Aix-Marseille Université

TP
Règles d'association et Motifs fréquents

Master 2 Informatique - Science et Ingénierie des Données

Nous allons utiliser le langage Python et la bibliothèque MLxtend [1] (Machine Learning extensions) pour extraire les motifs fréquents et découvrir les règles d'association. Bien que les associations (motifs et règles) soient présentes dans de nombreux logiciels libres ou commerciaux, elles ne sont pas présentes dans la célèbre bibliothèque Scikit-learn.

Le premier exercice est une prise en main de MLxtend pour découvrir des règles d'association sur un exemple classique, l'analyse du ticket de caisse (ou analyse du panier de la ménagère).

Le deuxième exercice portera sur des données issues du naufrage du célèbre paquebot Titanic. Le but sera de caractériser les deux classes ("survivant" et "non-survivant"). Ce jeu de données est simple et permet, avant tout, de s'entraîner et de maîtriser les outils utilisés.

Le troisième exercice permettra de comparer expérimentalement deux algorithmes d'extraction de motifs fréquents.

Exercice 1. Tickets de caisse

Cet exercice a pour but de vous familiariser avec le bibliothèque MLxtend pour la découverte de règles d'association. Les étapes sont classiques : chargement et préparation des données, extraction des motifs fréquents, découverte des règles et post-traitement des règles (filtrage, sélection, ...).

1. **Chargement des données**

   Les données que nous allons utiliser sont déjà préparées. Le fichier "market-basket.csv" est à télécharger sur la page AMeTICE du cours ("Fouille de données (Associations et Motifs)").

   Chaque ligne correspond à une transaction (i.e., un panier/ticket) et chaque colonne à un item (i.e., un produit). Un "1" (ou "True") code la présence d'un item (i.e., produit acheté), un "0" ou "False" son absence (i.e., produit non acheté).

   Chargez le fichier de données grâce à la bibliothèque Pandas et `read_csv()`.

```
import pandas as pd
from os import chdir

chdir("mettre␣le␣chemin␣vers␣votre␣dossier␣de␣travail")

dataset = pd.read_csv('market_basket.csv', header=0)
#vérifications
print(dataset.shape) #(1360, 303)
print(dataset.columns)

# transformation en valeurs booléennes (étape non obligatoire mais
    recommandée par MLxtend)
for col in dataset.columns:
  dataset[col] = dataset[col].map({1 : True, 0 : False})
  dataset[col].astype("boolean")
```

1. https://github.com/rasbt/mlxtend

2. **Extraction des motifs fréquents**

   Pour extraire les motifs fréquents, nous importons `apriori()` et l'appliquons sur les données chargées précédemment, avec un support minimum (`min_support`) fixé à 0,025 (i.e., 2,5%, soit 34 transactions), une cardinalité max (`max_len`) pour un motif fixée à 4 items et une utilisation des noms d'item (`use_colnames`) à vrai.

   Nous remarquons que les résultats (l'ensemble des motifs fréquents) sont stockés dans un `DataFrame` et qu'un motif est stockée dans une `Serie`. Pour chaque motif, nous avons la valeur de son support en relatif (%) et les items qu'il contient. Notons que les items sont stockés dans un `frozenset`, i.e., un ensemble non modifiable (non-mutable).

```
from mlxtend.frequent_patterns import apriori

taillemax = 4
freq_itemsets = apriori(dataset, min_support=0.025, max_len=taillemax,
    use_colnames=True, verbose=0)

#type du résultat
type(freq_itemsets)

#nombre de motifs obtenus
print(freq_itemsets.shape) # (603, 2) ; 603 motifs fréquents

#liste des colonnes
print(freq_itemsets.columns)

#affichage des 15 premiers motifs
print(freq_itemsets.head(15))

#type de la colonne 'itemsets'
print(type(freq_itemsets.itemsets))

#affichage du premier élément
print(freq_itemsets.itemsets[0])
```

3. **Génération des règles d'association**

   Afin de produire les règles d'association, il faut importer `association_rules()`. Nous allons choisir la mesure (`metric`) de la confiance et fixer le seuil minimal (`min_threshold`) à 0,75 (i.e., 75%).

   Les règles obtenues sont stockées dans un `DataFrame`. Pour chaque règle, nous avons l'antécédent, le conséquent, le support de la règle, la confiance, le lift, le leverage et la conviction. Nous avons aussi le support de l'antécédent et celui du conséquent.

```
arules = association_rules(freq_itemsets, metric="confidence",
   min_threshold=0.7)

#type du résultat
print(type(arules))

#nombre de règles générées
print(arules.shape) # (50, 9)

#liste des colonnes
print(arules.columns)

#affichage des 5 premières règles du résultat
print(arules.iloc[:5,:])
```

4. **Post-traitement des règles découvertes**

   A présent, nous allons trier les règles selon une mesure, filtrer les règles selon une mesure, et sélectionner des règles en fonction de la présence de certains items dans l'antécédent et le conséquent.

