RAPPORT DE TP

Module :

Fouille de données

Membres du binôme :

AKKOUCHE Adnane Aymen
SALHI Omar

Dans ce TP on nous a demandé d'implémenter l'algorithme **CLOSE** qui trouve l'ensemble d'itemset fréquents. Dans ce qui suit on utilise le dataset weather dans un notebook jupyter (python).

Lecture des données

Comme dit en page de garde on utilisera la dataset weather pour cette démonstration donc on commencera par l'ouvrir.

Entrée [2]:
```
import pandas as pd
import numpy as np
import itertools
```

Entrée [3]:
```
data = pd.read_csv('weather.csv')
```

Entrée [4]:
```
data.head()
```

Out[4]:

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	mild	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes

Pré-traitement

Contrairement aux données avec lesquelles on a travaillé en cours celles-ci ne sont pas binaires, on va donc modifier nos données pour obtenir des résultats semblables à l'emploi de variables muettes. Par exemple la colonne **outlook**, dans notre fonction ***binariser*** elle sera éclatée en trois ; **outlooksunny outlookovercast outlookrainy**

Entrée [5]:
```
def concateneListe(lst,mot):
    for i in range(len(lst)):
        lst[i]= mot+str(lst[i])
    return lst
def binariser(data):
    datbin = pd.DataFrame()
    incompatible = {}
    for col in data.columns:
        for val in data.loc[:,col].unique():
            datbin[col+str(val)] = (data.loc[:,col]==val)
            incompatible[col+str(val)] = concateneListe(data.loc[:,col].unique().tolist(),col)
    return datbin,incompatible
```

La fonction ***binariser*** renvoie donc les données binarisées ainsi qu'un dictionnaire contenant les colonnes et leurs incompatibilités dont on aura besoin plus tard lors de la génération des candidats pour l'itération suivante ça nous servira a élaguer les itemset impossibles comme **[outlooksunny outlookrainy]**.

La fonction ***concateneListe*** ne fait que concaténer touts les mots d'une liste avec un même mot.

Entrée [6]:
```
databin,incompat = binariser(data)
databin.head()
```

Out[6]:

	outlooksunny	outlookovercast	outlookrainy	temperaturehot	temperaturemild	temperaturecool	humidityhigh	humiditynormal	windyFalse	windyTrue	playno
0	True	False	False	True	False	False	True	False	True	False	True
1	True	False	False	True	False	False	True	False	False	True	True
2	False	True	False	False	True	False	True	False	True	False	False
3	False	False	True	False	True	False	True	False	True	False	False
4	False	False	True	False	False	True	False	True	True	False	False

Fonctions appelées dans CLOSE

Il nous faut l'implémentation des métriques vues en cours, plus spécifiquement le support surtout.

Entrée [7]:
```
def support(databin,itemset):
    arr = np.array(len(databin)*[True])
    for item in itemset:
        arr = np.logical_and(arr,databin.loc[:,item])
    return arr.sum()/len(databin)
def confiance(databin,A,B):
    return support(databin,A+B)/support(databin,A)
def lift(databin,A,B):
    return support(databin,A+B)/(support(databin,A)*support(databin,B))
```

Il nous faut aussi une fonction qui nous renvoie les itemsets candidats a chaque itération suivante. On fait ça en faisant le produit cartésien des candidats de l'étape précédente retenus avec toutes les colonnes du dataset puis filtrer les résultats incompatibles tout en gardant un unique exemplaire la ou il y a des doublons.

Entrée [8]:
```
def product(left,databin):
    t = itertools.product(left,databin.columns.tolist())
    res = []
    for i in t:
        res = res + [i[0]+[i[1]]]
    return res
```

Entrée [9]:
```
def prodCart(databin,incompat,left,deg):
    b = False
    res = []
    for i in product(left,databin):
        for d in range(deg-1):
            b = False
            temp = i[d]
            i = list(i)
            i[d] = ''
            if(len(list(set(i) & set(incompat[temp])))!=0):
                b = True
                break;
            i[d] = temp
        if(not b):
            if(set(i) not in res):
                res = res + [set(i)]
    return res
```

La fonction ***product*** nous fait un produit cartésien brut tandis que la fonction ***prodCart*** itère à travers les résultats de la première et avec des **if** ne garde que :

1. Les intersections entre l'itemset et toutes les incompatibilités de ses élements sont vides.
2. Ce n'est pas un itemset redondant.

Enfin il nous faut une fonction qui trouve la fermeture d'un itemset.

Entrée [10]:
```
def fermeture(databin,itemset):
    arr = np.array(len(databin)*[True])
    for item in itemset:
        arr = np.logical_and(arr,databin.loc[:,item])
    ferm = []
    for item in databin.columns:
        if(np.logical_and(arr,databin.loc[:,item]).sum()==arr.sum()):
            ferm = ferm + [item]
    return set(ferm)
```

Implémentation de CLOSE

Entrée [11]:
```
def CLOSE(data,minsup):
    databin,incompat = binariser(data)
    #Initialisation
    candidats = []
    for item in databin.columns:
        if(support(databin,[item])>=minsup):
            candidats = candidats + [([item],support(databin,[item]),fermeture(databin,[item]))]
    #La boucle
    ferm = []
    left = []
    regles = []
    deg = 2
    while(len(candidats)!=0):
        cand = []
        while(len(candidats)!=0):
            c = candidats.pop(0)
            s = support(databin,c[0])
            if(s>=minsup):
                left = left + [c[0]]
                c = list(c)
                c[1] = s; c[2]=fermeture(databin,c[0])
                ferm = ferm + [c[2]]
                c = tuple(c)
                cand = cand + [c]
        regles = regles + cand
        #Iteration suivante
        print("iteration")
        for i in prodCart(databin,incompat,left,deg):
            if(i not in ferm):
                candidats = candidats + [(list(i),0,[])]
        deg = deg + 1
        left = []
    return(regles)
```

Et voici le résultat final. La variable *deg* est le degré des itemsets recherchés, la liste *candidats* va contenir à chaque itération les itemsets sur lesquels travailler, *ferm* va contenir toutes les fermetures des itemsets retenus, *left* va contenir les itemsets retenus à l'itération actuelle et *regles* va contenir les résultat de l'algorithme, un ensemble d'itemsets avec leurs supports et leurs fermetures. Appliqué à notre dataset on obtient ceci.

Entrée [12]:
```
CLOSE(data,0.1)
```

```
iteration
```

Out[12]:
```
[(['outlooksunny'], 0.35714285714285715, {'outlooksunny'}),
 (['outlookovercast'], 0.2857142857142857, {'outlookovercast', 'playyes'}),
 (['outlookrainy'], 0.35714285714285715, {'outlookrainy'}),
 (['temperaturehot'],
  0.14285714285714285,
  {'humidityhigh', 'outlooksunny', 'playno', 'temperaturehot'}),
 (['temperaturemild'], 0.5714285714285714, {'temperaturemild'}),
 (['temperaturecool'],
  0.2857142857142857,
  {'humiditynormal', 'temperaturecool'}),
 (['humidityhigh'], 0.5, {'humidityhigh'}),
 (['humiditynormal'], 0.5, {'humiditynormal'}),
 (['windyFalse'], 0.5714285714285714, {'windyFalse'}),
 (['windyTrue'], 0.42857142857142855, {'windyTrue'}),
 (['playno'], 0.35714285714285715, {'playno'}),
 (['playyes'], 0.6428571428571429, {'playyes'}),
 (['outlooksunny', 'temperaturehot'],
  0.14285714285714285,
```