Suma Vangala (netid: svangal2)

Assignment 3 Frequent Pattern Mining Programming

Step 1: Mining Frequent Patterns for Each Topic

The algorithm which has been used for mining frequent patterns for each topic is "Apriori Algorithm". The algorithm proceeds by identifying the frequent individual items in the topic file and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the topic file. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation*), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.
The algorithm is based on the Apriori Property: Any subset of frequent itemset must be frequent
This general scheme used for the Apriori Algorithm is mentioned below. It is based on two main steps: candidate generation and pruning.
Determine the support of the one element item sets and discard the infrequent items.
• Form candidate item sets with two items (both items must be frequent), determine their support, and discard the infrequent item sets.
• Form candidate item sets with three items (all pairs must be frequent), determine their support, and discard the infrequent item sets.
• Continue by forming candidate item sets with four, five etc. items until no candidate item set is frequent.

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.

Choosing min support is quite subtle: a too small threshold may lead to the generation of thousands of patterns, whereas a too big one may often generate no answers. A small min_sup value may generate an exponential number of frequent patterns since such a small value only prunes a few item sets. As the number of derived frequent item sets is very large, it is not practical to produce and use such a huge result set.
Specifying appropriate minimum support is a challenging task as the choice of minimum support value is somewhat arbitrary. The algorithm was repeatedly executed, heuristically tuning the value of minimum support over a wide range, until the desired result is obtained, certainly, a very time consuming process. So setting the minimum support to 50, I found statistically significant patterns. This compressed the number of derived patterns to a considerable limit and thereby enhanced the quality of desired patterns.

Files:
Code file – AprioriFrequentPatternMining.py
Input files – topic-0.txt, topic-1.txt, topic-2.txt, topic-3.txt, topic-4.txt, vocab.txt
Output files – pattern-0.txt, pattern-1.txt, pattern-2.txt, pattern-3.txt, pattern-4.txt

Steps:

1. Extract data from the topic file
2. Generate 1-itemset candidates
3. Prune this 1-itemset candidates based on the minimum support
4. Continue by forming candidate item sets with two, three, four etc. items until no candidate item set is frequent.
5. Sort all the frequent items found in decreasing order of minimum support
6. Write all the frequent patterns found to a file pattern-i.txt by mapping the item ids to vocabulary using vocab.txt

Step 2: Mining Maximal/Closed Patterns

Max Pattern Mining
The frequent patterns found from topic files in Step 1 are used to generate the max patterns for topics. They are found by iterating through all the frequent patterns and for each pattern finding out if there exists any frequent super-pattern. And if there exist no frequent super pattern, then it is considered to be a maximal pattern.

Files:
Code File – MaxPatterns.py
Input files – pattern-0.txt, pattern-1.txt, pattern-2.txt, pattern-3.txt, pattern-4.txt
Output files – max-0.txt, max-1.txt, max-2.txt, max-3.txt, max-4.txt

Steps:
1. Import data from the frequent pattern file for the topic.
2. Iterate through the frequent patterns and for each item set find out if there exists any frequent superpattern
3. If a frequent super-pattern is not found then add it to the set of maximal patterns.
4. Sort all the maximal patterns found in descending order of minimum support.
5. Write all the maximal patterns found to file max-i.txt

Closed Pattern Mining
The frequent patterns found from topic files in Step 1 are used to generate the closed patterns for topics. They are found by iterating through all the frequent patterns and for each pattern finding out if there exists any super-pattern with the same support. And if there exist no super pattern with the same support, then it is considered to be a closed pattern.

Files:
Code File – ClosedPatterns.py
Input files – pattern-0.txt, pattern-1.txt, pattern-2.txt, pattern-3.txt, pattern-4.txt
Output files – closed-0.txt, closed-1.txt, closed-2.txt, closed-3.txt, closed-4.txt

Steps:
1. Import data from the frequent pattern file for the topic.
2. Iterate through the frequent patterns and for each item set find out if there exists any superpattern with the same support.
3. If a super-pattern with the same support is not found then add it to the set of closed patterns.
4. Sort all the closed patterns found in descending order of minimum support.
5. Write all the closed patterns found to file 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.

From the patterns mined, the domains could be figured out as listed below -
0 – Data Mining
1 – Machine learning

2 – Information Retrieval
3 – Theory
4 – Database
It is observed that the number of maximal/closed patterns generated are less in number compared to frequent patterns while maintaining the quality. It was also observed that too many frequent patterns were generated if the support threshold is too low. Mining Closed and Maximal frequent patterns fixed this but there was loss of information in case of maximal patterns.

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

Yes the result is satisfying. Although frequent pattern mining has an important role in many data mining tasks, however, it often generates a large number of patterns, which reduces its efficiency and effectiveness. Here looking into Max, or Closed patterns, they provide the same amount of information as frequent patterns, so mining all the frequent patterns may not be necessary. Comparing to frequent patterns, frequent Max, or Closed patterns generated were less in number, and therefore improved the efficiency and effectiveness.