   Par commodité, nous allons modifier les colonnes à afficher ainsi que le nombre de lignes et de colonnes à afficher.

```
my_rules = arules.loc[:,['antecedents', 'consequents', 'lift', '
    conviction']]
print(my_rules.shape)

#pour un affichage plus lisible
pd.set_option('display.max_columns', 6)
pd.set_option('display.max_rows', 10)
pd.set_option('display.precision', 3)

# sélection des colonnes à afficher
my_rules = arules.loc[:,['antecedents', 'consequents', 'lift', '
    conviction']]
print(my_rules.shape)

#affichage des 5 premières règles
print(my_rules[:10])

#affichage des règles avec un lift supérieur ou égal à 7
print(my_rules[my_rules['lift'].ge(7.0)])

#trier les règles selon le lift (ordre décroissant) et afficher les 10
    meilleures règles
print(my_rules.sort_values(by='lift', ascending=False)[:10])

#affichage des règles contenant 'Eggs' dans le conséquent
print(my_rules[my_rules['consequents'].ge({'Eggs'})])

#affichage des règles où le conséquent est exactement 'Eggs'
print(my_rules[my_rules['consequents'].eq({'Eggs'})])

#idem mais en triant selon le lift (ordre décroissant)
print(my_rules[my_rules['consequents'].eq({'Eggs'})].sort_values(by='
    lift', ascending=False))

#affichage des règles avec '2pct_Milk' contenu dans l'antécédent et '
    Eggs' correspondant au conséquent
print(my_rules[my_rules['antecedents'].ge({'2pct_Milk'}) & my_rules['
    consequents'].eq({'Eggs'})].sort_values(by='lift', ascending=False
    ))
```

5. **À vous de jouer**

   Les règles que nous avons ne sont pas vraiment très intéressantes. La valeur du support minimum est peut-être trop élevée.

   Déterminez les règles d'association avec cette fois $minsup$=0,015 et $minconf$=0,7 (la cardinalité max reste fixée à 4). On a alors 4 759 motifs et 3 265 règles.
   Remarque : vous pouvez essayer avec d'autres valeurs de $minsup$ et de $minconf$.

   Trouvez quelques règles qui vous semblent intéressantes.

   Avec la (les) règle(s) que vous avez jugée(s) la (les) plus intéressante(s), que suggérez-vous au gérant du magasin où les clients ont effectué tous ces achats ?

Exercice 2. Titanic

Le RMS Titanic est un paquebot transatlantique britannique qui fait naufrage dans l'océan Atlantique Nord en 1912 à la suite d'une collision avec un iceberg, lors de son voyage inaugural de Southampton à New York. C'est l'une des plus grandes catastrophes maritimes survenues en temps de paix et la plus grande pour l'époque. (Source : Wikipédia)

Nous disposons de données sur les passagers (âge, sexe, ...) et leur devenir (survivant ou non au naufrage). Le fichier "titanic3.xls" est à télécharger sur la page AMeTICE du cours.

Vous allez utiliser les règles d'association pour caractériser les deux classes (survivant et non-survivant) en fonction des autres attributs.

1. **Chargement des données**

```
import pandas as pd
from os import chdir

chdir("mettre le chemin vers votre dossier de travail")

df = pd.read_excel('titanic3.xls')

#vérification des données chargées
print(df.shape)
print(df.columns)
print(df.dtypes)
print(df.head())
...
```

