CS 570: Data Mining **(Due: 02/16/2022)**

Homework #1: Frequent Pattern Mining

Instructor: Carl Yang

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Please typeset your submissions in LaTeX.
- This is an individual assignment, which means it is OK to discuss with your classmates and your TAs regarding the methods, but it is not OK to work together or share code.
- Put your name along with an SPCA honor comment at the top of your code submitted to Canvas.
- Similar libraries or programs of frequent pattern mining algorithms can be found on-line, but you are prohibited to use these resources directly, which means you can not include public libraries, or modify existing programs, since the purpose of this programming assignment is to help you go through frequent pattern mining processing step by step.
- You can use either Java/C++/Python as your programming language.
- You will use a package working under OS with Unix kernel. It works well on Linux/MacOS. If you are a Windows user, you need to either (1) connect to a lab machine or (2) find other packages with the same function.
- You are asked to write a report about the assignment. So pay attention to the places with "Question to ponder"

Step 1: Get to know the Data

We collect paper titles from conferences in computer science of **5 domains**: **Data Mining(DM)**, **Machine Learning(ML)**, **Database(DB)**, **Information Retrieval(IR)** and **Theory(TH)**. You can download the raw data on Canvas named **paper_raw.txt**. (Note that we will not use this file directly for this assignment. But you can look at this file to see what the original titles look like.) Each line contains two columns, **PaperID** and **Title** of a paper, separated by Tab('\t'). Note that PaperID is **unique** in the whole data set. However, since this data is a subset of all the paper titles from a large corpus, PaperIDs are **not starting from 0** and not consecutive. The raw data looks like this:

PaperID	**Title**
7600	The Automatic Acquisition of Proof Methods
85825	Frequent pattern discovery with memory constraint

For this assignment, we pre-processed the raw data by removing stop words, converting the words to lower cases and lemmatization. You can download the processed data on Canvas named **paper.txt**. This is the actual dataset we will use for this assignment. In this file, each line is a **PaperID** followed by **a list of terms**.The format is PaperID Tab('\t') term1 Space(' ') term2 Space(' ') ... paper.txt looks like this:

PaperID	**[term1] [term2] [term3] [term4] ... [termN]**
7600	automatic acquisition proof method
85825	frequent pattern discovery memory constraint

Step 2: Preprocessing (20pts)

This step prepares input for LDA. You will generate two files based on paper.txt.

1. Generate a Dictionary (10 pts)

First, you need to generate a vocabulary from paper.txt. Please name your vocabulary file as **vocab.txt**. Each line in this file is a **unique** term extracted from paper.txt; each term should appear exactly once. Here is an example of the first five lines in vocab.txt. Note that the sequence of terms may be different.

```
automatic
acquisition
proof
method
philosophical
...
```

2. Tokenize Plain Text by Dictionary (10pts)

In this step, we represent each title as a sparse vector of word counts. We ask you to transform each title (i.e., each line in paper.txt) into the following format ([M] (space) [t1]:[c1] (space) [t2]:[c2] (space) ...):

```
[M] [term_1]:[count] [term_2]:[count] ... [term_N]:[count]
```

where [M] is the number of unique terms in the title, and the [count] associated with each term is how many times that term appeared in the title. For example, in paper.txt, suppose we have a title "automatic acquisition proof method". The corresponding line in the output file should be "4 0:1 1:1 2:1 3:1". Note that [term_i] is an integer which indexed a term in vocab.txt; it **starts from 0**. Please name the output file as **title.txt**.

Note: if you use any other lda package rather than the one mentioned in next step, please make sure the format in title.txt match what your lda package requires.

Step 3: Partitioning (10 pts)

Recall that we collect paper titles from conferences of **5 domains** in computer science. If we run frequent pattern mining algorithms directly on paper titles, the patterns would be independent of topics. Thus, if we want to mine frequent patterns in each domain, titles and terms should be partitioned into 5 domains. Note that the domain knowledge is not known to you. Instead, we apply topic modeling to uncover the hidden topics behind titles and terms automatically. Specifically, we apply LDA (you are not required to understand how exactly topic modeling works) to assign a topic to each term.

1. Assign a Topic to Each Term (5 pts)

(1) Download the LDA package here. Unzip it and you could see a list of source code.
(2) Open a terminal and go to the directory of the source code. Type **make**. Then an executable **lda** is generated.
(3) In lda-c-dist directory, there is a **settings.txt** file. You could use the following settings, but feel free to tune the parameters if you know how inference works for LDA.

```
var max iter 20
var convergence 1e-6
em max iter 100
em convergence 1e-4
alpha estimate
```

(4) Run LDA with the following command.

```
<dir>/lda-c-dist/lda est 0.001 5 <dir>/lda-c-dist/settings.txt <dir>/title.txt random <dir>/result
```

where <dir> is your own working directory, and 0.001 is the prior for topic proportion (it is just a parameter and you don't have to change it if you don't know how) given to LDA. 5 represents 5 topics. title.txt is the file you generate at Step 2. The output will be put into **result** directory.
(5) Check your output.

In the result directory, open file **word-assignments.dat**. Each line is of the form ([M] (space) [t1]:[k1] (space) [t2]:[k2] (space) ...):

```
[M] [term_1]:[topic] [term_2]:[topic] ... [term_N]:[topic]
```

where [M] is the number of unique terms in the title, and [topic] associated with each term is the topic assigned to it. Topic number starts from 0. One line could be: 004 0000:02 0003:02 0001:02 0002:02; which means that all terms in this title are assigned to the 3rd topic (topic 02). Note that you are not confined to use this package. Here is another option http://mallet.cs.umass.edu/topics.php.

