Generate and Visualize Association Rules

This example is found in **this page (http://intelligentonlinetools.com/blog/2018/02/10/how-to-create-data-visualization-for-association-rules-in-data-mining/)**

Association rule learning is used in machine learning for discovering interesting relations between variables. Apriori algorithm is a popular algorithm for association rules mining and extracting frequent itemsets with applications in association rule learning. It has been designed to operate on databases containing transactions, such as purchases by customers of a store (market basket analysis).
Besides market basket analysis this algorithm can be applied to other problems. For example in web user navigation domain we can search for rules like customer who visited web page A and page B also visited page C.

Python sklearn library does not have Apriori algorithm. **MLxtend (http://rasbt.github.io/mlxtend/)** has modules for different tasks. This example shows how to create data visualization for association rules in data mining using MLxtend for getting association rules and **NetworkX (https://networkx.github.io/documentation/stable/)** module for charting the diagram.

MLxtend is installed via command line with

```
conda install -c conda-forge mlxtend
```

Some bugs and deprecated solutions have been corrected by Claudio Sartori

- TransactionEncoder instead of OneHotTransaction
- .values instead of .as_matrix
- antecedent instead of antecedant
- in graph for consequents G1.add_nodes_from needed the parameter [c]

MLxtend for association rules

- **TransactionEncoder**: for the computation of the association rules with MLxtend, it is required that the data are provided as a dataframe in *one hot* representation; this function does the trick
  - the rows correspond to transactions, the columns correspond to items and have boolean values, true if the transaction contains the item corresponding to the column
  - according to the standards of **sklearn** the *TransactionEncoder* needs to be initialized, then *fit*, and then the fitted encoder is used to *transform* the dataset; the dataset can also be a list of lists, each component list being a transaction, composed by items (non-repeating)

MLxtend for association rules (cont)

- **apriori**: generates a dataframe with the frequent itemsets, taking as input the one-hot encoded dataframe and the minimum support threshold
  - the output dataframe has two columns: *support* and *itemsets* (the list of the items)
  - by default *itemsets* is a list of indexes, with _use*colnames*=*True* the list of items contains the column names, i.e. the item names

MLxtend for association rules (cont)

- **assiciation_rules**: generates a dataframe with the rules, in detail, the columns are:
  - antecedents (list)
  - consequents (list)
  - antecedent support
  - consequent support
  - support
  - confidence
  - lift
  - leverage
  - conviction

In [1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Prepare the environment
"""
Created on Tue Nov 27 12:28:04 2018
@author: http://intelligentonlinetools.com/blog/2018/02/10/how-to-create-data-visualization-for-associatio
n-rules-in-data-mining/
"""
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

In [2]:
```
# A toy dataset
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

The dataset is transformed in the *relational view* by fitting a **TransactionEncoder** and stored into a dataframe

In [3]:
```
# initialize the TransactionEncoder into "te", no parameters needed
te = TransactionEncoder()
# fit and transform the transaction encoder with dataset,
#.     store the result in te_ary and inspect its type
# te.fit(dataset)
# te_ary = te.transform(dataset)
te_ary=te.fit(dataset).transform(dataset)
type(te_ary)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
# transform te_ary into a dataframe and store it in df,
#.   the column names are taken from the columns_ attribute of "te"
df = pd.DataFrame(te_ary, columns=te.columns_)
print (df)
```

```
   Apple   Corn   Dill   Eggs  Ice cream  Kidney Beans   Milk  Nutmeg  Onion  \
0  False  False  False   True      False          True   True    True   True
1  False  False   True   True      False          True  False    True   True
2   True  False  False   True      False          True   True   False  False
3  False   True  False  False      False          True   True   False  False
4  False   True  False   True       True          True  False   False   True

   Unicorn  Yogurt
0    False    True
1    False    True
2    False   False
3     True    True
4    False   False
```

apriori is called to generate the frequent itemsets

In [5]:
```
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
print (frequent_itemsets)
```

```
    support                     itemsets
0       0.8                       (Eggs)
1       1.0               (Kidney Beans)
2       0.6                       (Milk)
3       0.6                      (Onion)
4       0.6                     (Yogurt)
5       0.8         (Eggs, Kidney Beans)
6       0.6                (Eggs, Onion)
7       0.6         (Milk, Kidney Beans)
8       0.6        (Onion, Kidney Beans)
9       0.6       (Kidney Beans, Yogurt)
10      0.6  (Eggs, Onion, Kidney Beans)
```

Then the association rules derived from the frequent itemsets are generated

In [6]:
```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
print (rules)
```

```
                antecedents            consequents  antecedent support  \
0                    (Eggs)         (Kidney Beans)                 0.8
1            (Kidney Beans)                 (Eggs)                 1.0
2                    (Eggs)                (Onion)                 0.8
3                   (Onion)                 (Eggs)                 0.6
4                    (Milk)         (Kidney Beans)                 0.6
5                   (Onion)         (Kidney Beans)                 0.6
6                  (Yogurt)         (Kidney Beans)                 0.6
7             (Eggs, Onion)         (Kidney Beans)                 0.6
8      (Eggs, Kidney Beans)                (Onion)                 0.8
9     (Onion, Kidney Beans)                 (Eggs)                 0.6
10                   (Eggs)  (Onion, Kidney Beans)                 0.8
11                  (Onion)   (Eggs, Kidney Beans)                 0.6

    consequent support  support  confidence  lift  leverage  conviction
0                  1.0      0.8        1.00  1.00      0.00         inf
1                  0.8      0.8        0.80  1.00      0.00         1.0
2                  0.6      0.6        0.75  1.25      0.12         1.6
3                  0.8      0.6        1.00  1.25      0.12         inf
4                  1.0      0.6        1.00  1.00      0.00         inf
5                  1.0      0.6        1.00  1.00      0.00         inf
6                  1.0      0.6        1.00  1.00      0.00         inf
7                  1.0      0.6        1.00  1.00      0.00         inf
8                  0.6      0.6        0.75  1.25      0.12         1.6
9                  0.8      0.6        1.00  1.25      0.12         inf
10                 0.6      0.6        0.75  1.25      0.12         1.6
11                 0.8      0.6        1.00  1.25      0.12         inf
```