2. **Préparation des données**

Pour préparer les données, nous allons sélectionner certains attributs ("pclass", "survived", "sex" et "age"), ignorer les exemples ayant une ou plusieurs valeurs manquantes, discrétiser l'âge (en deux), renommer des valeurs et changer le type des colonnes (en "category"). Enfin, nous mettrons les données préparées au bon format pour MLxtend.

```
#on enlève les attributs non voulus
df1 = df.drop(columns = ['name', 'ticket', 'home.dest', 'cabin', 'boat
    ', 'body', 'sibsp', 'parch', 'fare', 'embarked'])

print(df1.shape)
print(df1.columns)
print(df1.head())

#ignorer les exemples où l'âge est inconnu
print(df1['age'].isnull().sum())

df2 = df1.dropna()
print(df2.shape)
print(df2.head())

#reste-t-il des valeurs manquantes ?
print(df2.isnull().values.any())

#discrétisation de l'âge
data = df2['age']
agemin = data.min()
agemax = data.max()
datad = pd.cut(data, bins=[agemin, 18, agemax],
                labels=['child', 'adult'], include_lowest = True)
#ici, au lieu d'indiquer le nombre d'intervalles (par exemple, "bins
    =2"), on donne une liste de valeurs. Afin d'inclure le minimum, on
     ajoute "include_lowest=True".
```

```
print(datad.value_counts())

df2.insert(4, "aged", datad, True)
print(df2.head())
df2 = df2.drop(columns = ['age'])
df2 = df2.rename(columns={'aged': 'age'})
print(df2.head())

#renommage de valeurs
df2['pclass'] = df2['pclass'].map({1:'class=1st', 2:'class=2nd', 3:'
    class=3rd'})
df2['survived'] = df2['survived'].map({1:'survived=yes', 0:'survived=
    no'})
df2['sex'] = df2['sex'].map({'female':'sex=female', 'male':'sex=male'
    })
df2['age'] = df2['age'].map({'adult':'age=adult', 'child':'age=child'
    })

#changement de types
print(df2.dtypes)
df2['pclass'] = df2['pclass'].astype('category')
df2['survived'] = df2['survived'].astype('category')
df2['sex'] = df2['sex'].astype('category')
print(df2.dtypes)
print(df2.head())

#(facultatif) enregistrement des données préparées dans un fichier
df2.to_csv('titanic.csv', header=True, index=False)

#les données sont presque prêtes, il faut maintenant les mettre dans
    le format accepté par MLxtend
dataset = df2.values
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
dataset = pd.DataFrame(te_ary, columns=te.columns_)
#print(dataset)
```

3. **Découverte de règles d'association**

   Calculez les motifs fréquents avec $minsup$ fixé à 0,5.

   Générez les règles d'association avec $minconf$ fixé 0,8.

4. **Post-traitement et interprétation des résultats**

   Seulement deux règles sont produites :

   $sex = male \rightarrow age = adult$ et $survived = no \rightarrow sex = male$

   Calculez à la main le support, la confiance et le lift de la première règle. Cette règle vous semble-t-elle intéressante ?

   Que pensez-vous de la deuxième règle ?

5. **À vous de jouer**

   Continuez les expérimentations en faisant varier $minsup$ et $minconf$ afin de caractériser les survivants et les non-survivants.

   Remarques :
   - Si $minsup$ n'est pas assez bas, les enfants ne seront pas considérés.
   - Les membres d'équipage ("crew") ne sont pas présents dans nos données. Notez qu'en général, ils n'ont pas survécu.

Exercice 3. ***Apriori* vs. *FP-Growth***

Le but de cet exercice est de comparer expérimentalement les implémentations des algorithmes *Apriori* et *FP-Growth* de MLxtend sur deux jeux de données "benchmarks" : `mushroom` et `retail`. Ces derniers sont téléchargeables sur la page AMeTICE du cours.

1. Pour chacun des deux jeux de données, pour différentes valeurs de $minsup$, mesurez le temps d'exécution de chaque algorithme. Relevez également le nombre de motifs fréquents découverts. Mémorisez les résultats afin de les visualiser sous forme texte ou sous forme graphique (en utilisant, par exemple, Matplotlib).
2. Quel algorithme vous semble le plus performant sur `mushroom` ? Sur `retail` ? Y a-t-il une raison à cela selon vous ?

Remarques : vous allez devoir utiliser la format "sparse" (i.e., éparse) de MLxtend. Cette représentation permet d'économiser la mémoire lorsque les données contiennent beaucoup d'items. Pour mesurer le temps d'exécution pour pouvez utiliser `time`, par exemple.

Le code suivant pourra vous être utile :

```
import pandas as pd
import matplotlib.pyplot as plt

from os import chdir
from time import time

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import fpgrowth

chdir("mettre␣le␣chemin␣vers␣votre␣dossier␣de␣travail")

datasetname = 'mushroom.dat'

#chargement à partir d'un fichier texte
dataset = []
file = open(datasetname, "r")
line = file.readline()
while line:
    line = line.replace('␣\n','')
    dataset.append(line.split('␣'))
    line = file.readline()
file.close()

#données éparses (sparse dataset)
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset, sparse=True)
df = pd.DataFrame.sparse.from_spmatrix(te_ary, columns=te.columns_)
print(df.shape)

...

start = time()
...
end = time()
tempsexecution = end - start

...
```