaa
Matricule : 201400007564
HOUACINE Naila Aziza
Matricule : 201400007594
M2 SII Groupe:3

Professeur
Mme. BABA ALI

January 5, 2019

Contents

List of Figures

Introduction [1,2]

La quantité de données collectées quotidiennement ne cessent de croitre, celle-ci étant considérée comme une mine de connaissance qui n'attend qu'a être exploité. il en va de soit qu'analyser ces données est un besoin important. l'exploration de données (ou Data-Mining) peut répondre à ce besoin tel que:

L'exploration de données permet de trouver des informations précieuses cachées dans de grands volumes de données.

L'exploration de données est l'analyse des données et l'utilisation de techniques informatiques pour trouver des modèles et des régularités dans des ensembles de données (DataSet).

Le processus de Data-Mining est responsable de la recherche des modèles en identifiant les règles et fonctionnalités sous-jacentes des données.

Il est possible d'extraire des schémas jusqu'alors inconnus ou si évidents que personne ne les avait remarqués auparavant.

Le Data-Mining repose sur quelques technologies indispensables dont: l'algorithmique (et structure de données), les Statistiques, l'apprentissage automatique, Systèmes de bases de données et entrepôts de données ainsi que la recherche d'information...
En utilisant ces technologies le processus de Data-Mining passe par plusieurs étapes, celles-ci peuvent ce résumer comme suit:

1. **Pré-traitement des données**: c'est à dire : Traitement de valeurs manquantes, Nettoyage des données, Normalisation de données, Réduction de données, Discrétisation, ...

2. **Création de modèle de données**: application des outils intelligents de Data-Mining pour extraire des connaissances à partir de données, Pour cela le nombre d'outils et de techniques existants est suffisamment conséquent pour couvrir tous les types de problèmes, citons: l'extraction d'items fréquents et associations minières, classification supervisée : Arbres de décision, K-nn, Classification bayésienne, ..., classification non supervisée et clustering: K-means, K-medoid, Agnes, Dbscan, ...

3. **Tester le modèle**: les performances du modèle (par exemple, le rappel, la précision, l'exhaustivité, ...) sont testées sur des données indépendantes (non utilisées pour créer le modèle).

4. **Interprétation et évaluation**: En commençant par visualiser les résultats obtenus, Nous serons en mesure d'en extraire des observations et conclusions après les expérimentations, aussi nous serons en mesure d'apporter des améliorations aux techniques testées.

Chapter 1

Algorithme d'extraction d'items fréquents et associations minières

1.1 Introduction

Les algorithmes d'extraction d'items fréquents et règles d'associations sont très largement exploités car en plus de leur efficacité, nombreux sont les domaines d'application, nous citons à titre d'exemple:

1. Analyse des logs web sur un serveur web afin de découvrir le comportement utilisateur, et ainsi adapter et personnaliser le site à leurs attentes. [3]
2. Le panier de la ménagère qui décrit l'ensemble des achats au supermarché, tel que nous pouvons en extraire les produits qui se vendent ensemble, et ainsi décider de leur disposition dans les étagères selon le but du super marché. [3]

   Ou alors de choisi les réductions et promotions à effectuer sur des ensembles de produits.
3. ... etc.

Pour cela diverse techniques et algorithmes existent, dont les plus populaires et plus utilisés : **Apriori**, **FP-Growth** et **Eclat**
Dans ce qui suit nous allons nous interesser aux deux algorithmes **Apriori** et **FP-Growth**.

1.2 Apriori

1.2.1 Principe de fonctionnement

L'algorithme **Apriori** à été conçu en 1994, par *Rakesh Agrawal* et *Ramakrishnan Sikrant*, dans le domaine de l'apprentissage des règles d'association.[2]

Apriori emploie une approche itérative tel que chaque k-itemsets est utilisé pour explorer les (k+1)-itemsets.

Complexité:
La complexité Temporelle de **Apriori** est cubique, en **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 égal au maximum à (n * m), la complexité de la jointure est O($(n*m)^2$).

La complexité Spatial de **Apriori** est en **O(n^2)**.

Avec:
n : nombre d'items distincts du DataSet transactionnel.
m : nombre de transactions.
k : nombre d'itération (qui correspond aussi au max k-itemset).

1.2.2 Structures de données

Afin de bien résoudre notre problème qui est d'appliquer l'algorithme apriori sur un dataset transactionnel et d'extraire les items fréquents il a fallu modéliser et structurer nos données .

Transaction

une transaction sera modéliser comme suit:

- **idtransaction:** qui est le numéro de la transaction

SetofTransaction

- **set_T**(ensemble de transaction) qui est HashMap<Transaction, ArrayList<item>>

une HashMap est un dictionnaire qui prend comme clé des transactions et comme valeurs un ensemble d'items (itemset)

Item

- **Itemname** : est le nom de l'item exemple : i1,i2...

Itemset

- **items** : qui est un hashset d'item hashset est un ensemble d'éléments (un élément n'apparait qu'une seule fois sans répétition)
- **frequence** : qui est la fréquence (type integer) de l'itemset

SetItemset

- **set_itemset :** qui est un ensemble d

1.2.3 Pseudo-code

Algorithm 1 APRIORI

Input: D,base de données de transations ,support minimum
Output: L, itemset fréquent dans D

```
L1= find frequent 1-itemsets(D);
for k ← 2; L_{k-1}! = ∅; k++ do
    C_k ← apriori_gen(L_{k-1})
    for transaction t ∈ D do
        scan D for counts
        C_t=subset(C_k,t); génerer les subsets de t qui sont candidats
        for candidate c ∈ C_t do
            c.count++;
            L_k= c ∈ C_k , c.count>support minimum
return L = ⋃_k L_k
```

1.2.4 Explication

les differentes étapes :

1. Exploration de la base de données pour avoir le support de chaque 1-itemset (ensemble d'un seul item).
2. Comparer le support(fréquence) avec le **min_supp** .
3. Supprimer les 1-itemsets ayant un support inférieur au **min_supp** génerer alors $L1$.
4. Faire une jointure de L_{k-1} avec L_{k-1} pour générer les ensembles de candidate k-itemsets.
5. Verifier la propriété APRIORI pour élaguer les k-itemsets qui ne sont pas fréquents.
6. Exploration de la base de données pour avoir le support de chaque candidate k-itemset vérifiant la propriété apriori.
7. Comparer le support de chaque candidate k-itemset avec **min_supp**
8. Garder que l'ensemble des k-itemsets vérifiant la condition de **min_supp** et on aura ainsi L_k
9. si L_k est vide alors pour chaque frequent itemset 1 générer les subsets non vide de 1, et pour chaque s subsets non vide de 1, écrire la règle "s implique(1-s)" si la confidence C de la règle "s implique 1-s" satisfait le **support min de confiance**
10. sinon aller à 4

1.3 Fp-growth

1.3.1 Principe de fonctionnement

L'algorithme **Fp-growth** dont **Han** (2000) est le créateur, est une méthode efficace et évolutive pour l'exploitation minière.
Cet Algorithme utilise une structure d'arborescence de préfixes étendue pour le stockage d'informations compressées et cruciales sur les modèles fréquents nommés arborescence de modèles fréquents **FP-Tree**.[2]

FP-Growth construit l'arbre FP-Tree à partir des transactions données dans le dataset à traiter, puis filtre les ItemSets vérifiant le support minimal, ensuite de manière incrémentale en parcourant le FP-Tree il construit pour chaque Item ce qu'on appelle *Conditional Pattern Base* c'est à dire les motifs conditionnelles de base (les chemins vers l'item) qui servirons à leurs tour dans la génération des *Conditional FP-tree* ceux-ci préfixés par leur Item correspondant, formeront les **motifs fréquents** recherchés.[2]

Complexité:
La complexité Temporelle de **FP-Growth** est Polynomial, en **O(**n^2**)**.
La complexité Spatiale de **FP-Growth** est en **O(**n^2**)**.
Avec: n : nombre d'items distincts du DataSet transactionnel.

1.3.2 Structures de données

Item

Nous avons défini la structure de **Item**, celle-ci représente un item d'une transaction, tel qu'elle est composée de 2 attributs:

- **itemName:**chaine de caractère représentant le nom de l'item (*exemple: "i1"*)
- **Support:** entier (int), support de cet item c'est à dire ca fréquence dans le Dataset.

ItemSet

Il s'agit d'un ensemble d'Items ayant un support, ainsi un ItemSet est composé des 2 attributs suivants:

- **Items:** Liste d'Item qui peut correspondre à une branche de la FP-Tree.
- **Support:** il s'agit du support minimal des Item de la liste.

Tree

Il s'agit du FP-Tree, l'arbre permettant la représentation compressée des transactions du dataset, tel que chaque nœud possède les 3 éléments suivants:

- **Item:** l'item du nœud courant.

- **Children:** la liste les fils du noeud courant, qui est une liste de nœuds de la même structure *Tree*.
- **Dad:** Le nœud de même structure *Tree*, qui le parent du nœud courant.

Rule

Cette structure représente une règle d'association, celle-ci ayant la forme suivante:

Condition ⇒ Conclusion : Confiance = XXX%

Nous avons alors représenté chaque règle avec les 3 éléments suivants:

- **Condition:** un ItemSet représentant la partie gauche de la règle d'association.
- **Conclusion:** un ItemSet représentant la partie droite de la règle d'association.
- **Confiance:** (double) Degré de confiance de cette règle (en %).

fp-growth

Il s'agit d'une instance pour l'exécution de FP-Growth, telle que les éléments de base nécessaires pour cela y sont structurés comme suit:

- **DataSet:** le dataset contenant les transactions/items dont nous souhaitons étudier les motifs fréquents.
- **fp-tree:** l'arbre utiliser pour la représentation compressé du dataset.
- **Support_min:** support minimal à prendre en compte lors de la recherche d'ItemSet fréquents.
- **Confidence_min:** la confiance minimal à prendre en compte pour valider une règle.

1.3.3 Pseudo-code

Algorithm 2 FP-Growth

Input: D:base de données de transations ,$Supp_{min}$:support minimum
Output: L: itemset fréquent dans D

```
L= find frequent 1-itemsets(D);
//Create the root of an FP-tree, and label it as "null."
FP_tree = {null}
foreach Trans in D do
    Sort Items of (Trans) according to the order of (L)
    foreach Item in Trans do
        //FP_tree is the first element
        //P is the remaining list
        Insert_tree([FP_tree|P], Item)
L = FP-growth-Const(FP_tree, null)
return L
```

Algorithm 3 PROCEDURE INSERT_TREE

Procedure : Insert_tree([p—P], Item)
if *Item.has_a_child(N)* ***And*** *N.item-name = p.item-name* **then**
| N.freq ++
end
else
| Create_Node(N)
| N.freq = 1
| Link_To_Parent(Item, N)
end
if *P.empty() == False* **then**
| insert_tree(P, N)
end
return P

Algorithm 4 PROCEDURE FP-GROWTH-CONST

procedure FP_growth-Const(Tree, α)
if *Tree contains a single path P* **then**
| **foreach** *combination β of the nodes in the path P* **do**
| | generate pattern $\beta \cup \alpha$ with support_count = minimum support count of nodes in β;
else
| **foreach** *a_i in Q /*the header of Tree*/* **do**
| | generate pattern $\beta = a_i \cup \alpha$ with support_count = a_i.support_count
| | construct β conditional pattern base and then β conditional FP_tree $Tree_\beta$
| | **if** $Tree_\beta = \emptyset$ **then**
| | | FP_growth($Tree_\beta$, β)
| | set(Q) is the set of patterns generated
return α

1.3.4 Explication

Les différentes étapes :

1. Exploration de la base de données pour avoir le support de chaque 1-itemset (ensemble d'un seul item).
2. Comparer le support(fréquence) avec le **min_supp** .
3. Supprimer les 1-itemsets ayant un support inférieur au **min_supp** , et ordonner les autres Items dans l'ordre décroissant de nombre de support, générer alors L.
4. Initialiser notre FB_{Tree} a $null$
5. Ordonner les Item de chaque transaction selon l'ordre obtenu dans L
6. Construction de FB_{Tree} de manière itérative:

   Pour chaque Transaction insérer les items la formant dans le FB_{Tree}

tel que: soit on ajoute un nœud (si celui-ci n'existe pas), soit on incrémente la fréquence du nœud (dans le cas contraire).

7. Pour chaque Item de L toujours dans l'ordre décroissant de fréquence et en utilisant une stratégie de profondeur d'abord:

   On détermine l'arbre de cet Item (tous les chemins allant de la racine $\{\}$ à cet Item)

8. si l'on trouve un seul chemin P constitué d'un ensemble d'Items, alors on génère toutes les combinaisons β de ces Items.

9. On récupère les combinaisons β ayant un support $\geq$ **min_supp** que l'on ajoute à notre liste de motifs fréquents $\alpha \cup \beta$.

10. Sinon (plusieurs chemins) alors, faire l'union des chemins, tel que pour chaque item nous lui donnerons comme support la somme des supports des itemSet auxquels il appartient.

