CS412 - Introduction to Data Mining
Report on MP3

Abhishek Avinash Narwekar
UIN: 668601661, NetID: an41

November 2016

1 Steps 1-3

All the outputs of steps 1-3 are in the directory `data/`. The files include:

- `vocab.txt`
- `title.txt`
- `topic-0.txt...topic-4.txt`

2 Algorithms in Steps 4-6

2.1 Step 4: Mining Frequent Patterns For Each Topic

For each topic, I execute the **Apriori algorithm**. I have written the `Apriori` class in the file `Apriori.py` that contains all the functions to perform the frequent pattern mining. I have written a wrapper function, `getFrequentPatterns()` (present in the file `MinePatterns.py`), to call these functions, perform the sorting and write the output to files corresponding to each topic. The outputs are stored in the folder `data/patterns/`.

Optimizations: In each stage of the Apriori algorithm, we need to get the counts of potential frequent itemsets. This process can be extremely slow if we iterate through the entire dataset at every iteration. The Apriori algorithm can be sped up immensely by constructing a reverse index for the occurrences of all length-1 patterns. Occurrences of larger patterns can be easily found out by performing an intersection over the occurrence of all their constituent length-1 patterns. After this optimization, all the operations in Step 4 get executed in about 0.26s.

2.2 Step 5: Mining Closed and Maximal Patterns

I maintain a dictionary of dictionaries to store all the frequent patterns. Each key in the outer dictionary is a number beginning from 1 onwards, whose value is a dictionary containing the frequent patterns *of that length* and the value is the frequency of that pattern. This makes it very convenient for me to compute the closed and max patterns.

2.2.1 Closed Patterns

The closed patterns are frequent patterns that don't have and super-patterns with the same support as them. By this definitions, all frequent patterns of the largest length possible are closed patterns. To find closed patterns of smaller lengths, I iterate through patterns of each length in the dictionary. For each frequent pattern p of length k, I search for patterns of length $k+1$ that have the same support as p. If no such pattern exists, I add p to the list of closed patterns. The function to compute closed patterns is `getClosedPatterns()` in the file `MaxClosed.py`. The outputs are stored in the folder `data/closed/`.

2.2.2 Maximal Patterns

Maximal patterns are frequent patterns that don't have frequent super-patterns. Yet again, the frequent patterns of the largest length are maximal. For those of smaller lengths, we perform the same iterations as in section 2.2.1, except that now we check if the patterns of length of $k+1$ are *frequent*, regardless of whether they have the same count as p or not. The number of maximal patterns is always less than of equal to the number of closed patterns, a fact verified from the results. The function to compute closed patterns is `getMaxPatterns()` in the file `MaxClosed.py`. The outputs are stored in the folder `data/max/`.

2.3 Step 6: Re-ranking by Purity of Patterns

Here, we need to design a ranking mechanism that takes into account both the purity of a pattern and its support. Given the inherent disparity in the ranges of these two measures, I choose to consider only the *percentile score* of both the purity and the support of a pattern amongst all patterns in the topic. To do so, I sort the patterns of a given topic by purity and assign each pattern a percentile score (between 0-100). I repeat this for the support. I then add both the percentile scores (the result being in the range 0-200) and re-rank on the basis of this aggregate score. The function to perform the re-ranking is `rankPurity()` in the file `Rerank.py`. The outputs are stored in the folder `data/purity/`.

3 Questions to Ponder

3.1 How you choose min_sup for this task (step 4)? 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.

I choose min_sup to be 1% of the number of "transactions" in the file corresponding to a given topic. The number of frequent patterns from a topic that result are between 64 and 95.

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

I have written the function `getFrequentPatternsWords()` in `MinePatterns.py`, and the functions `getClosedPatternsWords()` and `getMaxPatternsWords()`, present in the file `MaxClosed.py`, that map the terms in the frequent, closed and max patterns respectively from numbers to words. I store the resultant patterns in the same folders as the patterns themselves with the suffix `.phrase`.

Based on the terms present in the files, my estimates for the topic assignment are shown in table 1 . I also specify the certainty with which I claim these

Topic-ID	Topic	Key Words	Certainty
0	Theory	logic, reasoning	moderate
1	Machine Learning	classifier, pattern	low
2	Data Mining	knowledge, data system	low
3	Information Retrieval	extraction, retrieval information	high
4	Database	system database, processing query	high

Table 1: Estimating topics using frequent patterns

assertions. The issue with topic-ID's 1 and 2 is that they *both* appear to be associated with data mining. For instance, topic-ID 1 contains patterns such as "data mining", "pattern mining" and "rule association" that seem related to data mining. But it also contains words such as "classifier", "data clustering", "nearest neighbour" that are machine learning related terms. On the other hand, topic-ID 2 too contains several data mining related terms but nearly *no machine-learning related terms*. Thus, the only reasonable association I can make is that of topic-ID 1 with machine learning and topic-ID 2 with data mining.

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

For the threshold I have chosen (0.01 times the transactions in the file), nearly all the frequent patterns mined appear to be **closed**, which makes sense, because there are rarely any phrases in the dataset that are *exactly* as frequent as their constituent words. Moreover, the number of max patterns is always less than or equal to the number of closed patterns, which agrees with the theory. The counts of frequent, closed and max patterns across topics is shown in table 2.

