Market Basket Analysis

YoungWoong Cho

August 2020

In this project, a market basket analysis is used as a recommendation algorithm. A priori algorithm is implemented in order to deduce the possible association rules from the dataset.

Dataset : Groceries Market Basket Dataset

Source : https://www.kaggle.com/irfanasrullah/groceries?select=groceries+-+groceries.csv

0.1 Preliminary works

1. Import libraries

[ ]:
```
import pandas as pd
import numpy as np
```

2. Get a dataset

[ ]:
```
from google.colab import files
data_to_load = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving market_basket.csv to market_basket.csv
```

[ ]:
```
data = pd.read_csv('market_basket.csv').iloc[:, 1:]

dataSet = []
for i in range(data.shape[0]):
  dataSet.append(data.iloc[i,:].dropna(axis='index').to_list())
dataSet[0]
```

[ ]:
```
['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups']
```

0.2 A priori algorithm

[ ]:
```
def createC1(dataSet): # Create frozen set for each item
  C1 = []
  for transaction in dataSet:
    for item in transaction:
```

```
      if [item] not in C1:
        C1.append([item])
        C1.sort()
  return list(map(frozenset, C1))

def scanD(D, Ck, minSupport): # Calculate support for itemsets
  ssCnt = {}
  for tid in D:
    for can in Ck:
      if can.issubset(tid):
        if can not in ssCnt: ssCnt[can]=1 # changed ssCnt.has_key(can) into can␣
 ↪not in ssCnt
        else: ssCnt[can] += 1
  numItems = float(len(D))
  retList = []
  supportData = {}
  for key in ssCnt:
    support = ssCnt[key]/numItems
    if support >= minSupport:
      retList.insert(0,key)
    supportData[key] = support
  return retList, supportData

def aprioriGen(Lk, k): #creates Ck
  """
  Input
    Lk : List of frequent itemsets
    k: size of the itemsets
  Output
    Ck : candidate itemsets with k elements(that meets )
  """
  retList = []
  lenLk = len(Lk)
  for i in range(lenLk):
    for j in range(i+1, lenLk):
      L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]
      L1.sort(); L2.sort()
      if L1==L2:
        retList.append(Lk[i] | Lk[j])
  return retList

def apriori(dataSet, minSupport):
  C1 = createC1(dataSet)
  D = list(map(set, dataSet))
  L1, supportData = scanD(D, C1, minSupport)
  L = [L1]
  k = 2
```

```
  while (len(L[k-2]) > 0):
    Ck = aprioriGen(L[k-2], k)
    Lk, supK = scanD(D, Ck, minSupport)
    supportData.update(supK)
    L.append(Lk)
    k += 1
  return L, supportData

def generateRules(L, supportData, minConf=0.7):
  bigRuleList = []
  for i in range(1, len(L)):
    for freqSet in L[i]:
      H1 = [frozenset([item]) for item in freqSet]
      if (i > 1):
        rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
      else:
        calcConf(freqSet, H1, supportData, bigRuleList, minConf)
  return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):
  prunedH = []
  for conseq in H:
    conf = supportData[freqSet]/supportData[freqSet-conseq]
    if conf >= minConf:
      print(freqSet-conseq,'-->',conseq,'conf:',conf)
      brl.append((freqSet-conseq, conseq, conf))
      prunedH.append(conseq)
  return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
  m = len(H[0])
  if (len(freqSet) > (m + 1)):
    Hmp1 = aprioriGen(H, m + 1)
    Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
    if (len(Hmp1) > 1):
      rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)
```

[ ]:
```
L,suppData=apriori(dataSet,minSupport=0.05)
rules=generateRules(L,suppData, minConf=0.3)
rules
```

```
frozenset({'rolls/buns'}) --> frozenset({'whole milk'}) conf:
0.30790491984521834
frozenset({'yogurt'}) --> frozenset({'whole milk'}) conf: 0.40160349854227406
frozenset({'other vegetables'}) --> frozenset({'whole milk'}) conf:
0.38675775091960063
```

```
[ ]: [(frozenset({'rolls/buns'}), frozenset({'whole milk'}), 0.30790491984521834),
      (frozenset({'yogurt'}), frozenset({'whole milk'}), 0.40160349854227406),
      (frozenset({'other vegetables'}),
       frozenset({'whole milk'}),
       0.38675775091960063)]
```