pt3

5 ноября 2020 г.

1 PRACTICAL TASK 3

[1]:
```
import pandas as pd
import apyori as apriori
import numpy as np
import collections
import itertools
from anytree import Node, RenderTree, search, Walker
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
```

1.1 TASK 1

[2]:
```
itemset = ["ABCD", "ACDF", "ACDEG", "ABDF", "BCG", "DFG", "ABG", "CDFG"]
itemset_list = [list(str) for str in itemset]
tid = [n for n in range(1, 9)]
```

1.2 APRIORI

[3]:
```
df = pd.DataFrame(tid, columns=["tid"])

unique_items = list(collections.OrderedDict.fromkeys("".join(itemset)).keys())

for i in range(len(unique_items)):
    df[unique_items[i]] = [1 if unique_items[i] in s else 0 for s in␣
 ↪itemset_list]

df.head()
```

[3]:
```
   tid  A  B  C  D  F  E  G
0    1  1  1  1  1  0  0  0
1    2  1  0  1  1  1  0  0
2    3  1  0  1  1  0  1  1
3    4  1  1  0  1  1  0  0
4    5  0  1  1  0  0  0  1
```

[4]:
```
support = 3/8
```

[5]:

```
F = []
```

[6]:

```
# первый этап
print(f"Step 0")

F.append(dict())

# подсчет поддержки для всех уникальных элементов
for item in unique_items:
    local_support = np.sum(df[item]) / df.shape[1]
    print(f"support of {item}: \t{local_support}")
    if local_support >= support:
        F[0][item] = np.sum(df[item]) / df.shape[1]

print(f"frequent sets: {F[0]}")
print()

# следующие шаги
k = 1

while bool(F[k-1]): # пока есть часто встречающиеся наборы
    print(f"Step {k}")
    C = dict()
    combinations = list(itertools.combinations(unique_items, k+1)) #получаем все␣
 ↪возможные комбинации элементов

    print(f"amount of possible combinations of {k+1} elements:␣
 ↪\t{len(combinations)}")

    # генерация комбинаций с учетом поддержки наборов меньшего размера
    for i in reversed(range(len(combinations))):
        if k > 1:
            local_combinations = list(itertools.combinations(combinations[i], k))
        else:
            local_combinations = [combinations[i][0], combinations[i][1]]

        for j in range(len(local_combinations)):
            if not F[k-1].get(local_combinations[j], False):
                combinations.pop(i)
                break

    print(f"amount of combinations after reduction: \t{len(combinations)}")
    print()

    # для каждой комбинации найдем поддержку
    for combination in combinations:
        col_mult = df[combination[0]].copy()
        for i in range(1, len(combination)):
            col_mult *= df[combination[i]]

        local_support = np.sum(col_mult) / df.shape[1]

        print(f"support of {combination}: \t{local_support}")

        # если поддержка больше установленной, то набор проходит на следующий␣
↪этап
        if local_support >= support:
            C[combination] = np.sum(col_mult) / df.shape[1]

    F.append(C)

    print(f"frequent sets: {F[k]}")
    print()

    k = k + 1
```

```
Step 0
support of A:   0.625
support of B:   0.5
support of C:   0.625
support of D:   0.75
support of F:   0.5
support of E:   0.125
support of G:   0.625
frequent sets: {'A': 0.625, 'B': 0.5, 'C': 0.625, 'D': 0.75, 'F': 0.5, 'G':
0.625}

Step 1
amount of possible combinations of 2 elements:  21
amount of combinations after reduction:         15

support of ('A', 'B'):  0.375
support of ('A', 'C'):  0.375
support of ('A', 'D'):  0.5
support of ('A', 'F'):  0.25
support of ('A', 'G'):  0.25
support of ('B', 'C'):  0.25
support of ('B', 'D'):  0.25
support of ('B', 'F'):  0.125
support of ('B', 'G'):  0.25
support of ('C', 'D'):  0.5
support of ('C', 'F'):  0.25
support of ('C', 'G'):  0.375
support of ('D', 'F'):  0.5
support of ('D', 'G'):  0.375
support of ('F', 'G'):  0.25
frequent sets: {('A', 'B'): 0.375, ('A', 'C'): 0.375, ('A', 'D'): 0.5, ('C',
'D'): 0.5, ('C', 'G'): 0.375, ('D', 'F'): 0.5, ('D', 'G'): 0.375}

Step 2
amount of possible combinations of 3 elements:  35
amount of combinations after reduction:         2

support of ('A', 'C', 'D'):     0.375
support of ('C', 'D', 'G'):     0.25
frequent sets: {('A', 'C', 'D'): 0.375}

Step 3
amount of possible combinations of 4 elements:  35
amount of combinations after reduction:         0

frequent sets: {}
```

1.3 FPG

[7]:

```
df = pd.DataFrame(zip(tid, itemset_list), columns=["tid", "itemset"])
df.head()
```