11. Supprimer les motifs de support $<$ **min_supp**

12. Construire la base de motifs conditionnels à partir de β

13. une fois toutes les combinaisons récupérées, on fait un appel récursif de FP-Growth avec la nouvelle liste β (retour à 8)

1.4 Comparaison et remarques

1.4.1 DataSet

Le DataSet utilisé pour nos expérimentation est un DataSet publique et reconnu, disponible sur le site : **UCI**. [4]
Celui-ci est un dataset transactionnel qui contient toutes les transactions intervenues et enregistrées entre le 01/12/2010 et le 09/12/2011 pour un détaillant en ligne basé au Royaume-Uni.[4]

Ainsi il contient 541909 Instances à 8 attributs (NumFacture, StockCode, Description, Quantité, DateFacture, PrixUnitaire, IDClient, Pays).

Que nous avons regrouper sous forme de 1 transaction/ ligne pour avoir le format de transaction souhaitable.

Puis nous n'avons pris que 100 transactions pour les expérimentations afin d'avoir un dataset suffisamment grand et diversifié tout en respectant les limites de nos machines.

1.4.2 Résultats d'Apriori et FP-Growth

Pour toutes les configurations qui vont suivres nous avons volontairement choisi la **confiance_min = 0** pour afficher toutes les règles générées, augmenter cette valeur filtrera les règles et n'affichera que celles dont la confiance calculée est $>$ confiance_min.

Configue 1 : Supp_min : 3

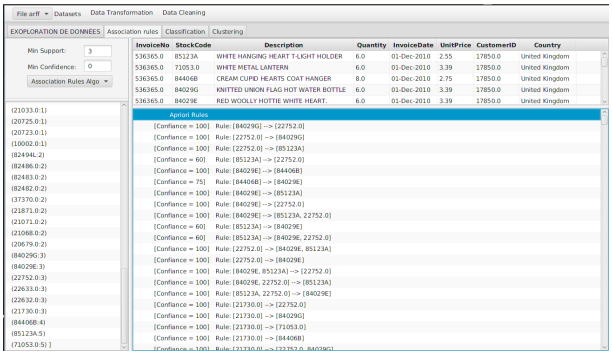


Figure 1.1: Exécution d'Apriori avec Supp_Min = 3

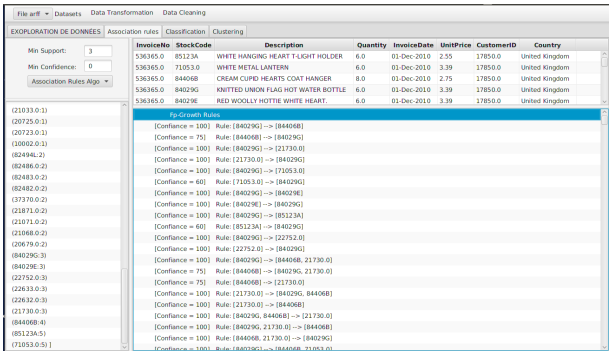


Figure 1.2: Exécution de FP-Growth avec Supp_Min = 3

Remarque:
-Nous remarquons qu'avec les deux méthodes les mêmes règles sont générées, ce qui est tout à fait normal.
-Nous avons généré une quantité importante de règles (plus de 100) avec des taux de confiance variant de **60% à 100%**.
Chose que nous justifiant par le supprot minimal qui est assez petit, ainsi le nombre de régles est plus important, car il porte sur plus d'ItemSet (non éliminés).
-Le temps d'exécution diffère entre les deux méthodes comme suit:

Apriori	40 s
FG-Growth	5 s

Configue 2 : Supp_min : 4

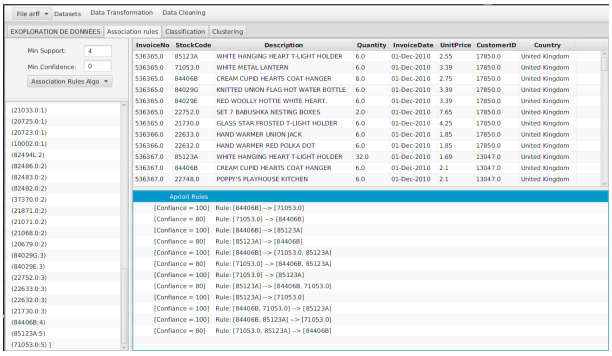


Figure 1.3: Exécution d'Apriori avec Supp_Min = 4

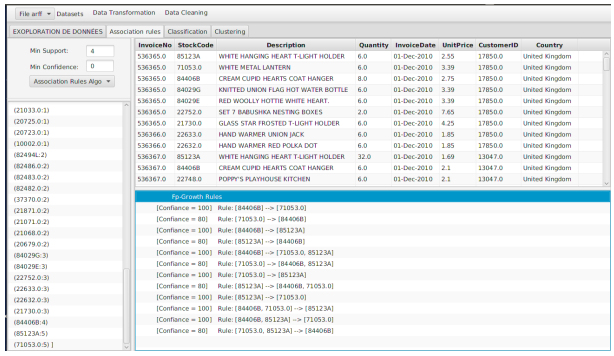


Figure 1.4: Exécution de FP-Growth avec Supp_Min = 4

Remarque:

-Nous remarquons qu'avec les deux méthodes les mêmes règles sont générées encore une fois, ce qui est tout à fait normal.

-Nous avons généré moins de règle (12 règles) comparé au nombre de règles générées avec un support minimal intérieur, et cela avec des taux de confiance de **80% et 100%**.

Ainsi un support minimal plus élevé que le précédent a généré moins de règles, car il porte sur plus d'ItemSet ayant un support plus élevé, ce qui augmente le nombre d'ItemSet éliminés et donc réduit les possibilités.

-Le temps d'exécution diffère entre les deux méthodes comme suit:

Apriori	0.960 s
FG-Growth	0.601 s

Configue 3 :Supp_min : 5

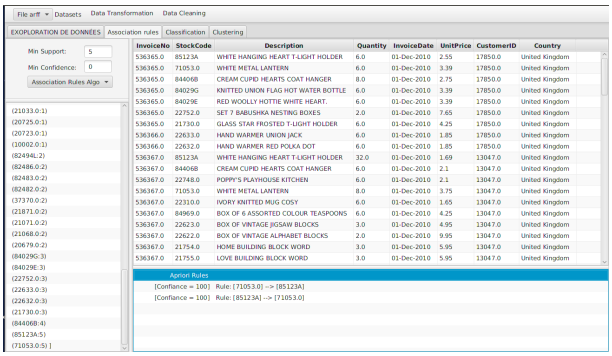


Figure 1.5: Exécution d'Apriori avec Supp_Min = 5

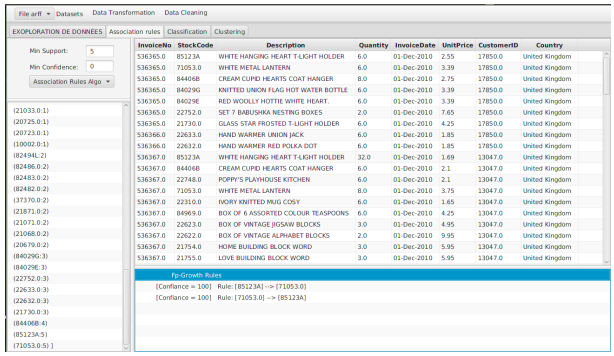


Figure 1.6: Exécution de FP-Growth avec Supp_Min = 5

Remarque:

-Nous remarquons qu'avec les deux méthodes les mêmes règles sont générées encore une fois, ce qui est tout à fait normal.

-Nous avons généré moins de règles (2 règles) comparé au nombre de règles générées avec un support minimal intérieur (3 et 4), et cela avec des taux de confiance de **100%** pour les deux règles.
Ainsi c'est le support minimal le plus élevé que l'on puisse utiliser pour obtenir des règles d'association car nous n'avons aucun Item ayant une fréquence supérieure à **5**.
-Le temps d'exécution diffère entre les deux méthodes comme suit:

Apriori	0.016 s
FG-Growth	0.006 s

1.5 Conclusion

Pour finir ce chapitre, nous tenons à énumérer les points importants auxquels nous avons fait face lors de nos expérimentations sur Apriori et FP-Growth, c'est-à-dire:

- Plus le nombre de transaction est grand plus le temps d'exécution est important (élevé)
- Plus le Supp_min est élevé moins les algorithmes ont d'itérations à faire, et donc plus le temps d'exécution est réduit.
- Les temps d'exécution de FP-Growth sont toujours plus faibles que ceux pour Apriori.

  Sauf lorsque le nombre d'itération est très petit, là nous avons pratiquement les mêmes temps d'exécution pour les deux approches.

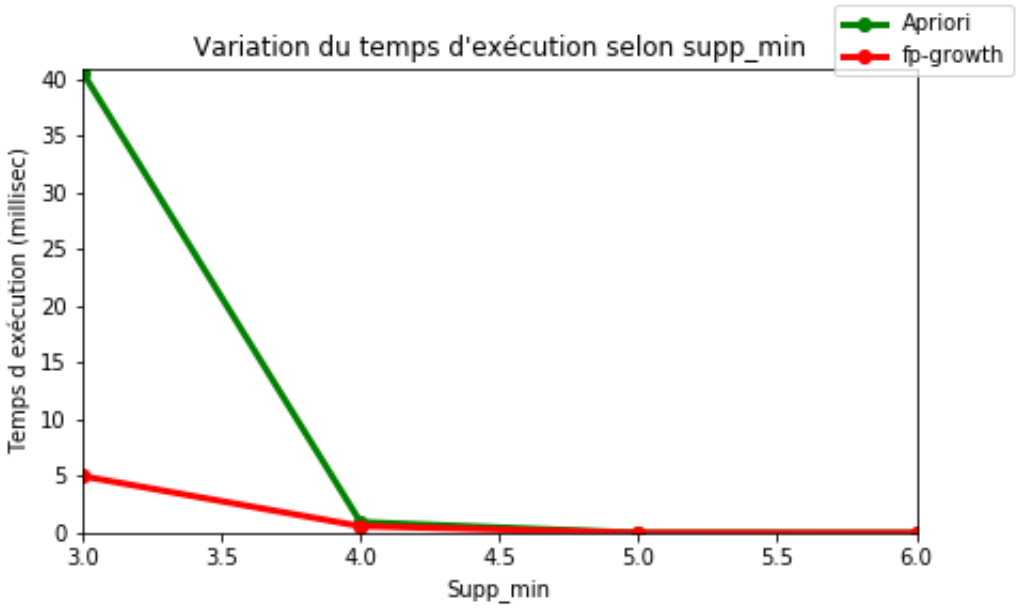


Figure 1.7: Variation du temps d'exécution selon supp_min

Aussi nous pouvons résumer notre étude comparative sur ces deux algorithme (Apriori et FP-Growth) en le tableau ci-dessous:

Paramètre	**Apriori**	**FG-Growth**
Structure de stockage	Tableau (matrice)	Arbre
Type de recherche	Largeur d'abord (toutes les combinaisons)	Diviser pour mieux régner + Profondeur d'abord
Techniques de bases	Génération de candidats, Jointure et filtrage	Extraire un sous arbre, Union et filtrage
Nombre de scans du DataSet	K+1 scans	2 scans
Espace mémoire	Grand	Petit
Temps d'exécution	Grand	Petit

En vu des résultats obtenus, on peut en conclure que l'algorithme FP-Growth est bien plus performant et intéressant à utiliser que l'algorithme Apriori.

Chapter 2

Algorithme de classification : K-NN

2.1 Introduction

K nearest neighbor où plutôt k plus proche voisins était décrit dans les **années 50** , c'est une méthode intensive qui demande beaucoup de temps de calcul pour arriver au résultat surtout quand il s'agit d'un large training sets, mais avec le développement des ordinateurs et la puissance de calcul son utilisation est revenu à l'actualité .[2]

K NN fait partie des algorithmes de classification qui apprenne par analogie , donc il se base sur des données d'entraînements (training set) pour prédire la classe des données de tests en faisant un calcul intensif de distance à chaque tuple des données de tests .

2.2 Principe de fonctionnement

On peut résumer le fonctionnement de l'algorithme K NN dans les points suivants mais avant on doit définir quelques notions:

2.2.1 Training set

un training set est un ensemble de données sur lequel s'opère l'analogie , ce choix est critique il doit être aussi diverse que possible pour couvrir toutes les possibilités dans le but de généraliser le mieux par la suite , plus le training set est grand et divers plus la classification est susceptible d'être plus précise .

2.2.2 Testing set

le testing set est l'ensemble de données sur lequel on va tester notre modèle entraîné , il faut noter que les données du training et du set son de mème type , autrement dit ce sont des instances du même dataset.

2.2.3 Distance

K nn se base globalement sur le calcul de distance entre les instances du training set et celui du test set.

