Assignment 3

Exercise 1 : Collaborating Filtering

1. Juccard distance between users A,B,C

	a	b	c	d	e	f	g	h
A	1	1		1	1		1	1
B		1	1	1	1	1	1	
C	1		1	1		1	1	1

Explanation : Juccard index for the users are usually defined by adding the numbers in a sets of the users divided by the number in either of the sets.

(i.e) Total number of members which are shared in the available sets/total number of members in all available sets.

J(A,B) = |A n B| / |A u B|

- Juccard distance between (A,B) = A n B / A u B

  Juccard(A,B) = 4/8 -> ½

- Juccard distance between (B,C) = B n C / B u C

  Juccard(B,C) = 4/8 -> ½

- Juccard distance between (C,A) = C n A / C u A

  Juccard(C,A) = 4/8 -> ½

2. Cosine distance between user A, B, C

	a	b	c	d	e	f	g	h
A	1	1		1	1		1	1
B		1	1	1	1	1	1	
C	1		1	1		1	1	1

Explanation : Cosine similarity is used to find the similarity between the documents or the vector users. It is defined by the formula

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

- Cosine distance between (A,B) = 1-A.B / sqrt(A) * sqrt (B)

Cosine(A,B) = 1+1+1+1/sqrt(6)* sqrt(6)

⇨ 1-0.666 => 0.34

- Cosine distance between (B,C) = 1-B.C / sqrt(B) * sqrt (C)

Cosine(B,C) = 1+1+1+1/sqrt(6)*sqrt(6)

⇨ 1-0.66 => 0.34

- Cosine distance between (C,A) = 1-C.A / sqrt(C) * sqrt (A)

Cosine(C,A) = 1+1+1+1/ sqrt(6)* sqrt(6)

⇨ 1-0.66 =>0.34

3. Treating 3,4,5 as 1 and 1,2 and black as 0-Juccard distance between users A,B,C

	a	b	c	d	e	f	g	h
A	1	1		1			1	
B		1	1	1				
C				1		1	1	1

- Juccard distance between (A,B) = A n B / A u B

Juccard(A,B) = 2/5 => 0.4

- Juccard distance between (B,C) = B n C / B u C

Juccard(B,C) = 1/6 => 0.16

- Juccard distance between (C,A) = C n A / C u A

Juccard(C,A) = 2/6 -> 1/3 =>0.33

4. Cosine distance between user A, B, C

	a	b	c	d	e	f	g	h
A	1	1		1			1	
B		1	1	1				
C				1		1	1	1

- Cosine distance between (A,B) = 1-A.B / sqrt(A) * sqrt (B)

Cosine(A,B) = 1+1/sqrt(1+1+1)* sqrt(1+1+1)

⇨ 1-0.577 =>0.423

- Cosine distance between (B,C) = 1-B.C / sqrt(B) * sqrt (C)

Cosine(B,C) = 1/sqrt(1+1+1)*sqrt(1+1+1+1)

⇨ 1-0.288 =>0.712

- Cosine distance between (C,A) = 1-C.A / sqrt(C) * sqrt (A)

Cosine(C,A) = 1+1/ sqrt(1+1+1+1)* sqrt(1+1+1+1)

⇨ 1-0.5 =>0.5

5. Normalize by subtracting -Juccard distance between users A,B,C

Average entry of User A = 4+5+5+1+3+2/6 => 3.33

Average entry of User B = 3+4+3+1+2+1/6 => 2.33

Average entry of User C = 2+1+3+4+5+3/6 => 3

	a	b	c	d	e	f	g	h
A	0.67	1.67		1.67	-2.33		-0.33	-1.33
B		0.67	1.67	0.67	-1.33	-0.33	-1.33	
C	-1		-2	0		1	2	0

- Juccard distance between (A,B) = A n B / A u B

Juccard(A,B) = 4/8 -> ½

- Juccard distance between (B,C) = B n C / B u C

Juccard(B,C) = 4/8 -> ½

- Juccard distance between (C,A) = C n A / C u A

Juccard(C,A) = 4/8 -> ½

6. Cosine distance between user A, B, C

	a	b	c	d	e	f	g	h
A	0.67	1.67		1.67	-2.33		-0.33	-1.33
B		0.67	1.67	0.67	-1.33	-0.33	-1.33	
C	-1		-2	0		1	2	0

- Cosine distance between (A,B) = 1-A.B / sqrt(A) * sqrt (B)

Cosine(A,B) = 1.1189+1.1189+3.09+0.4389/sqrt(13.33)* sqrt(7.33)

⇨ 1- 0.582 => 0.418

- Cosine distance between (B,C) = 1-B.C / sqrt(B) * sqrt (C)

Cosine(B,C) = -6.33/sqrt(7.33)*sqrt(10)

⇨ 1-(-0.739) => 1.739

- Cosine distance between (C,A) = 1-C.A / sqrt(C) * sqrt (A)

Cosine(C,A) = -1.33/ sqrt(10)* sqrt(13.33)

⇨ 1-0.115 => 1.115

Exercise 2 : Simple randomized Algorithm and comparison.

