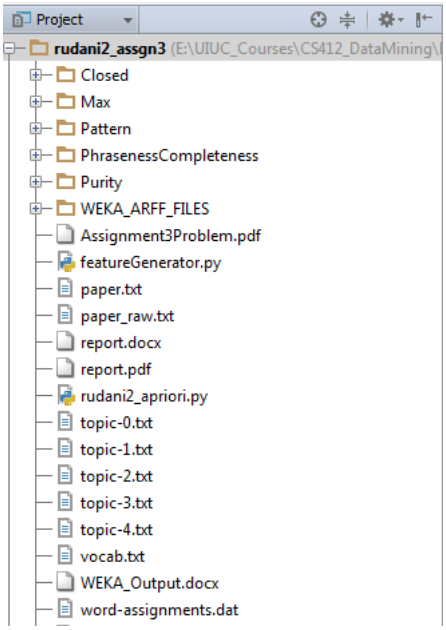
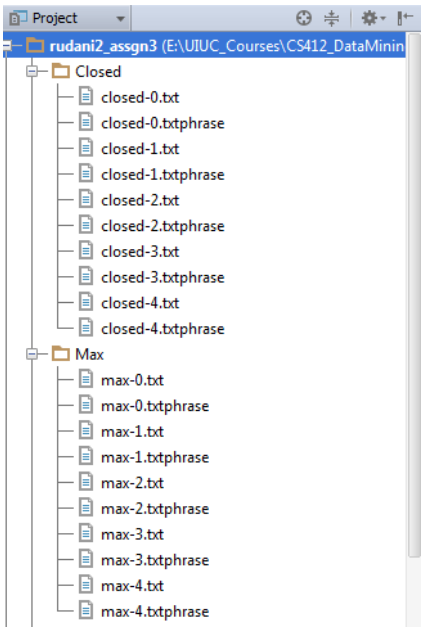
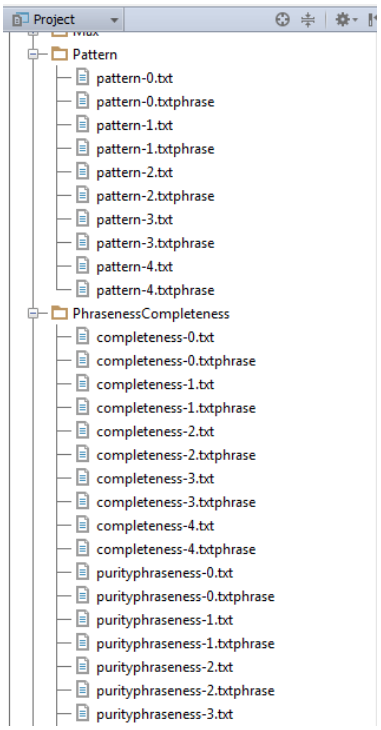
Jigar Rudani (rudani2)
CS 412 Data Mining

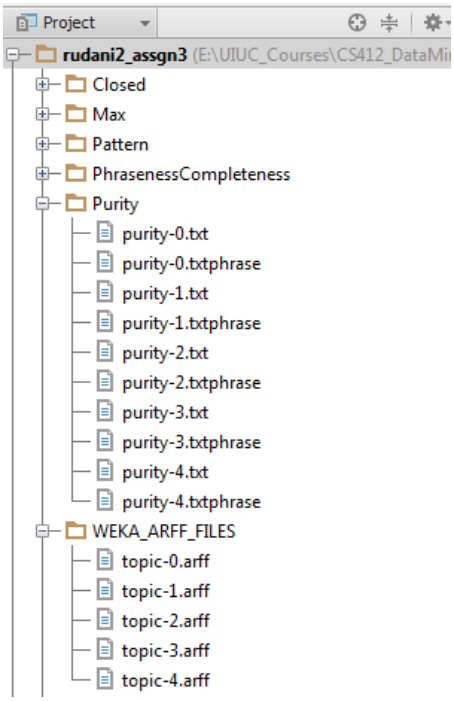
Assignment 3 Frequent Pattern Mining Programming (Fall 2014)

Organization









Algorithm

Step 1: Mining Frequent Patterns for Each Topic

Apriori pattern mining

1. Initially we consider each items in the transaction and generate Candidate C1 item set.
2. There are 2 important steps in Apriori pattern mining
    a. Prune Step
    b. Join Step
3. After generating C1 item set. We apply Prune step to it and generate L1 itemset
4. We perform Join step for each Lk itemset.
5. After Join step we generate remaining Ck item set (K >= 2) by applying Prune step to each item generated.
6. Repeat step 4 to step 5 till Ck = Null

