Lab2: Big Data

Frede Emnetu (100704524)

In [1]:
```
from apyori import apriori
import numpy as np
import pandas as pd
import matplotlib.pyplot as pl
import time
```

Output function

In [29]:
```
def output(pairs):
    for x in pairs:
        list_ = list(x)
        print('%d --> %d' % (list_[0],list_[1]))
```

Create combinations function

In [2]:
```
def createCk(Lk, k):
    cand_list = []
    len_Lk = len(Lk)
    for i in range(len_Lk):
        for j in range(i + 1, len_Lk):
            L1 = list(Lk[i])[:k - 2]
            L2 = list(Lk[j])[:k - 2]

            if L1 == L2:
                cand_list.append(Lk[i] | Lk[j])
#                 print(cand_list)
#         print("C",cand_list)
    return cand_list
```

Apyori function

In [3]:
```
def apyori(row, minsupport):
    counter = 0
    start = time.time()
#     percentChunk = 1
    min_support = minsupport
#     itemsets = itemsets[0:int(len(itemsets) * (sections)/100)]
    C1 = []
    for basket in row:
        for item in basket:
            if not [item] in C1:
                C1.append([item])
    C1 = [set(x) for x in C1]

    # print("C1 the first candidate list: ", C1)

    count = {}
    freq_item = []
    L1 = []

    for basket in row:
        for item in C1:
            if item.issubset(basket):
                candidate = frozenset(item)
                if candidate not in count:
                    count[candidate] = 1
                else:
                    count[candidate] += 1

    for key in count:
        support = count[key] / len(row)
        if support >= min_support:
            freq_item.insert(0, key)
            freq_item.insert(1, support)
            L1.insert(0, key)

#     print("Frequent Items: ", freq_item)

    C2 = createCk(L1, 2)

    # print("C2 the second candidate list: ", C2)

    count = {}
    freq_items = []
    L2 = []

    for basket in row:
        for item in C2:
            if item.issubset(basket):
                candidate = frozenset(item)
                if candidate not in count:
                    count[candidate] = 1
                else:
                    count[candidate] += 1
    for key in count:
        support = count[key] / len(row)
        if support >= min_support:
```

```
            freq_items.insert(0, key)
            freq_items.insert(1, support)
            L2.insert(0, key)

    end = time.time()
    return [end-start, L2]
```

Retail Data Set

In [4]:
```
retail = pd.read_csv("http://fimi.uantwerpen.be/data/retail.dat",delimiter=" ", c
itemsets = retail.values.tolist()
```

Removing Nan Values

In [5]:
```
retail_NN = []
for x in itemsets:
    retail_NN.append([i for i in x if str(i) != 'nan'])
```

Create sections

In [6]:
```
sections = [2,5,10,15, 20]
associationR = []
for i, x in enumerate(sections):
    new_list = retail_NN[0:int(len(retail_NN)*(sections[i]/100))]
    associationR.append(new_list)
```

In [25]:
```
for x in associationR:
    print(np.shape(x))
```

```
(1710,)
(4277,)
(8554,)
(12832,)
(17109,)
```

Using implemented function

In [11]:
```
times = []
pairslen = []
actualpairs = []
for x in range(5):
    results = apyori(associationR[x],0.01)
    times.append(results[0])
    pairslen.append(len(results[1]))
    actualpairs.append(results[1])
```

In [30]:
```
for x in range(len(sections)):
    print()
    print()
    print('with %g percent of the dataset I got %d pairs.' % (sections[x], pairsl
    print()
    print(output(actualpairs[x]))
```

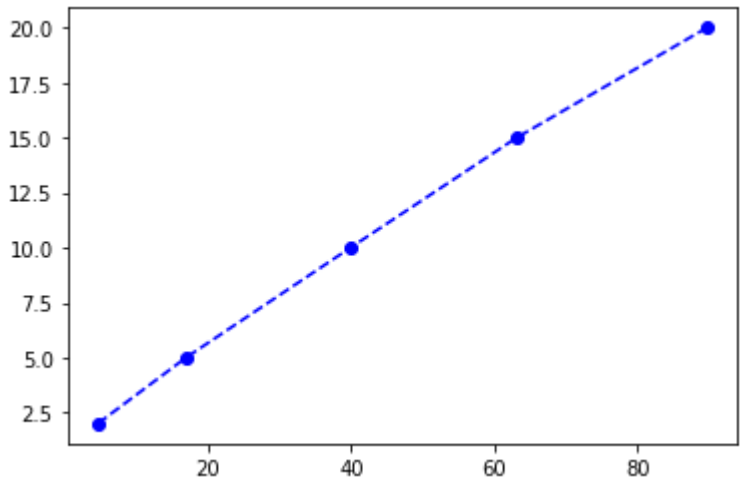
```
with 2 percent of the dataset I got 59 pairs.

38 --> 1327
41 --> 1327
39 --> 1327
48 --> 1327
589 --> 39
740 --> 39
664 --> 39
48 --> 110
604 --> 39
48 --> 310
41 --> 170
475 --> 39
48 --> 475
438 --> 39
48 --> 438
```

Time vs sections

In [31]:
```
pl.plot(times, [x for x in sections], 'bo--')
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x7f3c2d4fab20>]
```



Netflix Data Set

In [ ]:
```
netflix = pd.read_csv('netflix.data', delimiter=" ",engine='python', on_bad_lines
netflix_L = netflix.values.tolist()
```

removing Nan values

In [ ]:
```
netflix_NN = []
for x in netflix_L:
    netflix_NN.append([i for i in x if str(i) != 'nan'])
```

Creating sections

In [ ]:
```
sections = [2,5,10,15, 20]
associationN = []
for i, x in enumerate(sections):
    new_list = netflix_NN[0:int(len(netflix_NN)*(sections[i]/100))]
    associationN.append(new_list)
```

Use implemented function

In [ ]:
```
times = []
pairslen = []
actualpairs = []
for x in range(5):
    results = apyori(associationN[x],0.01)
    times.append(results[0])
    pairslen.append(len(results[1]))
    actualpairs.append(results[1])
```

Output pairs

In [ ]:
```
for x in range(len(sections)):
    print()
    print()
    print('with %g percent of the dataset I got %d pairs.' % (sections[x], pairsl
    print()
    print(output(actualpairs[x]))
```

Time vs section

In [ ]:
```
pl.plot(times, [x for x in sections], 'bo--')
```

Conclusion

In conclusion, I am confident that apyori algorithm was correctly implemented as the output of the graph was roughly the same as the one using the library function. Unfortunately I was unable to run the algorithm on the netflix dataset as loading the dataset consistently shutdown my kernel.