RAPPORT TP
DATA MINING

BOUCHALI NESMA HADIA

BAKOUR IMENE

MR NECIR

MASTER 1 IV

Juillet
2019

Table des matières

PRESENTATION DE JEU DE DONNEES

Onglet 1

Qu’est-ce qu’on a utilisé comme

Données ? :

- 340.723 cas d’accident
- 115 causes d’accident
- 134 tuples de test

PRÉ-TRAITEMENTS DES DONNEES

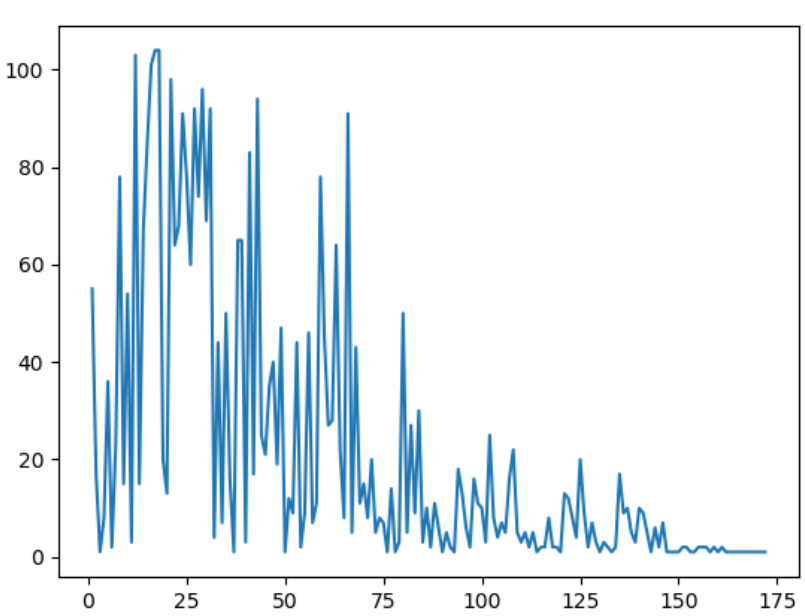
Nettoyage des données avant utilisation

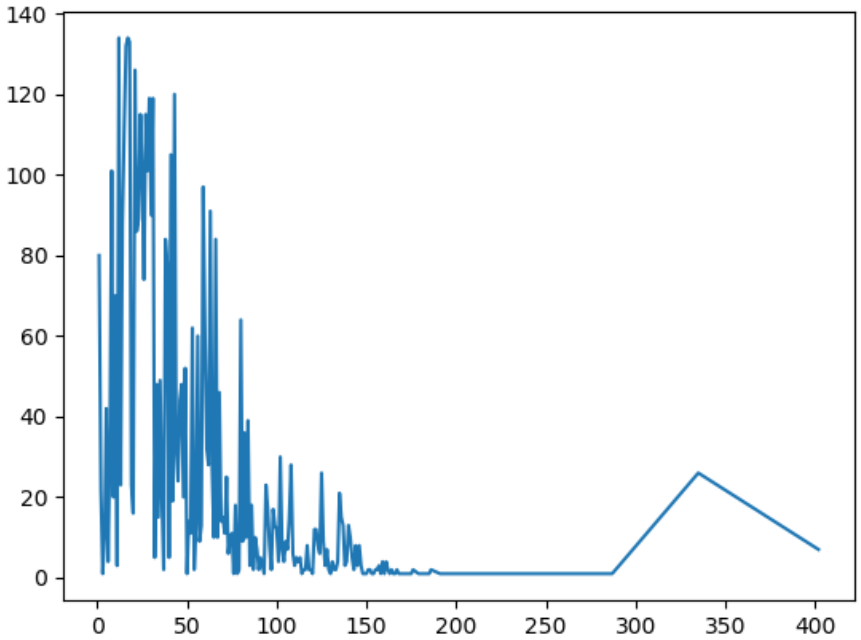
Comment ? :