2. Re-organize the Terms by Topic (5 pts)

You are asked to re-organize the terms by 5 topics. For the i-th topic, you should create a file named **topic-i.txt**. Separate each line in word-assignment.dat by topics assigned to them. For example, the lines in word-assignment.dat can be considered as the following form (Note that here we replace the integers with the real terms for better illustration):

```
004 automatic:02 acquisition:02 proof:02 method:02
005 classification:03 noun:02 phrase:03 concept:01 individual:03
```

Then your output files could be:

```
topic-0.txt
...
```

```
topic-1.txt
concept
...
```

```
topic-2.txt
automatic acquisition proof method
noun
...
```

```
topic-3.txt
classification phrase individual
...
```

In the real files, each term should be represented as the id corresponding to the dictionary generated from Step 2 (e.g., 0001, 0056). topic-i.txt looks like this:

```
[term1] [term2] ... [termN]
[term1] [term2] ... [termN]
...
```

Step 4: Mining Frequent Patterns for Each Topic (30 pts)

In this step, you need to implement a frequent pattern mining algorithm. You can choose whatever frequent pattern mining algorithms you like, such as Apriori, FP-Growth, ECLAT, etc. Note that you need to run your code on 5 files corresponding to 5 topics. The output should be of the form ([s] (space) [t1 (space) t2 (space) t3 (space) ...]), where each term should be represented as the term id:

#Support	[frequent pattern]
#Support	[frequent pattern]
...	

and frequent patterns are **sorted from high to low by #Support**. Your output files should be put into one directory named as **patterns**. The i-th file is named as **pattern-i.txt**.

(Hint: you need to figure out min_sup by yourself.)

Question to ponder A: How you choose min_sup for this task? Note that we prefer min_sup to be the consistent percentage (e.g. 0.05 / 5%) w.r.t. number of lines in topic files to cope with various-length topic files. Explain how you choose the min_sup in your report. Any reasonable choice will be fine.

Step 5: Mining Maximal/Closed Patterns (30 pts)

In this step, you need to implement an algorithm to mine maximal patterns and closed patterns. You could write the code based on the output of Step 4, or implement a specific algorithm to mine maximal/closed patterns, such as CLOSET, MaxMiner, etc.

The output should be of the same form as the output of Step 4. Max patterns are put into **max** directory. The i-th file is named as **max-i.txt**.Closed patterns are put into **closed** directory. The i-th file is named as **closed-i.txt**.

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

Question to ponder C: Compare the result of frequent patterns, maximal patterns and closed patterns, is the result satisfying? Write down your analysis.

Step 6: Re-rank by Purity of Patterns (10 pts)

In this paper, purity is introduced as one of the ranking measures for phrases. A phrase is pure in topic t if it is only frequent in documents (here documents refer to titles) about topic t and not frequent in documents about other topics. For example, 'query processing' is a more pure phrase than 'query' in the Database topic. We measure the purity of a pattern by comparing the probability of seeing a phrase in the topic-t collection $D(t)$ and the probability of seeing it in any other topic-t' collection ($t' = 1, ..., k, t'! = t$). In our case, $k = 4$. Purity essentially measures the distinctness of a pattern in one topic compared to that in any other topic. The definition is as follows:

$purity(p, t) = log[f(t, p)/|D(t)|] - log(max[(f(t, p) + f(t', p))/|D(t, t')|])$

Here, log base is 2. $f(t, p)$ is the frequency of pattern p appearing in topic t (i.e., support of p in topic-t.txt). We define $D(t)$ to be the collection of documents where there is at least one word being assigned the topic t. $D(t) = \{d|$topic t is assigned to at least one word in $d\}$. $D(t, t')$ is the union of $D(t)$ and $D(t')$. $|*|$ measures the size of a set. Actually $|D(t)|$ is exactly the number of lines in topic-i.txt. But note that $|D(t, t')| \neq |D(t)| + |D(t')|$.

Re-rank the patterns obtained from Step 4. The output should be of the form:

Purity	[frequent pattern]
Purity	[frequent pattern]
...	

and frequent patterns are **sorted from high to low by a measure which combines Support and Purity** (you will need to come up with how to combine them). Your output files should be put into one directory named as **purity**. The i-th file is named as **purity-i.txt**

Step 7: Report

Now you are ready to write your report. You should include the following sections in your report:
(1) Briefly explain what algorithms you use in Step4-Step6.
(2) Answer all the questions in Question to ponder.
(3) List your source file names and their corresponding Steps.

Step 8: Submission

Please submit a **pdf** file of your report and a **zip** file that contains **all** the files generated in the previous steps, and **all** the source codes written by you.

For every file in patterns/max/closed/purity, please map the numbers to terms based on your dictionary. Please use the same format with the only difference that each integer is replaced by a term, name each new file as **originalFileName.phrase** and put it in the same directory as the original file. For example, pattern-0.txt will be mapped to pattern-0.txt.phrase and it will be put in the directory patterns. This mapping step helps you to examine the quality of the mined patterns and see if they make sense in terms of forming phrases.

All of your code should be successfully compiled before submission. And don't forget to include the honor code statement in your source files. Copying source code from other students will get 0 grade. We'll run plagiarism detection software. Please do not cheat.