[7]:

```
   tid          itemset
0    1     [A, B, C, D]
1    2     [A, C, D, F]
2    3  [A, C, D, E, G]
3    4     [A, B, D, F]
4    5        [B, C, G]
```

[8]:

```
support = 2
```

[9]:

```
unique_items_count = dict()

for u_item in unique_items:
    count = 0
    for item in itemset:
        if u_item in item:
            count += 1
    unique_items_count[u_item] = count
```

[10]:

```
df["itemset_ordered"] = [sorted(df["itemset"][i], key=lambda x :␣
 ↪unique_items_count[x], reverse=True) for i in range(len(df["itemset"]))]

for key in unique_items_count.keys():
    if unique_items_count[key] < support:
        for i in range(len(df["itemset_ordered"])):
            for j in range(len(df["itemset_ordered"][i])):
                if df["itemset_ordered"][i][j] == key:
                    df["itemset_ordered"][i].pop(j)
                    print(f"delete {key}: support {unique_items_count[key] }␣
 ↪lower than min support {support}")
                    break

df.head(8)
```

```
delete E: support 1 lower than min support 2
```

```
[10]:    tid          itemset itemset_ordered
      0    1     [A, B, C, D]    [D, A, C, B]
      1    2     [A, C, D, F]    [D, A, C, F]
      2    3  [A, C, D, E, G]    [D, A, C, G]
      3    4     [A, B, D, F]    [D, A, B, F]
      4    5        [B, C, G]       [C, G, B]
      5    6        [D, F, G]       [D, G, F]
      6    7        [A, B, G]       [A, G, B]
      7    8     [C, D, F, G]    [D, C, G, F]
```

```
[11]: root = Node("root", ind=0)

      for i in range(len(df["itemset_ordered"])):
          print(f"transaction {i+1}:")

          prev = root
          for j in range(len(df["itemset_ordered"][i])):
              cur = None
              for child in prev.children:
                  if df["itemset_ordered"][i][j] == child.name:
                      cur = child

              if cur:
                  cur.ind += 1
              else:
                  cur = Node(df["itemset_ordered"][i][j], ind = 1, parent=prev)

              prev = cur

          for pre, fill, node in RenderTree(root):
              print(f"{pre} {node.name} {node.ind}")

          print()
```

```
transaction 1:
 root 0
  D 1
      A 1
          C 1
              B 1

transaction 2:
 root 0
  D 2
      A 2
          C 2
              B 1
              F 1

transaction 3:
 root 0
  D 3
      A 3
          C 3
              B 1
              F 1
              G 1

transaction 4:
 root 0
  D 4
      A 4
          C 3
             B 1
             F 1
             G 1
          B 1
              F 1

transaction 5:
 root 0
  D 4
     A 4
         C 3
            B 1
            F 1
            G 1
         B 1
             F 1
  C 1
      G 1
          B 1
```

```
transaction 6:
 root 0
  D 5
     A 4
        C 3
           B 1
           F 1
           G 1
        B 1
            F 1
     G 1
         F 1
  C 1
      G 1
          B 1

transaction 7:
 root 0
  D 5
     A 4
        C 3
           B 1
           F 1
           G 1
        B 1
            F 1
     G 1
         F 1
  C 1
     G 1
         B 1
  A 1
      G 1
          B 1

transaction 8:
 root 0
  D 6
     A 4
        C 3
           B 1
           F 1
           G 1
        B 1
            F 1
     G 1
        F 1
```

```
   C 1
       G 1
           F 1
C 1
   G 1
       B 1
A 1
    G 1
        B 1
```

[12]:
```
popular_sets = dict()

w = Walker()

for u_item in unique_items:
    nodes = search.findall_by_attr(root, u_item)

    if nodes == ():
        continue

    print(f"for {u_item}:")

    paths = []

    for node in nodes:
        p = w.walk(root, node)
        p = tuple(x for x in p[2] if x.name != node.name)

        if p == ():
            continue

        paths.append(p)

    if paths == []:
        continue

    local_root = Node("root", ind=0)

    for path in paths:
        prev = local_root
        for el in path:
            cur = None
            for child in prev.children:
                if el.name == child.name:
                    cur = child
            if not cur:
                cur = Node(el.name, ind = el.ind, parent=prev)

            prev = cur

    for pre, fill, node in RenderTree(local_root):
        print(f"{pre} {node.name} {node.ind}")

    items_path_count = dict()

    for _u_item in unique_items:
        nodes = search.findall_by_attr(local_root, _u_item)

        sum = 0
        for node in nodes:
            sum += node.ind
        if sum > 0:
            items_path_count[_u_item] = sum

    for key in list(items_path_count.keys()):
        if items_path_count[key] < support:
            items_path_count.pop(key)

    if items_path_count == {}:
        continue

    for i in range(1, len(items_path_count) + 1):
        combs = list(itertools.combinations(items_path_count.keys(), i))

        for comb in combs:
            min_in_comb = min([items_path_count[x] for x in comb])
            result_set = [x for x in comb]
            result_set.append(u_item)
            popular_sets[tuple(result_set)] = min_in_comb
```

```
for A:
 root 0
  D 6
for B:
 root 0
  D 6
     A 4
         C 3
  C 1
     G 1
  A 1
      G 1
```

```
for C:
 root 0
  D 6
      A 4
for D:
for F:
 root 0
  D 6
      A 4
         C 3
         B 1
      G 1
      C 1
          G 1
for G:
 root 0
  D 6
     A 4
        C 3
     C 1
  C 1
  A 1
```

