CS 412 HW3 Report

Name: Yining Wang

Netid: ywang282

UIN: 677886936

1. Algorithm explanation for step 4 – step 6

- Step 4: mining frequent patterns for each topic
  - I implemented the apriori algorithm
  - I defined several function modules:
    - load_topic_file(): load a topic file and convert the file into a python list
    - generate_C_1(): generate the candidates for the 1-itemset
    - get_frequent_itemset() : scan the DB and return the itemsets that meet the mi_sup requirement
    - joinable(): helper function to check whether two itemsets are self joinable
    - generate_next_C(): generate (k+1)-itemset candidates based on k-length frequent itemsets
    - run_apriori(): wrapper function to run the apriori algotirhm by calling all the functions defined above
    - write_result_to_pattern(): write the result to file
  - algorithm description:
    - initially scan the DB once to get the 1-itemset candidates and length-1 frequent itemsets
    - initialize k = 1
    - while the list of length (k) frequent itemset is not empty, generate the (k+1) itemset candidates, scan DB and get the length (k+1) frequent itemsets. Update k = k +1
    - after the while loop exits, return the entire frequent itemsets found and sort the result and write to file
  - repeat the above process for 5 times, once for each topic file

- Step 5: mining maximal/closed patterns
  - I used the output in step 4
  - Mining maximal patters:
    - I used a double for loop

- ❖ For each frequent pattern FP, compare it with the rest of the frequent patterns to check if there is any super pattern of FP
  - If no: output FP as a maximal pattern
  - If yes: discard FP

- Mining closed patterns:
  - ❖ I used a double for loop
  - ❖ For each frequent pattern FP, compare it with the rest of the frequent patterns to check if there is any super pattern of FP with the same support
    - If no: output FP as a closed pattern
    - If yes: discard FP

- Step 6: re-rank by purity of patterns
  - I defined several function modules:
    - ❖ load_word_assugnments(): load the 'word-assignments.dat'
    - ❖ load_pattern_file(): load the pattern-[0~4].txt
    - ❖ get_D_t(): get the list of the collection of documents where there is at least one word being assigned the topic t
    - ❖ get_D_t_size(): return the size of D(t)
    - ❖ get_frequency_dict(): get a dictionary of f(t, p) for a specified topic
    - ❖ get_frequency_dict_in_t_prime():get a dictionary of f(t', p) for topics other than the specified one
    - ❖ get_D_t_t_prime(): get |D(t, t')|
    - ❖ get_purity(): calculate the purity by calling the function modules defined above
    - ❖ get_combined_score(): get the weighted result of support and purity, sort the output from high to low by the combined measure
    - ❖ write_to_file(): write the result to file
  - algorithm description:
    - ❖ I used all the modules described above to calculate each

component of the formula given to us in the assignment wiki page. After gathering all the components, I applied the formula and arrived at the final purity score.

- ❖ I used 0.7 as a weight for support and 0.3 as a weight for purity because I value analyzing each topic as a whole rather than focusing too much on the purity of each pattern.
- ❖ The output makes lots of sense because the most important frequent patterns are still preserved while patterns that are not as pure are ranked lowerd by the combined score. For example, the pattern 'data' could appear a lot in both Data Mining and Database. After purifying the patterns, 'data''s rank become lower in both purity files because it is not so pure as to uniquely distinguish a domain from another.

2. Questions to ponder

- A: How did you choose the minimum support?
  - the minimum support I choose is 0.01
  - the total length of topic-[0~4].txt do not differ that much from each other so I will use topic-0.txt as an example here
  - I experimented different values for the min_sup for topic-0.txt and the result is as following:

min_sup	Frequent pattern count
0.005	206
0.007	121
0.01	70

0.02	34
0.03	13

- Analysis:
  - ❖ mim_sup > 0.01 does not give enough frequent patterns and the count of 2-itemsets are also low;
  - ❖ mim_sup < 0.005 gives too many frequent patterns and the patterns located towards the bottom of the file output is not as meaningful anymore;
  - ❖ 0.005 <= mim_sup <= 0.01 gives valuable frequent patterns but taking time complexity into consideration and without losing any important frequent patterns, 0.01 is the best choice.

- B: Can you figure out which topic corresponds to which domain based on patterns you mine?
  - After mapping all the term_id to term, its very obvious which topic corresponds to which domain based on the patterns I mined
  - Here is the result:

Topic index	Domain
0	Data Mining(DM)
1	Information Retrieval(IR)
2	Database(DB)
3	Machine Learning(ML)
4	Theory(TH)

- C: Compare the result of frequent patterns, maximal patterns and closed patterns, is the result satisfying?
    - From the output files, we can see that with mim_sup = 0.01, the frequent patterns and the closed patterns are exactly identical, or in other words, all the frequent patterns are already closed. However, the maximal patterns differ from the frequent patterns, which shortens the length of the file, making it easier and faster to detect the topic domains.

3. Source file names and their corresponding steps:

- Step 1: get to know your data

- Step 2: preprocessing
    - Source file: preprocess.py
    - How to run:
        - Open terminal and cd to current assignment directory
        - type in python preprocess.py
        - vocab.txt and title.txt will be generated in the same directory

- Step 3: partitioning
    - Run LDA
    - Reorganize terms by topic source file: organize_terms_by_topic.py
    - How to run:
        - Open terminal and cd to current assignment directory
        - type in python organize_terms_by_topic.py
        - topic-[0~4].txt will be generated in the same directory

- Step 4: mining frequent patterns for each topic

- Source file: fp_mining.py
- How to run:
  - Open terminal and cd to current assignment directory
  - type in python fp_mining.py
  - pattern-[0~4].txt will be generated in the patterns directory

- Step 5: mining maximal/closed patterns
  - Source file: closed_max_pattern.py
  - How to run:
    - Open terminal and cd to current assignment directory
    - type in python closed_max_pattern.py
    - max-[0~4].txt and closed-[0~4].txt will be generated in the max and closed directory respectively

- Step 6: re-rank by purity of patterns
  - Source file: purity.py
  - How to run:
    - Open terminal and cd to current assignment directory
    - type in python purity.py
    - purity-[0~4].txt will be generated in the purity directory

- Additional step: map term_id to term for patterns/max/closed/purity:
  - Source file: mapping.py
  - How to run:
    - Open terminal and cd to current assignment directory
    - type in python mapping.py
    - all the *.txt.phrase will be generated in purity/max/closed/patterns folder