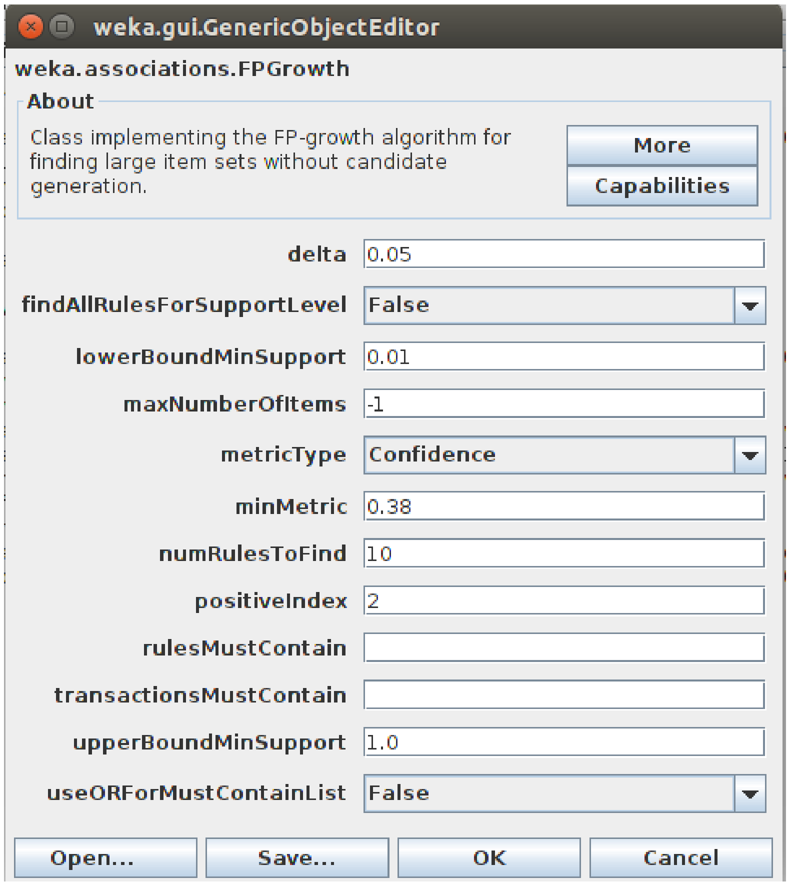
Step 3: Association Rule Mining by Weka

Steps:

1. Ran featureGenerator.py to generate the .arff files for all the topics
2. Imported the .arff file to Weka
3. Selected FP growth as the associator
4. Tuned the values of lowerBoundMinSupport and minMetric to generate at least 10 association rules for each topic. (The measures minimum support and minimum confidence were iteratively changed by a factor, until a required number of rules has been generated)

Below are the parameters min_sup and min_conf used to generate at least 10 association rules for each topic along with the screenshots.

Topic 0: min_support – 0.01; min_conf – 0.38


weka.gui.GenericObjectEditor

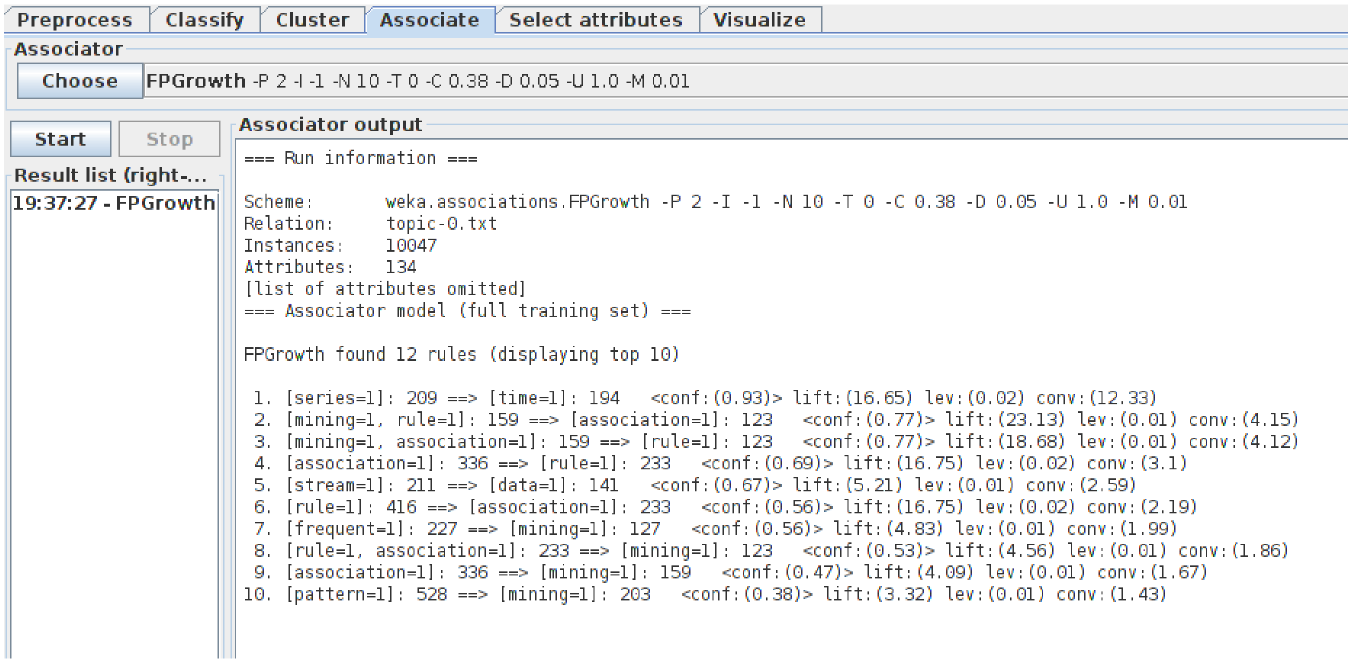
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.01
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.38
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

Suma Vangala (netid: svangal2)