Il existe pourtant plusieurs distances :

Distance Euclidienne

$$Distance(X_1, X_2) = \sqrt{\sum_{i=1}^{n}(X_{1i} - X_{2i})^2}$$

on l'utilise pour calculer la distance pour les attributs à valeurs numériques .

Distance Nominal

Pour la distance nominal on utilisé celle qui est "simple matching":

$$Distance(X_1, X_2) = \frac{p - m}{p} \tag{2.1}$$

où:
Mest le nombre d'attribut ou la valeur de l'instance X_1 est identique à celle de X_2 .
P est le nombre total d'attribut ayant un type nominal .

Distance Hybride

Afin de calculer la distance entre deux instances ayant des attributs de type nominal et numérique on a utilisé la somme des deux distances nominal et numérique .

$$Distance(X_1, X_2) = \sum \begin{cases} X_1 isnumeric & \sqrt{\sum_{i=1}^{n}(X_{1i} - X_{2i})^2} \\ X_1 isnominal & \frac{P-M}{P} \end{cases}$$

Plus la distance est grande moins les deux instances sont dites similaires et inversement .

2.2.4 Fonctionnement

Ainsi on peut résumer le principe de fonctionnement comme suit:

1. Partitionner les données en ensemble de test et training
2. Pour chaque instance de l'ensemble de test faire
3. Calculer la distance entre l'instance courante du test et toutes les instances du training set les mettre avec la classe correspondante , au fur et à mesure dans un tableau ou une liste.
4. Trier le tableau de distance par ordre croissant
5. prendre Les K premiers éléments de la liste et calculer le mode (la classe la plus fréquente)
6. Attribuer la classe trouvée en calculant le mode à l'instance du test set courante .

Complexité:

La complexité Temporelle de **Knn** est Polynomial quadratique, en **O(**$taille(trainingset)^2 * taille(testset)$**)**.

car on fait un tri sur un tableau comportant les distances de l'instance du test set à classifier avec tout le Training-set, la fonction de tri étant de complexité quadratique.

La complexité Spatial de **Knn** est en **O(n)**.

Avec: n : nombre d'instances du DataSet (test + training).

2.3 Structures de données

Dans le but d'implémenter notre algorithme de classification K NN on a opté pour la modélisation suivante:

2.3.1 Element

- **Classi**: qui est la classe de l'élément
- **distance:** qui est la distance associé à l'élément de type réel (float ou double)

2.3.2 VecteurD(Vecteur de distances)

- **ListofDistances:** qui est une liste contenant des ELEMENT(décrit plus haut)

la liste de distance étant nécessaire pour notre modélisation a fin de faire un tri décroissant sur cette liste selon la distance et récupérer ainsi les K PREMIÈRES Distances associées au instances puis ainsi on pourra extraire le mode dans ces K premières distances.

2.4 Pseudo-code

Algorithm 5 K NN:()

Input: TrainingSet : dataset qui sert à faire l'apprentissage,
TestSet : Données à classer,
K : nombre de voisins

Output: TestSetClassé : les données classifiées

```
for instance1 ∈ TestingSet do
    for instance2 ∈ TrainingSet do
        List_distance.add(CalculDistance(instance1,instance2))
    List_distance ← Trier(List_distance)
    Class ← K-MostcommonClass(List_distance,K)
    Instance1.Class ← Class
    TestSetClassé.add(Instance1)
return TestSetClasse
```

2.5 Résultats

Pour la méthode de classification supervisé KNN les paramètres empiriques sont le K et la taille du training et test set . on a donc fait varier K et la taille des trainingset et testset et voir comment ils influencent la précision.

Pour la précision on l'a calculé comme suit :

$$Accuracy = \frac{VraiPositives}{Taille(Testset)}$$

on a aussi décidé d'étudier les variations du temps d'exécution en fonction de nos paramètres les résultats sont les suivants :

2.5.1 Dataset: labor

Trainingset:90 et TestSET:10

Temps d'exécution + précision

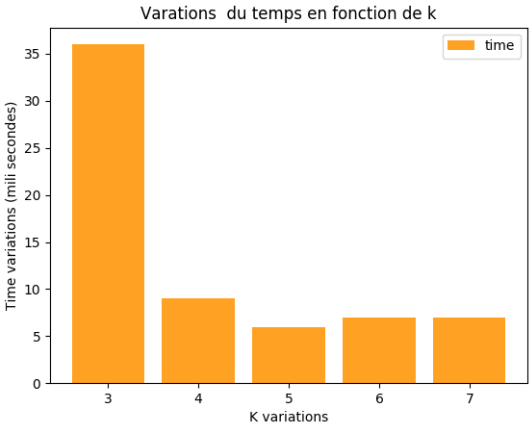


Figure 2.1: Variations du temps en fonction de K

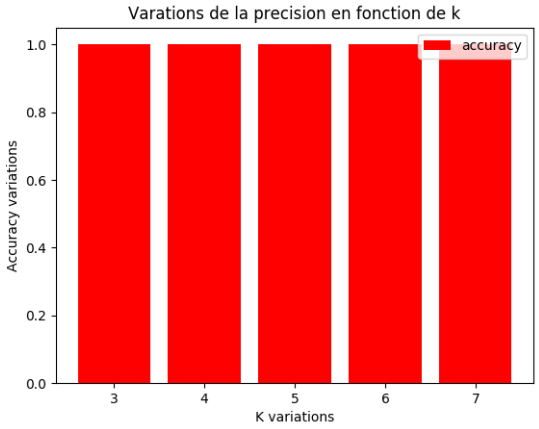


Figure 2.2: Variations de la précision en fonction de K

Trainingset:80 et TestSET:20

Temps d'exécution + précision

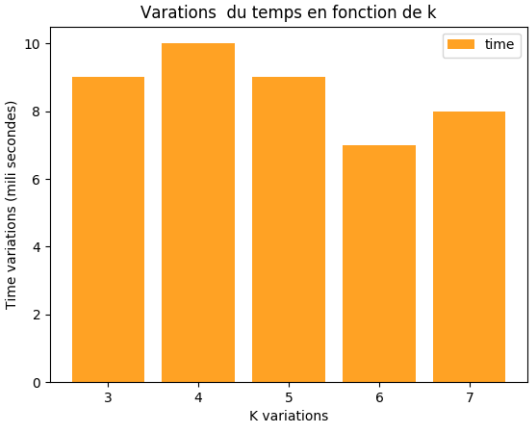


Figure 2.3: Variations du temps en fonction de K

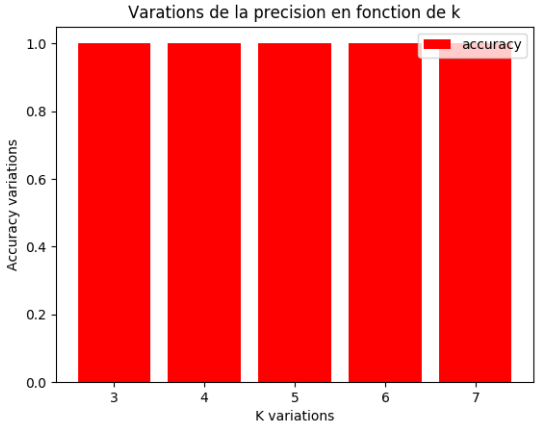


Figure 2.4: Variations de la précision en fonction de K

Trainingset:70 et TestSET:30

Temps d'exécution + précision

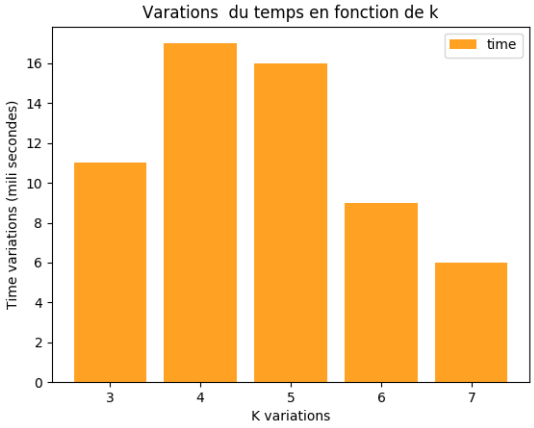


Figure 2.5: Variations du temps en fonction de K

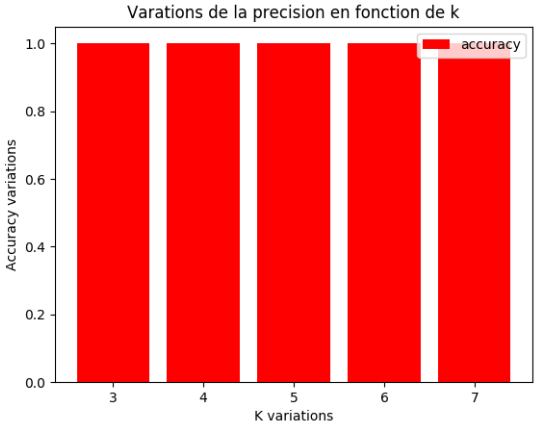


Figure 2.6: Variations de la précision en fonction de K

Trainingset:60 et TestSET:40

Temps d'exécution + précision

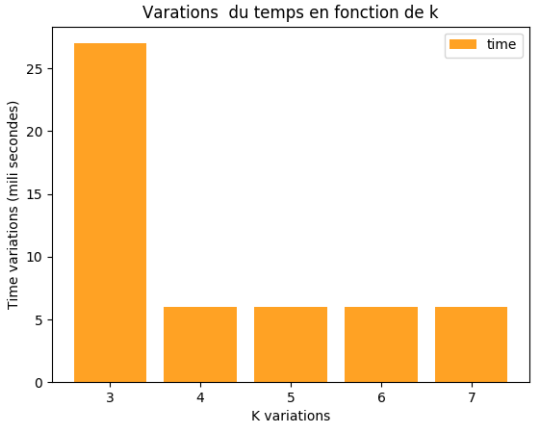


Figure 2.7: Variations du temps en fonction de K

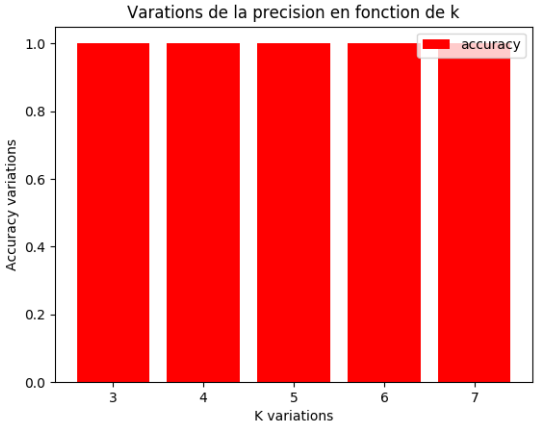


Figure 2.8: Variations de la précision en fonction de K

Trainingset:50 et TestSET:50

Temps d'exécution + précision

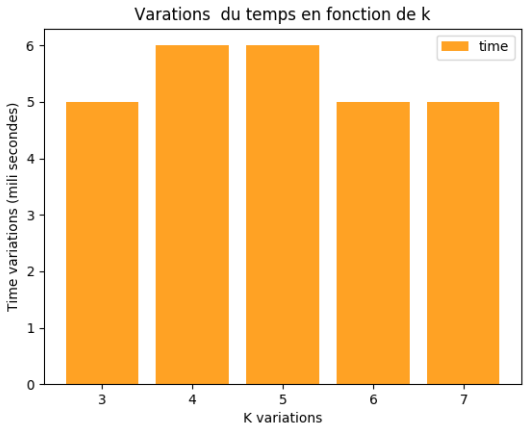


Figure 2.9: Variations du temps en fonction de K

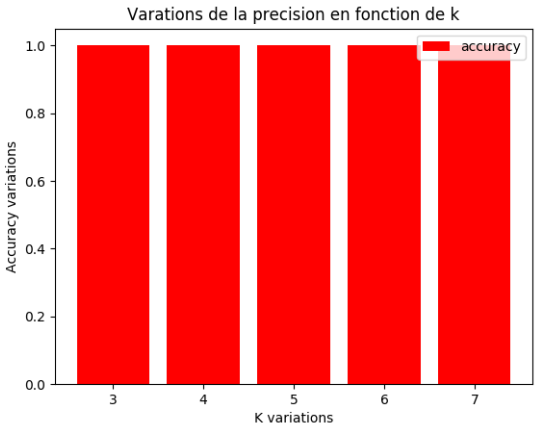


Figure 2.10: Variations de la précision en fonction de K

2.5.2 Dataset: Unbalanced (numérique)

unbalanced est un data set numerique avec une classe binaire (active ,non active) il comporte plus de 800 instanes et 37 attributs .