- Détection des doublons
- Détection des erreurs de frappe
- Représentation significative des données

Onglet 2

Onglet 2





Représentation de quantités de données dans deux différentes parties de la base de données

MOTIVATION

- Trouver les item set fréquents
- Déterminer les causes des accidents en relation
- Optimisation

CHOIX DE LANGAGE DE PROGRAMMATION

Pour les traitement de données, ainsi que l'implémentation de l'algorithme Close

Pourquoi Python ? :

- Implémentation des fonctionnalités avec moins de code
- Meilleure gestion de données
- peut être exécuté sur pratiquement toutes les plates-formes

ENJEUX

Les problèmes et difficulté qu'on a trouvé durant la réalisation de ce projet

En briefe :

- Multiples passages sur la base de données
- Ensemble de candidats très importants
- Calcul du support d'un motif

REALISATION

Etapes de développement :

- Trouver les 1-itemsets fréquents, puis trouver les 2-itemsets fréquents ....

- Calcul du support : • Une passe sur la base de données pour compter le support de tous les itemsets pertinents.

REALISATION

Choix du MinSup :

- A fin d'automatiser le processus d'élimination des items sets non fréquents le min sup a été automatisée

- Min Sup = le centre de gravité de l'ensemble des support * 2

REALISATION

```python
#ELIMINATION DES ITEMSET NON FREQUENTS
def createPostSet(dataBase, maxItemId, minSuppRelative):
    postSet = []
    #print(dataBase, maxItemId, minSuppRelative)
    for i in range(1, maxItemId + 1):
        try:
            tidSet = dataBase[i]
        except:
            continue
        if(len(tidSet) >= minSuppRelative):
            postSet.append(i)
    return np.array(postSet)
```

Fonction qui permet d'eliminer les items non fréquents dans l'étape 01


```python
#Calcul du MinSup
def MinSup(vecteur, v):
    MinS = 0
    for i in vecteur.values():
        MinS = MinS+i
    return (MinS/v)*2
```

Fonction qui permet de calculer le min sup

IMPLEMENTATION

- Trie des itemsets fréquent selon les supports
    - Extraction des fermetures
- Calcul des intersections entre les items set et les fermetures
    - Calcul des supports
- Refaires les étapes 1 ,2,3,4 jusqu'à ce qu'on n'y plus de combainaisons interessantes

```python
def intersectTIdSet(tIdSet1, tIdSet2):
    newTIdSet = []
    if(len(tIdSet1) > len(tIdSet2)):
        for tId in tIdSet2:
            if(np.any(tIdSet1 == tId)):
                newTIdSet.append(tId)
    else:
        for tId in tIdSet1:
            if(np.any(tIdSet2 == tId)):
                newTIdSet.append(tId)
    return np.array(newTIdSet)
```

Fonction qui fait l'intersection entre les itemsets et les fermetures pour extraire les ieme itemset

```python
def sortPostSet(postSet, dataBase):
    for i in range(len(postSet)):
        minimum = i
        for j in range(i + 1, len(postSet)):
            if(isSmallerAccordingToSupport(postSet[minimum], postSet[j], dataBase)):
                minimum = j
        postSet[minimum], postSet[i] = postSet[i], postSet[minimum]
    print(postSet)
    print(np.flip(postSet, 0))
    return np.flip(postSet, 0)
```

