Assignment 3 Frequent Pattern Mining Programming (Fall 2014, Distributed Oct. 9, 2014. Due Nov. 1, 2014)

Before Diving In

- Programming assignment might take more time than written assignments, so please start early.
- 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.
- 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/Matlab as your programming language.
- You are asked to write a report about the assignment. So pay attention to the places with "Question to ponder"

Objective

- Explore how frequent pattern mining can be applied to text mining to discover meaningful phrases.
- In this assignment, we run LDA on a corpus made up of titles from 5 domains' conference papers. Based on the results from LDA, a topic (representing a particular domain) is assigned to each word of each title. You are given 5 files in which each line contains words assigned to this topic. Then you write a frequent pattern mining algorithm to mine frequent patterns from each topic to get meaningful phrases. The mined frequent patterns may not necessarily be *meaningful* phrases for the topic. So you will consider the question of how to extract meaningful ones out of all the frequent patterns. The final goal is to output highly representative phrases for each topic.

Introduction

In this part, we introduce how we generate the input files for you. **You don't need to write code for these steps.** But it is the background of the following steps. So please read it carefully.

- **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).** The raw data is named as **paper_raw.txt**. Each line contains two columns, **PaperID** and **Title** of a paper, separated by **Tab ('\t')**. Recall the example in class. For each line in the file you can consider it as one transaction. Each word in the title is equivalent to an item in a transaction.

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 have pre-processed the raw data by removing stop words, converting the words to lower cases and lemmatization, named **paper.txt.** In this file, each line is **a list of terms**. Terms are separated by one space.

paper.txt looks like this:

[term1]	[term2]	[term3]	[term4]	[term5]	.....[termN]
automatic	acquisition	proof	method		
frequent	pattern	discovery	memory	constraint	

- ## Preprocessing and Partitioning

This step prepares input for LDA and partition paper titles based on the topic assignment of LDA. We will generate 6 files you are going to use for frequent pattern mining: **vocab.txt** and **topic-0.txt~topic-4.txt**

1 Generate a Dictionary and Tokenize Plain Text by Dictionary

First, we generate a vocabulary from paper.txt, named as **vocab.txt.** Each line in this file has two columns: the first column is the term index; the second column is a **unique** term extracted from paper.txt; columns are separated by **Tab('\t').** Each term should appear exactly once. Here are the first five lines in vocab.txt.

```
0    automatic
1    acquisition
2    proof
3    method
4    philosophical
...
```

Then, we tokenize paper.txt by vocab.txt. So each line in paper.txt will be like:

```
0 1 2 3

4 5 6 7

8 9 10 11 12 13 14 15 16 17
...
```

2 Assign a Topic to Each Term

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. We will use number **0~4** to represent these 5 domains. Instead, we apply topic modeling to uncover the hidden topics behind titles and terms automatically. Specifically, we apply Latent Dirichlet Allocation (LDA)to assign a topic to each term. You may refer to the original paper for further information(http://machinelearning.wustl.edu/mlpapers/paper_files/BleiNJ03.pdf).

We use a LDA packages to assign topic to each term. You can refer to for a comprehensive introduction to the package. (http://www.cs.princeton.edu/~blei/lda-c/index.html)

In the output of LDA, open file **word-assignments.dat**. (Note that every time LDA may generate different topic assignments to the same set of documents. So word-assignments.dat will be different in multiple runs.)

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, [term_i] is the term index in vocab.txt, and [topic] associated with each term is the topic assigned to it. Topic number starts from 0. The first three lines are as follows.

```
004 0000:03 0003:03 0001:03 0002:03

004 0004:01 0007:03 0006:01 0005:03

010 0015:01 0010:01 0017:02 0011:02 0012:02 0014:01 0009:01 0008:01 0016:02 0013:01
```

The first line means that terms 0, 3, 1, 2 are all assigned to topic3.

3 Re-organize the Terms by Topic

We re-organize the terms by 5 topics. For the i-th topic, we create a file named **topic-i.txt** (Note that these are the files you are going to work on to mine frequent patterns). We 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:03 method:03 acquisition:03 proof:03
004 philosophical:01 theory:03 learning:01 formal:03
010 mini:01 outdoor:01 challenge:02 robot:02 platform:02 abington:01 cost:01 low:01 grand:02 penn:01
```

The output files will be like:
//topic-0.txt

...

//topic-1.txt
philosophical learning

mini outdoor abington cost low penn

…

//topic-2.txt:
challenge robot platform grand
...

//topic-3.txt

automatic method acquisition proof

theory formal
...

//topic-4.txt
…

In the real files, each term is represented as index corresponding to the dictionary vocab.txt.

topic-1.txt looks like this:

```
4 6
15 10 14 9 8 13
...
```

Step 1: Mining Frequent Patterns for Each Topic (40pts)

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, etc. Note that you need to run your code on 5 files corresponding to 5 topics(topic-0.txt ~ topic-4.txt). The output should be of the form ([s] (space) [t1 (space) t2 (space) t3 (space) ...]):

```
#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**.