1. Implementation of Simple randomized algorithm :

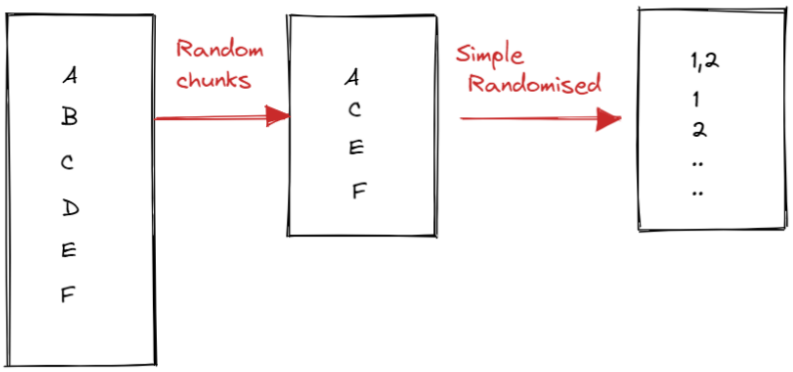
As the name suggests the randomized algorithm picks a certain amount of dataset and reflects as a entire dataset. With the adjusted amount of threshold and support the smaller number of brackets will look like a entire large dataset.

The support threshold of a dataset is considered as s and choosing the sample for 2% of the basket, it means we are looking for a sample of item set with at least s/100 in a baskets.

In order to select a random dataset from the given sets

- For all the data available in each basket generate a random value as a probability of the basket,
- if the probability is greater then 1-P , then the basket is taken for further process which is creating or generating item sets.
- All the values are stored in dictionary as key value pairs.
- Each basket data has to generate a item set which has to be updated in a final set as a result.
- Results are generated as singleton and pairs
- For each value in a item set, the value gets added it the item set is already present

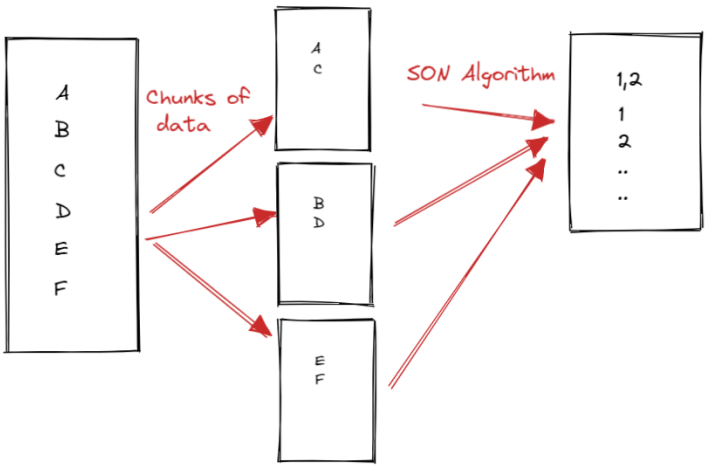
- If not, a new pair of item set is created and the values are added.
- Sort the item set as we can get the most frequent item sets.
- Lowering the confident to a certain value, there is a possibility that the we can get the expected confidence s for the portion of the data p.
- Lowering the confident to more like 0.9 ps e can achieve the exact confidence for any dataset.



2. Implementation of Son algorithm :

Sampling errors are mainly a disadvantage on using the previous algorithm. In order to solve the issue of having false positive and false negative we introduce and improvement to SON algorithm.

We need to give inputs as chunks and treat each sample as a chunk and create item sets. Once chunk item sets are created, they all are merged in to a single result file with all frequent item sets. Union of the chunks are called candidate item sets.



3. Comparing results of 2 Algorithms :

Implementation on both algorithm is different as SR picks random chunks of data and SON picks different chunks of data and gives final result.

There is a possibility of missing important information while picking chunks via random set, whereas its not the case in SON.

Example : Consider you are giving recommendations on Amazon products. If the user is buying a laptop, the recommendations will be mouse, keyboard having 5 star reviews. Considering SR algorithm there is a possibility of missing a product which may or may not be the part of recommendation. If we use Son algorithm there is a thing that all the items will be recorded as a result in SON.