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.38 -D 0.05 -U 1.0 -M 0.01

19:37:27 - FPGrowth

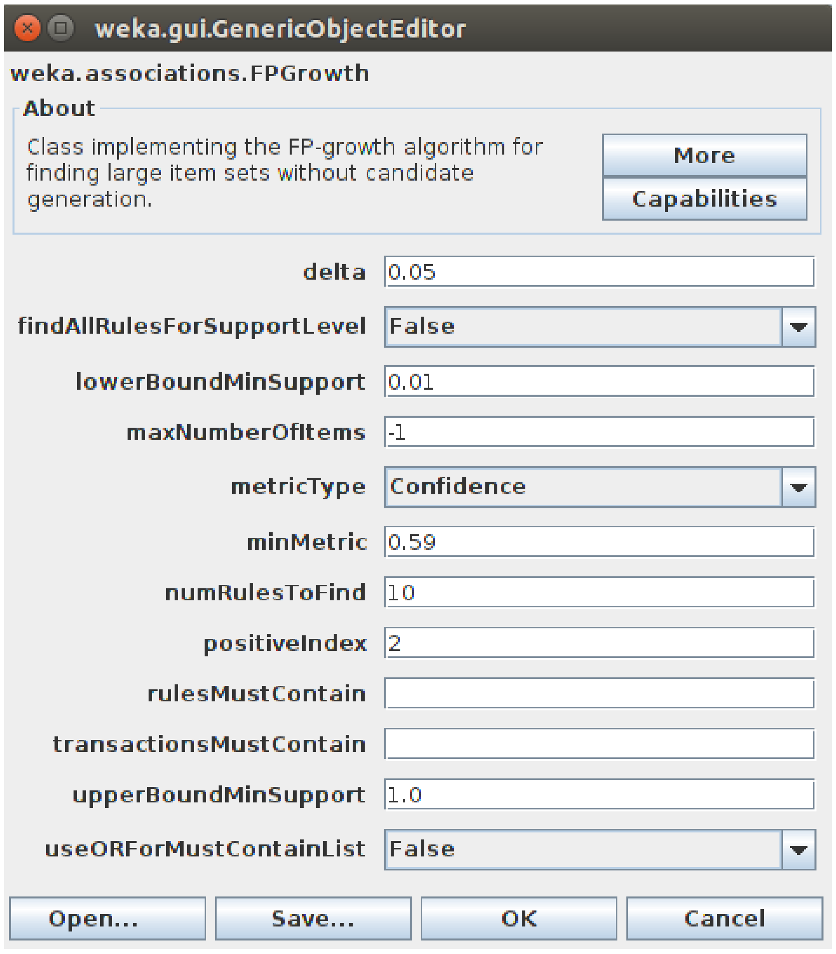
```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.38 -D 0.05 -U 1.0 -M 0.01
Relation:     topic-0.txt
Instances:    10047
Attributes:   134
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 12 rules (displaying top 10)

 1. [series=1]: 209 ==> [time=1]: 194    <conf:(0.93)> lift:(16.65) lev:(0.02) conv:(12.33)
 2. [mining=1, rule=1]: 159 ==> [association=1]: 123    <conf:(0.77)> lift:(23.13) lev:(0.01) conv:(4.15)
 3. [mining=1, association=1]: 159 ==> [rule=1]: 123    <conf:(0.77)> lift:(18.68) lev:(0.01) conv:(4.12)
 4. [association=1]: 336 ==> [rule=1]: 233    <conf:(0.69)> lift:(16.75) lev:(0.02) conv:(3.1)
 5. [stream=1]: 211 ==> [data=1]: 141    <conf:(0.67)> lift:(5.21) lev:(0.01) conv:(2.59)
 6. [rule=1]: 416 ==> [association=1]: 233    <conf:(0.56)> lift:(16.75) lev:(0.02) conv:(2.19)
 7. [frequent=1]: 227 ==> [mining=1]: 127    <conf:(0.56)> lift:(4.83) lev:(0.01) conv:(1.99)
 8. [rule=1, association=1]: 233 ==> [mining=1]: 123    <conf:(0.53)> lift:(4.56) lev:(0.01) conv:(1.86)
 9. [association=1]: 336 ==> [mining=1]: 159    <conf:(0.47)> lift:(4.09) lev:(0.01) conv:(1.67)
10. [pattern=1]: 528 ==> [mining=1]: 203    <conf:(0.38)> lift:(3.32) lev:(0.01) conv:(1.43)
```

Topic 1: min_support – 0.01; min_conf – 0.59



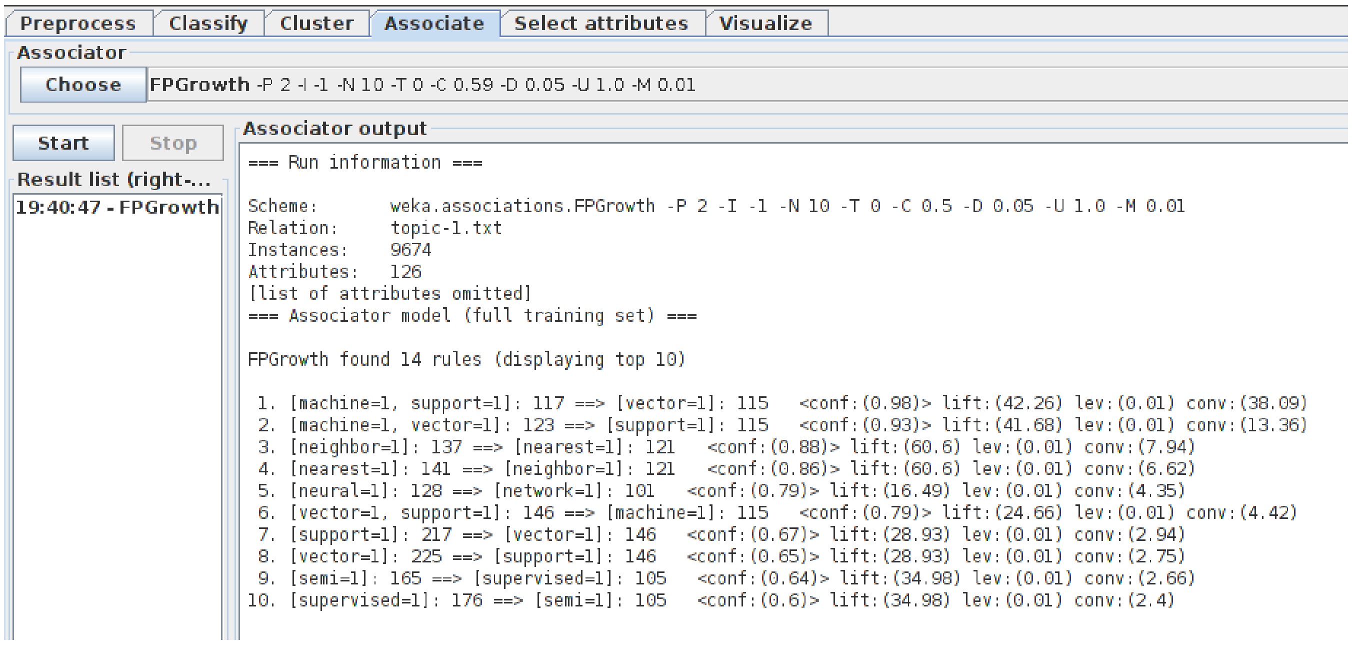
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.01
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.59
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

Suma Vangala (netid: svangal2)



