Homework 2 - Report

William Berg - willb@kth.se
Konrad Grudzinski - kjgr@kth.se

Implementation

Most of the description of our solutions is already in the code in the form of comments so we won't get in to details here. The requested functionality of the homework is implemented in the `functions.py` file. Here we have created a function called `a_priori_algorithm` which takes a list of the transactions and a support threshold and returns a collection of the frequent itemsets along with their support. This function has two helper functions:

1. `apriori_pass`: Does the k:th (given as a parameter) pass of the apriori algorithm and returns a list of frequent items of length k.
2. `filter_candidates`: Takes a list of candidate frequent items and filters out all candidate items that are not frequent enough according to the support threshold.

These functions address the first sub-problem. For sub-problem 2, there is one function of interest: `generate_association_rules`. This function takes the output of sub-problem 1 (i.e. the frequent items) and generates association rules, of confidence larger than c (given as a parameter). The procedure is roughly the same as explained in the course book (MMDS).

These functions are called to execute the program itself in the `main.py` file, which is responsible for handling the data and running the functions explained above together to produce a result. Relevant results are displayed for viewing by the `print_results` function.

Running the Program

As stated earlier the primary code is written in the `functions.py` file. These functions are then imported in the `main.py` file, which runs the program itself:

```
$ python main.py
```

Keep in mind that before running the program the data needs to be available at the path `data/transactions.dat`. The dataset can be retrieved here. By default, the support threshold is set to 1000 (1% of the baskets) and confidence is set to 0.5. Verbose is also set to `True`, which means that the resulting print out of the program will be more verbose (prints all itemsets found) than if it were set to `False` (only prints how many were found). These values can be adjusted in the `main.py` file.

Results

With regard to execution time, sub-problem 1 (the apriori algorithm) consistently takes about 10 seconds to execute; generating association rules, meanwhile, finishes basically instantly (usually takes around $1e-5$ seconds).

The results are easy to understand by just looking at the output of the program. The first example output below shows the output when `verbose` is set to `True`; in the second example `verbose` is set to `False`.

Verbose output:

```
Results with a support of 1000 and a confidence of 0.5:

Frequent itemsets:
Found frequent itemsets in: 10.69 seconds

Found 375 frequent itemsets of length 1
[25, 52, 240, 274, 368, 448, 538, 561, 630, 687, 775, 825, 834, 39, 120, 205,
401, 581, 704, 814, 35, 674, 733, 854, 950, 422, 449, 857, 895, 937, 964, 229,
283, 294, 381, 708, 738, 766, 853, 883, 966, 978, 104, 143, 569, 620, 798, 185,
214, 350, 529, 658, 682, 782, 809, 947, 970, 227, 390, 71, 192, 208, 279, 280,
496, 530, 597, 618, 675, 720, 914, 932, 183, 217, 276, 653, 706, 878, 161, 175,
177, 424, 490, 571, 623, 795, 910, 960, 125, 130, 392, 461, 862, 27, 78, 900,
921, 147, 411, 572, 579, 778, 803, 266, 290, 458, 523, 614, 888, 944, 43, 70,
204, 334, 480, 513, 874, 151, 504, 890, 73, 310, 419, 469, 722, 810, 844, 846,
918, 967, 326, 403, 526, 774, 788, 789, 975, 116, 198, 201, 171, 541, 701, 805,
946, 471, 487, 631, 638, 678, 735, 780, 935, 17, 242, 758, 763, 956, 145, 385,
676, 790, 792, 885, 522, 617, 859, 12, 296, 354, 548, 684, 740, 841, 210, 346,
477, 605, 829, 884, 234, 460, 649, 746, 600, 28, 157, 5, 115, 517, 736, 744, 919,
196, 489, 494, 641, 673, 362, 591, 31, 58, 181, 472, 573, 628, 651, 111, 154,
168, 580, 632, 832, 871, 988, 72, 981, 10, 132, 21, 32, 54, 239, 348, 100, 500,
48, 126, 319, 639, 765, 521, 112, 140, 285, 387, 511, 594, 93, 583, 606, 236,
952, 90, 593, 941, 122, 718, 1, 423, 516, 6, 69, 797, 913, 577, 110, 509, 611,
995, 343, 527, 33, 336, 989, 97, 574, 793, 598, 427, 470, 37, 992, 55, 897, 275,
51, 259, 45, 162, 378, 534, 906, 912, 576, 373, 716, 546, 665, 963, 349, 8, 197,
413, 749, 823, 94, 982, 984, 515, 692, 694, 567, 57, 800, 812, 41, 414, 923, 377,
752, 991, 998, 899, 710, 867, 170, 438, 563, 357, 332, 361, 322, 928, 75, 486,
440, 38, 784, 265, 686, 540, 468, 663, 819, 886, 429, 843, 129, 578, 510, 68,
860, 4, 887, 309, 804, 325, 826, 394, 707, 105, 815, 948, 308, 661, 634, 351,
405, 688, 949, 163, 893, 335, 173, 258, 85, 450, 428, 550, 769, 554, 366, 820,
207]

Found 9 frequent itemsets of length 2
[(39, 704), (39, 825), (704, 825), (227, 390), (789, 829), (368, 829), (217,
346), (368, 682), (390, 722)]
```

```
Found 1 frequent itemsets of length 3
[(39, 704, 825)]

Total number of frequent itemsets found: 385

Association rules:
Generated association rules in: 3e-05 seconds

704 → 39 (confidence: 0.62)
704 → 825 (confidence: 0.61)
227 → 390 (confidence: 0.58)
(704, 825) → 39 (confidence: 0.94)
(39, 825) → 704 (confidence: 0.87)
(39, 704) → 825 (confidence: 0.93)

Total number of association rules found: 6
```

Non-verbose output:

```
Results with a support of 1000 and a confidence of 0.5:

Frequent itemsets:
Found frequent itemsets in: 10.65 seconds

Found 375 frequent itemsets of length 1
Found 9 frequent itemsets of length 2
Found 1 frequent itemsets of length 3
Total number of frequent itemsets found: 385

Association rules:
Generated association rules in: 4e-05 seconds

704 → 39 (confidence: 0.62)
704 → 825 (confidence: 0.61)
227 → 390 (confidence: 0.58)
(704, 825) → 39 (confidence: 0.94)
(39, 825) → 704 (confidence: 0.87)
(39, 704) → 825 (confidence: 0.93)
```

```
Total number of association rules found: 6
```