Prune Step

1. We calculate the frequency of each item by counting the number of occurrences in the topic file.
2. We then compare frequency with min_support.
3. If frequency > min_support then
    a. Include the item in Frequent K Item Set
   Else
    a. Excludes item
4. We can check the Frequency of K (K > 1) itemset by checking whether each subset of itemset present in Lk-1 Itemset.
5. I have used min_support as 0.01 which I have explained in below step

Join Step

1. We perform self join operation on each Lk itemset.
    a. E.g [1,2] Join [1,3] → [1,2,3]

Output Path

1. I have created generalize method which helps to print desired output to each respective file in each respective folder.

Source Code →

1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***funcAprioriFreqPattern()***
   b. func_PruneItems()
   c. func_JoinItems()
   d. func_redirecttoOutputFolder()

```
# Define a func_PruneItems() function which performs pruning of ItemSet.
def func_PruneItems(transactionList,itemSet,listItemwithFreq,min_support):...

# Define a func_JoinItems() function which performs self join operation
def func_JoinItems(itemSet,tempItemSet,itemSize):...

# Define a funcAprioriFreqPattern() function which parsed the Transaction and get the Frequent Patterns out of each Topic.
def funcAprioriFreqPattern(transactionList,min_support):

    itemSet = set()
    listItemwithFreq = defaultdict(int)
    itemSize = 2
    tempItemSet = set()
    tempCandidateItem = set()

    for trans in transactionList:
        for items in trans:
            itemSet.add(frozenset([items]))

    #print "\n<-- C1 Candidate Set -->\n"
    #func_DisplayCandidate(itemSet)
    LItemSet,listItemwithFreq = func_PruneItems(transactionList,itemSet,listItemwithFreq,min_support)
    #print "\n<-- L1 ItemSet -->\n"
    #func_DisplayListItemFreq(LItemSet,listItemwithFreq,itemSize - 1)
    while (LItemSet != set([])):
        tempItemSet = LItemSet
        tempCandidateItem = func_JoinItems(LItemSet,tempItemSet,itemSize)
        #print "\n<-- C(%d) Candidate Set -->\n" % itemSize
        #func_DisplayCandidate(tempCandidateItem)
        LItemSet,listItemwithFreq = func_PruneItems(transactionList,tempCandidateItem,listItemwithFreq,min_support)
        #print "<-- L(%d) ItemSet -->\n" % (itemSize)
        #func_DisplayListItemFreq(LItemSet,listItemwithFreq,itemSize)
        itemSize += 1
    return tempItemSet,listItemwithFreq
```

Step 2: Mining Maximal/Closed Patterns

Closed Pattern Mining

1. Closed Patterns are patterns which doesn't have proper superset. If it has proper superset then frequency should not be same.

Source Code →
1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***func_ClosedMaxPattern()***

Maximal Pattern Mining

1. Maximal Patterns are pattern which doesn't have superset.
2. All Maximal patterns are expected to be closed as well
3. In order to optimize the code I have created same method for Closed and Max as finding superset operation is common to both

Source Code →
1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***func_ClosedMaxPattern()***

```
# Define a func_ClosedMaxPattern() function to find Closed and Maximal Pattern
def func_ClosedMaxPattern(listItemwithFreq,dt,indx,outputClosedPatternName,outputMaxPatternName,dictVocab):

    closedPattern = []
    maxPattern = []
    tempList = []
    isAppendClosedPattern = None
    isAppendMaxPattern = None
    patternDigitObj = re.compile('\d+')
    for items,freq in listItemwithFreq.items():
        tempList.append([freq,items])
    sorted_items= sorted(tempList,key=itemgetter(0),reverse=True)

    for listItems_1 in sorted_items:
        isAppendClosedPattern = True
        isAppendMaxPattern = True
        for listItems_2 in sorted_items:
            if (listItems_1[1] not in listItems_2[1]):
                if (listItems_2[1] > listItems_1[1]):
                    isAppendMaxPattern = False
                    if (listItems_2[0] == listItems_1[0]):
                        isAppendClosedPattern = False
        if (isAppendClosedPattern):
            closedPattern.append([listItems_1[0],patternDigitObj.findall(str(listItems_1[1]))])
        if (isAppendMaxPattern):
            maxPattern.append([listItems_1[0],patternDigitObj.findall(str(listItems_1[1]))])
    func_redirecttoOutputFolder(closedPattern,dt,indx,outputClosedPatternName,dictVocab)
    func_redirecttoOutputFolder(maxPattern,dt,indx,outputMaxPatternName,dictVocab)
```

