Apriori algorithm 2016025950 정재용

Summary

- Apriori algorithm을 구현하기 위해 이론 수업시간에 배운 수도코드를 이용했습니다.

- Pseudo-code:

  C_k: Candidate itemset of size k
  L_k : frequent itemset of size k

  L_1 = {frequent items};
  for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
  increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
  return $\cup_k L_k$;

- 프로그래밍 언어는 파이썬을 사용했습니다.
- 알고리즘에 사용된 함수는 다음과 같습니다.
  - getItemsetFromTdb - list 형태로 변형한 tdb에서 원소 하나짜리 itemset을 얻는 함수
  - getSupport - itemset X의 support를 구하는 함수
  - filterBySup - 주어진 itemset(candidate)에서 min_support를 만족하는 것만 추려내는 함수
  - getCandidate - Lk에서 self-join 과 pruning을 거쳐 candidate를 생성하는 함수
  - getFrequentItemset - tdb로부터 frequent itemset을 구하는 함수
  - getSubsets - 주어진 set의 공집합을 제외한 subset을 구하는 함수

- apriori - frequent itemset과 support을 구하고 dataframe 형태로 리턴하는 함수
- association_rule - dataframe으로 주어진 frequent itemset에서 association 관계를 구하는 함수

Description

getItemsetFromTdb(tdb: list)

```
def getItemsetFromTdb(tdb: list):
    # Get all single element sets from transaction DB
        # tdb: list - list of transaction(type of set)
        # return: list - single element itemsets

    return list(map(lambda x: {x}, (set([i for t in tdb for i in t]))))
```

개요

- transaction DB로 부터 tdb에 속한 모든 아이템의 리스트를 구하는 함수
- tdb는 각 transaction을 아이템으로 갖는 list로, 하나의 transaction은 set으로 이루어졌습니다.

디테일

- tdb list의 아이템을 모두 새로운 list에 넣습니다.
- set으로 만들어서 중복을 없애고, map을 통해 각각의 원소들을 set type으로 만들어 줍니다.
- map()을 이용해서 single element set의 list 형태로 return 해줍니다.

샘플코드

```
[3]  tdb = [{1, 16, 4, 2},
            {2, 3, 4, 7, 8},
            {11},
            {12,13, 17},
            {5, 9, 10, 15, 2, 3}]

     getItemsetFromTdb(tdb)
```

```
[{1},
 {2},
 {3},
 {4},
 {5},
 {7},
 {8},
 {9},
 {10},
 {11},
 {12},
 {13},
 {15},
 {16},
 {17}]
```

getSupport(tdb: list, item: set)

```
def getSupport(tdb: list, item: set):
    # Get support of item passed by argument
        # tdb: list - list of transaction(type of set)
        # return support by percentage

    _sup = 0
    for t in tdb:
        if item.issubset(t):
            _sup += 1

    return _sup / len(tdb)
```

개요

- 특정 itemset에 대해 transaction DB에서의 support를 구합니다

디테일

- tdb를 순회하며 transaction의 subset일 경우 count해줍니다.

- tdb의 길이로 나눠 확률 형태로 return 해줍니다.

샘플코드

```
[7]  tdb = [{1, 16, 4, 2},
            {2, 3, 4, 7, 8},
            {11},
            {12,13, 17},
            {5, 9, 10, 15, 2, 3}]

     item = {2, 3}

     getSupport(tdb, item)

     0.4
```

filterBySup(tdb: list, itemset: list, sup: float)

```
def filterBySup(tdb: list, itemset: list, sup: float):
    # filtering candidate by its support
        # tdb: list - list of transaction(type of set)
        # itemset: list - list of itemsets
        # sup: int - percentage of minimun support
        # return: list - frequent itemsets (L(k))

    return [i for i in itemset if getSupport(tdb, i) >= sup]
```

개요

- candidate itemsets 에서 minimum support를 만족하는 itemset만 골라냅니다.
- sup 인자는 minimum support를 퍼센트 형식 (ex. 0.02)으로 전달받습니다.
- 결과적으로 C(k)로부터 L(k)를 얻어냅니다.

디테일

- 전달 받은 itemset을 순회하면서 getSupport함수를 통해 support를 구하고 인자로 받은 minimum support와 비교합니다.