Trainingset:90 et TestSET:10

Temps d'exécution + précision

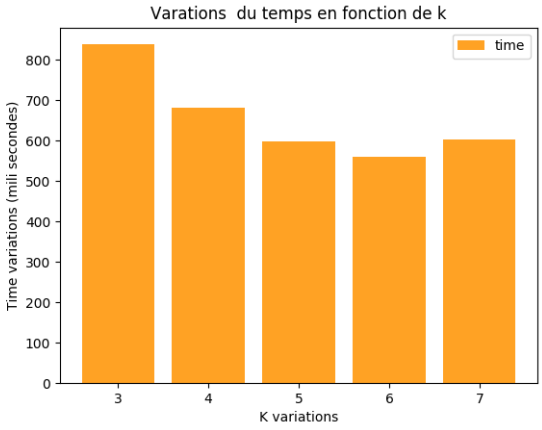


Figure 2.11: Variations du temps en fonction de K

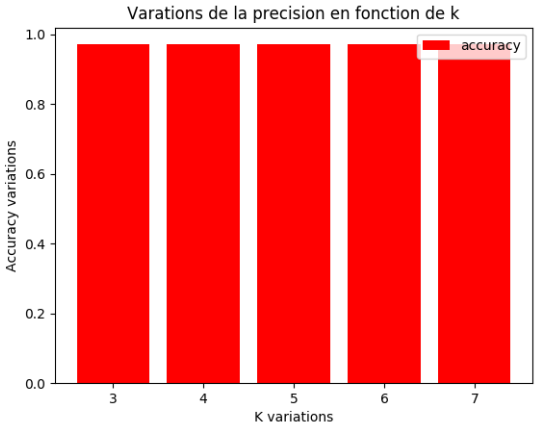


Figure 2.12: Variations de la précision en fonction de K

Trainingset:80 et TestSET:20

Temps d'exécution + précision

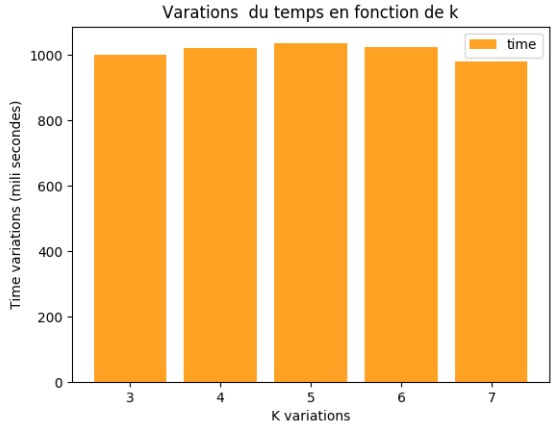


Figure 2.13: Variations du temps en fonction de K

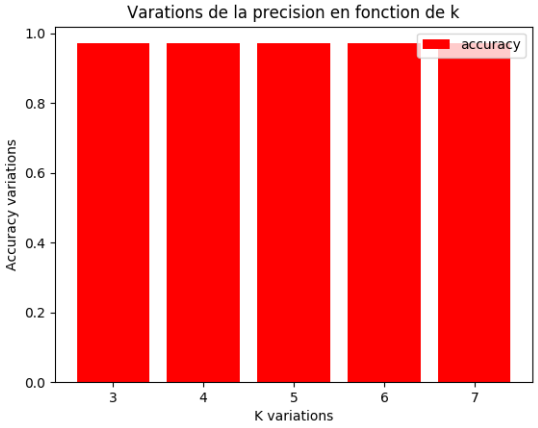


Figure 2.14: Variations de la précision en fonction de K

Trainingset:70 et TestSET:30

Temps d'exécution + précision

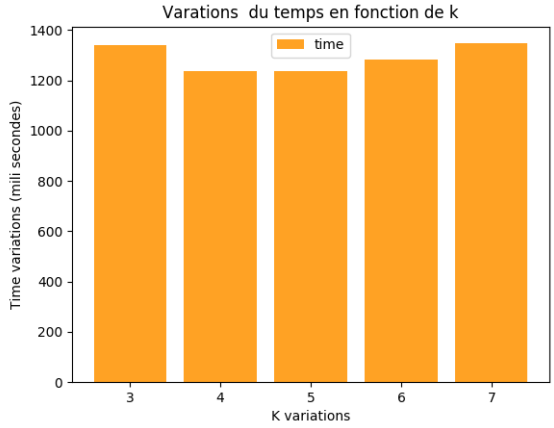


Figure 2.15: Variations du temps en fonction de K

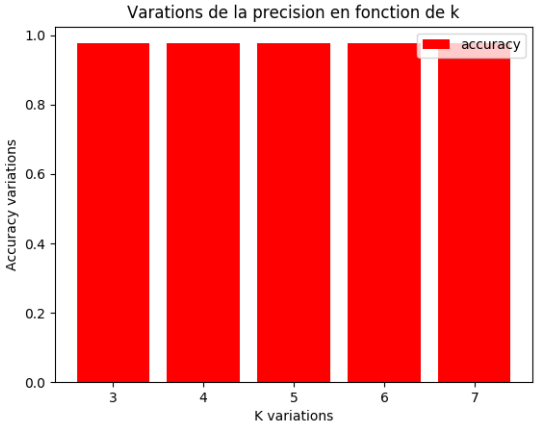


Figure 2.16: Variations de la précision en fonction de K

Trainingset:60 et TestSET:40

Temps d'exécution + précision

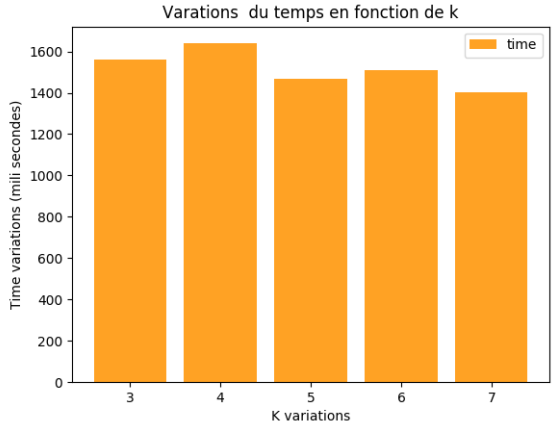


Figure 2.17: Variations du temps en fonction de K

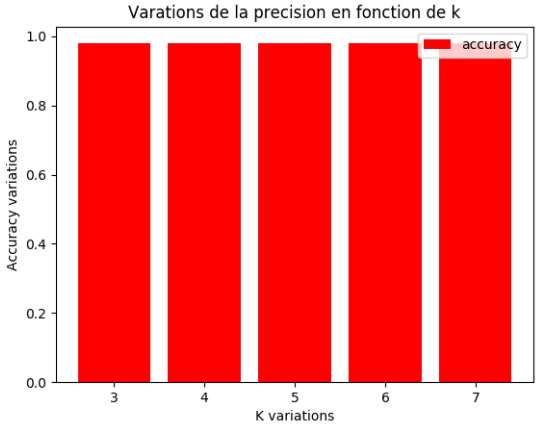


Figure 2.18: Variations de la précision en fonction de K

Trainingset:50 et TestSET:50

Temps d'exécution + précision

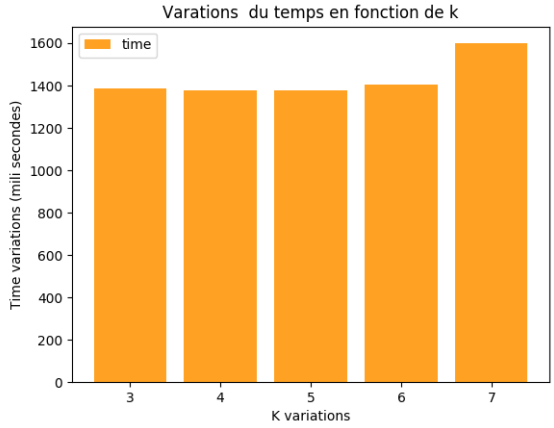


Figure 2.19: Variations du temps en fonction de K

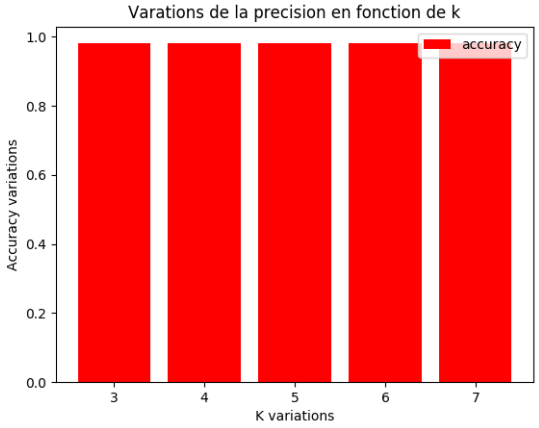


Figure 2.20: Variations de la précision en fonction de K

Trainingset:80 et test:20 avec K=7,13,20

on a pri **Temps d'exécution + précision**

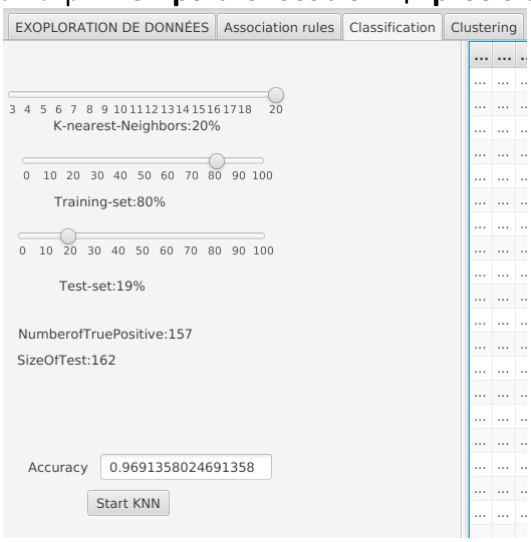


Figure 2.21: Précision pour K=20 pour dataset unbalanced

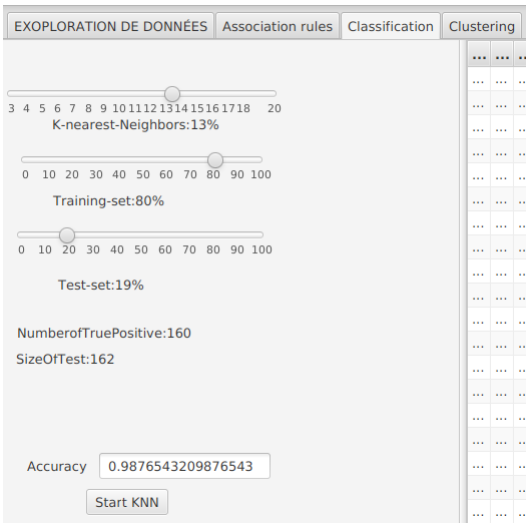


Figure 2.22: Précision pour k=13 pour dataset unbalanced

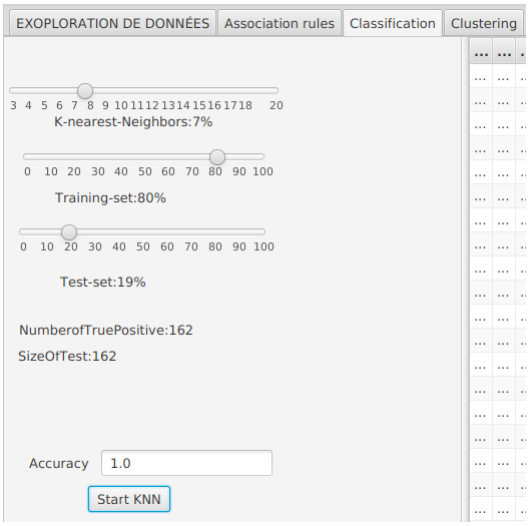


Figure 2.23: Précision pour k=7 pour dataset unbalanced

Remarques On remarque clairement que pour le dataset unbalanced après normalisation et nettoyage on a eu :

K	7	13	20
Précision	100%	98%	96%

K et la précision sont inversement proportionnelle c'est à dire si K augmente on peut noter que la précision diminue cela peut être expliqué par la confusion que rencontre KNN lorsque le nombre de plus proche voisins à considérer est important , la classe modale varie et on va prendre des voisins qui sont de plus en plus loin de notre instance à classifier.

2.6 Conclusion

Selon nos expérimentations on a pu remarquer les points suivants :

- L'approche K-nn est caractérisé par le paramètre **K** qui est le nombre de voisin à prendre en compte pour classer une instance, ce paramètre doit être fixé suite à plusieurs expérimentations.
- Les variations de K (de 3 jusqu'à 7) n'influencent pas énormément la précision par contre elle augmente parfois le temps d'exécution. mais si on augmente K à des valeurs conséquentes par exemple 20 etc , on remarque que la précision diminue car on s'intéresse plus aux classes des premières instances vu qu'on va prendre 20 voisins et la classe modale par rapport à ces voisins (20), une solution serait de pondérés les K élément en donnant un poids à chaque instance trouvée dans les K voisins en mettant en priorité celles qui sont proches (un peu comme dans la recherche d'information).
- les variations du training set et test set ont influencé la précision on remarque que plus on augmente le testset et on diminue le training set plus la précision diminue, ceci peut être expliqué par le fait que le training set ne devient plus assez diverse (il est même possible qu'une classe entière ne soit pas représenté), donc les données de test deviennent inconnues et il ne saura pas comment les classifier.
- On a noté aussi que KNN est trés sensible au bruit et à la normalisation , par exemple on a eu des précisions de 45% pour le dataset labor SANS normalisation et sans nettoyage alors que aprés le prétraitement on eu prés de 98% ceci est en relation direct avec le calcul de distance , un attribut dominera la classification (il convergera cette dernière) si il a des valeurs assez grandes et inversement un attribut aura moins d'effet si ces valeurs sont petites on parle ici du cas numérique.

