Homework 2 Report – Frequent Itemsets & Association Rules

Data Mining – ID2222 – Homework Group 23

Adriano Mundo (mundo@kth.se), Edoardo Realini (erealini@kth.se)

November 16, 2020

1. Introduction

The goal of the *homework assignment 2* was to implement the stages for discovering association rules between item-sets in a dataset. Indeed, the main problem is the discovery of frequent item-sets, also known as the "discovery of association rules". The objective is to identify items that frequently occur together in sets of items. Association rules have many applications in real-life scenarios and one of the main algorithms used in this context is the A-Priori algorithm developed by Agrawal et. al [1].
The following report explains the implementation and how to run the code for the two problems of: finding frequent item-sets given a support s, by leveraging on the A-Priori algorithm and generating association rules with confidence at least c from the previously founded item-sets. The implementation has been done by using Python and we leveraged on the knowledge from the book [2] and lectures [3].

2. Implementation

The code is organized in classes, each code file attached with this report represents a class. For each task of the assignment there is a class with the respective functions. They are used for running the pipeline of the A-Priori algorithm in the *main.py* file. In this section we will explain what each file contains and what is used for

- ***Apriori.py**:* it contains the class *Apriori* which is responsible for implementing the A-Priori algorithm. It creates objects which store filename, baskets derived from the dataset, the candidates and filtered dictionaries containing the items with their supports generated at each step and the support selected by the user for the pipeline. It has different functions:
  - *load_dataset:* the function reads the file line by line, removes blank spaces, new lines and stores the baskets in a list of lists. Hence, we have a list of baskets where each element is a list that represents a single basket.
  - *generate_candidates_ck:* the function is responsible, depending on the step of the iteration, of generating and storing in the *Apriori* object the dictionaries of candidate items with their support at each stage of the pipeline. It uses another function of this class. It takes as input the baskets, the item-sets generated at the previous iteration and the stage of the computation.
  - *filter_candidates*: the function by taking as input a candidate dictionary and the support, it filters the candidates and returns a dictionary with the items which occurrences are greater than the specified support.
  - *generate_support_ck*: the function takes as input the baskets, the frequent items-set from the previous iteration and the step of the pipeline. It is used as a support function for the candidate's generation. If we are in the first stage, the function iterates over baskets and counts the frequency of each item. Otherwise, we are in the following stages, hence we

iterate over baskets with a smarter approach. A naïve solution could be to iterate over baskets and all possible combinations, but the result would be poor. Thus, we derive a list of all the items from the previous step and we iterate over the baskets. For each basket we do an intersection between the basket and the list of items. In this way, we derive the common elements and then we can compute all the possible combinations. If we are in the second stage, we just create a dictionary with the new occurrences while if we are in later stages, before doing that we check if every combination of the subsets is present or not in the list of subsets from the previous iteration. By doing so, we are sure to have only the right candidates' tuples at the end of the computation in respect with the intuition coming from [1].

- *filtered_plot:* the function displays a plot which shows the distribution of frequent items over each stage
- *candidates_plot:* the function displays a plot which shows the distribution of candidate items over each stage
- *timelapsed_plot:* the function displays a plot which shows the time distribution for each stage

- ***AssociationRules.py****:* this file contains the implementation of the class *AssociationRules* that is responsible for finding association rules among given item-sets.
  The main methods of this class are the following:
  - *generate_all_rules:* given in input a confidence threshold value, computes all the rules from the input item-sets. The input item-sets are given when the object is created. For each itemset, by calling *generate_association_rule* generates all the possible subsets and hence each rule that comes from each subset. Subsequently calls *filter_association_rules* to get only the rules that respect the confidence threshold. This method returns a list of rules. Each rule is itself a list built in the following way: the first element of the list is the set of items that composes the antecedent of the rule, the second element of the list is again a set of items that composes the consequence of the rule, last in third position of the list we store the confidence of the particular rule.
  - *generate_association_rule:* the function, given one single itemset as input, the function generates all the possible association rules in the form of a list of sets. The itemset given as input is a set itself. In the first set we have the antecedent, in the second set we have the consequence in the following form: [set(antecedent), set(consequence)].
  - *filter_association_rules:* the method iterates over a list of given association rules and finds the ones that respect the confidence value given as input.
  - *compute_confidence:* it computes the confidence of a given rule in input by using the input data of item-sets. In the moment of construction of an object of the class *AssociationRules*, the item-sets given as a list of dictionaries (one per each cardinality value) is flattened to one single dictionary and kept as an attribute. By leveraging on this dictionary, the computation for the confidence of a single rule is immediate.