샘플코드

```
[33] tdb = [{1, 16, 4, 2},
            {2, 3, 4, 7, 8},
            {11},
            {12,13, 17},
            {5, 9, 10, 15, 2, 3},
            {1, 2, 3, 4, 5},
            {1, 2, 4, 6, 7, 9}]

     itemset = getItemsetFromTdb(tdb)
     filterBySup(tdb, itemset, 0.2)
```

```
[{1}, {2}, {3}, {4}, {5}, {7}, {9}]
```

getCandidate(itemset: list, length: int)

```
def getCandidate(itemset: list, length: int):
    # Get next-step candidates from frequent itemsets
        # itemset: list - list of set type items (C(k))
        # length: int - length of set type items of current itemsets
        # return: list - C(k + 1)

    _ck = []

    if length == 1:
        [_ck.append(i.union(j)) for i in itemset for j in itemset if i != j]
    else:
        for (i, j) in combinations(itemset, 2):
            _sd = i.symmetric_difference(j)
            if len(_sd) == length and _sd in itemset:
                _ck.append(i.union(j))

    result = []
    [result.append(i) for i in _ck if i not in result]

    return result
```

개요

- L(k) itemset을 전달받아 self-joining 과 pruning을 거쳐 C(k+1)를 생성합니다.
- 전달받는 itemset의 원소 개수를 length 인자로 전달받습니다.

디테일

- length가 1이라면 원소간 단순 합집합으로 길이 2의 set list를 만듭니다.
- 두 itemset을 join할 때, 두 set의 교집합을 제외한 합집합이 기존 itemset list에 있어야 조건을 만족합니다.
- 조건을 만족하는 candidate를 return list에 저장하고, return 하기 전 중복되는 itemset을 제거해줍니다.

샘플코드

```
tdb = [{1, 16, 4, 2},
       {2, 3, 4, 7, 8},
       {11},
       {12,13, 17},
       {5, 9, 10, 15, 2, 3},
       {1, 2, 3, 4, 5},
       {1, 2, 4, 6, 7, 9}]

itemset = getItemsetFromTdb(tdb)
f_itemset = filterBySup(tdb, itemset, 0.2)
getCandidate(f_itemset, 1)
```

```
[{1, 2},
 {1, 3},
 {1, 4},
 {1, 5},
 {1, 7},
 {1, 9},
 {2, 3},
 {2, 4},
 {2, 5},
 {2, 7},
 {2, 9},
 {3, 4},
 {3, 5},
 {3, 7},
 {3, 9},
 {4, 5},
 {4, 7},
 {4, 9},
 {5, 7},
 {5, 9},
 {7, 9}]
```

getFrequentItemset(tdb: list, sup: float)

```python
def getFrequentItemset(tdb: list, sup: float):
    # get Frequent itemsets from transaction DB
        # tdb: list - list of transaction (type of set)
        # sup: float - percentage of minimun support
        # return: list - list of frequent itemsets

    _Ck = getItemsetFromTdb(tdb)
    _Lk = filterBySup(tdb, _Ck, sup)
    _L = []
    _length = 1

    while len(_Lk) != 0:
        _L.extend(_Lk)
        _Ck = getCandidate(_Lk, _length)
        _Lk = filterBySup(tdb, _Ck, sup)
        _length += 1

    return _L
```

개요

- transaction DB와 minimum support를 전달받아 전체 apriori 알고리즘을 진행합니다.
- frequent한 itemset의 list를 리턴합니다.

디테일

- 먼저 getItemsetFromTdb() 함수를 통해 C1을 생성하고, filterBySup()함수로 L1을 구합니다.
- Lk에 원소가 없을 때까지 반복문을 돌면서 candidate를 생성하고 필터링을 통해 frequent itemset을 만드는 과정을 반복합니다.
- 생성되는 frequent itemset을 list에 모아 리턴합니다.

샘플코드

```
[40] tdb = [{1, 16, 4, 2},
            {2, 3, 4, 7, 8},
            {11},
            {12,13, 17},
            {5, 9, 10, 15, 2, 3},
            {1, 2, 3, 4, 5},
            {1, 2, 4, 6, 7, 9}]

     itemset = getItemsetFromTdb(tdb)
     getFrequentItemset(tdb, 0.2)
```

```
     [{1},
      {2},
      {3},
      {4},
      {5},
      {7},
      {9},
      {1, 2},
      {1, 4},
      {2, 3},
      {2, 4},
      {2, 5},
      {2, 7},
      {2, 9},
      {3, 4},
      {3, 5},
      {4, 7},
      {1, 2, 4},
      {2, 3, 4},
      {2, 3, 5},
      {2, 4, 7}]
```

getSubsets(itemset: set)

```python
def getSubsets(itemset: set):
    # get subsets of set except null set
        # itemset: set - itemset
        # return: list - list of subsets

    _ssl = []
    _item_chain = chain([combinations(itemset, i+1) for i in range(len(itemset))])
    [_ssl.append(set(i)) for c in _item_chain for i in c ]

    return _ssl
```