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.59 -D 0.05 -U 1.0 -M 0.01

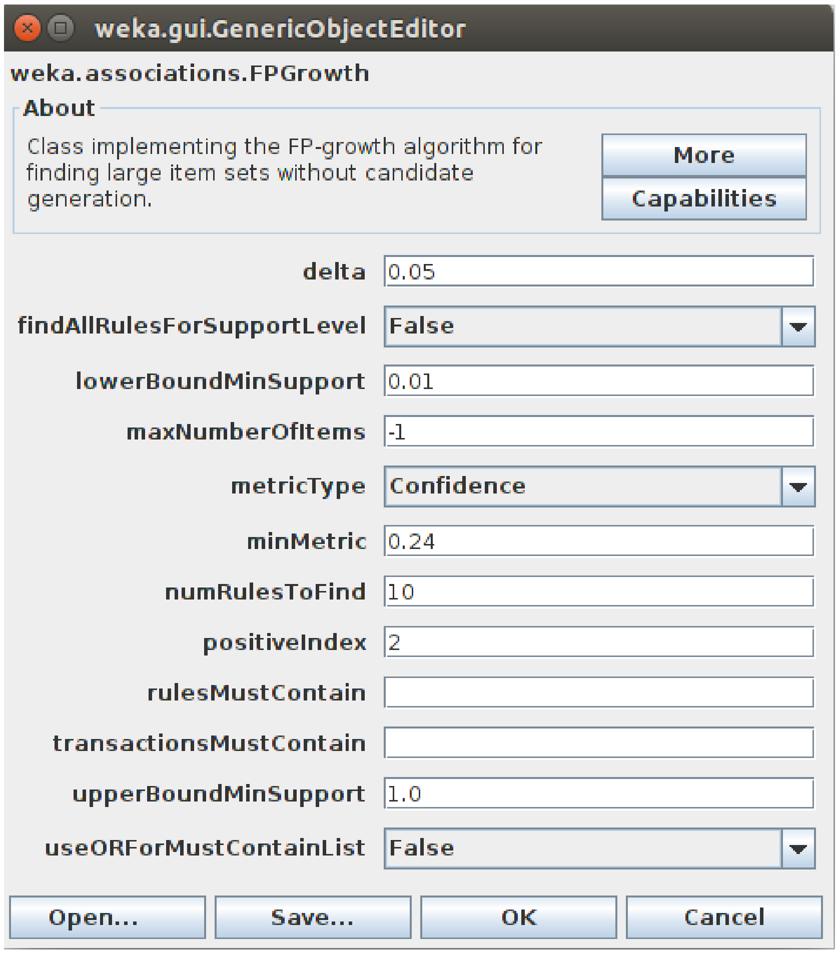
```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.5 -D 0.05 -U 1.0 -M 0.01
Relation:     topic-1.txt
Instances:    9674
Attributes:   126
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 14 rules (displaying top 10)

 1. [machine=1, support=1]: 117 ==> [vector=1]: 115   <conf:(0.98)> lift:(42.26) lev:(0.01) conv:(38.09)
 2. [machine=1, vector=1]: 123 ==> [support=1]: 115   <conf:(0.93)> lift:(41.68) lev:(0.01) conv:(13.36)
 3. [neighbor=1]: 137 ==> [nearest=1]: 121   <conf:(0.88)> lift:(60.6) lev:(0.01) conv:(7.94)
 4. [nearest=1]: 141 ==> [neighbor=1]: 121   <conf:(0.86)> lift:(60.6) lev:(0.01) conv:(6.62)
 5. [neural=1]: 128 ==> [network=1]: 101   <conf:(0.79)> lift:(16.49) lev:(0.01) conv:(4.35)
 6. [vector=1, support=1]: 146 ==> [machine=1]: 115   <conf:(0.79)> lift:(24.66) lev:(0.01) conv:(4.42)
 7. [support=1]: 217 ==> [vector=1]: 146   <conf:(0.67)> lift:(28.93) lev:(0.01) conv:(2.94)
 8. [vector=1]: 225 ==> [support=1]: 146   <conf:(0.65)> lift:(28.93) lev:(0.01) conv:(2.75)
 9. [semi=1]: 165 ==> [supervised=1]: 105   <conf:(0.64)> lift:(34.98) lev:(0.01) conv:(2.66)
10. [supervised=1]: 176 ==> [semi=1]: 105   <conf:(0.6)> lift:(34.98) lev:(0.01) conv:(2.4)
```

Topic 2: min_support – 0.01; min_conf – 0.24



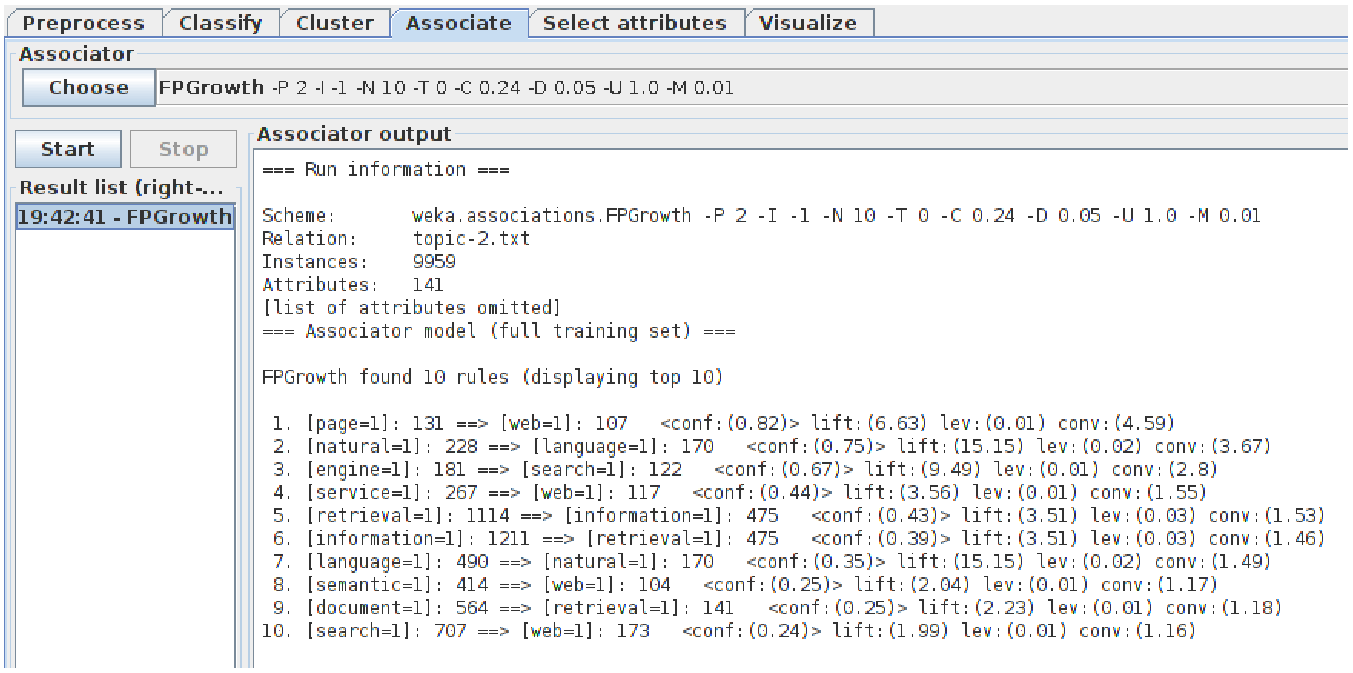
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.01
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.24
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

