Apriori Algorithm Project Report

2016024893 오 성준

1. Summary of Algorithm

다음의 순서로 프로그램이 실행됩니다.

1) Input.txt 파일을 읽습니다. 각각의 transaction들을 저장하고 그 과정에서 전체 transaction에 있는 item들을 모두 저장해서 무슨 item이 있는지도 파악합니다.
2) 저장해둔 item들의 support를 구해서 minimum support를 가지는 item들을 filtering합니다.
3) 본격적인 Apriori알고리즘을 실행합니다..

   3-1) k-1개의 원소를 가지는 frequent pattern들을 입력으로 받습니다. 앞으로 이 것들을 k-1 patterns라고 하겠습니다.

   3-2) 그런 다음에 k-1 patterns를 읽으면서 self-joining합니다. 그 이후에 pruning을 해서 가능한 k개의 원소를 가지는 set들을 모아둡니다 이를 k patterns라고 하겠습니다.

   3-3) 이제 만들어진 k patterns를 돌면서 각각의 set들에서 다시 k-1개의 원소를 가지는 set들을 다시 만들고 이 set들이 k-1 patterns에 존재하는지 확인합니다. 하나라도 없다면 이 k pattern은 candidate가 될 수 없습니다.

   3-4) candidate들을 받은 후 이 candidate 각각의 support를 구해서 minimum support를 가지는지 확인하고 filtering합니다.

   3-5) 3-1부터 3-4까지의 과정을 candidate가 없거나 새로 만들어진 frequent patterns가 없을 때까지 반복합니다.
4) 구해진 frequent patterns에서 각각의 pattern에서 가능한 association rule 후보들과 confidence를 구해서 output.txt 파일에 집어넣습니다.

2. Detailed Description of codes (for each function)

1) file_read (input_name) function

=> Input_name에 해당하는 파일을 읽어서 transaction들을 저장하고, 모든 item들을 모아서 all_items에 저장합니다. 우선 transaction은 list로 저장했고 all_items는 합집합 연산을 통해서 쉽게 구하기 위해서 set을 사용했습니다.

2) get_support (self, candidate) function

=> candidate를 받아서 그 candidate의 support를 구합니다. 이 때, candidate는 set으로 받고 각각의 transcation들을 set으로 변환해서 issubset함수를 사용합니다. 이 때, candidate가 transaction의 subset이 맞으면 cnt를 1 올려주고 마지막에 구한 cnt를 return합니다.

3) get_cadidates (self, before_patterns, k) function

=> 1번에서 3-1부터 3-3까지에 해당하는 함수입니다. K-1개의 원소를 가지는 frequent pattern들의 집합인 before_patterns로부터 각각 k개의 원소를 가지는 candidate를 구합니다. 자료구조가 set이기 때문에 |(합집합)을 이용해서 간단히 self-join했고 Itertools.combination을 사용해서 pruning과 다시 k-1개의 원소를 가지는 set으로 나누는 과정을 실행했습니다. Candidate를 return합니다.

4) get_freq_items (self) function

=> all_items가 self에 items로 저장된 상태에서 각각의 support를 구해서 앞으로 사용할지 말지를

결정합니다. 이 때는 item들 중에서만 구하는 것으로 candidate에 대한 연산은 따로 구현합니다.

5) get_freq_pat (self) function

=> 이제는 본격적으로 frequent pattern인지를 검사합니다. 우선 get_freq_items로 item들을 filtering 하고 나서, 이를 before_patterns에 넣어주고 candidate를 구합니다. 그리고 나서 candidate를 통해서 after_patterns를 구해주고 다시 이 걸 before_patterns에 넣어줍니다. 이와 같은 과정을 반복하는데 만약에 candidate나 after_patterns이 더 이상 나오지 않으면 for문을 break합니다.

6) get_association_patterns (self) function

=> self.freq_pats에는 5번에서 구한 frequent pattern들이 모두 저장되 있습니다. 그리고 나서 이 pattern에서 association rule이 될 수 있는 것들을 모두 구해서 각각의 confidence를 구하고 list를 만든 다음에 association_patterns에 넣어주고 마지막에 return 해줍니다.

7) set2write (set_input) function

=> set을 file write해주기 위해서 output.txt에 해당하는 form으로 바꾼 한 줄의 string을 return 합니다.

8) main (min_s, input_f, output_f) function

=> Apriori class를 통해서 association pattern들을 구하고 이를 output에 format에 맞게 write합니다.

3. Instructions for compiling your source codes at TA's computer (e.g screenshot)

```
seongjun-o@seongjun-oui-MacBookPro DataScience % python apriori.py 5 input.txt output.txt
(2, 190, 190)
(3, 1140, 77)
(4, 23, 16)
283
seongjun-o@seongjun-oui-MacBookPro DataScience % python apriori.py 3 input.txt output.txt
(2, 190, 190)
(3, 1140, 462)
(4, 488, 62)
(5, 7, 7)
721
seongjun-o@seongjun-oui-MacBookPro DataScience %
```

=> 명세대로 실행되도록 했습니다. 그리고 실제파일에선 print를 모두 주석처리 해 뒀습니다.

4. Any other specification of your implementation and testing

=> There is no specification of my implementation and testing