- ***main.py****:* this file contains the call functions for showing the execution of algorithms for mining association rules and plotting some graphs. It loads the file, reads the command arguments specified by the user and creates the *Apriori* object. After that, by using the functions of the two classes, run the entire pipeline while printing some useful information. In addition to the results, it also shows some plots about candidates' size, filtered candidates' size and elapsed time. With respect to the phase of generation of association rules, our class perfectly integrates with the previous stages. First, we instantiate one new object of the class *AssociationRules* by passing to the constructor the data structure filled with item-sets. The single-entry point of the class is the method *generate_all_rules*. The results are then printed.

The program has been tested by using the dataset provided by the course teachers in Canvas.

3. How to Run the Code

In this section it is explained how to run the code provided with this short report. The recommended environment is Ubuntu >= 18.04 LTS. It is also requested to have installed on the machine Python3.

The pipeline can be run by using the following command in the directory *src*.

```
$ python3 main.py
```

It is possible to use command line arguments to set specific values for the n-tuples to generate, the support, the confidence values and the displaying of plots. The default configuration is the one below.

```
$ python3 main.py -k 6 -s 700 -c 0.9 -p False
```

4. Results

By running the program, it is possible to see the following output which shows each step of the pipeline correctly executed. Below, it is possible to find the output with the plotting option set.

