Lab3: Frede Emnetu

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

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]
            L1.sort()
            L2.sort()
            if L1 == L2:
                cand_list.append(Lk[i] | Lk[j])
    return cand_list
```

output function

In [18]:
```
def output(data):
    for x in data:
        list_ = list(x)
        print('%d --> %d' % (list_[0],list_[1]))
```

Hash function

In [3]:
```
def hash(i, j, numBuckets):
    return (i * j) % numBuckets
```

Implemented PCY algorithm

In [14]:
```
def PCY(data, minsupport):
    start = time.time()
    min_support = minsupport
    numBuckets = 2000  # number of buckets for hashmap


    C1 = []
    for basket in data:
        for item in basket:
            if not [item] in C1:
                C1.append([item])
    C1 = [set(x) for x in C1]

    count = {}
    freq_items = []
    L1 = []
    countBuckets = [0] * numBuckets

    for basket in data:
        for item in C1:
            if item.issubset(basket):
                candidate = frozenset(item)
                if candidate not in count:
                    count[candidate] = 1
                else:
                    count[candidate] += 1
        # get all the pairs possible from each basket
        pairs = list(combinations(basket, 2))
        # hash each pair to hashmap
        for pair in pairs:
            listPair = list(pair)
            numbers = [int(x) for x in listPair]
            countBuckets[hash(numbers[0], numbers[1],numBuckets)] += 1

    # convert bucket hashmap to bitmap
    # changing bucket values to 1 if it meets support requirement (frequent bucke
    for i in range(0, len(countBuckets) - 1):
        if countBuckets[i] / len(itemsets) >= min_support:
            countBuckets[i] = 1
        else:
            countBuckets[i] = 0

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

#     print("Frequent Items: ", freq_items)

    freq_pairs = []
    L2 = []
    C2 = []
    final = []
    count = {}
```

```
    test = [list(x) for x in L1]

    for i in test:
        final.append(int(i[0]))
    # construct pair candidates
    for basket in data:
        pairs = list(combinations(basket, 2))
        for pair in pairs:
            listPair = list(pair)
            numbers = [int(x) for x in listPair]
            # if the pair hashes to a frequent bucket and each value in pair is f
            if countBuckets[hash(numbers[0], numbers[1],numBuckets)] == 1 and num
                C2.append(pair)

    for basket in data:
        for item in C2:
            item = frozenset(item)
            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(itemsets)
        if support >= min_support:
            freq_pairs.insert(0, key)
            freq_pairs.insert(1, support)
            L2.insert(0, key)

#     print("Frequent Pairs: ", freq_pairs)


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

Retail Dataset

In [5]:
```
retail = pd.read_csv("http://fimi.uantwerpen.be/data/retail.dat",delimiter=" ", c
itemsets = retail.values.tolist()
```

remove Nan values

In [6]:
```
retail_NN=[]
for x in itemsets:
    retail_NN.append([i for i in x if str(i) != 'nan'])
```

Create sections

In [7]:
```
sections = [2,5,10,20,50]
associationR = []
for i, x in enumerate(sections):
    new_list = retail_NN[0:int(len(retail_NN)*(sections[i]/100))]
    associationR.append(new_list)
```

In [10]:
```
for x in associationR:
    print(np.shape(x))
```

```
(1710,)
(4277,)
(8554,)
(17109,)
(42774,)
```

Using PCY algorithm

In [15]:
```
times = []
pairslen = []
actualpairs = []
count = 0;
for x in range(len(sections)):
    print(count)
    results = PCY(associationR[x],0.01)
    times.append(results[0])
    pairslen.append(len(results[1]))
    actualpairs.append(results[1])
    count += 1
```

```
0
1
2
3
4
```

Output pairs

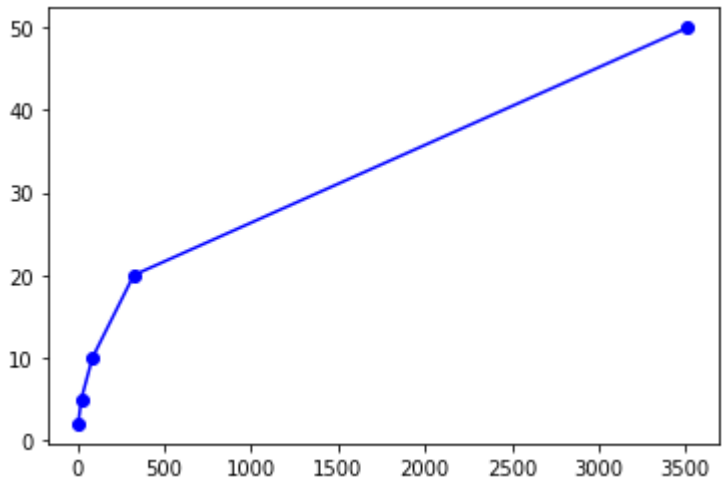
In [19]:
```
for x in range(len(sections)):
    print()

    print()
#     print(pairslen[x])
    print('with %g percent of the dataset I got %d pairs.' % (sections[x], pairsl
    print()
    output(actualpairs[x])
```

```
with 2 percent of the dataset I got 0 pairs.



with 5 percent of the dataset I got 3 pairs.

48 --> 41
48 --> 39
41 --> 39


with 10 percent of the dataset I got 9 pairs.

32 --> 38
32 --> 48
32 --> 39
48 --> 41
```

Plot graph

In [20]:
```
pl.plot(times, [x for x in sections], 'bo-')
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x7f53c86b2a30>]
```



Netflix DataSet

In [ ]:
```
netflix = pd.read_csv('netflix.data',delimiter=" ", on_bad_lines='skip',skip_blar
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'])
```

In [ ]:
```
sections = [2,5,10,20,50]
associationN = []
for i, x in enumerate(sections):
    new_list = retail_NN[0:int(len(retail_NN)*(sections[i]/100))]
    associationN.append(new_list)
```

In [ ]:
```
times = []
pairslen = []
actualpairs = []
count = 0;
for x in range(len(sections)):

    results = PCY(associationR[x],0.01)
    times.append(results[0])
    pairslen.append(len(results[1]))
    actualpairs.append(results[1])
```

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

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

Conclusions

Comparing PCY to Apriori, it is evident that Apriori has a much more efficient run time compared to the PCY algorithm. This is the trade off when constructing an algorithm for memory efficiency. Unfortunatly I was not able to run the netflix data set due to the same reasons as discussed in previous labs. You have noticed the (0,1,2,3,4) outputs in the 'using pcy function" cell, these were used to determine how far in the pcy algorith was