Chapter 3

Algorithme de clustering : DBSCAN

3.1 Introduction

Le Clustering consiste en un regroupement d'objets (instances) dans un même cluster/groupe selon la similitude qui les rassemble, c'est à dire en comparant les caractéristiques de ces objets (instances) nous obtenons une distance que l'on cherche à minimiser entre deux objets du même cluster et à maximiser entre deux clusters.

Parmi les méthodes de clustering nous avons quatre catégories:

- Méthodes basées Partitionnement
- Méthodes Hiérarchiques
- Méthodes basées Densité
- Méthodes basées Grille

3.2 Principe de fonctionnement

DBSCAN a été proposé en **1996** par Martin.E et al. Il est basé sur la densité, Il procède à la création des clusters de manière incrémentale c'est-à-dire qu'il crée et fini de remplir un cluster avant de passer au suivant, ainsi cet algorithme prend deux paramètres empiriques qui sont:

- **epsilon (ϵ) :** la distance maximale entre deux objets (instances) pour qu'ils soient considérés comme voisins.
- **MinPts :** Le nombre minimum de voisins pour créer un cluster.

Complexité:
La complexité Temporelle de **dbscan** est Polynomial, en **O(n^2)**.
La complexité Spatiale de **dbscan** est en **O(n)**.
Avec: n : nombre d'instances du DataSet.

3.3 Structures de données

Cluster

Nous avons défini la structure de **Cluster**, celle-ci représente un Cluster contenant l'ensemble des instances qu'il regroupe, telle qu'elle est composée de 2 attributs:

- **clusterName:**chaine de caractère représentant le nom du Cluster. (*exemple: "cluster1"*)
- **listInstances:** La liste des instances appartenant a ce Cluster.

dbscan

Il s'agit d'une instance pour l'exécution de dbscan, telle que les éléments de base nécessaires pour cela y sont structuré comme suit:

- **DataSet:** le dataset contenant les instances que nous souhaitons clusteriser.
- **epsilon (ϵ):** distance maximal pour considérer une instance comme un voisin d'une autre instance.
- **MinPts:** le nombre minimal de voisins pour un point afin de le regrouper dans un cluster.

3.4 Pseudo-code

Algorithm 6 DBSCAN

Input: D:Dataset à clusteriser ,$Epsilon(\epsilon)$:distance minimal d'un voisin, $MinPts$: nombre minimum de voisins

Output: $Clusters$: Liste des clusters

```
24
   foreach Point P ∈ D do
25     MarquéVisité(P)
       PtsVoisins ← Voisinage(D,P,epsilon)

26     if Taille(PtsVoisins) < MinPts then
27         BruitList.add(P)
           //marquer P comme du bruit
28     else
29         NewCluster ← CreationCluster()
           EtendreCluster(D, P, PtsVoisins, NewCluster, epsilon, MinPts)
           Clusters.add(NewCluster)
30 return Clusters
```

Algorithm 7 PROCEDURE ETENDRECLUSTER

EtendreCluster(D, P, PtsVoisins, NewCluster, epsilon, MinPts)
$NewCluster \leftarrow$ Inserer(P)
foreach $PointP' \in PtsVoisins$ **do**
  if P' *nonVisité* **then**
    MarquéVisité(P')
    $PtsVoisins' \leftarrow$ Voisinage(D,P',epsilon)
    if $Taille(PtsVoisins') \geq MinPts$ **then**
      $PtsVoisins = PtsVoisins \cup PtsVoisins'$
    end
  end
  if $P' \notin AnyCluster$ **then**
    $NewCluster \leftarrow$Inserer(P')
  end
end

Algorithm 8 VOISINAGE

Voisinage(D, P, epsilon)
foreach $PointsPts \in D$ **do**
  if $Distance(P,Pts) \leq epsilon$ **then**
    $listVoisins$.add(Pts)
  end
end
return $listVoisins$

3.5 Explication

1. Sélectionner aléatoirement un point **p**, le marqué comme *"visité"* c'est-à-dire clusterisé.
2. Récupérer tous les points accessibles en densité à partir de **p**, c'est-à-dire distance entre **p** et ces autres points $\leq$ Epsilon (ϵ), on appelle ces points les voisins de **p**.
3. Si le nombre de voisins de **p** est $\geq$ MinPts,Alors **p** est un **corps**, on forme un nouveau cluster.

   Puis on procède de la même manière pour ces voisins de **p**, mais cette fois on ne crée pas un nouveau cluster mais on les ajoute au cluster de **p**.
4. Sinon **p** est un point frontière, c'est-à-dire aucun point n'est accessible en densité à partir de **p**.
5. DBSCAN visite le prochain point du Dataset non clusterisé.
6. Si un point ne peux être ajouté dans aucun cluster et ne peux pas être un point corps pour un nouveau cluster (epsilon,MinPts non compatible) alors marquer le point comme bruit.
7. Continuez le processus jusqu'à ce que tous les points aient été traités.

Évaluation:

Il existe des mesures d'évaluation du clustering obtenu, nous citons: **L'Inertie Inter-Cluster** et **Intra-Cluster**, tel que:

- **Intra-Cluster :** elle représente la somme des inerties de tous les clusters La formule d'évaluation est comme suit:

$$\sum_{i=1}^{nbrCluster} \sum_{X_i \in C_i} d^2(X_i, G_i)$$

Avec:
C_i : i^{eme} cluster.
G_i : Centre de gravité du i^{eme} cluster.
X_i : Instance du i^{eme} cluster.
d : distance euclidienne.

- **Inter-Cluster :** cette mesure permet de mesurer la disparité des clusters, c'est à dire la distance entre clusters, celle-ci doit être élevée pour rester cohérent dans la logique du clustering.
La formule d'évaluation est la suivante:

$$\sum_{i=1}^{nbrCluster} d^2(G_i, Gl)$$

Avec:
G_i : Centre de gravité du i^{eme} cluster.
Gl : Centre de gravité globale (de tous le nuage de points).
d : distance euclidienne.

Remarque 1:

Il existe la version normalisée de ces mesures, qui sont comme suit:

Intra : $\langle \sum_{i=1}^{nbrCluster} \sum_{X_i \in C_i} d^2(X_i, G_i) \rangle / nbrInstancesClusterisees$

Inter : $\langle \sum_{i=1}^{nbrCluster} d^2(G_i, Gl) \rangle / nbrCluster$

Remarque 2:

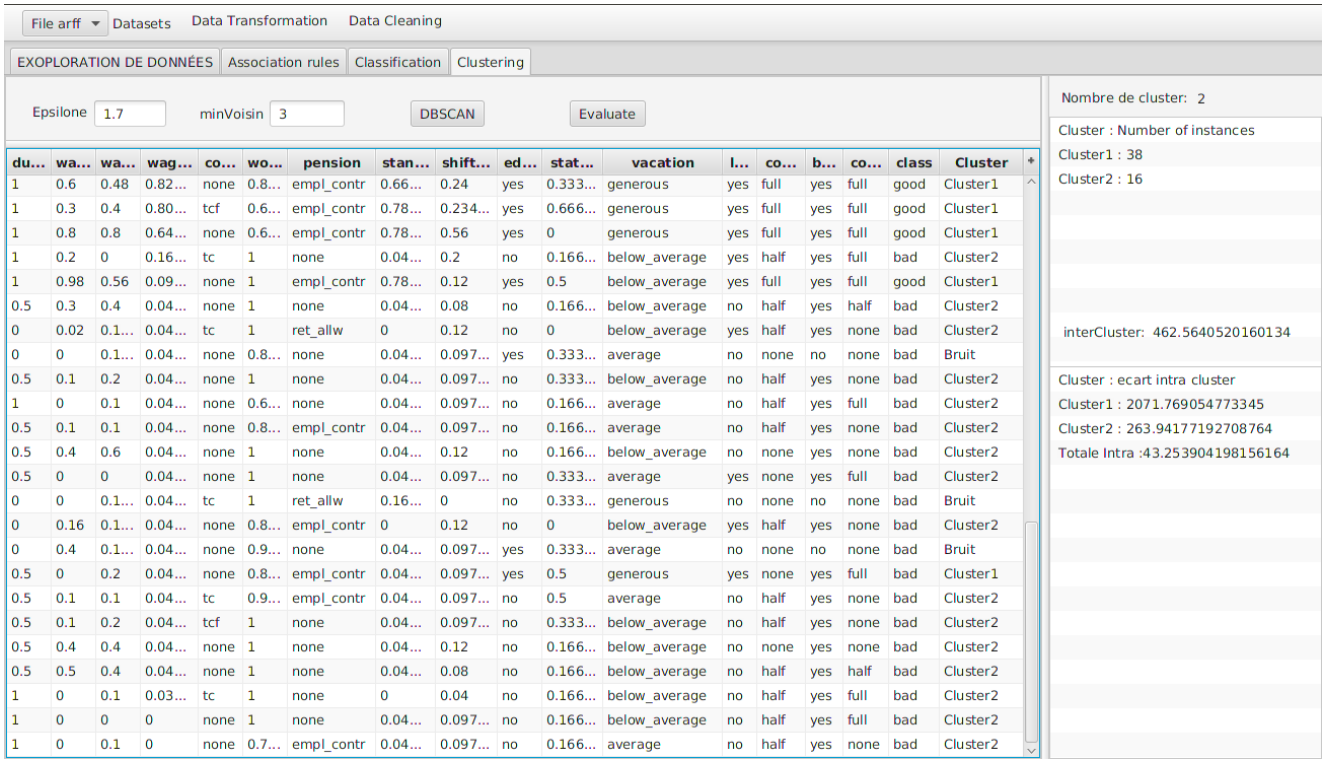
Ces deux mesures sont mises en comparaison, pour dire que nous avons un bon clustering, il faut que 2 points soient visibles :

- Nos Clusters soient danses $\Leftrightarrow$ minimiser la mesure intra-Cluster.
- Nos Clusters soient éloignés (clairement disjoint) $\Leftrightarrow$ maximiser la mesure inter-Cluster.

3.6 Résultats

DataSet : LABOR (57 instance/17 attributs)

- **Mixte (Nominal + numérique)**
- **Epsilon = 1.7**
- **MinPts = 3**



du...	wa...	wa...	wag...	co...	wo...	pension	stan...	shift...	ed...	stat...	vacation	l...	co...	b...	co...	class	Cluster
1	0.6	0.48	0.82...	none	0.8...	empl_contr	0.66...	0.24	yes	0.333...	generous	yes	full	yes	full	good	Cluster1
1	0.3	0.4	0.80...	tcf	0.6...	empl_contr	0.78...	0.234...	yes	0.666...	generous	yes	full	yes	full	good	Cluster1
1	0.8	0.8	0.64...	none	0.6...	empl_contr	0.78...	0.56	yes	0	generous	yes	full	yes	full	good	Cluster1
1	0.2	0	0.16...	tc	1	none	0.04...	0.2	no	0.166...	below_average	yes	half	yes	full	bad	Cluster2
1	0.98	0.56	0.09...	none	1	empl_contr	0.78...	0.12	yes	0.5	below_average	yes	full	yes	full	good	Cluster1
0.5	0.3	0.4	0.04...	none	1	none	0.04...	0.08	no	0.166...	below_average	no	half	yes	half	bad	Cluster2
0	0.02	0.1...	0.04...	tc	1	ret_allw	0	0.12	no	0	below_average	yes	half	yes	none	bad	Cluster2
0	0	0.1...	0.04...	none	0.8...	none	0.04...	0.097...	yes	0.333...	average	no	none	no	none	bad	Bruit
0.5	0.1	0.2	0.04...	none	1	none	0.04...	0.097...	no	0.333...	below_average	no	half	yes	none	bad	Cluster2
1	0	0.1	0.04...	none	0.6...	none	0.04...	0.097...	no	0.166...	average	no	half	yes	full	bad	Cluster2
0.5	0.1	0.1	0.04...	none	0.8...	empl_contr	0.04...	0.097...	no	0.166...	average	no	half	yes	none	bad	Cluster2
0.5	0.4	0.6	0.04...	none	1	none	0.04...	0.12	no	0.166...	below_average	no	none	yes	none	bad	Cluster2
0.5	0	0	0.04...	none	1	none	0.04...	0.097...	no	0.333...	average	yes	none	yes	full	bad	Cluster2
0	0	0.1...	0.04...	tc	1	ret_allw	0.16...	0	no	0.333...	generous	no	none	no	none	bad	Bruit
0	0.16	0.1...	0.04...	none	0.8...	empl_contr	0	0.12	no	0	below_average	yes	half	yes	none	bad	Cluster2
0	0.4	0.1...	0.04...	none	0.9...	none	0.04...	0.097...	yes	0.333...	average	no	none	no	none	bad	Bruit
0.5	0	0.2	0.04...	none	0.8...	empl_contr	0.04...	0.097...	yes	0.5	generous	yes	none	yes	full	bad	Cluster1
0.5	0.1	0.1	0.04...	tc	0.9...	empl_contr	0.04...	0.097...	no	0.5	average	no	half	yes	none	bad	Cluster2
0.5	0.1	0.2	0.04...	tcf	1	none	0.04...	0.097...	no	0.333...	below_average	no	half	yes	none	bad	Cluster2
0.5	0.4	0.4	0.04...	none	1	none	0.04...	0.12	no	0.166...	below_average	no	none	yes	none	bad	Cluster2
0.5	0.5	0.4	0.04...	none	1	none	0.04...	0.08	no	0.166...	below_average	no	half	yes	half	bad	Cluster2
1	0	0.1	0.03...	tc	1	none	0	0.04	no	0.166...	below_average	no	half	yes	full	bad	Cluster2
1	0	0	0	none	1	none	0.04...	0.097...	no	0.166...	below_average	no	half	yes	full	bad	Cluster2
1	0	0.1	0	none	0.7...	empl_contr	0.04...	0.097...	no	0.166...	average	no	half	yes	none	bad	Cluster2