We could see some of the items like 58,60,52 are same in both the version and have minor differences. We could see the singleton and doubleton pairs are same in both the cases. (29,58) in both the cases.

Since we took full dataset for Son algorithm and divided in to chunks we get a total count of these values and in the SR we get count only for the random chunks we took.

Run time for Random Algorithm is `finish in 0:00:00.972927`

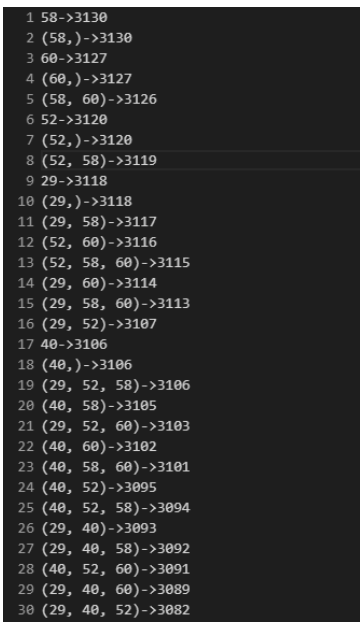
Run time for SON is `Wall time: 3.21 s`

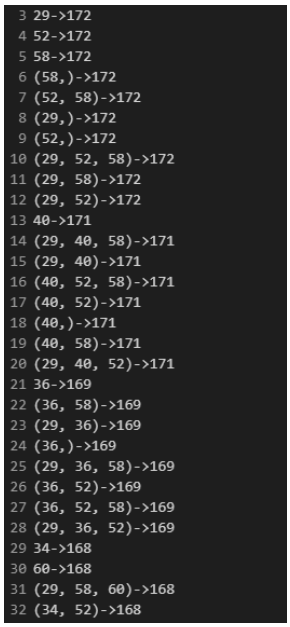
Comparing the run time we can say that SR is faster than SON, but with lesser better results than SON.

Similarly it can be seen in other datasets too.

Here, the screenshots are from Son and SR algorithm for Chess dataset respectively.

SON vs SR algorithm :





4. Performance experiments :

Analyzing the results obtained,

Assuming the results of the Son are little accurate than the chunks at random, we could see that

Running random set algo : Top 5 results of a Chess dataset are :

P=1%	P=2%	P=5%	P=10%	P=20%
(48, 58)->31	(58,)->66	(52, 58)->153	(58,)->310	(58,)->625
(40,48,58)->31	(60,)->66	(58,)->153	(52, 58)->309	(29,)->624
(52,60)->31	(58, 60)->66	(52,)->153	(52,)->309	(52, 58)->624
(40,48,60)->31	(29, 58, 60)->65	(40,)->152	(29,)->308	(29, 58)->624
(40)-31	(52, 60)->65	(40, 52, 58)->152	(29, 58)->308	(52,)->624

Running a SON algo : Top 5 results of a chess dataset are :

SON C=100%
(58,)->3130
(60,)->3127
(58, 60)->3126
(52,)->3120
(52, 58)->3119

P with 20% has 3 correct frequent itemset
P with 10% has 2 correct itemset
P with 5% has 3 correct itemset
P with 2% has 1 correct itemset
P with 1% has 0 correct itemset.

This indicated that more the value of P more the accurate results of frequent itemset and incorrect results may be due to the false positives.

Similar results could be analyzed in other dataset too.

Exercise 3 : Greedy Algorithms and Random Algorithm for Ad words

Ads are the short term acronym for advertisements which are widely placed in many websites today. Ads creates impression based on the advertisements which are placed on a particular search query. These impressions creates a lot of attention to the people who tend to visit links and increase sales or whatever the reason they are using ads.

Ads are created and works on different algorithms which we will see shorty. Big companies like Amazon, Google have their own way of creating ads and impressions.

Before diving deep in to the algorithms, we will see why we are using algorithms for the ads.

The problem :

- As allocated we have a certain set of advertisers and queries, each query can be bid by any advertisers based on the budget they have.
- CTR- click through rate is to be determined for each keyword
- Wallet allowance or budget for each advertiser
- Limitless on ads, (i.e) Duration of the ads to stay as per the budget.
- We need to find advertisers who can bid a certain amount to display ads on the internet.

Greedy Approach :

The main goal was to maximize the search engines revenue. With many search queries q1,q2... and several advertisers we need an online algorithm that will decide which ads to come first and which shouldn't.

