5. Association Rules Mining

Apriori algorithm

„Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis." [1 [https://en.wikipedia.org/wiki/Apriori_algorithm]]

Association rules

- Association rule: implication in the form X ⇒ Y, where X, Y is the subset of the set of items I
- Support: Percentag of occurrence of X U Y in a transaction database D
- Confidence: Ratio of the count of transactions in the set of transactions D that contain X and Y to the count of transactions in the set D that contain X.

Apriori algorithm - Observation

- Subset of each frequent itemset occurres at least as often as the original itemset.
- No itemset occurres more often than any of its subests.

Apriori - computation

- L_k: Set of frequent itemsets of size k (with min support)
- C_k: Set of candidate itemset of size k (potentially frequent itemsets)

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) **do**
  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
return $\cup_k L_k$;

Apriori - candidates generation

Input: L_{i-1} : set of frequent itemsets of size *i-1*

Output: C_i: set of candidate itemsets of size *i*

C_i = *empty set;*

for each itemset J in L_{i-1} **do**

  for each itemset K in L_{i-1} s.t. K<> J do

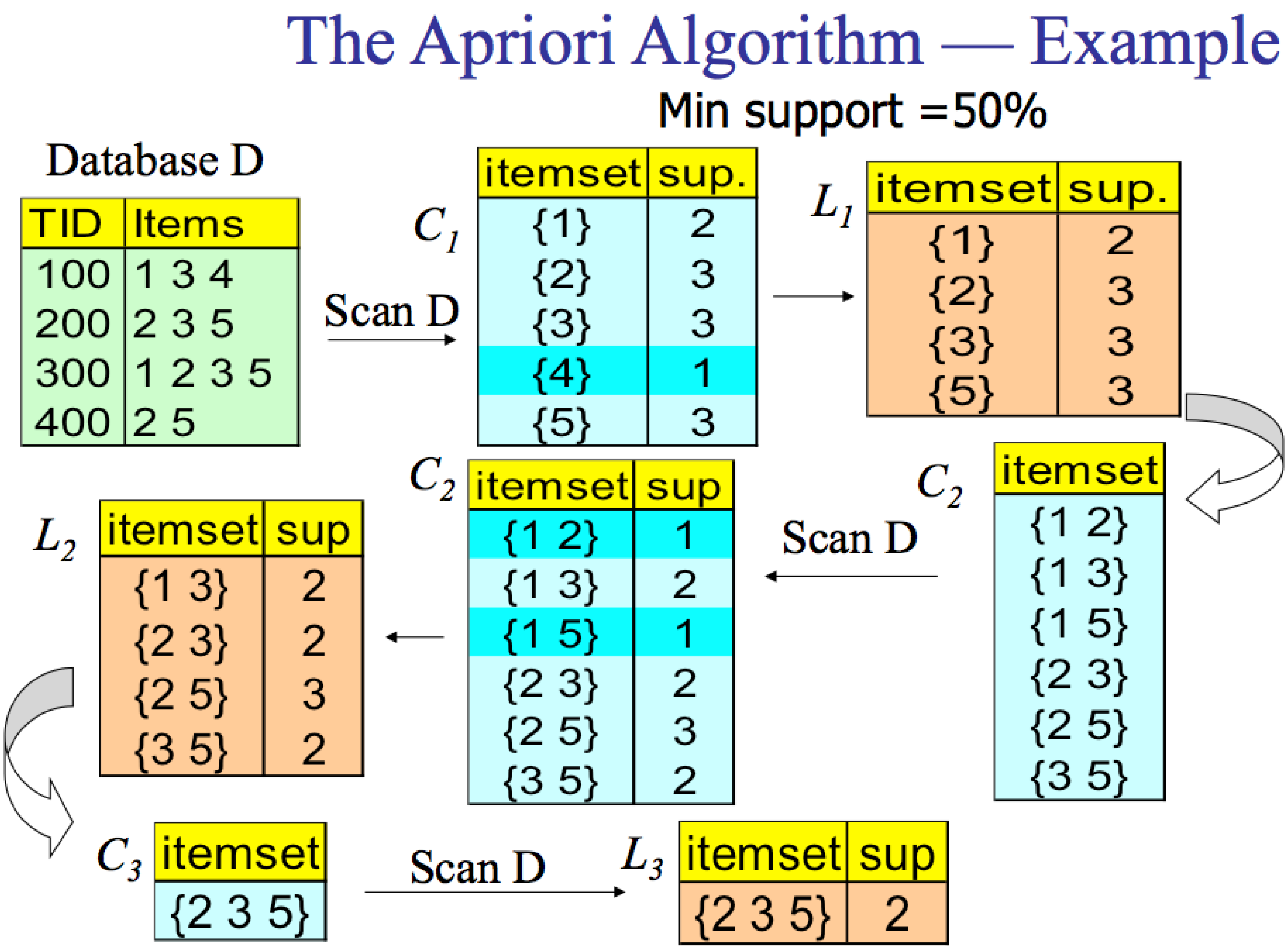
    if *i-2* of the elements in J and K are equal **then**

      if all subsets of $\{K \cup J\}$ are in L_{i-1} **then**

        $C_i = C_i \cup \{K \cup J\}$

return C_i;