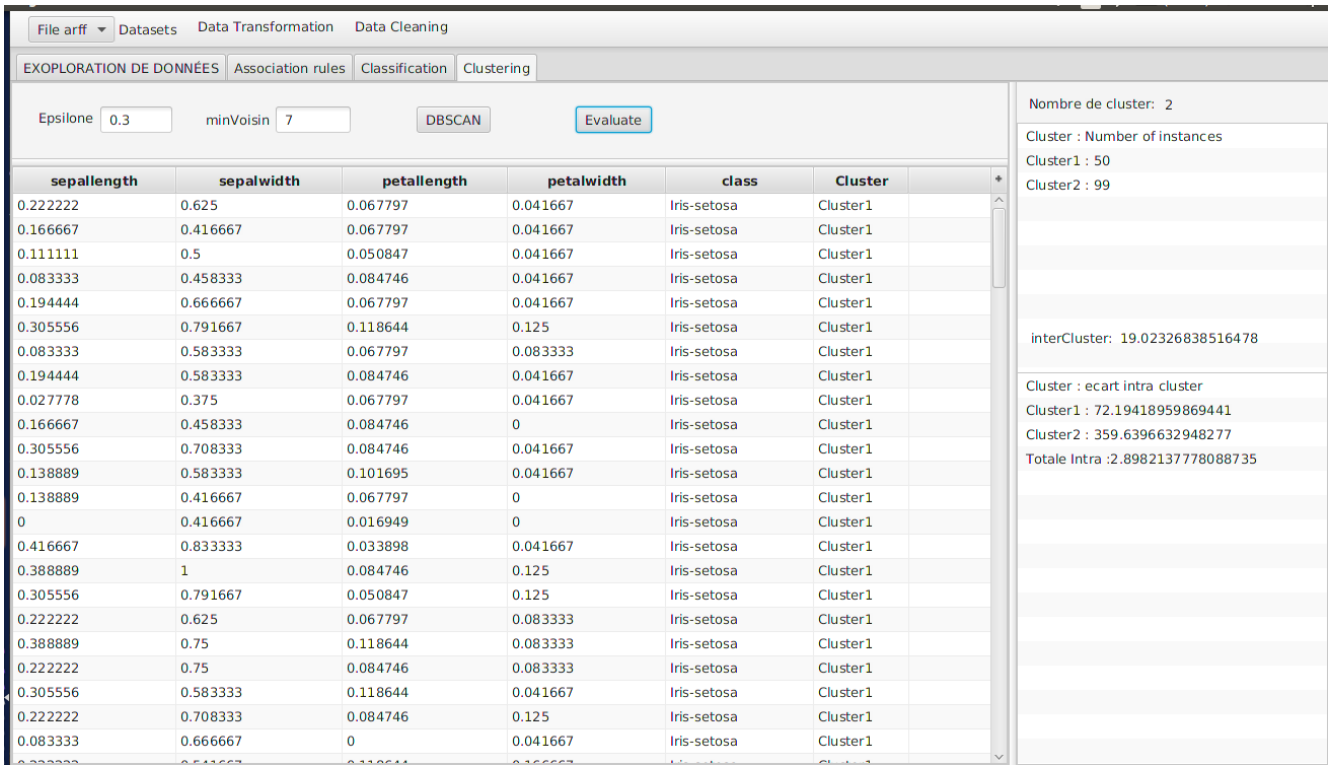
Figure 3.1: Exécution de DBSCAN sur le DataSet LABOR

Résultats obtenus :

- **2 Clusters:**
  - Cluster 1 : 38 instances
  - Cluster 2 : 16 instances
  - Bruit : 3 instances
- **Inertie:**
  - Inter-Cluster : 462.56
  - Intra-Cluster : 43.25
- **Temps d'exécution : 3ms**

DataSet : IRIS (150 instance/5 attributs)

- **Numérique**
- **Epsilon = 0.3**
- **MinPts = 7**



sepallength	sepalwidth	petallength	petalwidth	class	Cluster
0.222222	0.625	0.067797	0.041667	Iris-setosa	Cluster1
0.166667	0.416667	0.067797	0.041667	Iris-setosa	Cluster1
0.111111	0.5	0.050847	0.041667	Iris-setosa	Cluster1
0.083333	0.458333	0.084746	0.041667	Iris-setosa	Cluster1
0.194444	0.666667	0.067797	0.041667	Iris-setosa	Cluster1
0.305556	0.791667	0.118644	0.125	Iris-setosa	Cluster1
0.083333	0.583333	0.067797	0.083333	Iris-setosa	Cluster1
0.194444	0.583333	0.084746	0.041667	Iris-setosa	Cluster1
0.027778	0.375	0.067797	0.041667	Iris-setosa	Cluster1
0.166667	0.458333	0.084746	0	Iris-setosa	Cluster1
0.305556	0.708333	0.084746	0.041667	Iris-setosa	Cluster1
0.138889	0.583333	0.101695	0.041667	Iris-setosa	Cluster1
0.138889	0.416667	0.067797	0	Iris-setosa	Cluster1
0	0.416667	0.016949	0	Iris-setosa	Cluster1
0.416667	0.833333	0.033898	0.041667	Iris-setosa	Cluster1
0.388889	1	0.084746	0.125	Iris-setosa	Cluster1
0.305556	0.791667	0.050847	0.125	Iris-setosa	Cluster1
0.222222	0.625	0.067797	0.083333	Iris-setosa	Cluster1
0.388889	0.75	0.118644	0.083333	Iris-setosa	Cluster1
0.222222	0.75	0.084746	0.083333	Iris-setosa	Cluster1
0.305556	0.583333	0.118644	0.041667	Iris-setosa	Cluster1
0.222222	0.708333	0.084746	0.125	Iris-setosa	Cluster1
0.083333	0.666667	0	0.041667	Iris-setosa	Cluster1

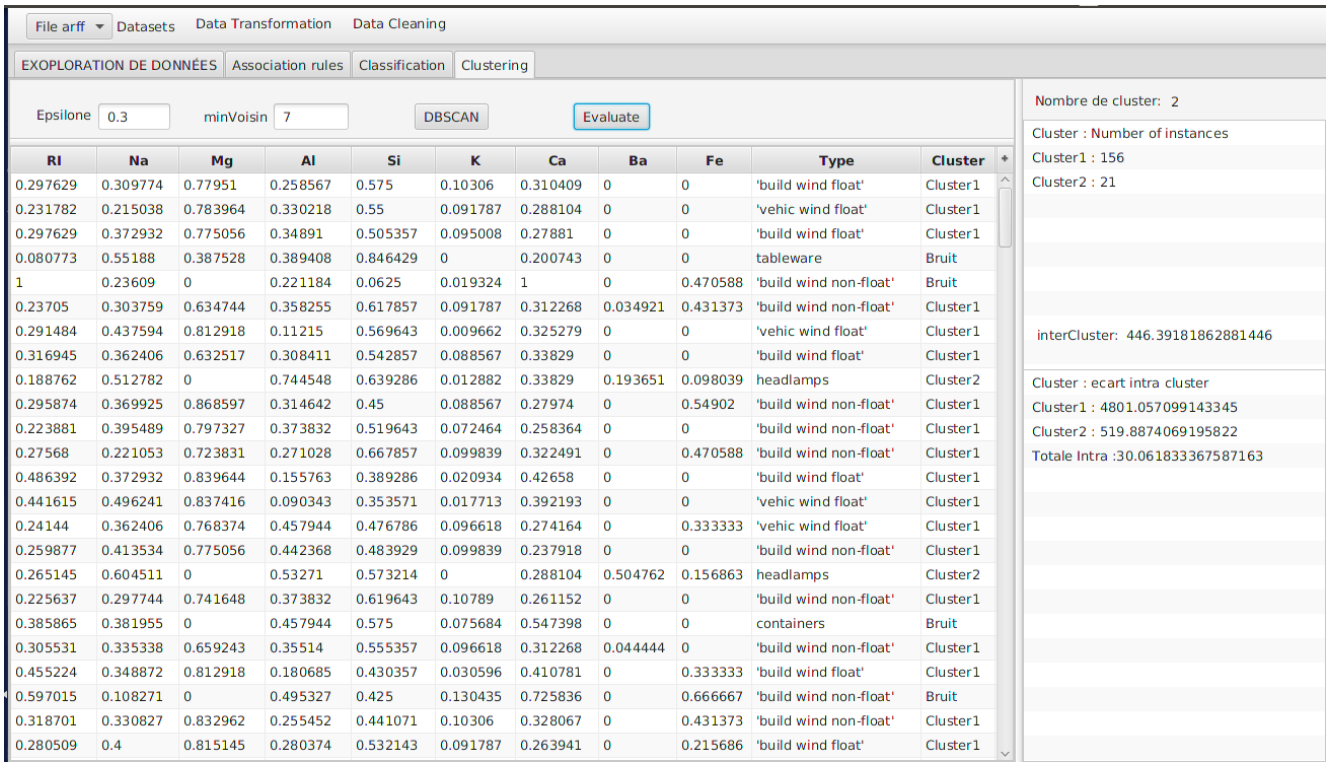
Figure 3.2: Exécution de DBSCAN sur le DataSet IRIS

Résultats obtenus :

- **2 Clusters:**
  - Cluster 1 : 50 instances
  - Cluster 2 : 99 instances
  - Bruit : 1 instances
- **Inertie:**
  - Inter-Cluster : 19.02
  - Intra-Cluster : 2.89
- **Temps d'exécution : 12ms**

DataSet : GLASS (214 instance/10 attributs)

- **Mixte (Nominal + numérique)**
- **Epsilon = 0.3**
- **MinPts = 7**



RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type	Cluster
0.297629	0.309774	0.77951	0.258567	0.575	0.10306	0.310409	0	0	'build wind float'	Cluster1
0.231782	0.215038	0.783964	0.330218	0.55	0.091787	0.288104	0	0	'vehic wind float'	Cluster1
0.297629	0.372932	0.775056	0.34891	0.505357	0.095008	0.27881	0	0	'build wind float'	Cluster1
0.080773	0.55188	0.387528	0.389408	0.846429	0	0.200743	0	0	tableware	Bruit
1	0.23609	0	0.221184	0.0625	0.019324	1	0	0.470588	'build wind non-float'	Bruit
0.23705	0.303759	0.634744	0.358255	0.617857	0.091787	0.312268	0.034921	0.431373	'build wind non-float'	Cluster1
0.291484	0.437594	0.812918	0.11215	0.569643	0.009662	0.325279	0	0	'vehic wind float'	Cluster1
0.316945	0.362406	0.632517	0.308411	0.542857	0.088567	0.33829	0	0	'build wind float'	Cluster1
0.188762	0.512782	0	0.744548	0.639286	0.012882	0.33829	0.193651	0.098039	headlamps	Cluster2
0.295874	0.369925	0.868597	0.314642	0.45	0.088567	0.27974	0	0.54902	'build wind non-float'	Cluster1
0.223881	0.395489	0.797327	0.373832	0.519643	0.072464	0.258364	0	0	'build wind non-float'	Cluster1
0.27568	0.221053	0.723831	0.271028	0.667857	0.099839	0.322491	0	0.470588	'build wind non-float'	Cluster1
0.486392	0.372932	0.839644	0.155763	0.389286	0.020934	0.42658	0	0	'build wind float'	Cluster1
0.441615	0.496241	0.837416	0.090343	0.353571	0.017713	0.392193	0	0	'vehic wind float'	Cluster1
0.24144	0.362406	0.768374	0.457944	0.476786	0.096618	0.274164	0	0.333333	'vehic wind float'	Cluster1
0.259877	0.413534	0.775056	0.442368	0.483929	0.099839	0.237918	0	0	'build wind non-float'	Cluster1
0.265145	0.604511	0	0.53271	0.573214	0	0.288104	0.504762	0.156863	headlamps	Cluster2
0.225637	0.297744	0.741648	0.373832	0.619643	0.10789	0.261152	0	0	'build wind non-float'	Cluster1
0.385865	0.381955	0	0.457944	0.575	0.075684	0.547398	0	0	containers	Bruit
0.305531	0.335338	0.659243	0.35514	0.555357	0.096618	0.312268	0.044444	0	'build wind non-float'	Cluster1
0.455224	0.348872	0.812918	0.180685	0.430357	0.030596	0.410781	0	0.333333	'build wind float'	Cluster1
0.597015	0.108271	0	0.495327	0.425	0.130435	0.725836	0	0.666667	'build wind non-float'	Bruit
0.318701	0.330827	0.832962	0.255452	0.441071	0.10306	0.328067	0	0.431373	'build wind non-float'	Cluster1
0.280509	0.4	0.815145	0.280374	0.532143	0.091787	0.263941	0	0.215686	'build wind float'	Cluster1