Suma Vangala (netid: svangal2)



Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Associator

Choose FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.24 -D 0.05 -U 1.0 -M 0.01

Start Stop

Result list (right-...

19:42:41 - FPGrowth

Associator output

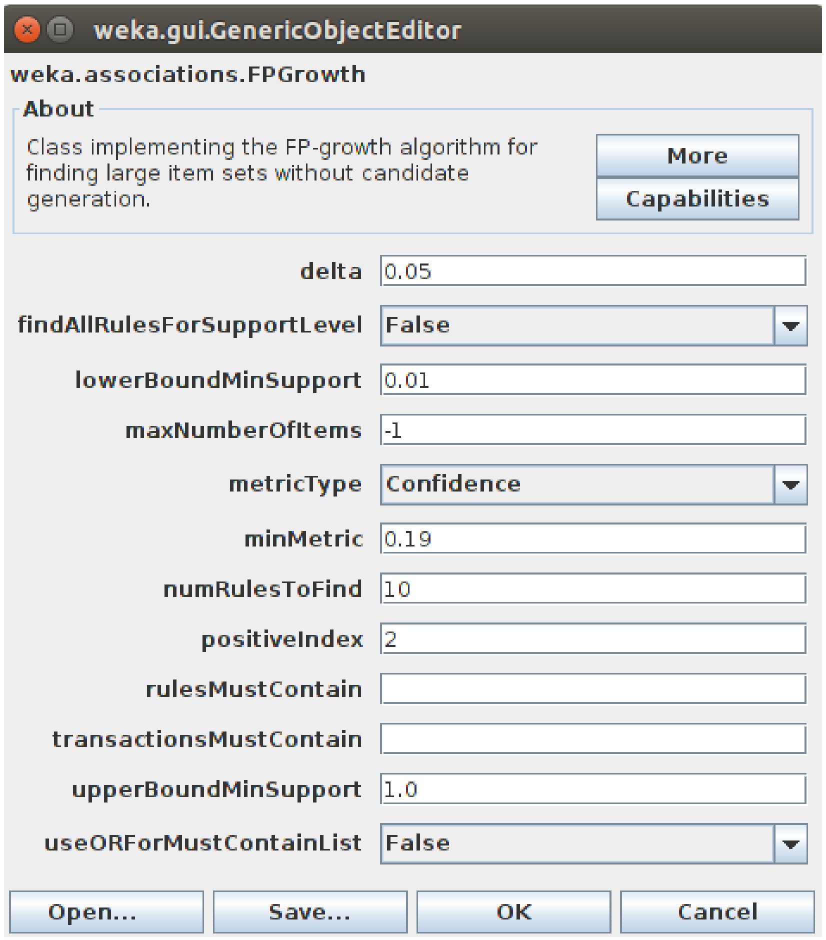
```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.24 -D 0.05 -U 1.0 -M 0.01
Relation:     topic-2.txt
Instances:    9959
Attributes:   141
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 10 rules (displaying top 10)

 1. [page=1]: 131 ==> [web=1]: 107   <conf:(0.82)> lift:(6.63) lev:(0.01) conv:(4.59)
 2. [natural=1]: 228 ==> [language=1]: 170   <conf:(0.75)> lift:(15.15) lev:(0.02) conv:(3.67)
 3. [engine=1]: 181 ==> [search=1]: 122   <conf:(0.67)> lift:(9.49) lev:(0.01) conv:(2.8)
 4. [service=1]: 267 ==> [web=1]: 117   <conf:(0.44)> lift:(3.56) lev:(0.01) conv:(1.55)
 5. [retrieval=1]: 1114 ==> [information=1]: 475   <conf:(0.43)> lift:(3.51) lev:(0.03) conv:(1.53)
 6. [information=1]: 1211 ==> [retrieval=1]: 475   <conf:(0.39)> lift:(3.51) lev:(0.03) conv:(1.46)
 7. [language=1]: 490 ==> [natural=1]: 170   <conf:(0.35)> lift:(15.15) lev:(0.02) conv:(1.49)
 8. [semantic=1]: 414 ==> [web=1]: 104   <conf:(0.25)> lift:(2.04) lev:(0.01) conv:(1.17)
 9. [document=1]: 564 ==> [retrieval=1]: 141   <conf:(0.25)> lift:(2.23) lev:(0.01) conv:(1.18)
10. [search=1]: 707 ==> [web=1]: 173   <conf:(0.24)> lift:(1.99) lev:(0.01) conv:(1.16)
```