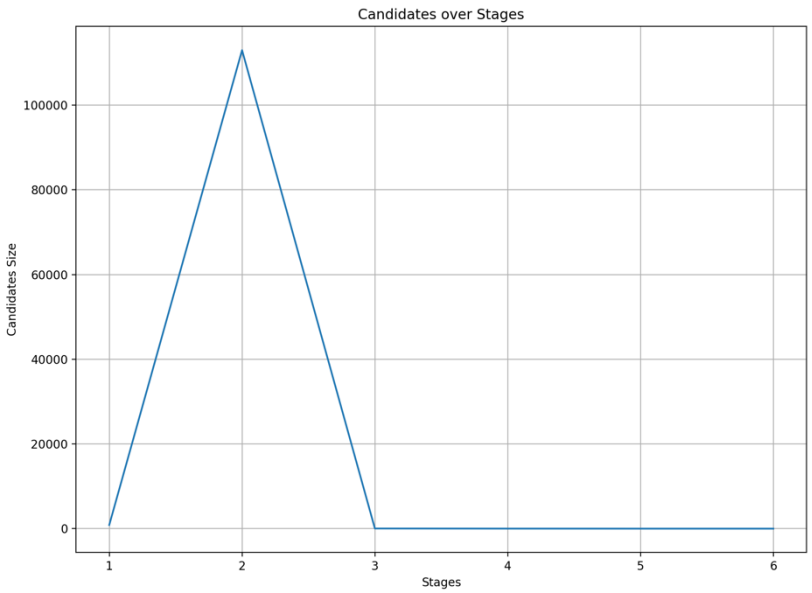
```
A-propri algorithm for Association Rule Mining
Adriano Mundo & Edoardo Realini
KTH Royal Institute of Technology - 2020

Parameter values:
Number of k-tuples:  6
Support:  700
Confidence threshold:  0.9
Plotting:  False

Start Apriori Pipeline

L2:  {(448, 538): 723, (39, 704): 1107, (39, 825): 1187, (704, 825): 1102, (529, 782): 862,
(227, 390): 1049, (496, 626): 761, (175, 910): 713, (571, 623): 783, (571, 795): 838, (571,
853): 793, (623, 795): 805, (623, 853): 791, (795, 853): 806, (392, 862): 881, (411, 803):
826, (208, 290): 803, (208, 458): 796, (208, 888): 829, (208, 969): 806, (290, 458): 786,
(290, 888): 826, (290, 969): 797, (458, 888): 796, (458, 969): 788, (888, 969): 810, (192,
935): 775, (471, 678): 810, (789, 829): 1194, (296, 829): 736, (392, 489): 866, (368, 829):
1194, (72, 541): 846, (368, 937): 701, (6, 676): 769, (21, 793): 765, (21, 857): 729, (793,
857): 734, (529, 598): 943, (598, 782): 800, (214, 354): 775, (12, 132): 785, (538, 878):
767, (57, 937): 782, (33, 217): 852, (33, 283): 845, (33, 346): 844, (33, 515): 824, (217,
283): 926, (217, 346): 1336, (217, 515): 843, (283, 346): 910, (283, 515): 835, (346, 515):
849, (69, 362): 757, (923, 947): 859, (75, 438): 772, (75, 684): 797, (438, 684): 825, (21,
413): 756, (413, 793): 766, (413, 857): 731, (70, 684): 860, (70, 765): 789, (70, 819):
798, (684, 765): 812, (684, 819): 800, (765, 819): 787, (32, 947): 720, (788, 956): 733,
(470, 534): 764, (470, 995): 754, (534, 995): 750, (494, 862): 720, (75, 775): 787, (368,
682): 1193, (368, 494): 860, (494, 682): 714, (368, 692): 928, (12, 722): 740, (227, 722):
995, (390, 722): 1042, (675, 886): 823, (401, 571): 708, (217, 947): 716, (39, 145): 734,
(471, 960): 935}
```