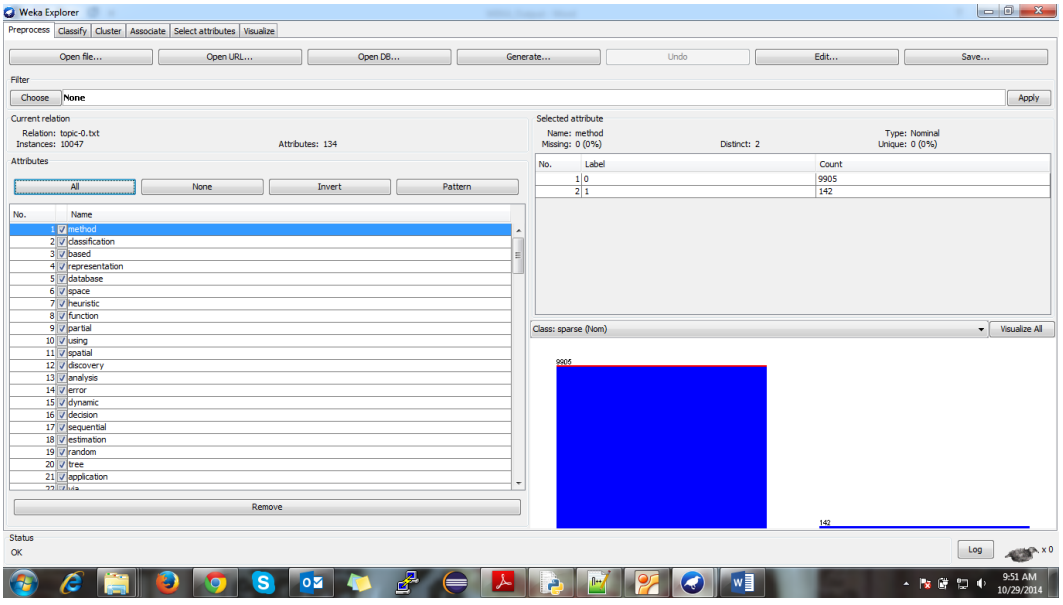
Step 3: Association Rule Mining by Weka

1. Genrated .arff files for each topics. Below are the screen shots for it

