CS570 Homework 1: Pattern Mining

Daniela Chanci Arrubla

February 21, 2022

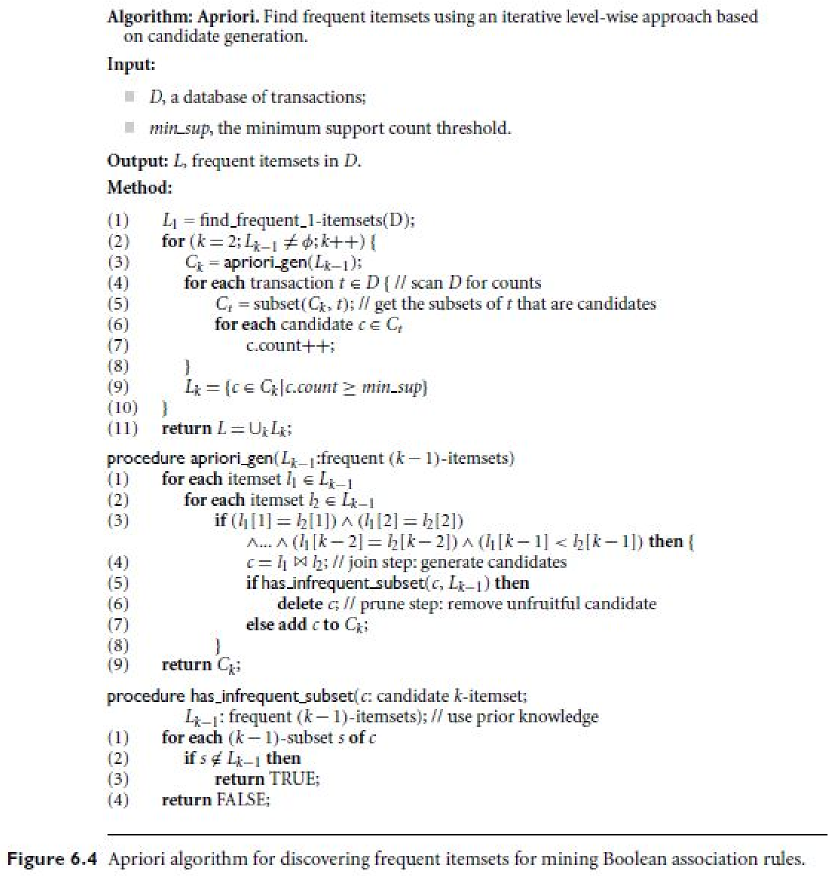
1 Algorithms

In this section, I present a brief description of the algorithms used from Step 4 to Step 6 from the homework.

1.1 Step 4

For the mining of frequent patterns for each topic, I selected the Apriori algorithm. I used the slides from the class and the theory from the book to create the code [1]. More specifically, I used the pseudocode provided in page 253 of the book as a guide (See Figure 1). Basically, this algorithm can be described as follows: Apriori is an iterative algorithm that identifies frequent patterns based on the scanning of the dataset, and the creation and selection of k-itemsets. This k-itemsets are then used to identify possible (k+1)-itemsets. For a k-itemset to be considered a frequent pattern, it has to have a support greater than the established *minsup*, value that serves as a threshold. Additionally, this algorithm uses the Apriori property, which says that the subsets of a frequent pattern must also be frequent. This helps to make the process more efficient because if a k-itemset is not a frequent pattern, and it is a subset of a (k+1)-itemset, the latter can be discarded because, based on what we already know, it is not a frequent pattern.

The code was implemented on Python. I looped through the files corresponding to the 5 topics, and extracted the frequent patterns for each of them. I started by creating the 1-itemsets, making sure that the terms were not repeated in the set. Then, I found the support for each 1-itemset using percentages (i.e. *frequency/total of titles topic-i*), and discarded the ones for which the support was less than the *minsup*. For this task, I selected *minsup* to be 0.01. Then, I looped through the possible (k+1)-itemsets, following a similar process to the one explained above. This is: In each iteration, I created the (k+1)-itemsets, discarded the ones that contained non-frequent (k)-itemsets, scanned the database, computed the support for the remaining (k+1)-itemsets, and discarded the ones that were below the threshold. This process continued until there were no more (k+1)-itemsets. Finally, I saved the all the frequent patterns following the required format.



Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

```
(1)   $L_1$ = find_frequent_1-itemsets(D);
(2)   for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
(3)       $C_k$ = apriori_gen($L_{k-1}$);
(4)       for each transaction $t \in D$ { // scan $D$ for counts
(5)           $C_t$ = subset($C_k$, $t$); // get the subsets of $t$ that are candidates
(6)           for each candidate $c \in C_t$
(7)               c.count++;
(8)       }
(9)       $L_k = \{c \in C_k | c.count \geq min\_sup\}$
(10)  }
(11)  return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
(1)   for each itemset $l_1 \in L_{k-1}$
(2)       for each itemset $l_2 \in L_{k-1}$
(3)           if ($l_1[1] = l_2[1]$) $\wedge$ ($l_1[2] = l_2[2]$)
                  $\wedge ... \wedge$ ($l_1[k-2] = l_2[k-2]$) $\wedge$ ($l_1[k-1] < l_2[k-1]$) then {
(4)               $c = l_1 \bowtie l_2$; // join step: generate candidates
(5)               if has_infrequent_subset($c$, $L_{k-1}$) then
(6)                   delete $c$; // prune step: remove unfruitful candidate
(7)               else add $c$ to $C_k$;
(8)           }
(9)   return $C_k$;

procedure has_infrequent_subset($c$: candidate $k$-itemset;
          $L_{k-1}$: frequent $(k-1)$-itemsets); // use prior knowledge
(1)   for each $(k-1)$-subset $s$ of $c$
(2)       if $s \notin L_{k-1}$ then
(3)           return TRUE;
(4)   return FALSE;
```