Topic 3: min_support – 0.01; min_conf – 0.19



weka.gui.GenericObjectEditor

weka.associations.FPGrowth

About

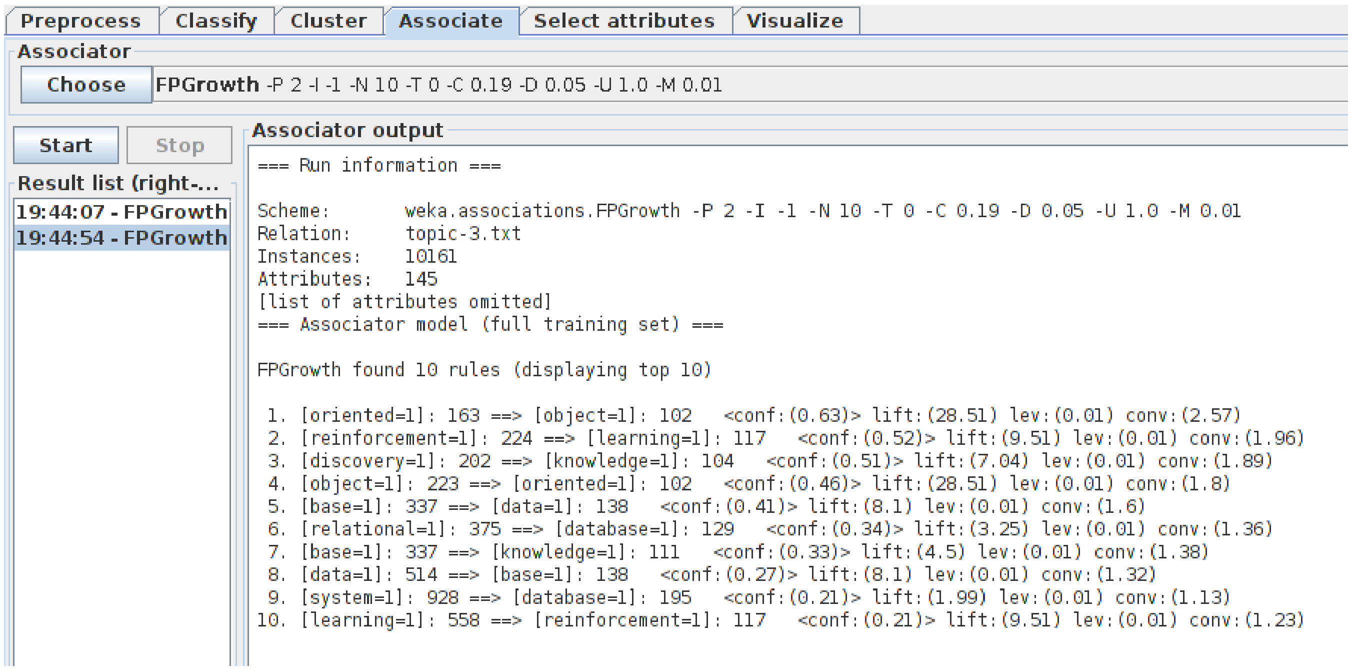
Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

More | Capabilities

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.01
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.19
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

Open... | Save... | OK | Cancel

Suma Vangala (netid: svangal2)



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.19 -D 0.05 -U 1.0 -M 0.01

19:44:07 - FPGrowth
19:44:54 - FPGrowth

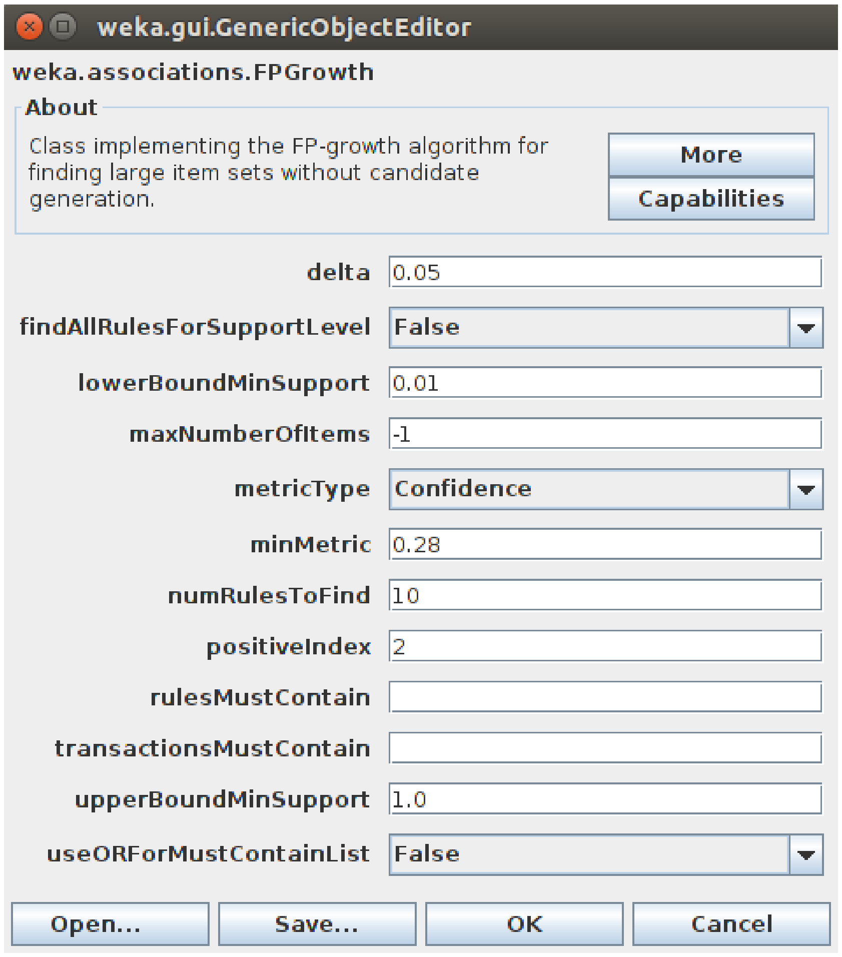
```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.19 -D 0.05 -U 1.0 -M 0.01
Relation:     topic-3.txt
Instances:    10161
Attributes:   145
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 10 rules (displaying top 10)

 1. [oriented=1]: 163 ==> [object=1]: 102   <conf:(0.63)> lift:(28.51) lev:(0.01) conv:(2.57)
 2. [reinforcement=1]: 224 ==> [learning=1]: 117   <conf:(0.52)> lift:(9.51) lev:(0.01) conv:(1.96)
 3. [discovery=1]: 202 ==> [knowledge=1]: 104   <conf:(0.51)> lift:(7.04) lev:(0.01) conv:(1.89)
 4. [object=1]: 223 ==> [oriented=1]: 102   <conf:(0.46)> lift:(28.51) lev:(0.01) conv:(1.8)
 5. [base=1]: 337 ==> [data=1]: 138   <conf:(0.41)> lift:(8.1) lev:(0.01) conv:(1.6)
 6. [relational=1]: 375 ==> [database=1]: 129   <conf:(0.34)> lift:(3.25) lev:(0.01) conv:(1.36)
 7. [base=1]: 337 ==> [knowledge=1]: 111   <conf:(0.33)> lift:(4.5) lev:(0.01) conv:(1.38)
 8. [data=1]: 514 ==> [base=1]: 138   <conf:(0.27)> lift:(8.1) lev:(0.01) conv:(1.32)
 9. [system=1]: 928 ==> [database=1]: 195   <conf:(0.21)> lift:(1.99) lev:(0.01) conv:(1.13)
10. [learning=1]: 558 ==> [reinforcement=1]: 117   <conf:(0.21)> lift:(9.51) lev:(0.01) conv:(1.23)
```

