EE412 Foundation of Big Data Analytics, Fall 2021 HW2

20170616 정희진

※ Because I submitted late, I will use token.

1. MapReduce and Spark

(1) Explanation of code

First, I put the text file in RDD

```
conf = SparkConf()
sc = SparkContext(conf = conf)
lines = sc.textFile(sys.argv[1])
```

Before explaining the next line, I'll explain find_common_friends function. The next 4 lines divide the text into one person and their friends. The friends list is sorted.

```
find_common_friends
me_and_friends = l.split('₩t')
friends = me_and_friends[1]
friends = friends.split(",")
friends.sort()
```

The two people who have the same friend("one person" mentioned above) are grouped in to a tuple and put on the key_list. The third factor is set to 1 because later it will be the number of common friends the two people have, and it will be obtained by adding up. The find_common_friends function returns key_list.

```
find_common_friends
key_list = []
for i in range(len(friends)-1):
    for j in range(i+1, len(friends)):
        key_list.append(((friends[i], friends[j]), 1))

return key_list
```

Returning to the main code, if the find_common_friends function is applied to each line, we get the list of (person1, person2, the number of common friends) values, and they are flatten.

```
common_friends1 = lines.flatMap(find_common_friends)
```

Now I will explain the find_friends function. First, it processes the text like the find_common_friends. But it doesn't have to be sorted.

```
find_friends
me_and_friends = l.split('₩t')
me = me_and_friends[0]
```

```
friends = me_and_friends[1]
friends = friends.split(",")
```

Next, put a person and their friends tuple on the key_list. Since they are each other's friends, there is no need to create a pair repeatedly. So the conditional statement "`if me <= friends[i]:`" is included. The value of 1 can be ignored.

```
find_friends
key_list = []
for i in range(len(friends)):
    if me <= friends[i]:
        key_list.append(((me, friends[i]), 1))
```

Returning to the main cod again, as in the above line, the find_friends function is applied to each line and flatmap.

```
friends = lines.flatMap(find_friends)
```

For "finding pairs of users who are not friends with each other", the pair obtained from find_common_friends function is filtered by the pair obtained from find_friends function.

```
common_friends2= common_friends1.subtractByKey(friends)
```

To find the number of friends in common between two people, we have to run the code.

```
count = common_friends2.reduceByKey(lambda x, y: x + y)
```

Sort the tuple according to the condition "The lines must be sorted by their counts in descending order. In case there are ties in counts, the lines are sorted by the first user ID integer in ascending order and then the second user ID integer in ascending order".

```
sorted_count = count.sortBy(lambda c: (-c[1],c[0][0],c[0][1]))
```

Get the top 10 results and print them.

```
result = sorted_count.take(10)

for i in range(10):
    if len(result) > i:
        print(result[i][0][0],'₩t',result[i][0][1],'₩t',result[i][1])
```

(2) Result and Elapsed time

```
18739    18740    100
31506    31530    99
31492    31511    96
31511    31556    96
31519    31554    96
31519    31568    96
31533    31559    96
31555    31560    96
31492    31556    95
31503    31537    95
It took  637.2849524021149 seconds.
```

Using time.time() function, elapsed time is measured.

2. Frequent Itemsets

(a) The number of triangular-matrix method pairs is $\frac{N\times(N-1)}{2}$ because it generate pairs using frequent items

The triangular-matrix size is $\frac{N\times(N-1)}{2}\times 4 = 2N^2 - 2N$ because integer is 4 bytes.

The number of triple method pairs is $10^6 + M$ because it generates pairs whose counts are not 0 only using frequent items.

The triple size is $(M + 10^6) \times 3 \times 4 = 12(M + 10^6)$ because they consist of 3 integers.

Therefore, the minimum number of bytes of main memory needed to execute the A-Priori Algorithm is min$(2N^2 - 2N,\ 12(M + 10^6))$.