To note that in pattern-i.txt files, the term index should be mapped to word based on vocab.txt. This requirement is the same with the next two steps too.

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

***Question to ponder A**: How do you choose min_sup for this task? Explain how you choose the min_sup in your report. Any reasonable choice will be fine.*

Step 2: Mining Maximal/Closed Patterns (30pts)

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 1, 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 1. 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 3: Association Rule Mining by Weka (15pt)

In this step, you need to use Weka to mine association rules. In class, we have demonstrated how to use Weka to mine association rules. You can download the slides "Weka-Associate" on schedule page to review the process.

1 Generate .ARFF Files

In this step, you need to convert the text file into .ARFF files. We have provided you with the code **featureGenerator.py**. Please put it in the same folder as topic-i.txt and vocab.txt. In the command line, type:

python featureGenerator.py

Then it will generate 5 ARFF files corresponding to each topic-i.txt file.

**Note: If you are MacOS/Linux user, you can easily do so. If you are windows user, you can get access to Linux system EWS lab on campus

2 Get Association Rules by Weka

(1) Download and install Weka from http://www.cs.waikato.ac.nz/ml/weka/downloading.html.

(2) Double click on the icon on the application file of Weka.

(3) Choose "Explorer".

(4) Click on "Preprocess" tag.

(5) Click on “Open file”, choose “topic-i.arff” , click on “Choose” and wait Weka until it finishes loading the data.

(6) Click on “Associate” tag. Click on “Choose” and select “FPGrowth” as the mining algorithm.

(7) Click on the parameter line. You need to modify **lowerBoundMinSupport** and **minMetric** to generate **at least 10** association rules. You may keep the default value of other parameters.

(Hint: lowerBoundMinSupport is the min_sup and minMetric is the min_conf of the association rules. If you set it min_sup to high, it won’t find any association rules.)

Please put the parameter settings and association rules in your report.

Question to ponder D*: What are the quality of the phrases that satisfies both min_sup and min_conf? Please compare it to the results of Step1 and put down your observations.*

Step 4: Re-rank by Purity of Patterns (15pts)

In this paper (http://arxiv.org/abs/1306.0271), 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'= 0, 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, f(t,p) is the frequency of pattern p appearing in topic t. We define D(t) to be the set of documents where there is at least one word being assigned the topic t. Set D(t) = { d | topic t is assigned to at least one word in document d } and 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') | != | D(t) | + | D(t') |.

In the following, we give you the exact number of D(t) and D(t, t’). The elements on the diagonal are D(t). The others are D(t,t’). You need to iterate through all the patterns of each topic to get the f(t,p) and calculate purity(p,t).

D(t,t’)	topic0	topic1	topic2	topic3	topic4
topic0	10047	17326	17988	17999	17820
topic1	-	9674	17446	17902	17486
topic2	-	-	9959	18077	17492

topic3	-	-	-	10161	17912
topic4	-	-	-	-	9845

Re-rank the patterns obtained from Step 1. 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.**

Step5: Bonus (10pts)

Could you come up with other filtering/ranking criteria to improve the quality of your mined phrase lists? Implement your algorithm and put your analysis in your report.

(Hint: Some related papers describe strategies to deal with this problem by balancing max patterns and closed patterns.

CATHY: http://www.cs.uiuc.edu/~hanj/pdf/kdd13_cwang.pdf

KERT: http://arxiv.org/abs/1306.0271.)

Step6: Report (10pts)

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 Step1~Step6.

(2) Answer all the questions in *Question to ponder*.

(3) List your source file names and their corresponding Steps.

Project Organization and Submission

The structure should be like this: (Files enclosed with "<>" are the source codes written by you. Those followed by "|--------" are directory names.)

```
yourNetId_assign3|--------
                                    <frequent mining source files>
                                  patterns          |--------pattern-0.txt~pattern-4.txt
                                  <max/closed mining source files>
```

max |--------max-0.txt~max-4.txt

closed |--------closed-0.txt~closed-4.txt

<Re-rank source files>

purity |------purity0.txt~purity-4.txt

report.pdf

For every file in patterns/max/closed/purity, please map the indices to terms based on your dictionary. 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. After you finish your implementation and report, please compress your project into zip format, and rename your zip file to **yourNetId_assign3.zip *and submit this file through *Compass2g**.

Grading Policy

- Completeness of your work
- Compilable source code
- Reasonable results with explanations
- Answer all questions
- Copying source code from other students will get 0 grade. We will run plagiarism detection software. Please do not cheat.