Figure 3.3: Exécution de DBSCAN sur le DataSet GLASS

Résultats obtenus :

- **2 Clusters:**
  - Cluster 1 : 156 instances
  - Cluster 2 : 21 instances
  - Bruit : 37 instances
- **Inertie:**
  - Inter-Cluster : 446039
  - Intra-Cluster : 30.06
- **Temps d'exécution : 24ms**

DataSet : VOTE (425 instance/17 attributs)

- **Nominal**
- **Epsilon = 1.5**
- **MinPts = 10**

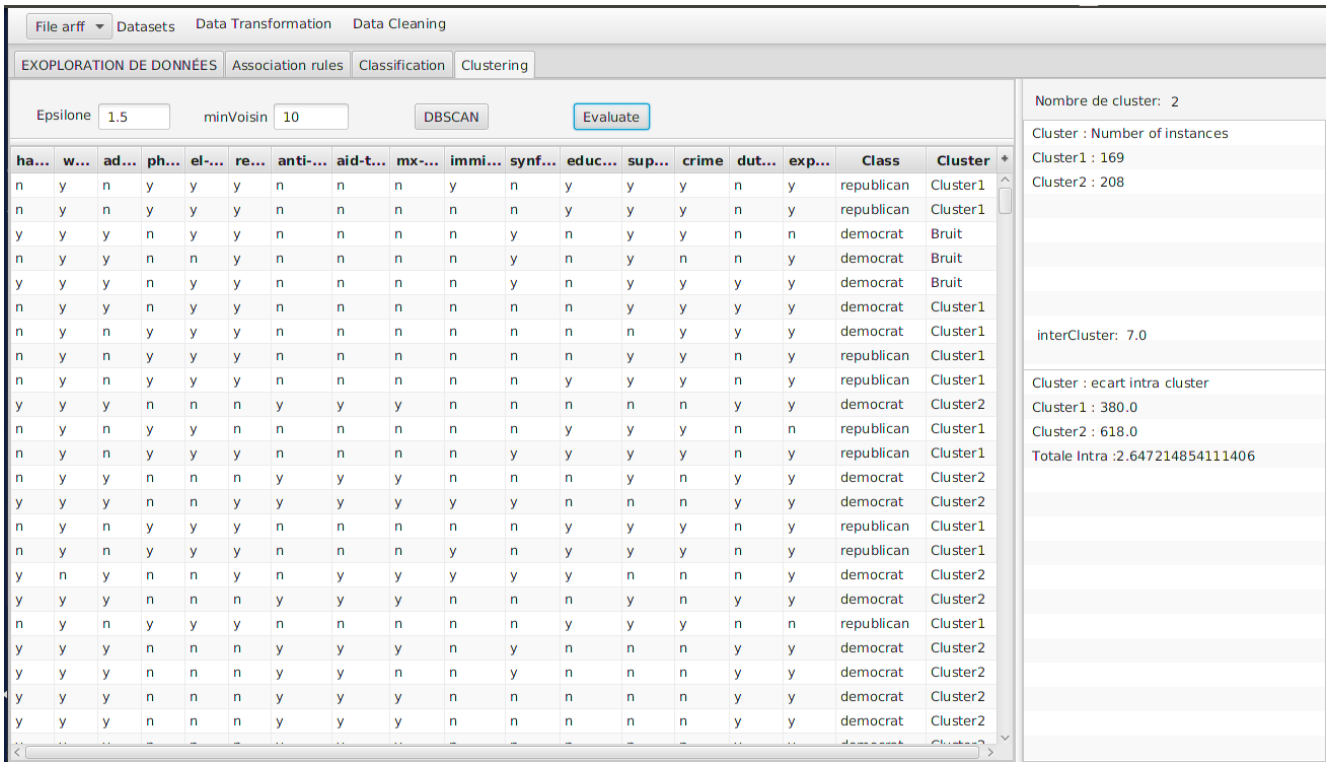


Figure 3.4: Exécution de DBSCAN sur le DataSet VOTE

Résultats obtenus :

- **2 Clusters:**
  - Cluster 1 : 169 instances
  - Cluster 2 : 208 instances
  - Bruit : 58 instances
- **Inertie:**
  - Inter-Cluster : 7
  - Intra-Cluster : 2.64
- **Temps d'exécution : 122ms**

DataSet : UNBALANCED (856 instance/33 attributs)

- **Numérique**
- **Epsilon = 1**
- **MinPts = 7**

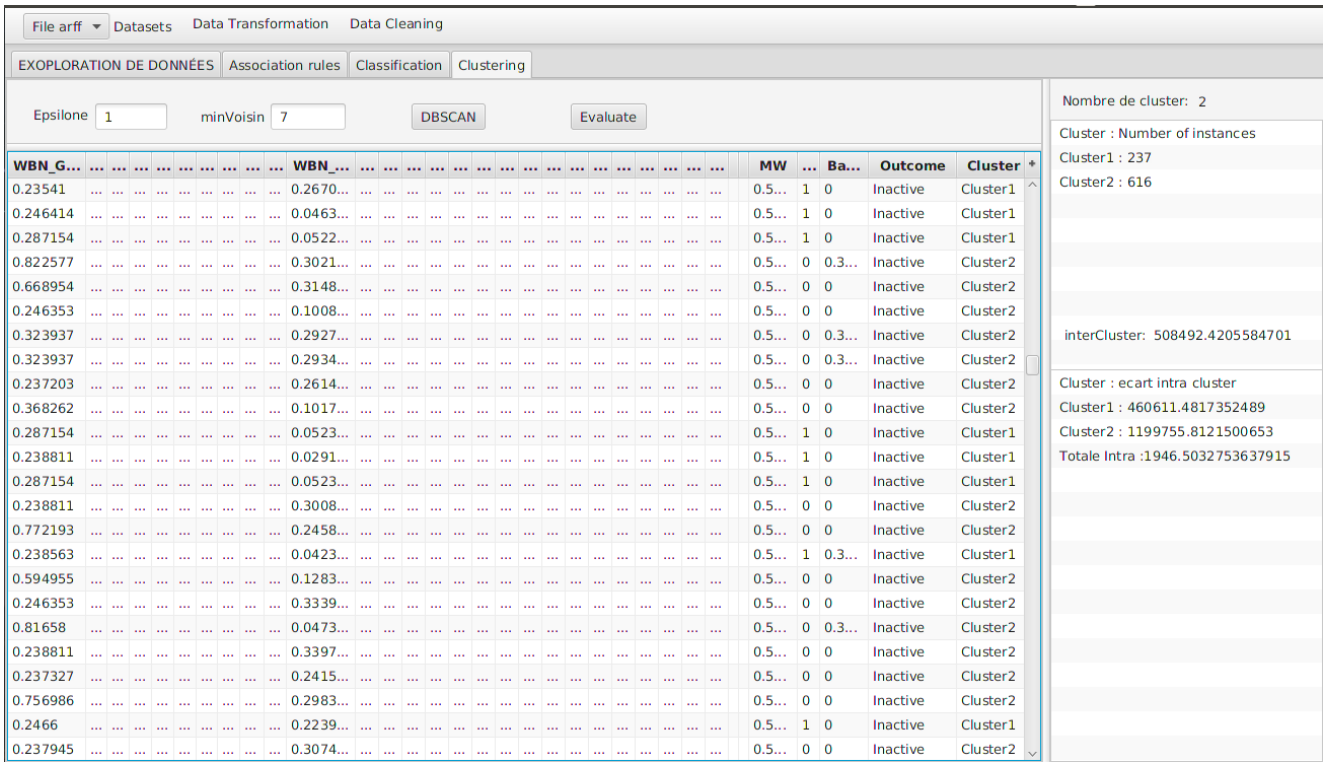


Figure 3.5: Exécution de DBSCAN sur le DataSet UNBALANCED

Résultats obtenus :

- **2 Clusters:**
  - Cluster 1 : 237 instances
  - Cluster 2 : 616 instances
  - Bruit : 3 instances
- **Inertie:**
  - Inter-Cluster : 508492.42
  - Intra-Cluster : 1946.5
- **Temps d'exécution : 779ms**

récapitulatif des résultats

Dans le tableau ci-dessous, nous avons regroupé les résultats des expérimentations mentionnées un peu plus haut:

DataSet	Epsilon	MinPts	Cluster1	Cluster2	Bruit	Inter	Intra	Temps
Labor	1.7	3	38	16	3	462.56	43.25	3ms
IRIS	0.3	7	50	99	1	19.02	2.89	12ms
Glass	0.3	7	156	21	37	446.39	30.06	24ms
Vote	1.5	10	169	208	58	7	2.64	122ms
Unbalanced	1	7	237	616	3	508492.42	1946.5	779ms

3.7 Conclusion

En conclusion, après implémentation de DBSCAN, et suite à plusieurs expérimentations, nous sommes capables de noter remarques importantes suivantes:

- Un avantage de DBSCAN est que Nous n'avons **pas besoin** de connaitre à l'avance ni le nombre de clusters (K) ni les centres de ces derniers (centriodes).
- **Epsilon**(ϵ) et **MinPts** sont des paramètres empiriques qu'il faut bien fixer après maintes expérimentations pour avoir les meilleurs résultats escomptés.
- Nous avons la possibilité d'évaluer notre clustering avec les mesures d'Inertie, et pour cela nous devons obtenir l'inertie **inter-Cluster** $>>$ **intra-Clutser**.

  Cela est bien visible via les résultats que nous avons obtenus (voir tableau des résultats)
- Il est **obligatoire** de passer par l'étape 1 qui est le pré-traitement,le nettoyage et l'étude du DataSet (**Missing-value** et **normalisation**), Aussi si la classe est numérique passer par la **discrétisation** (voir ANNEXE A).
- Comme le graphe ci-dessous l'illustre: le temps d'exécution est proportionnel à la taille du dataset (nombre d'instances/ nombre d'attributs), c'est-à-dire, plus le nombre d'instance est grand plus le temps d'exécution sera grand.

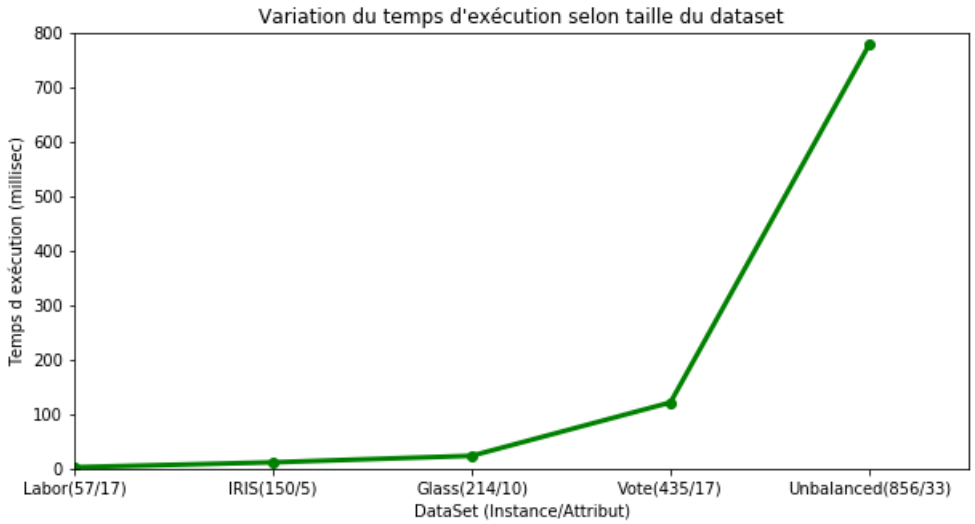


Figure 3.6: Variation du temps d'exécution de DBSCAN selon la taille du DataSet

- Le temps d'exécution reste assez réduit même pour de grand DataSets, nous expliquons cela par la complexité de dbscan qui est en $O(n^2)$ comme cité précédemment.
- Les clusters ne correspondent pas forcement à la classe car le clustering étudie les similarités et dissimilarités entre instances.