(b) (1) Explanation of code

First, define the variables. C1 is a dictionary.

```
file_path = sys.argv[1]
support = 200
C1 = {}
num_fre_items = 0
```

It is first pass. Set the item as the key and count the number.

```
with open(file_path, 'r') as f:
    for line in f:
       basket = line.split()
       for i in basket:
           C1[i] = C1.get(i, 0) + 1
```

Count the number of frequent items. If it is not a frequent item, the value is set to -1. If it is a frequent item, the index is set.

```
num_fre_items = 0
for key, value in C1.items():
    if value < support:
        C1[key] = -1
    else:
        C1[key] = num_fre_items
        num_fre_items = num_fre_items + 1
```

It is second pass. C2 is the triangular-matrix. Temp tells whether there is the frequent item for each basket or not. Therefore, if there is the item in the basket, it is set to true. Next, pick two frequent items set to true, and calculate the position(index) of the triangular-

matrix, and count. To check for frequent items for each basket, set temp to false all before passing to the new basket.

```
C2 = [0 for i in range(int(num_fre_items*(num_fre_items - 1.0) / 2.0))]
temp = [False for i in range(num_fre_items)]

with open(file_path, 'r') as f:
    for line in f:
        basket = line.split()

        for i in basket:
            if C1[i] != -1:
                temp[C1[i]] = True

        for i in range(num_fre_items - 1):
            for j in range(i + 1, num_fre_items):
                if temp[i] == True and temp[j] == True:
                    index = int(i*(num_fre_items-i/2.0-1.0/2.0)+j-i-1.0)
                    C2[index] = C2[index] + 1
        temp = [False for i in range(num_fre_items)]
```

This is just turning the C1 from dictionary to list.

```
temp = []
for key, value in C1.items():
    if value == -1:
        temp.append(key)
for key in temp:
    del C1[key]
C1 = sorted(C1.items(), key = lambda item: item[1])
for i in range(len(C1)):
    C1[i] = C1[i][0]
```

Remove all pairs except frequent item pairs in triangular-matrix. It also turns into (item1, item2, count). The list is sorted from highest count to lowest count.

```
k = 0
for i in range(len(C1) - 1):
    for j in range(i+1, len(C1)):
        if C2[k] < support:
            del C2[k]
        else:
            C2[k] = (C1[i], C1[j], C2[k])
            k = k + 1
C2 = sorted(C2, key = lambda item: item[2], reverse = True)
```

Print the number of frequent items, the number of frequent item pairs, and top-10 most frequent item pairs and their count.

```
print(num_fre_items)
print(len(C2))
for i in range(10):
    if i < len(C2):
        print('%s₩t%s₩t%d'%(C2[i][0],C2[i][1],C2[i][2]))
```

(2) Result and Elapsed time

```
363
328
DAI62779        ELE17451        1592
FRO40251        SNA80324        1412
FRO40251        DAI75645        1254
FRO40251        GRO85051        1213
DAI62779        GRO73461        1139
DAI75645        SNA80324        1130
FRO40251        DAI62779        1070
DAI62779        SNA80324        923
DAI62779        DAI85309        918
ELE32164        GRO59710        911
It took 537 seconds
```

3. Finding Similar Items

(a) I used the website https://www.desmos.com. The red line is y=x. And, the other tick lines are constructed probability. Here are the exact equations. The order of equations are same as the order of problems.

