EE412 Foundation of Big Data Analytics, Fall 2021

HW1

Name: 함태욱

Student ID: 20180716

Discussion Group (People with whom you discussed ideas used in your answers):

I did it alone

On-line or hardcopy documents used as part of your answers:

- https://smlee729.wordpress.com/2016/06/29/spark-map-flatmap-operations/

- <https://pubdata.tistory.com/38>

## Answer to Problem 1

함수 nC2는 txt문서 한 라인의 tab기준 우측의 사람들 간의 pair를 만들어 리턴.

함수make\_pairs는 tab좌측의 사용자와 tab우측의 사람들 리스트를 리턴

함수 make\_friends는 이미 친구인 사람은 (pair,0) 새로운 친구가 될 가능성이 있는 사람은 (pair,1)로 구성, 각각의 key,value 값을 담은 리스트 pairs를 리턴

각 라인별 map을 통해 make\_pairs를 적용후 flatMap을 통해 make\_friends를 적용한다. 이때 (pair,v)로 구성된 rdd 들을 생성 . 이때 groupByKey로 그룹화 후 이미 친구인 경우는 제외하고 mutual friends로 가장 언급이 많이 된 10쌍을 출력한다.

<Result>

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

time : 968.3367385864258[s]

## Answer to Problem 2

(a) traingular matrix 사용시 frequent items로 lower traingle을 만드므로 총 선택지는 개이다.

따라서 사용하는 메모리는

Triple method 사용시 0이 아닌 값들로 채워지므로 의 메모리를 사용한다.

이 성립할 때 최소 ,그 외의 경우일 때 최소 를 사용한다.

(b) Problem1에서 사용한 nC2를 그대로 사용.

함수 search는 각 줄에서 pair를 찾고 임계를 넘으면 빈도수를 반환. 그렇지 않다면 0 반환.

아이템 네임의 해시 테이블로 파이썬 딕셔너리를 이용했다.(0~n-1)

파일 각 줄을 읽고 단일 아이템이 각각 몇 번 등장했는지 딕셔너리에 기록.

이후 value가 threshold를 넘는 아이템만 추출 후 새롭게 해싱(1~m). 이 아이템들끼리 pair를 형성해 triangular matrix를 만든다. 이때 넘파이 어레이를 활용하였다.

함수 search를 이용해 각 pair가 몇번 등장하는 지 파악후 matrix 해당 부분을 채워 넣는다. 임계치를 넘는 부분에서 결과를 추출하고 이후 해당 row, column 값을 통해 해시테이블에서 item pair를 찾는다.

<Result>

363

328

ELE17451 DAI62779 1592

FRO40251 SNA80324 1412

FRO40251 DAI75645 1254

FRO40251 GRO85051 1213

GRO73461 DAI62779 1139

SNA80324 DAI75645 1130

DAI62779 FRO40251 1070

DAI62779 SNA80324 923

DAI62779 DAI85309 918

GRO59710 ELE32164 911

('time :', 1073.2845270633698)[s]

## Answer to Problem 3

(a)

|  |  |  |  |
| --- | --- | --- | --- |
|  | amplification | If p=0.8 | If p=0.4 |
| A 2-way AND construction followed by a 3-way OR construction. |  | 0.9533 | 0.4073 |
| A 3-way OR construction followed by a 2-way AND construction. |  | 0.9841 | 0.6147 |
| A 2-way AND construction followed by a 2-way OR construction, followed by a 2-way AND construction. |  | 0.7576 | 0.0867 |
| 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. |  | 0.0588 | 0.8342 |

(b)

Minhashing과 LSH계산에서 2~3차원 벡터가 필요할 것 같아 리스트보다 계산 속도가 빠른 넘파이 어레이를 주로 사용했다. Shingles의 중복등록을 막고 임의의 번호를 부여하기위해 딕셔너리를 사용했다. 처음 파일을 읽어서 전체 shingle을 구한다. Shingle 딕셔너리 등록 순으로 한 shingle의 각 문서안 등장여부에 따라 1 아니면 0을 vec\_Arr어레이에 순차적으로 저장하고 그 어레이를 vec\_Arrs에 저장. 이후 20\*6 개의 랜덤 해시함수를 생성하고 shingle 딕셔너리의 value vector를 인풋으로 하여 각각120개의 아웃풋 벡터(120개의 random permutation)를 생성. signature matrix에서 최소값을 구할 때 vec\_Arr의 각 원소와 permutation vecotr의 각 원소를 곱하고 그 값들 중 최소값을 구한다. 이 과정을 문서별로 반복하고 다시 전체 과정을 함수별로 120번 반복한다. signature matrix에 최소값 0을 구해 알맞은 위치에 넣어준다. 이후 각 밴드별로 벡터값이 일치하는 후보군들을 추린 후 자카드 유사도를 통해 최종 결과를 출력한다.

<Result>

t448 t8535

t8413 t269

t980 t2023

t1621 t7958

t3268 t7998

('time :' 847.9405519962311)