개요

- 전달받은 set의 공집합을 제외한 부분집합을 구합니다.
- frequent itemset으로 부터 association rule 관계를 나타낼 때 사용하기 위해 만든 함수입니다.

디테일

- itertools의 combination을 사용해서 원소끼리 조합을 한 후
- chain으로 묶고 순회하면서 set type으로 바꾼 후 list에 저장합니다.

샘플코드

```
[44] itemset = {1, 2, 3}
     getSubsets(itemset)

     [{1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}]
```

apriori(tdb: list, sup: float)

```
def apriori(tdb: list, sup: float):
    # get frequent itemsets and its support from transaction DB with minimum support
        # tdb: list - list of transaction (type of set)
        # sup: float - percentage of minimun support
        # return: dataframe(columns=[itemset, support]) - frequent itemsets and its support

    _freq_itemsets = getFrequentItemset(tdb, sup)

    _df = pd.DataFrame(data=pd.Series(_freq_itemsets), columns=['itemset'])
    _df['support'] = _df['itemset'].map(lambda x: getSupport(tdb, x))

    return _df
```

개요

- transaction DB와 minumum support를 전달 받아 frequent itemset list를 구하고, 각 itemset의 support와 함께 리턴합니다.
- pandas.DataFrame 형태로 정리해서 리턴합니다.

- column은 'itemset' (type: set) 과 'support' (type: float) 입니다.

디테일

- getFrequentItemset()함수를 통해 frequent itemset list를 구합니다.
- 데이터프레임 형태로 frequent itemset list를 나타낸 후
- 모든 itemset에 대해 getSupport()함수를 적용해서 파생변수 'support'를 만듭니다.

샘플코드

```
tdb = [{1, 16, 4, 2},
       {2, 3, 4, 7, 8},
       {11},
       {12,13, 17},
       {5, 9, 10, 15, 2, 3},
       {1, 2, 3, 4, 5},
       {1, 2, 4, 6, 7, 9}]

apriori(tdb, 0.4)
```

	itemset	support
0	{1}	0.428571
1	{2}	0.714286
2	{3}	0.428571
3	{4}	0.571429
4	{1, 2}	0.428571
5	{1, 4}	0.428571
6	{2, 3}	0.428571
7	{2, 4}	0.571429
8	{1, 2, 4}	0.428571

association_rule(frequent_itemsets)

```python
def association_rule(frequent_itemsets):
    # get association and its confidence from frequent itemsets
        # frequent_itemsets: dataframe - frequent itemsets and its support
        # return: dataframe(columns=[antecedent, consequent, support, confidence])

    _ar = []

    for row in frequent_itemsets.iterrows():
        _itemset = row[1]['itemset']
        if len(_itemset) == 1:
            continue

        _subsets = getSubsets(_itemset)
        for _ant in _subsets:
            _cons = _itemset.difference(_ant)
            if len(_cons) > 0:
                _union_sup = frequent_itemsets[frequent_itemsets['itemset'] == _itemset].reset_index().loc[0, 'support']
                _ant_sup = frequent_itemsets[frequent_itemsets['itemset'] == _ant].reset_index().loc[0, 'support']
                _conf = _union_sup / _ant_sup
                _ar.append((_ant, _cons, round(_union_sup * 100, 2), round(_conf * 100, 2)))

    return pd.DataFrame(_ar, columns = ['antecedent', 'consequent', 'support', 'confidence'])
```

개요

- apriori() 함수로 만들어진 frequent itemset 데이터프레임으로 부터 association rule과 그 confidence를 구하고 다시 데이터프레임 형태로 정리합니다.
- 반환되는 데이터프레임의 column은 'antecedent' (type: set), 'consequent' (type: set), 'support' (type: float), 'confidence' (type: float) 입니다.