1 $y = x$

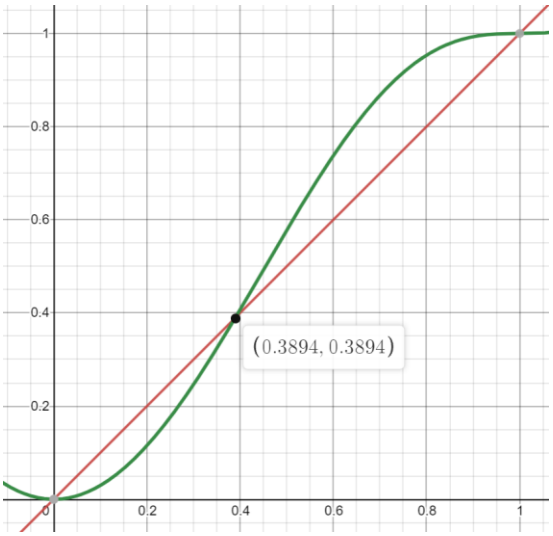
2 $y = 1 - (1 - x^2)^3$

3 $y = (1 - (1 - x)^3)^2$

4 $y = (1 - (1 - x^2)^2)^2$

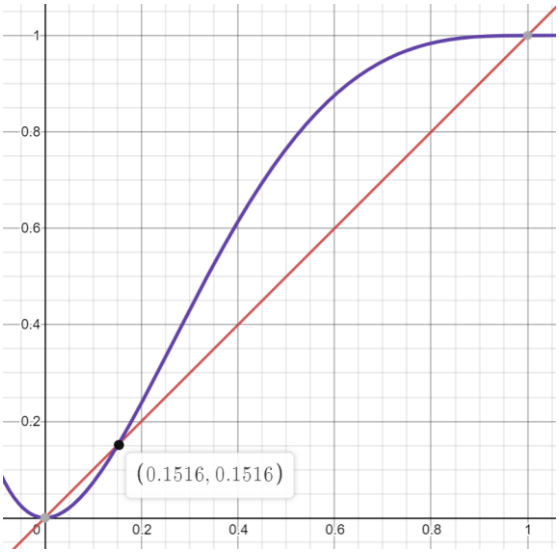
5 $y = (1 - (1 - (1 - (1 - x)^2)^2)$

(a) A 2-way AND construction followed by a 3-way OR construction.



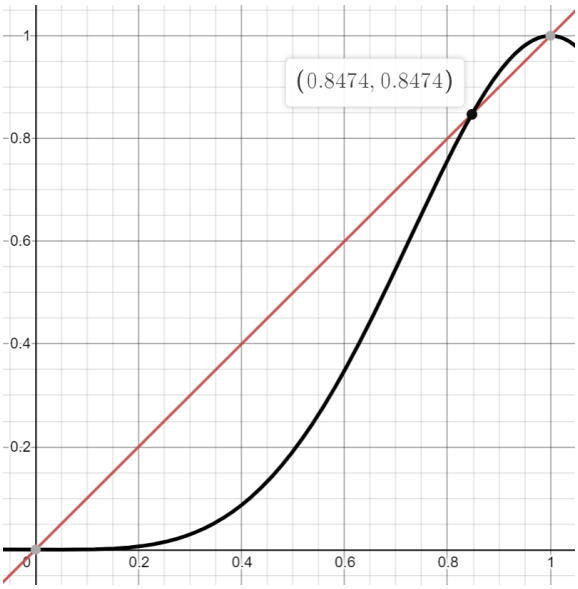
It will be higher when the probability of being declared a candidate is higher than 0.3894, and lower when the probability is lower than 0.3894.

(b) A 3-way OR construction followed by a 2-way AND construction.



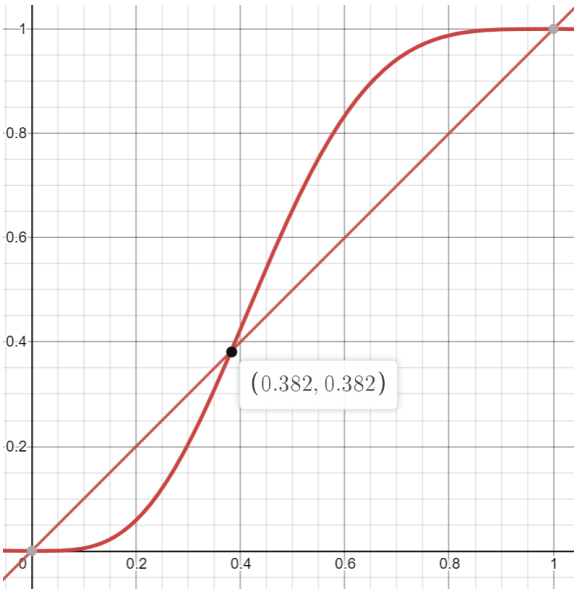
It will be higher when the probability of being declared a candidate is higher than 0.1516, and lower when the probability is lower than 0.1516.

