1 Règles d'association

Scikit learn est très pauvre pour la recherche de patterns. Il existe cependant quelques librairies qui ont été développées. La recherche de patterns nécessite souvent des algorithmes très performants pour trouver les résultats. Il est donc courant de devoir passer par une API vers un code développé en C++ par exemple. Nous présentons ici deux approches différentes.
La première utilise une librairie développée en Python qui permet de pouvoir extraire des itemsets et des règles d'association. Il s'agit de la librairie MLxtend : http://rasbt.github.io/mlxtend/ (http://rasbt.github.io/mlxtend/).

La seconde utilise une implémentation d'un algorithme de recherche de règles d'association développé en Java. Il s'agit de SPMF. SPMF propose un très grand choix d'algorithmes pour extraire des règles d'association, des motifs séquentiels, des sous graphes. Il propose également un grand nombre de jeu de données. Il est très utilisé dans la communauté Data Mining : http://www.philippe-fournier-viger.com/spmf/ (http://www.philippe-fournier-viger.com/spmf/).

Rappel de la problématique de la recherche de règles d'association

Soit $\mathrm{I} = \{i_1, i_2, \ldots, i_m\}$ un ensemble d'items. Soit $\mathrm{T} = \{t_1, t_2, \ldots, t_n\}$ un ensemble de transactions, telles que t_i soit un sous-ensemble de I (i.e. $t_i \subseteq I$). Une règle d'association s'exprime sous la forme : $\mathrm{X} \rightarrow \mathrm{Y}$, où $\mathrm{X} \in \mathrm{T}, \mathrm{Y} \in \mathrm{T}$, et $\mathrm{X} \cap \mathrm{Y} \neq \emptyset$.

Initialement la problématique de la recherche de règles d'association etait définie par : A partir d'une base de données de transactions, l'objectif est d'extraire l'ensemble des règles qui sont telles que le support de la règle est $\geq$ support_minimal et la confiance (*confidence*) est $\geq$ confiance_minimale où support_minimal confiance_minimale sont deux paramètres définis par l'utilisateur final. Cependant au cours des années différentes mesures ont été proposées notamment pour remplacer la confiance.

Le *Support* d'une règle $\mathrm{X} \rightarrow \mathrm{Y}$, noté $Support(\mathrm{X}, \mathrm{Y})$ indique la fréquence de la règle dans les transactions, i.e. combien de fois les items X et Y apparaissent ensemble, i.e $Support(\mathrm{X} \cup \mathrm{Y})$.

La *Confidence* d'une règle $\mathrm{X} \rightarrow \mathrm{Y}$ est définie par : $Confidence(\mathrm{X} \rightarrow \mathrm{Y}) = p(Y|X) = \frac{support(X, Y)}{suppport(X)}$. Range = [0,1]

Différentes mesures ont été proposées dans la littérature.

$Lift(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{p(Y|X)}{p(Y)} = \frac{suppport(X, Y)}{suppport(X) \times suppport(Y)} = \frac{Confidence(\mathrm{X} \rightarrow \mathrm{Y})}{support(X)}$.
Range [0,∞]

Si des règles ont un lift de 1 cela voudrait dire que la probabilité de l'antécédent et du conséquent sont indépendantes l'une de l'autres. Quand deux événements sont indépendants il est donc difficile d'en tirer une règle. Si le lift est supérieur à 1 cela montre que les deux parties sont dépendantes et qu'une règle est peut être utile pour prédire le conséquent à partir de l'antécédent.

$Leverage(\mathrm{X} \rightarrow \mathrm{Y}) = support(X, Y) - support(A) \times support(C)$. Range = [-1,1]

Une valeur de leverage nulle indique que X et Y sont indépendantes.

$Conviction(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{1 - suppport(Y)}{1 - Confidence(\mathrm{X} \rightarrow \mathrm{Y})}$. Range [-1,1]

Une valeur de conviction de 1 indique que les événements sont indépendants.

1.1 Utilisation de MLxtend

Dans l'exemple nous considérons l'exercice fait en cours. Les données sont organisées de la manière suivante :
Chaque ligne correspond à un client, les items achetés sont précisés.