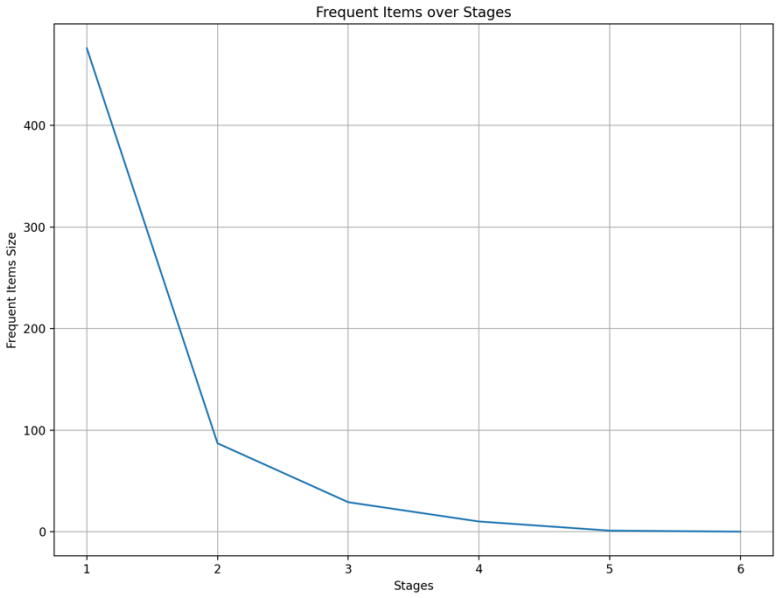
```
L3:  {(39, 704, 825): 1035, (571, 623, 795): 704, (571, 623, 853): 713, (571, 795, 853):
715, (623, 795, 853): 720, (208, 290, 458): 751, (208, 290, 888): 764, (208, 290, 969):
764, (208, 458, 888): 757, (208, 458, 969): 755, (208, 888, 969): 771, (290, 458, 888):
750, (290, 458, 969): 748, (290, 888, 969): 765, (458, 888, 969): 756, (529, 598, 782):
737, (33, 217, 283): 793, (33, 217, 346): 802, (33, 217, 515): 789, (33, 283, 346): 802,
(33, 283, 515): 786, (33, 346, 515): 797, (217, 283, 346): 827, (217, 283, 515): 799, (217,
346, 515): 809, (283, 346, 515): 806, (21, 413, 793): 700, (70, 684, 819): 700, (227, 390,
722): 907}
L4:  {(208, 290, 458, 888): 720, (208, 290, 458, 969): 719, (208, 290, 888, 969): 734,
(208, 458, 888, 969): 725, (290, 458, 888, 969): 720, (33, 217, 283, 346): 766, (33, 217,
283, 515): 755, (33, 217, 346, 515): 764, (33, 283, 346, 515): 763, (217, 283, 346, 515):
773}
L5:  {(33, 217, 283, 346, 515): 732}
L6:  {}

End Apriori Pipeline, time elapsed:  6.067410230636597  sec.


Association rules generation
Confidence threshold set to:  0.9

       Rule            Confidence

 {969} --> {208}      0.9493521790341578
 {969} --> {290}      0.9387514723203769
 {969} --> {458}      0.928150765606596
 {969} --> {888}      0.9540636042402827
 {704, 825} --> {39}         0.9392014519056261
 {704, 39} --> {825}         0.9349593495934959
 {571, 623} --> {853}        0.9106002554278416
 {853, 623} --> {571}        0.9013906447534766
 {571, 853} --> {795}        0.9016393442622951
 {853, 623} --> {795}        0.9102402022756005
 {208, 458} --> {290}        0.9434673366834171
 {208, 290} --> {458}        0.9352428393524284
 {458, 290} --> {208}        0.955470737913486
 {208, 290} --> {888}        0.9514321295143213
 {208, 888} --> {290}        0.9215922798552473
 {888, 290} --> {208}        0.9249394673123487
 {208, 969} --> {290}        0.9478908188585607
 {208, 290} --> {969}        0.9514321295143213
 {969, 290} --> {208}        0.958594730238394
 {208, 458} --> {888}        0.9510050251256281
 {208, 888} --> {458}        0.9131483715319663
 {888, 458} --> {208}        0.9510050251256281
 {208, 969} --> {458}        0.9367245657568238
 {208, 458} --> {969}        0.9484924623115578
 {969, 458} --> {208}        0.9581218274111675
 {969} --> {208, 888}        0.9081272084805654
 {208, 969} --> {888}        0.956575682382134
 {208, 888} --> {969}        0.9300361881785284
 {888, 969} --> {208}        0.9518518518518518
 {888, 458} --> {290}        0.9422110552763819
 {888, 290} --> {458}        0.9079903147699758
 {458, 290} --> {888}        0.9541984732824428
 {458, 290} --> {969}        0.9516539440203562
 {969, 458} --> {290}        0.949238578680203
 {969, 290} --> {458}        0.9385194479297365
 {969} --> {888, 290}        0.901060070671378
 {888, 969} --> {290}        0.9444444444444444
 {888, 290} --> {969}        0.9261501210653753
 {969, 290} --> {888}        0.9598494353826851
 {888, 969} --> {458}        0.9333333333333333
 {888, 458} --> {969}        0.949748743718593
 {969, 458} --> {888}        0.9593908629441624
 {598, 782} --> {529}        0.92125
```