(c) A 2-way AND construction followed by a 2-way OR construction, followed by a 2-way AND construction.



It will be higher when the probability of being declared a candidate is higher than 0.8474, and lower when the probability is lower than 0.8474.

(d) A 2-way OR construction followed by a 2-way AND construction, followed by a 2-way OR construction followed by a 2-way AND construction.



It will be higher when the probability of being declared a candidate is higher than 0.382, and lower when the probability is lower than 0.382.

(b) (1) Explanation of code

First, define the variables.

```
file_path = sys.argv[1]
E = set()
candidate = set()
band = 6
row = 20
n = band * row
num_news = 0
ids = []
```

Put article ID in ids list and shingles in E.

```
with open(file_path, 'r') as f:
    for line in f:
        num_news = num_news + 1
        news_id = line.split()[0]
        ids.append(news_id)
        news = line.lstrip(news_id)
        news = news.lstrip()
        news = news.lower()
        news = re.sub('[^a-z ]',"",news)
        news = re.sub('₩s+'," ",news)
```

```
        for i in range(len(news)-3+1):
            E.add(news[i:i+3])
```

C is the part of the characteristic matrix that tells if the article contains shingles. The index of C represent article.

```
E = list(E)
C = [[] for i in range(len(E))]

with open(file_path, 'r') as f:
    for line in f:
        for i in range(len(E)):
            C[i].append(E[i] in line)
```

This is the function that tells if it is prime. If n is divisible by the number less than n, it is not prime.

```
is_prime
for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

p is the smallest prime number larger than or equal to length of C (the number of shingles). a and b is the list of random numbers less than p. The length is n=120.

```
p = len(C)
while not is_prime(p):
    p = p + 1
a = np.random.randint(p, size = n).tolist()
b = np.random.randint(p, size = n).tolist()
```

H is the hash function values list. The row of H means shingles' index. The column of H means hash functions that are made of a, b, p.

```
H = [[] * n for i in range(len(C))]
for i in range(len(C)):
    for j in range(n):
        H[i].append((a[j]*i+b[j])%p)
```

S is the signature matrix. Whenever an element in C has the true value, the hash function value in H is compared with the element in S, and the element in S is set to smaller value.

```
S = [[float('inf') for i in range(num_news)] for j in range(n)]
for i in range(len(C)):
    for j in range(num_news):
        if C[i][j] == True:
            for k in range(n):
                if S[k][j] > H[i][k]:
                    S[k][j] = H[i][k]
```

Divide S into 6 bands of 20 rows. Then each column is compared, and candidate pairs are obtained and put into the candidate set.

```
S = np.array(S)
for i in range(band):
    for j in range(num_news-1):
        for k in range(j+1,num_news):
            sig1 = S[i*row:(i+1)*row,j]
            sig2 = S[i*row:(i+1)*row,k]
            if np.array_equal(sig1, sig2):
                candidate.add((j, k))
```

Find the similarity in each candidate pair and print if it is greater than or equal to 0.9.

```
candidate = list(candidate)
C = np.array(C)
for i in range(len(candidate)):
    set1 = C[:,candidate[i][0]]
    set2 = C[:,candidate[i][1]]
    set1_and_set2 = np.logical_and(set1, set2)
    set1_num = sum(set1)
    set2_num = sum(set2)
    and_num = sum(set1_and_set2)
    sim = float(and_num) / float(set1_num + set2_num - and_num)
    if sim >= 0.9:
        print("%s₩t%s"%(ids[candidate[i][0]], ids[candidate[i][1]]))
```

(2) Result and Elapsed time.

```
t448    t8535
t8413   t269
t980    t2023
t1621   t7958
t3268   t7998
It took 335 seconds
```