Apriori - example



Generation of association rules

for each frequent itemset I **do**
  for each subset C of I **do**
    if (support(I) / support(I - C) >= minconf) **then**
      output the rule (I - C) $\Rightarrow$ C,
      with confidence = support(I) / support (I - C)
        and support = support(I)

Tools

- pandas - Python Data Analysis Library
  - Great for data munging and preparation, but less so for data analysis and modeling.
  - Helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R.
  - A fast and efficient DataFrame object for data manipulation with integrated indexing;
  - Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
    - http://pandas.pydata.org/ [http://pandas.pydata.org/]

Virtualenv

```
# https://virtualenv.pypa.io/en/stable/userguide/

# create env
virtualenv ddw-tutorial-5
# virtualenv --system-site-packages ddw-tutorial-5

# activate
source ddw-tutorial-5/bin/activate

# operations ...
pip install ...

# deactivate and remove
deactivate
rm -r ./ddw-tutorial-5
```

Installation

```
pip install pandas
```

Basic Operations

```
# Reading csv to a data frame
import pandas as pd
df = pd.read_csv('bank-data.csv')

# print head(tail) of the data frame
print(df.head()) # df.tail()

# select column
print(df[['age', 'car']])

# select by index
print(df.iloc[3:6,5:9])

# delete column
del df["id"]
print(df.head())

# discretize continous values to categorical values
df["income"] = pd.cut(df["income"],10)
print(df.head())
```

Output

```
# print head(tail) of the data frame
        id  age     sex      region   income married  children  car save_act  \
0  ID12101   48  FEMALE  INNER_CITY  17546.0      NO         1   NO       NO
1  ID12102   40    MALE        TOWN  30085.1     YES         3  YES       NO
2  ID12103   51  FEMALE  INNER_CITY  16575.4     YES         0  YES      YES
3  ID12104   23  FEMALE        TOWN  20375.4     YES         3   NO       NO
4  ID12105   57  FEMALE       RURAL  50576.3     YES         0   NO      YES

  current_act mortgage  pep
0          NO       NO  YES
1         YES      YES   NO
2         YES       NO   NO
3         YES       NO   NO
4          NO       NO   NO

# select column
     age  car
0     48   NO
1     40  YES
2     51  YES
3     23   NO
...
[600 rows x 2 columns]

# select by index
  married  children car save_act
3     YES         3  NO       NO
4     YES         0  NO      YES
5     YES         2  NO      YES

# delete column
   age     sex      region   income married  children  car save_act  \
```

```
0   48  FEMALE  INNER_CITY  17546.0      NO         1   NO       NO
1   40    MALE        TOWN  30085.1     YES         3  YES       NO
2   51  FEMALE  INNER_CITY  16575.4     YES         0  YES      YES
3   23  FEMALE        TOWN  20375.4     YES         3   NO       NO
4   57  FEMALE       RURAL  50576.3     YES         0   NO      YES

  current_act mortgage  pep
0          NO       NO  YES
1         YES      YES   NO
2         YES       NO   NO
3         YES       NO   NO
4          NO       NO   NO

# discretize continous values to categorical values
   age     sex      region                    income married  children  car  \
0   48  FEMALE  INNER_CITY  (16637.388, 22448.977]      NO         1   NO
1   40    MALE        TOWN  (28260.566, 34072.155]     YES         3  YES
2   51  FEMALE  INNER_CITY  (10825.799, 16637.388]     YES         0  YES
3   23  FEMALE        TOWN  (16637.388, 22448.977]     YES         3   NO
4   57  FEMALE       RURAL  (45695.333, 51506.922]     YES         0   NO

  save_act current_act mortgage  pep
0       NO          NO       NO  YES
1       NO         YES      YES   NO
2      YES         YES       NO   NO
3       NO         YES       NO   NO
4      YES          NO       NO   NO
```