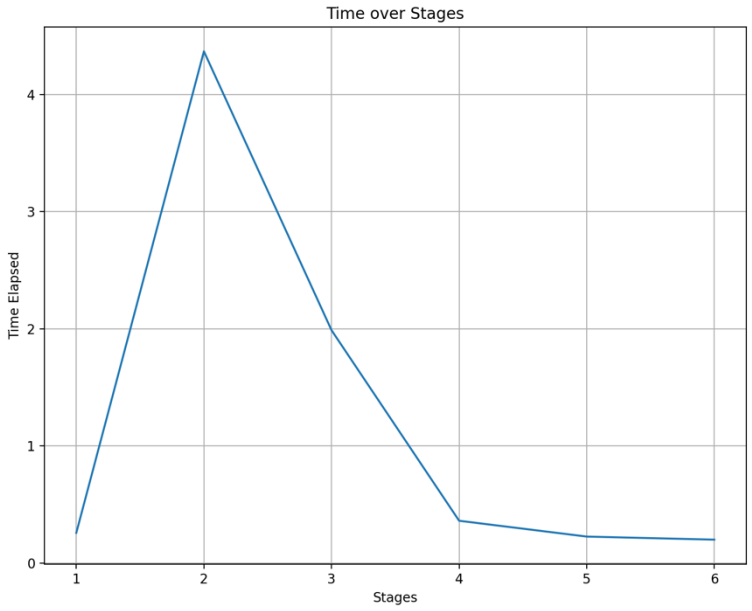
```
{33, 283} --> {217}   0.9384615384615385
 {33, 217} --> {283}         0.9307511737089202
 {33, 346} --> {217}         0.9502369668246445
 {33, 217} --> {346}         0.9413145539906104
 {33, 515} --> {217}         0.9575242718446602
 {33, 217} --> {515}         0.926056338028169
 {217, 515} --> {33}         0.9359430604982206
 {33, 346} --> {283}         0.9502369668246445
 {33, 283} --> {346}         0.9491124260355029
 {33, 515} --> {283}         0.9538834951456311
 {283, 515} --> {33}         0.9413173652694611
 {33, 283} --> {515}         0.9301775147928995
 {33, 346} --> {515}         0.9443127962085308
 {33, 515} --> {346}         0.9672330097087378
 {346, 515} --> {33}         0.9387514723203769
 {346, 283} --> {217}        0.9087912087912088
 {217, 515} --> {283}        0.9478054567022538
 {283, 515} --> {217}        0.9568862275449102
 {217, 515} --> {346}        0.9596678529062871
 {346, 515} --> {217}        0.9528857479387515
 {346, 515} --> {283}        0.9493521790341578
 {283, 515} --> {346}        0.9652694610778443
 {793, 21} --> {413}         0.9150326797385621
 {793, 413} --> {21}         0.9138381201044387
 {21, 413} --> {793}         0.9259259259259259
 {722, 227} --> {390}        0.9115577889447236
 {208, 458} --> {888, 290}   0.9045226130653267
 {458, 290} --> {208, 888}   0.916030534351145
 {888, 458} --> {208, 290}   0.9045226130653267
 {208, 458, 290} --> {888}   0.9587217043941412
 {208, 458, 888} --> {290}   0.9511228533685601
 {208, 290, 888} --> {458}   0.9424083769633508
 {888, 458, 290} --> {208}   0.96
 {208, 458} --> {969, 290}   0.9032663316582915
 {458, 290} --> {208, 969}   0.9147582697201018
 {969, 458} --> {208, 290}   0.9124365482233503
 {969, 290} --> {208, 458}   0.9021329987452948
 {208, 458, 290} --> {969}   0.9573901464713716
 {208, 969, 458} --> {290}   0.952317880794702
 {208, 969, 290} --> {458}   0.9410994764397905
 {969, 458, 290} --> {208}   0.9612299465240641
 {208, 969} --> {888, 290}   0.9106699751861043
 {208, 290} --> {888, 969}   0.9140722291407223
 {969, 290} --> {208, 888}   0.9209535759096612
 {888, 969} --> {208, 290}   0.9061728395061729
 {208, 969, 290} --> {888}   0.9607329842931938
 {208, 969, 888} --> {290}   0.9520103761348897
 {208, 290, 888} --> {969}   0.9607329842931938
 {888, 969, 290} --> {208}   0.9594771241830066
 {208, 458} --> {888, 969}   0.9108040201005025
 {969, 458} --> {208, 888}   0.9200507614213198
 {888, 458} --> {208, 969}   0.9108040201005025
 {208, 969, 458} --> {888}   0.9602649006622517
 {208, 969, 888} --> {458}   0.940337224383917
 {208, 458, 888} --> {969}   0.9577278731836195
 {888, 969, 458} --> {208}   0.958994708994709
 {888, 458} --> {969, 290}   0.9045226130653267
 {458, 290} --> {888, 969}   0.916030534351145