Topic-ID	Frequent Patterns	Closed Patterns	Max Patterns
0	64	64	59
1	95	95	74
2	74	74	60
3	83	83	65
4	78	78	64

Table 2: Count of frequent, closed and max patterns across topics for $min_{sup} = 0.01$

4 Source File Names And The Steps

The source files in the project are:

- `PreprocessLDA.py`: Contains the functions to perform the preprocessing for LDA
- `Reorganize.py`: Distributes the output of LDA into topics
- `Apriori.py`: Contains the class `Apriori` that executes the Apriori algorithm
- `MinePatterns.py`: Performs the frequent pattern mining using the Apriori algorithm
- `MaxClosed.py`: Mines the closed and the max patterns from the frequent patterns
- `Rerank.py`: Reranks the patterns based on purity and support
- `EC.py`: Ranks the patterns according to an advanced ranking mechanism as described in the paper on the KERT system [1]

All the source files can be **directly executed** in python without any parameters. The libraries required to execute the files are:

- `time`: To compute the runtime

- `math`: To compute the log to the base 2
- `operator`: For intermediate sorting operations
- `numpy`: For several mathematical operations

5 Advanced Ranking (Extra Credit)

5.1 Formulation

CATHY [2] and KERT[1] contain ranking mechanisms that combine measures such as the support, phrase completeness and purity. First, KERT defines a measure that balances closed and max patterns. It is called phrase completeness, defined as:

$$\pi_t^{com}(p) = 1 - \frac{max_w f_t(p \cup \{w\})}{f_t(p)} \tag{1}$$

Here, $f_t(p)$ is the absolute support of pattern p in topic t. w iterates over all the words in the dictionary. The measure essentially computes how much more frequent the pattern $p \cup \{w\}$ is as compared to p. If the supremum over all the words is high, it tells us that there is a super-pattern that is nearly as frequent as the pattern itself. The **phraseness** of a pattern is defined as:

$$\pi_t^{phr}(p) = log\frac{f_t(p)}{|D_t|} - \sum_{w \in p} log\frac{f_t(w)}{|D_t|} \tag{2}$$

The phraseness of a pattern is high if the log probability of occurrence of a pattern is relatively larger than that all of its component words combined. It is therefore, large for phrases that are unusually more common than their component words. We combine all of these measures to get a combined score for a pattern ($r_t(p)$), as follows:

$$r_t(p) = \begin{cases} 0 & \text{if } \pi_t^{com} \leq \gamma \\ \pi_t^{cov}[(1-\omega)\pi_t^{pur} + \omega\pi_t^{phr}](p) & \text{o.w.} \end{cases} \tag{3}$$

5.2 Results

By setting a lower bound for the value of completeness (γ), we can filter out patterns (such as "support vector") that are frequent, but are often part of a larger pattern (such as "support vector machine"). To perform aggressive filtering and get representative patterns, I set $\gamma = 0.9$. This results in several patterns getting a score of 0. I weigh both phraseness and purity equally by setting $\omega = 0.5$. Since the purity of a pattern and the phraseness of patterns can be negative, there are patterns having a high value for completeness but a negative overall score $r_t(p)$.

Coming to the question of estimating the topic in each file, we rank the patterns in each file using the mechanism proposed in KERT. However, the

highest ranked entries may not necessarily agree with the rest of the entries. For instance, the pattern "support vector machine" is the highest ranked pattern in topic-3. However, there are strong indicators that topic-3 is **IR**, given the phrases "search engine", "web search", "document retrieval", "information retrieval", etc. Thus, we conclude that KERT is effective as a filtering tool, but may not necessarily rank the *most indicative patterns* at the top. We perform the analysis from table 1 again for the new rankings, considering only the *indicative patterns* (picked manually) in the classification. The results are shown in table 3. While my certainty about this topic allocation is definitely higher than the estimates in table 1, I note that **machine learning** does not seem to be associated significantly with any file in particular.

Topic-ID	Topic	Key Words	Certainty
0	Theory	optimization, model learning	moderate
1	Data Mining	data mining, rule mining association	moderate
2	Machine Learning	natural language, knowledge base	low
3	Information Retrieval	information retrieval, web search	high
4	Database	object oriented database, query processing	high

Table 3: Estimating topics using representative frequent patterns after filtering using the KERT system

The results obtained after performing this ranking are stored in the folder `data/EC/` with the formats `pattern-i.txt` and `pattern-i.txt.phrase`.

References

[1] Marina Danilevsky, Chi Wang, Nihit Desai, Jingyi Guo, and Jiawei Han. Kert: Automatic extraction and ranking of topical keyphrases from content-representative document titles. *arXiv preprint arXiv:1306.0271*, 2013.

[2] Chi Wang, Marina Danilevsky, Nihit Desai, Yinan Zhang, Phuong Nguyen, Thrivikrama Taula, and Jiawei Han. A phrase mining framework for recursive construction of a topical hierarchy. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 437–445. ACM, 2013.