Figure 6.4 Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Figure 1: Apriori pseudocode extracted from the book

1.2 Step 5

For mining the maximal and closed patterns, I did not used any specific algorithm. Instead, I used the results obtained in the previous step, and found the maximal and closed patterns following the conditions provided in the class slides. Basically, for both of them I looped through the created files, and at the same time, I looped through the extracted frequent patterns. For the closed patterns, I checked that the examined pattern was not a subset of a larger pattern with the same support number. If this was the case, the pattern was discarded. However, for the frequent patterns that I got and the *minsup* that I selected, almost all of the frequent patterns were also closed patterns. Then, for the max patterns, I checked that the examined pattern was not contained in a larger frequent pattern, disregarding the support number.

1.3 Step 6

For finding the purity of the patterns, I used the topic-i.txt files, the pattern-i.txt files, and the given formula. Basically, I looped through the files and the patterns, computing the corresponding purity value. To achieve this, I used the frequency of the pattern in the topic t, which had been already computed and was located in the corresponding pattern-i.txt file. I also computed the total number of lines in the corresponding topic-i.txt file. Then, I checked if the examined pattern was also present in any of the other topics, and if it was, I used the support value found in the corresponding pattern file. Lastly, I computed the union of the documents (titles) present in each of the topics, avoiding any duplicates. All these values were plugged into the given formula, and the purity value was computed for each pattern. Considering that the values were smaller than the ones obtained for the support (percentage), more importance was given to the purity. This means, purity and support were combined into a final value. The combination was 0.9 of the purity and 0.1 of the corresponding support.

2 Questions to ponder

2.1 Question A

How did you choose min_sup for this task?

Following the recommendations provided in the Homework PDF, a percentage *minsup* was used: I chose 0.01. My reasoning for this selection was made by observing the total number of documents, or titles (lines), in each of the five topics. For all of them, the value was close to 10,000. Considering this information, I decided that a pattern was frequent if it appeared more than 100 times, approximately. In addition, as a part of the selection process, I explored using other *minsup* values greater than 0.01, and studied the results. With these values, almost all the frequent patterns obtained were composed only by one or two terms, i.e., the three-terms frequent patterns basically disappeared. Since I wanted to have frequent patterns containing three terms, and 0.01 seemed like a reasonable choice, this was my final selection.

2.2 Question B

Can you figure out which topic corresponds to which domain based on patterns you mine? Write your observations in the report.

Yes, the domains are the following:

- 0: Mechine Learning
- 1: Information Retrieval
- 2: Database
- 3: Theory
- 4: Data Mining

This was a very straight forward task, considering that for each topic the mined patterns with higher support values included very representative terms of each domain, and most of them, included directly the name of the domain —or part of the name— in the patterns located within the first lines.

2.3 Question C

Compare the result of frequent patterns, maximal patterns and closed patterns, is the result satisfying? Write down your analysis.

The results of the frequent patterns, closed patterns, and max patterns were satistying, and were in agreement with what I was expecting based on the theory covered during class and the examples. The frequent patterns for each topic were consistent within the topic, and that is the reason why it was easy to identify the different domains. It was possible to find representative terms from each domain, with the most important ones having higher support values. Therefore, I deduced that my implementation was correct. For the closed patterns, I obtained almost the same frequent patterns, extremely few were removed. However, this makes sense because even though there were frequent patterns that were subset of larger frequent patterns, they did not share the same support value. Then, this result was satisfying. Finally, for the maximal patterns, I did see a reduction from the original set of frequent patterns. Therefore, since I was expecting more closed patterns than maximal patterns based on the theory, this was also a satisfying result.

3 Source Files

3.1 HW1_Step2.py

- Open required file.
- Loop through the titles.
- Populate dictionary with each term.
- Check for duplicated items.
- Save required file.
- Loop through the titles.
- Check if term is repeated within the title.
- Find corresponding index for the term.
- Organize the required format.
- Save the files.

3.2 HW1_Step3.py

- Open required files.
- Loop through the word-assignments.dat.
- Assign the terms to the corresponding file.
- Organize the files in the required format
- Save the required files.

3.3 HW1_Step4.py

- Open required files.
- Loop through the topics and the titles.
- Generate 1-itemsets.
- Compute support.
- Discard patterns that did not meet the condition.
- Loop through possible (k+1)-itemsets.
- Generate (k+1)-itemsets.
- Discard non-frequent (k+1)-itemsets based on non-frequent k-itemsets.
- Compute support.
- Discard patterns that did not meet the condition.
- Sort patterns.
- Organize file format.
- Save files.

3.4 HW1_Step5_closed.py

- Open required files.
- Loop through files and patterns.
- Check if there are super-patterns with the same support for each pattern.
- Discard non-closed patterns.
- Organize file format.
- Save the required files.

3.5 HW1_Step5_max.py

- Open required files.
- Loop through files and patterns.
- Check if there are super-patterns for each pattern.
- Discard non-max patterns.
- Organize file format.
- Save the required files.

3.6 HW1_Step6.py

- Open required files.
- Loop through the topics and titles.
- Compute D(t,t') and D(t).
- Loop through patterns.
- Compute f(t,p) and f(t',p).
- Compute given equation.
- Combine support and purity.
- Sort patterns.
- Organize file format.
- Save files.

3.7 mapping.py

- Open required files.
- Loop through all the files and the patterns.
- Find the corresponding term for each index.
- Organize file format.
- Save files.

References Cited

[1] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques.* Elsevier, 2011.