Lets, take an example :

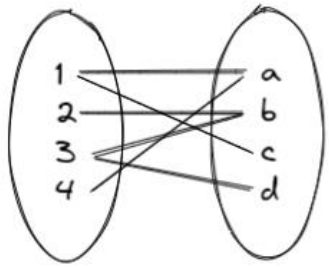
With the stated above example, we can just give A as they are the highest bid and they will help in achieving the goal. Basically the ideology works but it's not an optimal solution.

Hence we introduce a column called CTR, which is click through rate where it tells the percentage of people click the ads. The advertisers get charged only if the users click the ads. Else they are free to view.

CTR behaves as a critical point to decide which brings more income and displays the das accordingly. A can bring only one penny as it has only 1% of CTS whereas B can bring 1.5 cents as they have 2% of CTR. So we can decide to give B a priority than A and C.

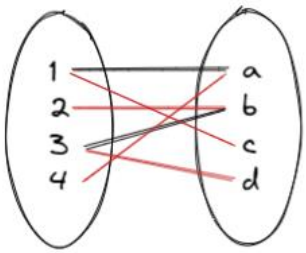
Instead of sorting the advertises bid, sorting the CTR by expected revenue will works efficiently than the heuristic approach. Based on the wallet balance we can give advertisers a chance to showcase their ads.

Consider a greedy match with nodes and edges as (1,a) (1,c) (2,b) (3,b) (3,d)(4,a) . In this the edges (1,a) becomes a super match and (1,c) will not have any match because it has already been hooked up with a.
Next edge (2,b) is selected and the next edge is rejected as it has been already matched. (3,d) is matched and the next is rejected for having a match already. Final edge (4,a) is been rejected because of the same reason as a is been matched already.



With the greedy approach been proceeded we will get the results but not the optimal values with efficiency. Edges matched with greedy are {(1,a) (2,b) (3,d)}

It could also been done in other way, so that all the edges will have a perfect match and this is considered as a major con in greedy algorithm. We could match whats needed and leave the rest un attended as shown in the figure.



There should be atleast A items such that |A|<= |B| and it can get an optimal match. |A| <= |B| <= |Mgreedy|

Greedy approach Simplification:

For each query, pick an advertiser
- Who bids 1 on each query
- Has some budget left for advertising.

Competitive ratio of Greedy:
Competitive ratio is defined as the revenue which are close as possible to what we obtain from it. Its basically the ratio of the revenue we invest and the revenue we get from it.

Competitive ratio = Revenue of our algorithm / reveue of the optimal algorithm.

Competitive ratio for greedy : ½

Balance Algorithm :

Balance algorithm is similar to Greedy approach but with similar modifications. The algorithm assign the query to the advertiser who is having larger budget balance. Ties can be worked based on arbitrary approach.

Consider advertisers A and B with budget 2
Queries x and y with A bidding and z and B bidding on both x and y
Sequence of queries be xxyy

Algorithm works in the following steps :

- Put first x to B
- Put the next x to A as it has highest balance
- Put y to B
- B has no budget to serve second y
- Revenue is 3

Competitive ratio is no more than ¾, its proved that the CR is more than greedy for just 2 advertisers. The competitive ration gets lower but not lower than 0.63. Lower CR can be defined by the formula(1-1/e)

1. Greedy approach to assign atleast 4 out 6 queries.

Constrains : Queries z,y,z
Budget : 2
A can bid only on x
B can bid on x and y
C can bid on x,y,z

Query sequence : xxyyzz

Consider a scenario where we start with B and C as they have more possibility of bidding queries. With this approach A will not be assigned on both of the first queries as they have only one query for bidding. We are restricting the bidding with only B and C.

B and C can bid on first two queries as they have the eligibility to bid on x. Its regardless whether B or C bids on first of them. Next, query Y and we know that it can also be bid by B and C, the third and fourth queries will be assigned to them. Leaving last two queries on the sequence as both of the advertiser budget B and C has been exhausted and there is no way that A can bid as query x has already been matched.

Hence, we can conclude that greedy algorithm will assign at least 4 of the 6 queries xxyyzz

2. Find another sequence of queries such that the greedy algorithm can assign as few as half the queries that the optimal offline algorithm would assign to that sequence.

Considering a sequence xxyyyy with the constrains A can bid on x,y,z. C on y and B on x

The optimal algorithm will be able to assign 4 queries to the advertiser out of 6. Advertiser A and hold 2 and C can hold 2, with that their budget will be exhausted and out of 6, 4 has been matched.

With greedy approach on this query, We can able to tell that A be allocated to first 2 queries as they can hold for x,y and z. With this A budget will be exhausted and B and C will neither not be bidding on those queries.

Hence, the greedy algorithm can achieve only 2 matches compared to the optimal assignment of 4.