Topic 4: min_support – 0.01; min_conf – 0.28



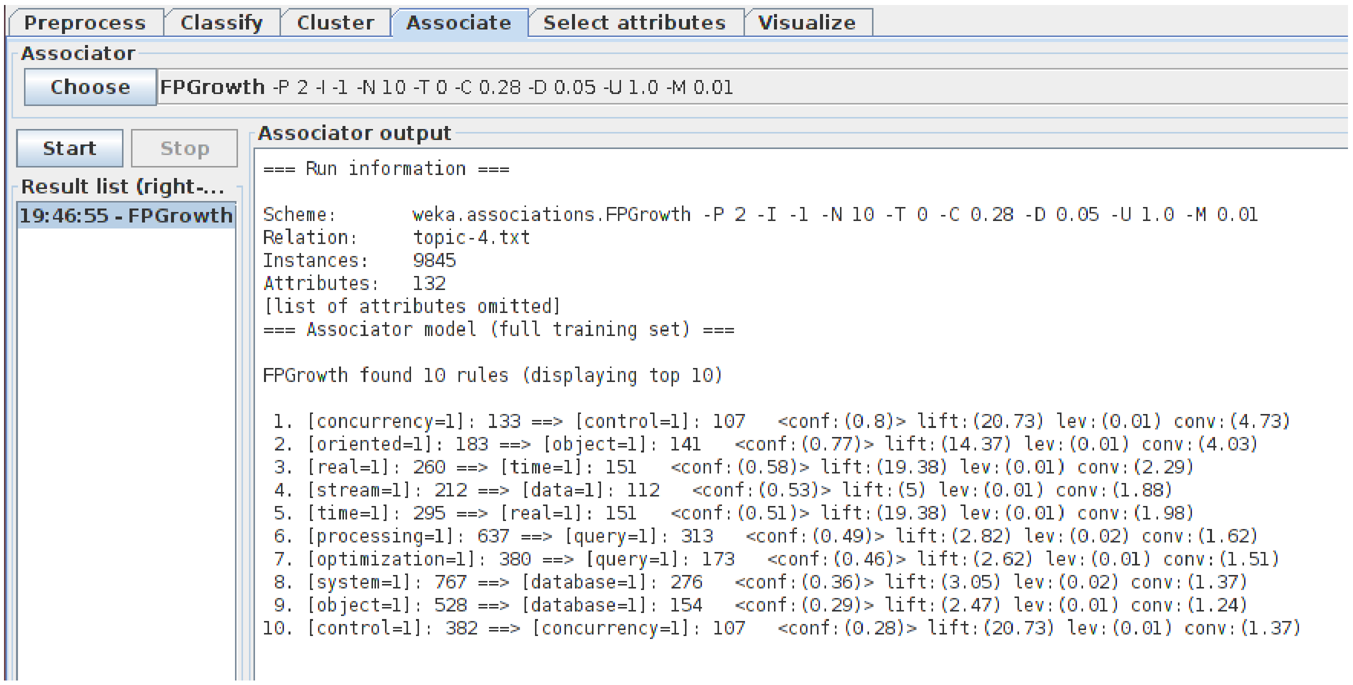
weka.associations.FPGrowth

About

Class implementing the FP-growth algorithm for finding large item sets without candidate generation.

delta	0.05
findAllRulesForSupportLevel	False
lowerBoundMinSupport	0.01
maxNumberOfItems	-1
metricType	Confidence
minMetric	0.28
numRulesToFind	10
positiveIndex	2
rulesMustContain	
transactionsMustContain	
upperBoundMinSupport	1.0
useORForMustContainList	False

Suma Vangala (netid: svangal2)



FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.28 -D 0.05 -U 1.0 -M 0.01

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.28 -D 0.05 -U 1.0 -M 0.01
Relation:     topic-4.txt
Instances:    9845
Attributes:   132
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 10 rules (displaying top 10)