디테일

- frequent itemset 데이터프레임의 행을 순회하면서 각 itemset을 가져옵니다.
- 원소가 1개 이상인 itemset에 대해 subset을 구합니다.
- subset들을 순회하면서 antecedent와 consequence(합집합 - antecedent)을 정합니다.
- consequence에 원소가 있을 때 (antecedent가 전체 itemset이 아닐 때)
- itemset의 support와, 조건부 확률을 통해 confidence를 구합니다.
- list에 튜플로 저장하면서 support와 confidence는 소수점 아래 둘째자리에서 반올림 해줍니다.
- 데이터프레임 형태로 만들어서 리턴합니다.

샘플코드

```
tdb = [{1, 16, 4, 2},
       {2, 3, 4, 7, 8},
       {11},
       {12,13, 17},
       {5, 9, 10, 15, 2, 3},
       {1, 2, 3, 4, 5},
       {1, 2, 4, 6, 7, 9}]

ap = apriori(tdb, 0.4)
association_rule(ap)
```

	antecedent	consequent	support	confidence
0	{1}	{2}	42.86	100.0
1	{2}	{1}	42.86	60.0
2	{1}	{4}	42.86	100.0
3	{4}	{1}	42.86	75.0
4	{2}	{3}	42.86	60.0
5	{3}	{2}	42.86	100.0
6	{2}	{4}	57.14	80.0
7	{4}	{2}	57.14	100.0
8	{1}	{2, 4}	42.86	100.0
9	{2}	{1, 4}	42.86	60.0
10	{4}	{1, 2}	42.86	75.0
11	{1, 2}	{4}	42.86	100.0
12	{1, 4}	{2}	42.86	100.0
13	{2, 4}	{1}	42.86	75.0

main(sup, input_file, output_file)

```
def main(sup, input_file, output_file):
    # main function
        # sup: str - minimum support from command line
        # input_file: str - name of input file
        # output_file: str - name of output file

    # file  open
    f = open(input_file, 'r')
    lines = f.readlines()
    f.close()

    # make transaction list from input.txt
    tdb = []
    for l in lines:
        tdb.append(set(l.split()))

    # running algorithm
    ap = apriori(tdb, int(sup) / 100)
    ar = association_rule(ap)

    # make ouput.txt
    f = open(output_file, 'w')
    for row in ar.iterrows():
        d  = row[1].to_dict()
        f.write('{{{0}}}\t{{{1}}}\t{2:.2f}\t{3:.2f}\n'. format(','.join(d['antecedent']), ','.join(d['consequent']), d['support'], d['confidence']))
    f.close()


if __name__ == '__main__':
    main(sys.argv[1], sys.argv[2], sys.argv[3])
```

개요

- 프로그램의 main 함수입니다.
- command line을 통해 인자를 받아서 apriori 알고리즘을 실행하고 association rule을 구해 최종 output.txt를 만듭니다.

디테일

- 인자로 받은 input 파일을 오픈하고, transaction line들을 집합의 list 형태로 가공합니다.
- apriori() 함수와 association_rule() 함수를 실행해서 결과를 얻습니다.
- 인자로 받은 대로 output 파일을 생성하고, string formatting을 통해 형식에 맞게 결과를 작성합니다.
- support와 confidence는 소수점 아래 둘째자리가 0이더라도 그 자리를 채우도록 해줍니다.

프로그램 실행

커맨드라인에서 다음과 같이 프로그램을 실행합니다.

```
(base) simon ~/Desktop/DataScience/assignment/apriori python3 apriori.py 2 input.txt output.txt
(base) simon ~/Desktop/DataScience/assignment/apriori
```

실행 후 output.txt 파일의 모습입니다.

```
{1}     {8}     15.40   51.68
{8}     {1}     15.40   34.07
{1}     {5}     10.00   33.56
{5}     {1}     10.00   39.68
{1}     {15}    10.80   36.24
{15}    {1}     10.80   38.57
{1}     {4}     9.20    30.87
{4}     {1}     9.20    37.40
{1}     {9}     9.60    32.21
{9}     {1}     9.60    34.53
{1}     {13}    10.20   34.23
{13}    {1}     10.20   34.46
{1}     {12}    8.20    27.52
{12}    {1}     8.20    33.61
{1}     {2}     9.00    30.20
{2}     {1}     9.00    34.09
{18}    {1}     8.00    28.99
{1}     {18}    8.00    26.85
{1}     {17}    6.80    22.82
{17}    {1}     6.80    28.57
```