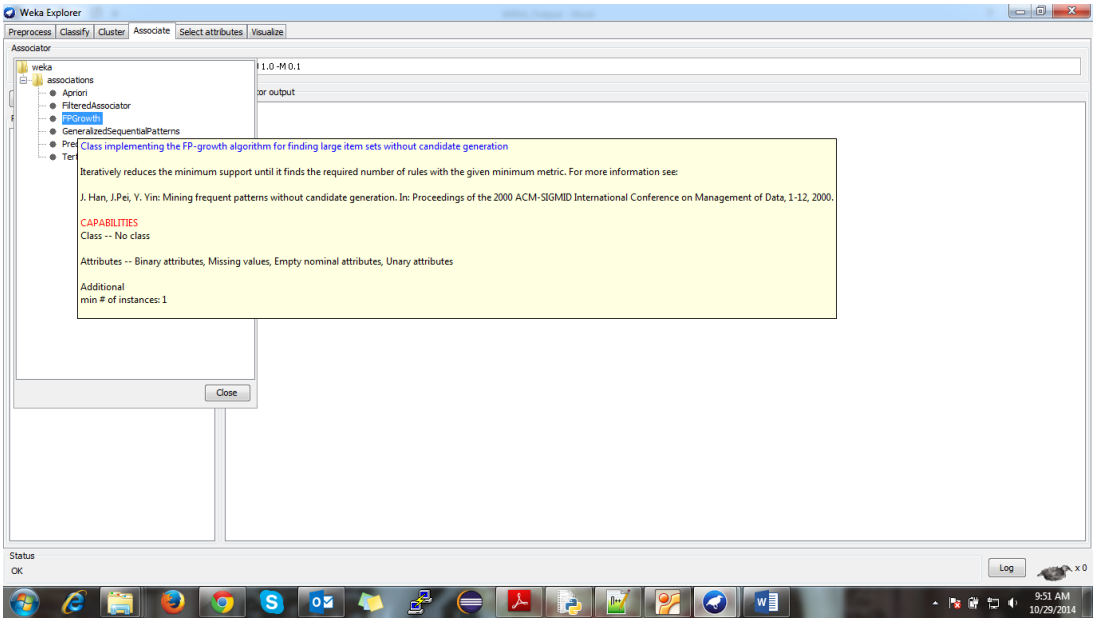
WEKA

Filename: - topic-0.arff

Preprocess →



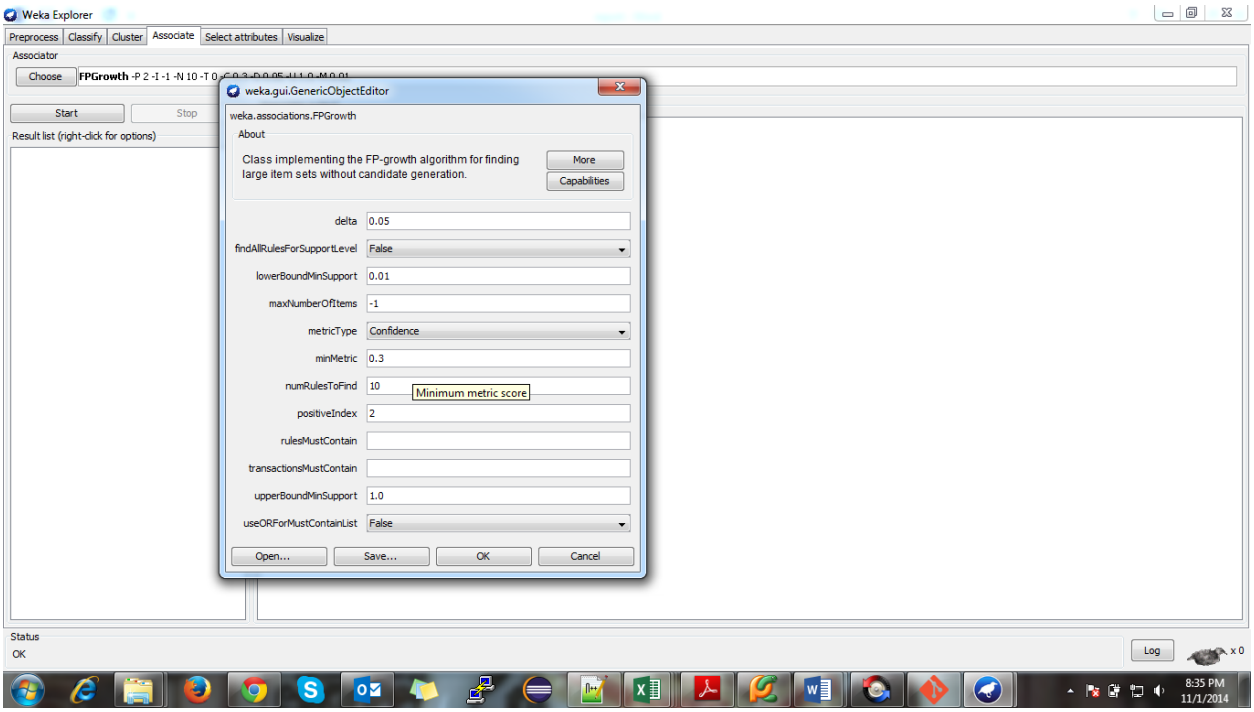
Associator Choose → FP GROWTH



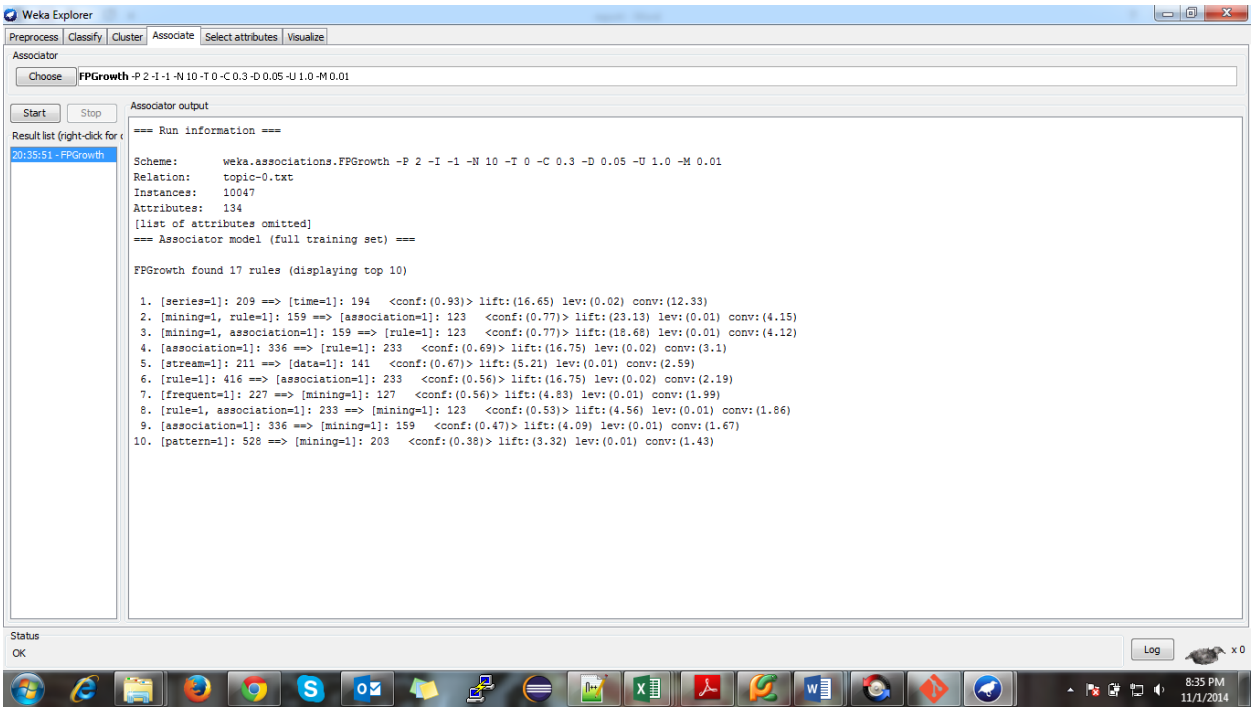