[13]:

```
for key in popular_sets.keys():
    print(f"rule: {key} supp: {popular_sets[key] / 8}")
```

```
rule: ('D', 'A') supp: 0.75
rule: ('A', 'B') supp: 0.625
rule: ('C', 'B') supp: 0.5
rule: ('D', 'B') supp: 0.75
rule: ('G', 'B') supp: 0.25
rule: ('A', 'C', 'B') supp: 0.5
rule: ('A', 'D', 'B') supp: 0.625
rule: ('A', 'G', 'B') supp: 0.25
rule: ('C', 'D', 'B') supp: 0.5
rule: ('C', 'G', 'B') supp: 0.25
rule: ('D', 'G', 'B') supp: 0.25
rule: ('A', 'C', 'D', 'B') supp: 0.5
rule: ('A', 'C', 'G', 'B') supp: 0.25
rule: ('A', 'D', 'G', 'B') supp: 0.25
rule: ('C', 'D', 'G', 'B') supp: 0.25
rule: ('A', 'C', 'D', 'G', 'B') supp: 0.25
rule: ('A', 'C') supp: 0.5
rule: ('D', 'C') supp: 0.75
rule: ('A', 'D', 'C') supp: 0.5
rule: ('A', 'F') supp: 0.5
rule: ('C', 'F') supp: 0.5
rule: ('D', 'F') supp: 0.75
```

```
rule: ('G', 'F') supp: 0.25
rule: ('A', 'C', 'F') supp: 0.5
rule: ('A', 'D', 'F') supp: 0.5
rule: ('A', 'G', 'F') supp: 0.25
rule: ('C', 'D', 'F') supp: 0.5
rule: ('C', 'G', 'F') supp: 0.25
rule: ('D', 'G', 'F') supp: 0.25
rule: ('A', 'C', 'D', 'F') supp: 0.5
rule: ('A', 'C', 'G', 'F') supp: 0.25
rule: ('A', 'D', 'G', 'F') supp: 0.25
rule: ('C', 'D', 'G', 'F') supp: 0.25
rule: ('A', 'C', 'D', 'G', 'F') supp: 0.25
rule: ('A', 'G') supp: 0.625
rule: ('C', 'G') supp: 0.625
rule: ('D', 'G') supp: 0.75
rule: ('A', 'C', 'G') supp: 0.625
rule: ('A', 'D', 'G') supp: 0.625
rule: ('C', 'D', 'G') supp: 0.625
rule: ('A', 'C', 'D', 'G') supp: 0.625
```

1.4 TASK 2

Каков размер области поиска наборов элементов, если ограничиваться только наборами, состоящими из простых элементов?

[14]:

```python
simple_items_n = 11

size = (2 ** simple_items_n) - 1
print(size)
```

```
2047
```

Предположив, что минимальный уровень поддержки = 7/8. Найдите все часто встречающиеся наборы элементов, состоящие только из элементов высокого уровня в таксономии. Имейте в виду, что если в транзакции появляется простой элемент, предполагается, что все его предки высокого уровня также присутствуют в транзакции.

[15]:

```python
itemsets = pd.Series(["2 3 6 7",
                      "1 3 4 8 11",
                      "3 9 11",
                      "1 5 6 7",
                      "1 3 8 10 11",
                      "3 5 7 9 11",
                      "4 6 8 10 11",
                      "1 3 5 8 11"])

for i in itemsets.keys():
    itemsets[i] = itemsets[i].split()
    for j in range(len(itemsets[i])):
        if 2 <= int(itemsets[i][j]) <= 5:
            itemsets[i][j] = "14"

        if 7 <= int(itemsets[i][j]) <= 11:
            itemsets[i][j] = "15"

enc = TransactionEncoder()

itemsets = enc.fit_transform(itemsets)

df = pd.DataFrame(itemsets, columns=enc.columns_)

frequent_itemset = apriori(df, min_support=7/8, use_colnames=True)

print(frequent_itemset)
```

```
   support  itemsets
0      1.0      (14)
1      1.0      (15)
2      1.0  (15, 14)
```