# CS570 HW1 Report

Ruochen Kong

Friday 18th February, 2022

## 1 Explanations

### 1.1 Frequent Patterns – Step 4

In this step, I implemented the Apriori algorithm to find the frequent patterns. The 'pattern.py' file contains the code for this step. During implementation, I first defined a list of sets *Data* to collect the set of single items of each line in a specific topic. Let use *topic 0* as the example, then the first few elements of *Data* would be:

| index | content |
|---|---|
| 0 | {'0019', '0020'} |
| 1 | {'0095','0096','0056'} |
| 2 | {'0097','0099','0100'} |
| ... | ... |

Before loading the topic file, I also created a dictionary that uses the itemset as keys and their support numbers as values. During loading, I counted the support numbers of every single item, and after loading, I removed the ones with support numbers less than min support. The **min support is fixed at 1%** of the total lines of each topic.

Then in each iteration of Apriori, I collected the remaining frequent patterns from the previous iteration and joined them with themselves to form the patterns that need to be checked in the current iteration. I checked each pattern with *Data* to count the support numbers and collect the ones with support numbers higher than the min support number. The iteration stopped when there is no remaining frequent pattern from the previous step. Finally, I output this dictionary to the txt file.

**** Codes for this step are in the pattern.py file.***

### 1.2 Maximal/Closed Patterns – Step 5

These patterns are calculated from the output in step 4. For the **closed pattern**, I read the pattern file into dictionaries with the support number as the keys and the set of patterns as the values. Then to check if a pattern is a closed pattern, it is only to compare with the ones in the same list with it. For example, some elements of the dictionary for *topic 3* would be:

| keys | values |
|---|---|
| ... | ... |
| 101 | [{'0233'}, {'0676'}, {'0397'}] |
| 100 | [{'0368'}, {'0487'}] |
| 99 | [{'0206'}] |
| 98 | [{'1889'}, {'0105','0129'}, {'0105','0196'}] |
| ... | ... |

**** Codes for this step are in the closed.py file.***

For the **max pattern**, I made a list of (pattern, support number) pairs as the result. When loading the pattern files, I checked whether the pattern is a supper pattern of one element of the result list. If it is a supper pattern, then change the (pattern, support number) pair with the super pattern with its support number; otherwise, add this pattern with its support number into the result list as a new element.

**** Codes for this step are in the maxpattern.py file.***

### 1.3 Purity – Step 6

I first calculated the $D(t, t')$ for every possible $(t, t')$ pairs, and stored them into a $5 \times 5$ matrix $Dtt$ with $Dtt[t][t] = D(t)$ and $Dtt[t][t'] = D(t, t')$. Then I load the pattern files as a list of dictionaries $P$ with $P[i]$ represents the mining pattern of the $i$th topic. Each dictionary uses the patterns as the keys and their support number as the values. In this way, the $f(t, p)$ can be easily accessed through this list. For finding the $f(t', p)$, I loaded the topic files and counted the appearance of $p$. By the provided formula, I generated a dictionary of each topic with patterns as the key and their purity as the value. To combine the purity with the support number, I calculated a reranking value by:

$$val = \text{support number} \times e^{\text{purity}}$$

As the purity ranged from positive to negative, by this equation, all the resulted values will be positive and a pattern with negative purity but a large support number may have a chance to receive a higher rank than the one with positive purity but a small support number. Hence, in general, a pattern with a larger support number and higher purity will receive a higher rank.
**** Codes for this step are in the purity.py file.***

# 2 Questions

## 2.1 Ponder A

I chose $min\_sup = 1\%$, because when I tried with $min\_sup = 2\%$, the number of frequent patterns is not enough for me to create an overview of the topic. With 1%, around 180 frequent patterns are extracted from each topic, which is the most suitable number for me.

## 2.2 Ponder B

By looking through the several pattern files, I think :

*topic 0* Query Processing

*topic 1* Natural Language Processing

*topic 2* Data Mining

*topic 3* Database System

*topic 4* Algorithms

For *topic 0*, most of its frequent patterns are about queries and efficiency, so I think this topic should be related to query processing models.
For *topic 1*, it has 'image', 'detection', 'semantic', so I think it should be NLP.
For *topic 2*, it is quite clear than contains 'data mining'
For *topic 3*, similarly, it has 'database' as a frequent pattern with a high rank.
For *topic 4*, as the same condition, it has 'algorithm' as its most frequent pattern.

## 2.3 Ponder C

Based on my results, the *closed-i.txt* is identical to the *pattern-i.txt* and most of the patterns contain only one term, so I think the terms are not likely to appear together in our dataset. The *max-i.txt* files successfully show some difference than the previous twos. When guessing the topic, I found

it would be more helpful to look through the max files instead of the patterns. This feeling is probably because the max files removed the single term patterns that have supper patterns and then enhanced the understandability. Thus, I think the max patterns are satisfied to determine the topics, while the closed and patterns are not.

## 3 Sources

**LDA** for preprocessing.
**os** to create the directories in step 4 to 6.
**math** to calculate the 2 based log for the purity in setp 6.
Other structures including sets and dictionaries are self-contained in Python.
Note, several files may be started with 'from closed import *'. These files are importing functions from my own file, 'closed.py'.