Apriori algorithm implementation

```
from collections import Counter

def frequentItems(transactions, support):
    counter = Counter()
    for trans in transactions:
        counter.update(frozenset([t]) for t in trans)
    return set(item for item in counter if counter[item]/len(transactions) >= support), counter

def generateCandidates(L, k):
    candidates = set()
    for a in L:
        for b in L:
            union = a | b
            if len(union) == k and a != b:
                candidates.add(union)
    return candidates

def filterCandidates(transactions, itemsets, support):
    counter = Counter()
    for trans in transactions:
        subsets = [itemset for itemset in itemsets if itemset.issubset(trans)]
        counter.update(subsets)
    return set(item for item in counter if counter[item]/len(transactions) >= support), counter

def apriori(transactions, support):
    result = list()
    resultc = Counter()
    candidates, counter = frequentItems(transactions, support)
    result += candidates
    resultc += counter
    k = 2
    while candidates:
        candidates = generateCandidates(candidates, k)
        candidates,counter = filterCandidates(transactions, candidates, support)
        result += candidates
        resultc += counter
        k += 1
    resultc = {item:(resultc[item]/len(transactions)) for item in resultc}
    return result, resultc
```

Example

```
dataset = [
    ['bread', 'milk'],
    ['bread', 'diaper', 'beer', 'egg'],
    ['milk', 'diaper', 'beer', 'cola'],
    ['bread', 'milk', 'diaper', 'beer'],
    ['bread', 'milk', 'diaper', 'cola'],
]

frequentItemsets, supports = apriori(dataset, 0.1)
for f in frequentItemsets:
    print("{} - {}".format(f,supports[f]))
```

```
frozenset({'cola'}) - 0.4
frozenset({'beer'}) - 0.6
frozenset({'milk'}) - 0.8
frozenset({'bread'}) - 0.8
frozenset({'diaper'}) - 0.8
frozenset({'bread', 'beer'}) - 0.4
frozenset({'diaper', 'milk'}) - 0.6
frozenset({'bread', 'milk'}) - 0.6
frozenset({'diaper', 'beer'}) - 0.6
frozenset({'bread', 'diaper'}) - 0.6
frozenset({'beer', 'milk'}) - 0.4
frozenset({'milk', 'cola'}) - 0.4
frozenset({'diaper', 'cola'}) - 0.4
```

```
frozenset({'bread', 'diaper', 'milk'}) - 0.4
frozenset({'diaper', 'milk', 'cola'}) - 0.4
frozenset({'bread', 'diaper', 'beer'}) - 0.4
frozenset({'diaper', 'beer', 'milk'}) - 0.4
```

Tasks

Implementation

- Complete the implementation about the generation of association rules
- Allow setting of minimum confidence value for each rule
  - the output can be printed to the console
    - provide information about antecedent, consequent, support and confidence of the rule
    - e.g. {a=1,b=2} -> {c=3}, support=0.5, confidence=0.3
    - optionally allow sorting by support, confidence, or rule length
- Implement other metric of your choice [1 [https://en.wikipedia.org/wiki/Association_rule_learning]]
  - $lift(X \rightarrow Y) = \frac{support(X \cup Y)}{support(X) \times support(Y)}$
  - $conviction(X \rightarrow Y) = \frac{1 - support(Y)}{1 - confidence(X \rightarrow Y)}$

Data Analysis

- Perform association rules mining on the example dataset containing bank data.
  - bank-data.zip
- Experiment with different settings of metrics (confidence, optionally lift and conviction). Which settings and metric works best for your use case.
- Try another dataset from UCI repository
  - e.g. subset of datasets in CSV http://repository.seasr.org/Datasets/UCI/csv/ [http://repository.seasr.org/Datasets/UCI/csv/]

Code Example

```
def genereateRules(frequentItemsets, supports, minConfidence):
    ...
    print(" .... ")

# bank dataset preprocessing
import pandas as pd
df = pd.read_csv("./bank-data.csv")
del df["id"]
df["income"] = pd.cut(df["income"],10)
dataset = []
for index, row in df.iterrows():
    row = [col+"="+str(row[col]) for col in list(df)]
    dataset.append(row)
frequentItemsets, supports = apriori(dataset, 0.3)
genereateRules(frequentItemsets, supports, 0.5)

# ...
# {'car=YES'} => married=YES, 0.3233333333333333, 0.6554054054054054
# ...
# {'married=YES', 'save_act=YES'} => current_act=YES, 0.3433333333333333, 0.7436823104693141
# ...
```

/mnt/www/courses/MI-DDW.16/data/pages/tutorials/05/start.txt · Poslední úprava: 2017/04/23 08:19 autor: kuchajar