 {969, 458} --> {888, 290}   0.9137055837563451
 {969, 290} --> {888, 458}   0.903387703889586
 {888, 458, 290} --> {969}   0.96
 {888, 969, 458} --> {290}   0.9523809523809523
 {888, 969, 290} --> {458}   0.9411764705882353
 {969, 458, 290} --> {888}   0.9625668449197861
 {33, 346} --> {217, 283}    0.9075829383886256
 {33, 283} --> {217, 346}    0.906508875739645
```

```
{33, 346, 283} --> {217}     0.9551122194513716
 {33, 346, 217} --> {283}    0.9551122194513716
 {33, 283, 217} --> {346}    0.9659520807061791
 {217, 346, 283} --> {33}    0.9262394195888755
 {33, 515} --> {217, 283}    0.9162621359223301
 {283, 515} --> {33, 217}    0.9041916167664671
 {283, 33, 515} --> {217}    0.960559796437659
 {33, 515, 217} --> {283}    0.9569074778200254
 {283, 217, 515} --> {33}    0.9449311639549437
 {33, 283, 217} --> {515}    0.9520807061790668
 {33, 346} --> {217, 515}    0.9052132701421801
 {33, 515} --> {217, 346}    0.9271844660194175
 {217, 515} --> {33, 346}    0.9062870699881376
 {33, 346, 515} --> {217}    0.958594730238394
 {33, 346, 217} --> {515}    0.9526184538653366
 {33, 515, 217} --> {346}    0.9683143219264893
 {217, 346, 515} --> {33}    0.9443757725587144
 {33, 515} --> {346, 283}    0.9259708737864077
 {283, 515} --> {33, 346}    0.9137724550898204
 {33, 346} --> {283, 515}    0.9040284360189573
 {33, 283} --> {346, 515}    0.9029585798816568
 {33, 346, 515} --> {283}    0.9573400250941029
 {283, 33, 515} --> {346}    0.9707379134860051
 {283, 346, 515} --> {33}    0.9466501240694789
 {33, 346, 283} --> {515}    0.9513715710723192
 {217, 515} --> {346, 283}   0.9169632265717675
 {346, 515} --> {217, 283}   0.9104829210836278
 {283, 515} --> {217, 346}   0.925748502994012
 {217, 346, 515} --> {283}   0.9555006180469716
 {283, 217, 515} --> {346}   0.967459324155194
 {283, 346, 515} --> {217}   0.9590570719602978
 {217, 346, 283} --> {515}   0.9347037484885127
 {33, 515, 217} --> {346, 283}      0.9277566539923955
 {33, 346, 515} --> {217, 283}      0.918444165621079
 {283, 33, 515} --> {217, 346}      0.9312977099236641
 {33, 346, 217} --> {283, 515}      0.912718204488778
 {33, 283, 217} --> {346, 515}      0.9230769230769231
```

In the figures below, it represented the distribution of candidates, frequent item-sets and time over the stages of the pipeline.







References

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. *Fast Algorithms for Mining Association Rules in Large Databases.* In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 487–499.

[2] Chapter 6 in *Mining of Massive Datasets,* by Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman, 2nd edition, Cambridge University Press, 2014

[3] Frequent Itemsets lecture, ID2222, Data Mining, KTH Royal Institute of Technology