Data Mining - Homework 2

Robert-Andrei Damian and Alice De Schutter

Discovery of Frequent Itemsets and Association Rules

November 19, 2021

Task

(1) Write a program that implements the A-Priori algorithm for finding frequent item-sets with support at least s in a dataset of sales transactions.
(2) Bonus task: Develop and implement an algorithm for generating association rules between frequent item-sets discovered with the A-Priori algorithm. The rules must have support at least s and confidence at least c, where s and c are given as input parameters.

1 Detailed information

Apache Spark was used.

- Our implementation of (1) does the following:
  1. Counts frequent singleton occurrences with support s.
  2. Recursively calls a function called "gather_all_groups()" which counts frequent doubletons, triple-tons... etc. with support size s. The creation of larger groups is done with a "helper" DataFrame which consists of the frequent singletons joined with the baskets in which they are included in.
     Result is first shown with a spark DataFrame where each item is represented by a group (column: "group"; the occurrences are stored in column: "occurrences").
     Result is then also shown in a second DataFrame which displays the group_sizes that exist with the given support size s, and their associated group_size counts (i.e. "there exists 9 doubletons").
  3. Cached solutions for values of support-size $s = \{700, 1000, 2000\}$ can be found in the "data" folder.
- **BONUS:** Our implementation of (2) does the following:
  1. Finds all of the possible association rules which correspond to the frequent item groups (double-tons, triple-tons etc.). This is achieved with the itertools permutations function.
  2. Confidence is computed for all of these association rules. The rules which have confidence c or higher are kept, others are discarded.

Instructions on how to build and run the program

– pip install pyspark

– spark-submit frequent_items.py

Figure 1 shows information about optional command line parameters:

```
optional arguments:
  -h, --help            show this help message and exit
  --support-size SUPPORT_SIZE
                        Set the minimum support size
  --disable-cache       Do not use cached data
  --confidence-threshold CONFIDENCE_THRESHOLD
                        Set the confidence_threshold
```

Figure 1: Optional arguments

Since the implementation of our A-priori algorithm is slow for certain values of support values s (i.e s = 700), we included the option to enable/ disable the cache. This means that the program can use previously cached data in order to perform the Bonus task. By default, the cache is enabled.

2 Results

The results shown below are found with command line parameters:

- *support_size = 1000* (default value)
- *confidence_threshold = 0.5* (default value)
- - - disable-cache

Task 1:

```
Checking for 1010228 pairs. Aiming for 1000 support!
Checking for 6530628 pairs. Aiming for 1000 support!
Identified 9 new supported groups!
Checking for 19432 pairs. Aiming for 1000 support!
Identified 1 new supported groups!
Checking for 0 pairs. Aiming for 1000 support!
Identified 0 new supported groups!
+-----+-----------+
|group|occurrences|
+-----+-----------+
|[471]|       2894|
|[496]|       1428|
|[392]|       2420|
|[540]|       1293|
|[897]|       1935|
|[623]|       1845|
|[516]|       1544|
| [31]|       1666|
|[580]|       1667|
| [85]|       1555|
|[458]|       1124|
|[883]|       4902|
|[804]|       1315|
|[970]|       2086|
|[472]|       2125|
|[853]|       1804|
|[296]|       2210|
|[513]|       1287|
|[322]|       1154|
| [78]|       2471|
+-----+-----------+
only showing top 20 rows

+----------+-----------------+
|group_size|count(group_size)|
+----------+-----------------+
|         1|              375|
|         2|                9|
|         3|                1|
+----------+-----------------+
```

Figure 2: Spark DataFrame output for task 1

Task 2 (bonus):

group	premise	conclusion	group_support	premise_support	confidence
[39, 704, 825]	[704, 825]	[39]	1035	1102	0.9392014519056261
[39, 704, 825]	[39, 704]	[825]	1035	1107	0.9349593495934959
[39, 704, 825]	[39, 825]	[704]	1035	1187	0.8719460825610783
[39, 704]	[704]	[39]	1107	1794	0.617056856187291
[704, 825]	[704]	[825]	1102	1794	0.6142697881828316

Figure 3: Spark DataFrame output for task 2