In [1]:

```
basket = [['Biscuit', 'Creme', 'Couches',
           'Pain', 'Confiture'],
          ['Oeufs', 'Salade', 'Pain'],
          ['Pain', 'Biscuit', 'Couches',
           'Lait','Camembert','Biere','Poudre de cacao'],
          ['Biere', 'Lait', 'Creme',
           'Pain', 'Salade','Couches','Poudre de cacao'],
          ['Camembert', 'Confiture',
           'Poudre de cacao', 'Lait'],
          ['Lait','Biere','Perrier',
           'Oeufs','Couches'],
          ['Biscuit', 'Creme', 'Oeufs',
           'Pain', 'Couches','Lait',
           'Poudre de cacao', 'Confiture'],
          ['Camembert','Biere','Perrier',
           'Oeufs','Salade','Confiture','Couches'],
          ['Salade','Confiture','Pain','Oeufs',
           'Creme','Perrier','Biscuit']]

```

Il faut à présent transformer les données pour qu'elles correspondent à une matrice où les items sont les colonnes, les clients les lignes et faire apparaître un 0 ou un 1 lorsqu'un client a acheté ou pas un item.

mlxtend propose une fonction TransactionEncoder pour réaliser cette opération.

In [2]:

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
te_ary = te.fit(basket).transform(basket)
df = pd.DataFrame(te_ary, columns=te.columns_)

```

Pour connaître l'ensemble des itemsets fréquents il suffit d'appeler l'algorithme apriori.

In [3]:

```
from mlxtend.frequent_patterns import apriori

frequent_itemsets = apriori(df, min_support=0.3, use_colnar

display(frequent_itemsets.head(10))
```

	support	itemsets
0	0.444444	(Biere)
1	0.444444	(Biscuit)
2	0.333333	(Camembert)
3	0.555556	(Confiture)
4	0.666667	(Couches)
5	0.444444	(Creme)
6	0.555556	(Lait)
7	0.555556	(Oeufs)
8	0.666667	(Pain)
9	0.333333	(Perrier)

L'obtention des règles se fait par la fonction association rules. Il est possible de spécifier différentes metrique (Cf. remarque précédente).

In [4]:

```
from mlxtend.frequent_patterns import association_rules
print ('Utilisation de la confiance\n')
rules = association_rules(frequent_itemsets,
                          metric="confidence",
                          min_threshold=0.2)

display(rules.sample(10))
```

```
Utilisation de la confiance
```

	antecedents	consequents	antecedent support	consequent support	support	con
105	(Couches)	(Poudre de cacao, Pain)	0.666667	0.333333	0.333333	
117	(Creme, Pain)	(Confiture, Biscuit)	0.444444	0.333333	0.333333	
59	(Confiture, Biscuit)	(Pain)	0.333333	0.666667	0.333333	
74	(Pain)	(Creme, Biscuit)	0.666667	0.333333	0.333333	
40	(Salade)	(Oeufs)	0.444444	0.555556	0.333333	
98	(Couches)	(Poudre de cacao, Lait)	0.666667	0.444444	0.333333	
112	(Pain, Creme, Confiture)	(Biscuit)	0.333333	0.444444	0.333333	
97	(Poudre de cacao)	(Lait, Couches)	0.444444	0.444444	0.333333	
113	(Creme, Confiture, Biscuit)	(Pain)	0.333333	0.666667	0.333333	
101	(Poudre de cacao, Couches)	(Pain)	0.333333	0.666667	0.333333	

In [5]:

```
print ('Utilisation du lift\n')
rules = association_rules(frequent_itemsets,
                          metric="lift")

display(rules.head(10))
```

```
Utilisation du lift
```

	antecedents	consequents	antecedent support	consequent support	support	confic
0	(Couches)	(Biere)	0.666667	0.444444	0.444444	0.6(
1	(Biere)	(Couches)	0.444444	0.666667	0.444444	1.0(
2	(Lait)	(Biere)	0.555556	0.444444	0.333333	0.6(
3	(Biere)	(Lait)	0.444444	0.555556	0.333333	0.7[
4	(Confiture)	(Biscuit)	0.555556	0.444444	0.333333	0.6(
5	(Biscuit)	(Confiture)	0.444444	0.555556	0.333333	0.7[
6	(Couches)	(Biscuit)	0.666667	0.444444	0.333333	0.5(
7	(Biscuit)	(Couches)	0.444444	0.666667	0.333333	0.7[
8	(Creme)	(Biscuit)	0.444444	0.444444	0.333333	0.7[
9	(Biscuit)	(Creme)	0.444444	0.444444	0.333333	0.7[

Le code suivant permet de pouvoir ajouter une colonne qui contient la taille de la partie antecedent pour ne pouvoir afficher que les règles qui ont un certain nombre d'items dans la partie antécédent.

In [6]:

```
rules["antecedent_len"] = rules["antecedents"].apply(lambda
```

Affichage des règles dont la partie antécédent est supérieure à 1.

In [7]:

```
display (rules.loc[(rules["antecedent_len"]>1)].head(10))
```

	antecedents	consequents	antecedent support	consequent support	support	confi
46	(Lait, Couches)	(Biere)	0.444444	0.444444	0.333333	
47	(Lait, Biere)	(Couches)	0.333333	0.666667	0.333333	
48	(Couches, Biere)	(Lait)	0.444444	0.555556	0.333333	
52	(Creme, Confiture)	(Biscuit)	0.333333	0.444444	0.333333	
53	(Creme, Biscuit)	(Confiture)	0.333333	0.555556	0.333333	
54	(Confiture, Biscuit)	(Creme)	0.333333	0.444444	0.333333	
58	(Pain, Confiture)	(Biscuit)	0.333333	0.444444	0.333333	
59	(Confiture, Biscuit)	(Pain)	0.333333	0.666667	0.333333	
60	(Pain, Biscuit)	(Confiture)	0.444444	0.555556	0.333333	
64	(Pain, Biscuit)	(Couches)	0.444444	0.666667	0.333333	

1.2 Utilisation d'une API extérieure (SPMF)

Le code java de SPMF est disponible ici :
http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php
(http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php)
Il faut le télécharger et sauvegarder le fichier jar dans le répertoire courant.

Comme vous pouvez le constater il existe un très grand nombre d'algorithmes de recherche de règles d'association disponibles. Nous allons utiliser ici FP_Growth.

Pour obtenir les itemsets : FPGrowth_itemsets
Pour obtenir les règles d'association : FPGrowth_association_rules
Pour obtenir les règles d'association avec le lift :
FPGrowth_association_rules_with_lift

Dans l'exemple nous utiliserons le dernier. Se reporter à la page de documentation pour connaître les différents paramètres.

L'appel se fait simplement via :
java -jar spmf.jar run FPGrowth_association_rules_with_lift
NomduFichierFormatSPMF Sortie.txt support% confiance% valeurlift"

Dans l'exemple nous prenons le fichier contextIGB.txt, support 50%, confiance 90%, lift 1.2.

In [8]:

```
1  import os
2
3  os.system("java -jar spmf.jar run FPGrowth_association_rule
4
5
6
```

Out[8]:

0

In [9]:

```
#Affichage des résultats
f=open("output.txt")
for ln in f:
    display (ln)
f.close()
```

'1 ==> 5 #SUP: 4 #CONF: 1.0 #LIFT: 1.2\n'

'1 2 ==> 5 #SUP: 4 #CONF: 1.0 #LIFT: 1.2\n'

'4 5 ==> 1 #SUP: 3 #CONF: 1.0 #LIFT: 1.5\n'

'1 4 ==> 5 #SUP: 3 #CONF: 1.0 #LIFT: 1.2\n'

'2 4 5 ==> 1 #SUP: 3 #CONF: 1.0 #LIFT: 1.5\n'

'1 2 4 ==> 5 #SUP: 3 #CONF: 1.0 #LIFT: 1.2\n'