- Aussi nous avons remarqué que le paramètre **Epsilon** influe sur le nombre de cluster, tel que: Plus on augmente la valeur d'**Epsilon** plus il y aura de clusters.

  Et le paramètre **MinPts** quant à lui influe sur la densité des clusters, tel que: Plus **MinPts** sera pris petit plus les cluster seront dense.

  Ainsi si l'on prend **Epsilon** assez petit et **MinPts** assez grand nous obtiendrons plus d'instance classées comme du bruit.

- On peut exploiter le bruit (outliers) dans des datasets pour un objectif donné, car dbscan arrive à les détecter.

Conclusion

Suite aux nombreuses expérimentations sur nos Quatre algorithmes qui sont :

Apriori, **Fp-growth**, **K-nn** et **Dbscan**

Nous avons pu relever plusieurs informations et remarques intéressantes, nous commençons par les deux premiers algorithmes qui concernent:

l'extraction des items fréquents sur des dataSets transactionnels généralement , on a pu observer les points suivants sur **APRIORI**:

- Très simple à implémenter et à comprendre
- Face à grandes instances (dataset de plus de 200 transactions) il devient très long et prend beaucoup de temps à s'exécuter (sur nos machines) ceci est en relation direct avec le calcul intensif qu'apriori fait et qui fait sa signature qui est la jointure à chaque étape pour le calcul les itemset candidats, puis il repasse encore sur le dataset transactionnel pour calculer le support de chaque itemset candidats afin de filtrer ses derniers ceci augmente la complexité d'apriori et le rend ainsi Long.
- Efficace il arrive au résultat avec une bonne précision c'est à dire qu'il extrait exactement tous les items fréquents.

Il existe des versions d'apriori qui sont plus rapide et qui ont proposé une solution à sa lenteur comme :

Pour la jointure on code les itemset sur un tableau de taille des items on mettra 0 quand l'item n'apparait pas dans l'itemset et 1 sinon ainsi on pourra remplacer la jointure par :

- l'opérateur OU logique .(exemple : soit un dataset ayant que trois items distincts et deux items set de longeur deux un exemple de jointure entre les deux sera comme suit :$[i_1 : 0, i_2 : 1, i_3 : 1] OU [i_1 : 1, i_2 : 1, i_3 : 0] = [i_1 : 1, i_2 : 1, i_3 : 1]$)

Représentation du dataset verticalement (principe de la recherche d'information fichier inverse) c'est à dire pour chaque item va pointer vers un tableau de taille des transactions on mettera un 0 quand l'item n'apparaît pas dans la case de la transaction et 1 sinon ainsi pour le calcul de support d'un item set on fera un ET logique qui sera comme suit:

- l'opérateur Et logique .(exemple : soit un dataset ayant que trois Transactions distincts et deux items set de longueur deux un exemple de jointure entre les deux sera comme suit :$[t_1 : 0, t_2 : 1, t_3 : 1] ET [t_1 : 1, t_2 : 1, t_3 : 0] = [t_1 : 0, t_2 : 1, t_3 : 0]$) le support de l'itemset sera le nombre de 1 .

Ceci réduit considérablement le temps d'exécution . mais il existe un autre algorithme proposé par Han FP-growth dans lequel il s'est intéressé à une modélisation et implémentation complètement différente en exploitant les arbres.

Plus Concrètement, **FP-GROWTH** utilise de tout autres structures de données, il s'agit d'une représentation compressée du Dataset sous forme d'arbre (FP-Tree), ainsi sa construction ne nécessite que 2 scans du DataSet, puis la suite de l'exécution se fait en accédant uniquement au FP-Tree. Soient les points suivants nos principales remarques à ce sujet:

- Implémentation plus compliqué.
- Nette réduction du temps d'exécution par rapport à Apriori, même pour de grandes instances (DataSet) les temps d'exécution restent très raisonnables.
- C'est aussi un algorithme complet car il arrive toujours a extraire tous les items fréquents.

Classification supervisée et non supervisée Dans ce projet on eu affaire à deux algorithmes de classification :

- **K nearest neighbors** "K plus proche voisins" dans la classe des algorithmes de classification supervisé
- **DBscan** dans la classe des algorithmes de classification non supervisé

Dans la classification avec K-NN on peut noter les points essentiels suivants:

- L'étape du prétraitement est primordiale pour une bonne précision.
- c'est un algorithme de classification sensible au bruit principalement si on prend par exemple un nombre de voisin assez petit par exemple trois voisins et deux de ces voisins sont des bruits "mal classifié" on va alors classifier notre instance avec la mauvaise classe .
- K NN se base sur un principe simple (similarité) donc le choix des fonctions de distances influencera de manière considérable la classification .
- les données d'entraînement doivent couvrir toutes les classes potentiels et doivent être assez diverse ce qui implique plusieurs instances pour une classe sinon K nn sera incapable de prédire la classe de l'instance à classifier dans les données de testes car la classe n'existe pas dans les données d'entraînement.
- le temps d'exécution augmente de façon exponentielle quand nos données d'entraînements et de testes dépasse les 1000 instances , on peut expliquer ça par le fait que pour avoir les plus proches voisins on est obligé de passer par le tri (complexité quadratique) qui ralentit le processus de classification , ainsi que le calcul de distances entre l'instance à classifier et toutes les données d'entraînements.
- K est un paramètre empirique qu'il faut fixer lors des expérimentations pour une bonne classification choisir et qui dépend directement du nombre de classe de nos données à classifier .

Des améliorations par contre peuvent être suggérées comme :

On pourrait éviter le tri après le calcul de distances et utiliser à la place une liste de taille K triée généralement K n'est pas très grands comparer à la taille des données ,ainsi à chaque calcul de distance de l'instance avec les données d'entraînements:

1. si la distance qu'on est entrain de calculer est supérieur à toutes les instances de la liste alors on l'ajoute , on pourra comparer toujours avec le dernier car notre liste est triée (de k éléments).

2. sinon alors cette instance doit être ajouté en la remplaçant avec celle qui lui est ai supérieur en utilisant une fonction d'insertion dans une liste triée .

Dans le clustering avec DBSCAN nous avons constaté les points suivants:

- L'étape du pré-traitement (aussi bien le traitement des valeurs manquantes que la normalisation) est NON négligeable pour une bonne précision, sachant que la distance Epsilon dépend directement de la normalisation.

- Contrairement aux algorithmes de classification supervisée nous n'avons pas le nombre de cluster/class au préalable, ce qui permet de réellement clusteriser selon les similitudes et dissimilitudes des objets/instances.

- Par contre, pour DBSCAN étant basé densité nous avons 2 paramètres à fixer après expérimentation, qui sont Epsilon et MinPts,ceux là permettent de décider de la densité (concentration) des objets/instance dans les clusters.

- DBSCAN étant simple à implémenter, a en plus la particularité de ne pas être couteux en temps d'exécution, vu sa complexité.

- Cet algorithme est tout aussi efficace dans la détection d'outlier, car certaines études s'intéressent justement au valeurs aberrantes, ainsi Dbscan permet leur extraction.

- Le paramètre Epsilon étant une mesure de distance, dbscan tout comme K-nn dépend de la fonction de mesure de distance utilisée (euclidienne,manathan, ..)

Certaines améliorations peuvent être suggérées:

Il est à noter que les objets/instances d'un dataset n'ont pas tous la même densité, ainsi il se peut qu'en choisissant des valeurs de Epsilon et MinPts trop petites, certain clusters soient bien formés et d'autres absolument pas, ou meme que beaucoup d'instances soient classées comme bruit sans l'être.

- C'est pour cela qu'un solution serais de seuiller ses deux paramètres, par exemple avec un Min-Epsilon et Max-Epsilon, de même pour le nombre de voisins: Min-Pts et Max-Pts.

Comparons les trois catégories de techniques: Les 3 catégories d'algorithme de Data-Mining vu au cours de ce projet sont chacune spécifique à une catégorie de problème, tel que:

- Pour l'étude des motifs fréquents, des associations courantes des attributs d'un dataset, ou l'étude de dataset transactionnel, nous utilisons la $1^e re$ catégorie qui regroupe: **Apriori** et **FP-Growth**.
- Pour la classification d'objets/Instances dont on a connaissance des classes que nous souhaitons obtenir, et que nous avons à disposition des datasets de training, nous avons plus intérêt à utiliser les algorithmes de classification supervisées dont **K-NN**.
- Pour l'étude des similitudes et dissimilitudes entre objets/Instances, la détection d'outliers, ou la classification mais que nous ne possédons pas de dataset de training, nous nous dirigerons alors aux algorithmes de classification non-supérvisées (clustering) dont **DBSCAN**.

Ainsi nous concluons que chaque problème possède des méthodes / techniques qui lui sont le plus appropriées, c'est pour quoi il faut avoir connaissance de l'ensemble du problème sous tous ses aspects pour savoir vers quelles techniques se diriger, cependant dans une même catégorie il existe plusieurs algorithmes, qu'ils faut maitriser afin de choisir le plus adéquat selon les avantages et inconvénients de chacun.

ANNEXE A

Nettoyage et traitement du dataset avant clustering:n
(DataSet : Unbalanced : Missing-value + Normalisation)

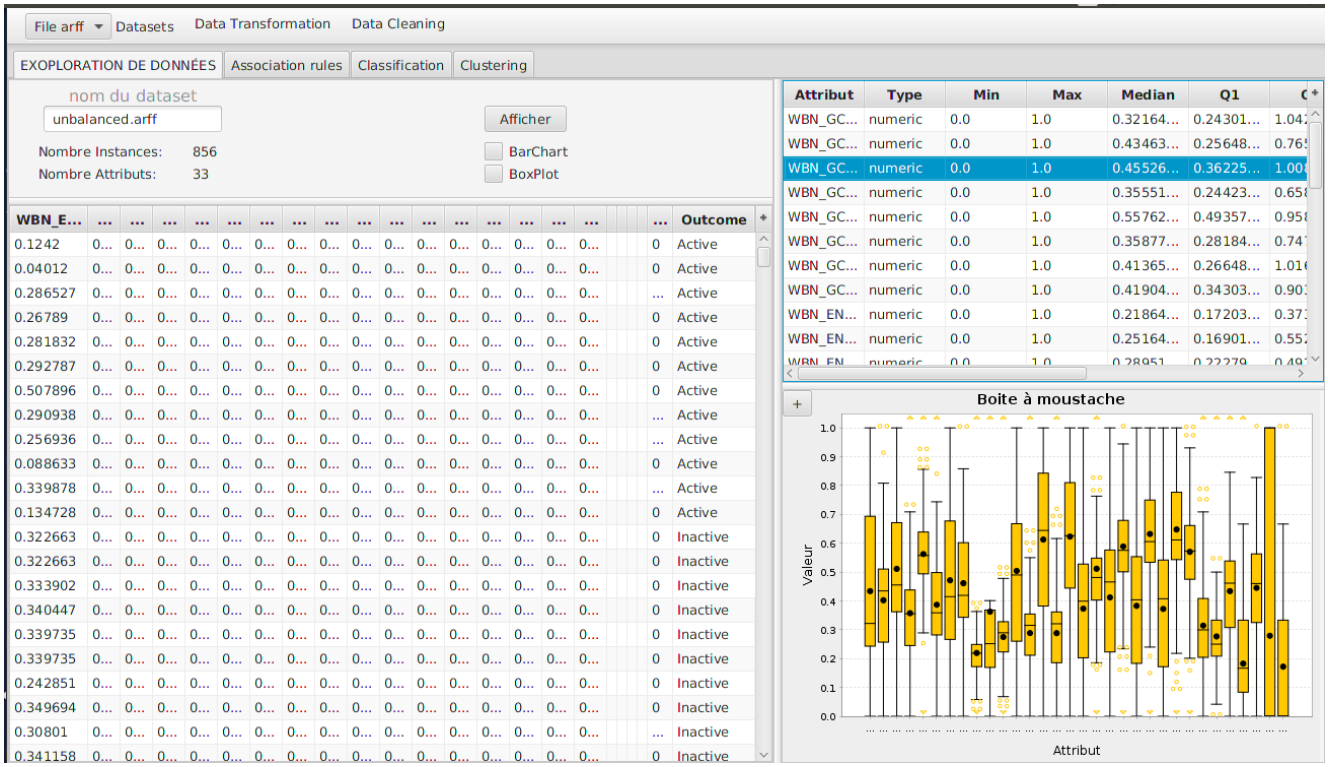


Figure 3.7: Nettoyage et traitement du dataset avant clustering

Références bibliographiques

[1] : http://www.cs.ccsu.edu/ markov/ccsu_courses/DataMining-1.html

[2] : Han's book : DataMining conceptis and technique third edition Morgan kaufmann.

[3] : https://fr.slideshare.net/ssuserab1db8/les-algorithmes-de-gnration-des-rgles-d-association

[4] : https://archive.ics.uci.edu/ml/datasets/Online+Retail