Fonction qui fait le tri des supports des itemset

```python
def dci_closed(isFirstTime, closedSet, closedSetTIds, postSet, preSet, minSupp, dataBase, f_out):
    for m in range(1, len(postSet)):
        i = postSet[m]
        newGenTIds = []

        if(isFirstTime):
            newGenTIds = dataBase[i]
        else:
            newGenTIds = intersectTIdSet(closedSetTIds, dataBase[i])

        if(len(newGenTIds) >= minSupp):
            newGen = np.append(closedSet, np.array([i]))

            if(isDup(newGenTIds, preSet, dataBase) == False):
                closedSetNew = np.array(newGen)

                if(isFirstTime):
                    closedNewTIds = dataBase[i]
                else:
                    closedNewTIds = np.array(newGenTIds)

                postSetNew = []

                for j in postSet:

                    if(isSmallerAccordingToSupport(i, j, dataBase)):

                        if(set(newGenTIds).issubset(dataBase[j])):
                            closedSetNew = np.append(closedSetNew, [j])

                            jTIds = dataBase[j]

                            closedNewTIds = intersectTIdSet(closedNewTIds, jTIds)
                        else:
                            postSetNew = np.append(postSetNew, [j])

                # print(numpy.sort(closedSetNew), len(closedNewTIds))
                f_out.write(' '.join(map(repr, closedSetNew.astype(int))) +  " #SUP: " + str(len(closedNewTIds)) + "\n")

                preSetNew = np.array(preSet)
                dci_closed(False, closedSetNew, closedNewTIds, postSetNew, preSetNew,  minSupp, dataBase, f_out)

                preSet = np.append(preSet, [i])
```

Fonction Close , qui extrait les combanaisons fréquents fermés des itemset

CALCUL DISTRIBUE

- Pouvoir évaluer les résultats en peu de temps

- Solution très intéressante et évidente pour le traitements des data émense

- Python nous fournit de differents outil de parallelisme

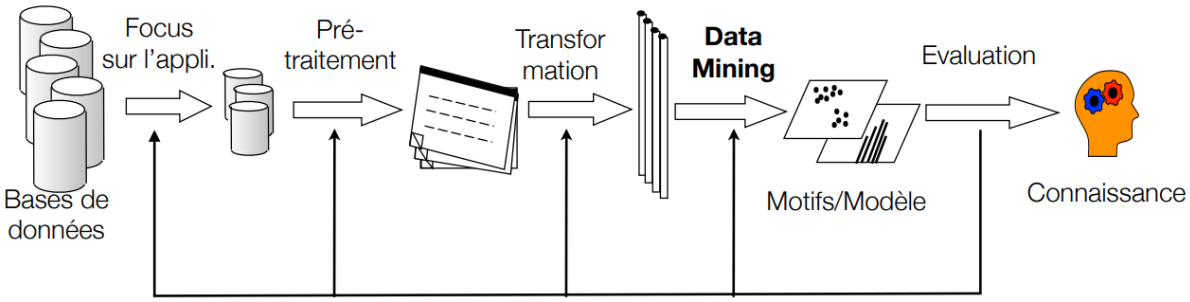


Schéma explicatif des étapes d'extraction des connaissances en dataMining

Les méthodes de calcul distribué travaillent sur les données et les taches, pour les données on a réfléchit a partitionner notre base de donnée en 2 partitions verticalement, et lancer 2 threads chacun pour extraire les informations supports et vérifie les données fréquents.

De l'autre part pour le parallélisme des taches, la meilleure solution sera d'implémenter un des modèles distribués de l'algorithme Close.

Pour le code on a réussi a implémenter notre première solution, mais la deuxième non pour l'instant malheureusement

```python
class Support_computing(Thread):
    def __init__(self, support, file):
        Thread.__init__(self)
        self.support = support
        self.file = file


    def run(self):
        sp.Sup(self.file, self.support)
        attente = 0.2
        attente += random.randint(1, 60) / 100
        time.sleep(attente)
def Calculer_vite():
    thread_1 = Support_computing(sup1,f_one)
    thread_2 = Support_computing(sup2,f_two)

    # Lancement des threads
    thread_1.start()
    thread_2.start()

    # Attend que les threads se terminent
    thread_1.join()
    thread_2.join()
```

Distribution des données

INTERFACE

Comment utiliser :

- Cliquer sur le buton ‘lancer le traitement’

- Visualiser les résultats en cliquant sur ‘voir le résultat’