 1. [concurrency=1]: 133 ==> [control=1]: 107   <conf:(0.8)> lift:(20.73) lev:(0.01) conv:(4.73)
 2. [oriented=1]: 183 ==> [object=1]: 141   <conf:(0.77)> lift:(14.37) lev:(0.01) conv:(4.03)
 3. [real=1]: 260 ==> [time=1]: 151   <conf:(0.58)> lift:(19.38) lev:(0.01) conv:(2.29)
 4. [stream=1]: 212 ==> [data=1]: 112   <conf:(0.53)> lift:(5) lev:(0.01) conv:(1.88)
 5. [time=1]: 295 ==> [real=1]: 151   <conf:(0.51)> lift:(19.38) lev:(0.01) conv:(1.98)
 6. [processing=1]: 637 ==> [query=1]: 313   <conf:(0.49)> lift:(2.82) lev:(0.02) conv:(1.62)
 7. [optimization=1]: 380 ==> [query=1]: 173   <conf:(0.46)> lift:(2.62) lev:(0.01) conv:(1.51)
 8. [system=1]: 767 ==> [database=1]: 276   <conf:(0.36)> lift:(3.05) lev:(0.02) conv:(1.37)
 9. [object=1]: 528 ==> [database=1]: 154   <conf:(0.29)> lift:(2.47) lev:(0.01) conv:(1.24)
10. [control=1]: 382 ==> [concurrency=1]: 107   <conf:(0.28)> lift:(20.73) lev:(0.01) conv:(1.37)
```

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.

The quality of patterns that satisfy both min_support and min_confindence are more revealing than the patterns just satisfying min_support as in Step 1.
Association rules generated by using Weka on topic files revealed interesting relationships. From Step 1, we only got the set of all possible relationships that satisfied minimum support. But the result generated here through Weka satisfying both min_sup and min_conf are not the set of all possible relationships, but the set of all interesting ones. These rules helped to uncover relationships between seemingly unrelated words from titles.

Step 4: Re-rank by Purity of Patterns

The output obtained from Step 1 has been used in order to re rank them using purity. The algorithm used in step 1 to generate frequent patterns is Apriori algorithm which uses a breadth-first search strategy to count the support of item sets and uses a candidate generation function which exploits the downward closure property of support. Then purity is calculated for those generated patterns. Purity of a pattern is measured 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).

Purity essentially measures the distinctness of a pattern in one topic compared to that in any other topic. It is calculated by using the following formula:

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

Then support is incorporated in order to re rank the patterns. For all the items with the same purity, minimum support is used as another measure and they are re ordered in decreasing order of support.

Steps:

1. Import data from the pattern files generated in Step 1
2. For each topic, purity of items is calculated by using the above mentioned formula
3. All the items are ranked in descending order of purity
4. For items with same purity, the items are re-ranked again in descending order of minimum support. This further ensures that the most frequent and pure items are always at the top.

Files:

Code file – ReRankByPurity.py

Input files – pattern-0.txt, pattern-1.txt, pattern-2.txt, pattern-3.txt, pattern-4.txt, vocab.txt

Output files – purity-0.txt, purity-1.txt, purity-2.txt, purity-3.txt, purity-4.txt

Step5: Bonus
In order to improve the quality of mined phrases, an algorithm has been implemented which takes into account coverage, purity and phraseness. The measures are defined as follows:
Coverage: A representative keyphrase for a topic should cover many documents within that topic.
Purity: A phrase is pure in a topic if it is only frequent in documents belonging to that topic and not frequent in documents within other topics
Phraseness: A group of words should be combined together as a phrase if they co-occur significantly more often than the expected chance co-occurrence frequency, given that each term in the phrase occurs independently.

Coverage:
A representative keyphrase for a topic should cover many documents within that topic. We directly quantify the coverage measure of a phrase as the probability of seeing a phrase in a random topic-t word set

$$\pi_t^{cov}(p) = P(e_t(p)) = \frac{f_t(p)}{|D_t|}$$

Purity:
A phrase is pure in topic t if it is only frequent in documents about topic t and not frequent in documents about other topics. We measure the purity of a keyphrase by comparing the probability of seeing a phrase in the topic-t collection of word sets and the probability of seeing it in any other topic-t collection (t = 0, 1, . . . , k,t = t). The purity of a keyphrase compares the probability of seeing it in the topic-t collection and the maximal probability of seeing it in any reference collection:

$$\begin{aligned}\pi_t^{pur}(p) &= \log \frac{P(e_t(p))}{\max_{t'} P(e_{t,t'}(p))} \\ &= \log \frac{f_t(p)}{|D_t|} - \log \max_{t'} \frac{f_t(p) + f_{t'}(p)}{|D_{t,t'}|}\end{aligned}$$

Phraseness:
A group of words should be grouped into a phrase if they co-occur significantly more frequent than the expected co-occurrence frequency given that each word in the phrase occurs independently. For example, while ‘active learning’ is a good keyphrase as in the Machine Learning topic ‘learning classification’ is not, since the latter two words co-occur only because both of them are popular in the topic. We therefore compare the probability of seeing a phrase p = {$w_1 \ldots w_n$} and seeing the n words $w_1 \ldots w_n$ independently in topic-t documents:

$$\begin{aligned}\pi_t^{phr}(p) &= \log \frac{P(e_t(p))}{\prod_{w \in p} P(e_t(w))} \\ &= \log \frac{f_t(p)}{|D_t|} - \sum_{w \in p} \log \frac{f_t(w)}{|D_t|}\end{aligned}$$

Then the rank is calculated using the above mentioned three measures using the formula:

$$\pi_t^{cov}[(1-\omega)\pi_t^{pur} + \omega\pi_t^{phr}](p)$$

Steps:

1. Import data from the topic and pattern files generated in Step 1
2. For each topic, purity, coverage and phraseness of items is calculated by using the above mentioned formulae
3. Rank which is a combined measure of purity, phraseness and coverage is calculated
4. All the items are ranked in descending order of rank

Files:
Code file – ReRanking.py
Input files – pattern-0.txt, pattern-1.txt, pattern-2.txt, pattern-3.txt, pattern-4.txt, vocab.txt
Output files – purity-0.txt, purity-1.txt, purity-2.txt, purity-3.txt, purity-4.txt

<u>Topic-0 top 20 frequent patterns re-ranked by using the measures described above</u>

```
0.0206  mining data
0.0179  rule association
0.0167  mining
0.0163  graph
0.0149  series time
0.0123  pattern mining
0.0113  mining rule association
0.0100  mining association
0.0097  mining rule
0.0082  mining frequent
0.0071  approximation algorithm
0.0060  preserving privacy
0.0052  pattern frequent
0.0050  association
0.0050  lower bound
0.0049  reduction dimensionality
0.0046  dimensional data
0.0043  algorithm
0.0043  set data
0.0043  approximation
```

For example: above are the top 20 frequent patterns generated from the algorithm for topic-0. As compared to just the patterns generated using purity or minimum support, this algorithm generates patterns which are much more related and interesting. It can also be observed from the above generated patterns is that we could directly compare and rank phrases of different lengths.
It was also observed from the re-ranked patterns that an item with low coverage turned out to be of low quality. Also the tradeoff between purity and phraseness is controlled by ω. So ω was tuned such that the frequent phrases are ranked higher.

Suma Vangala (netid: svangal2)

Bash Script:

Bash script GeneratePatterns.sh has been included as a part of the solution. It can be executed on Linux environment by using the command: ./GeneratePatterns.sh

This will generate all the pattern-i.txt, max-i.txt, closed.txt and purity-i.txt files for all the topics.