Parameter Settings →

lowerBoundMinSupport → 0.01

minMetric → 0.3

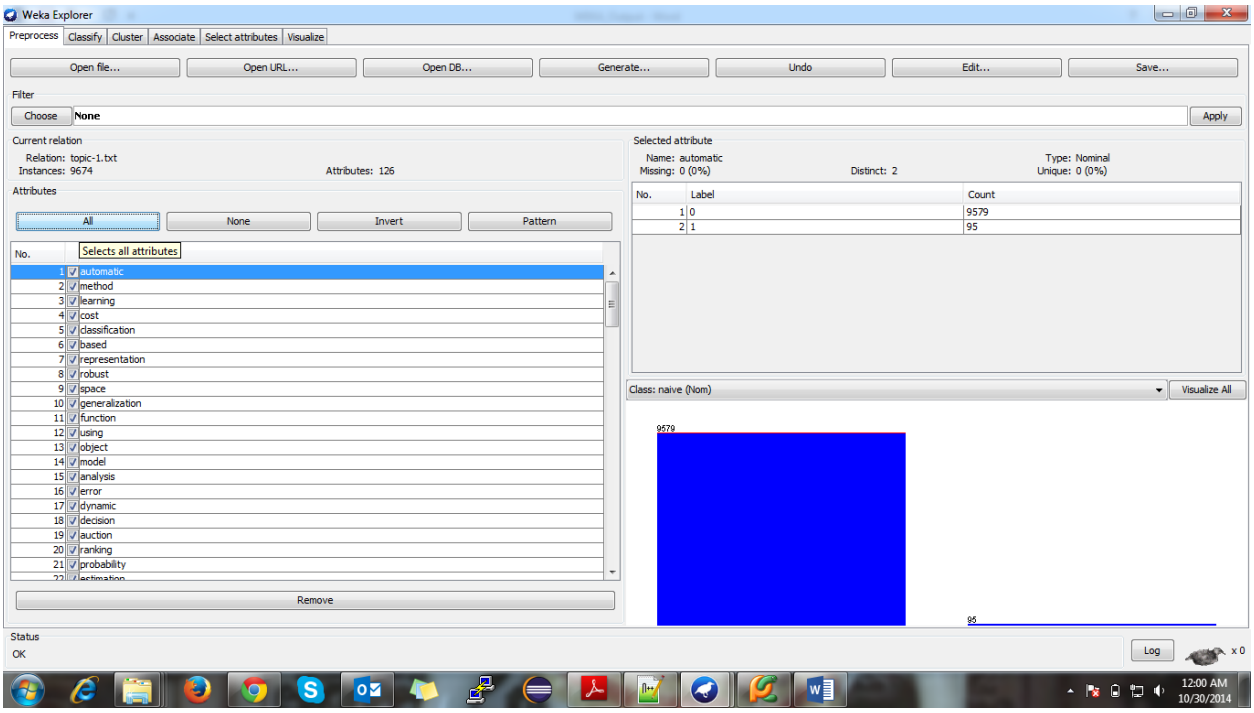


Association Rules →



Filename: - topic-1.arff

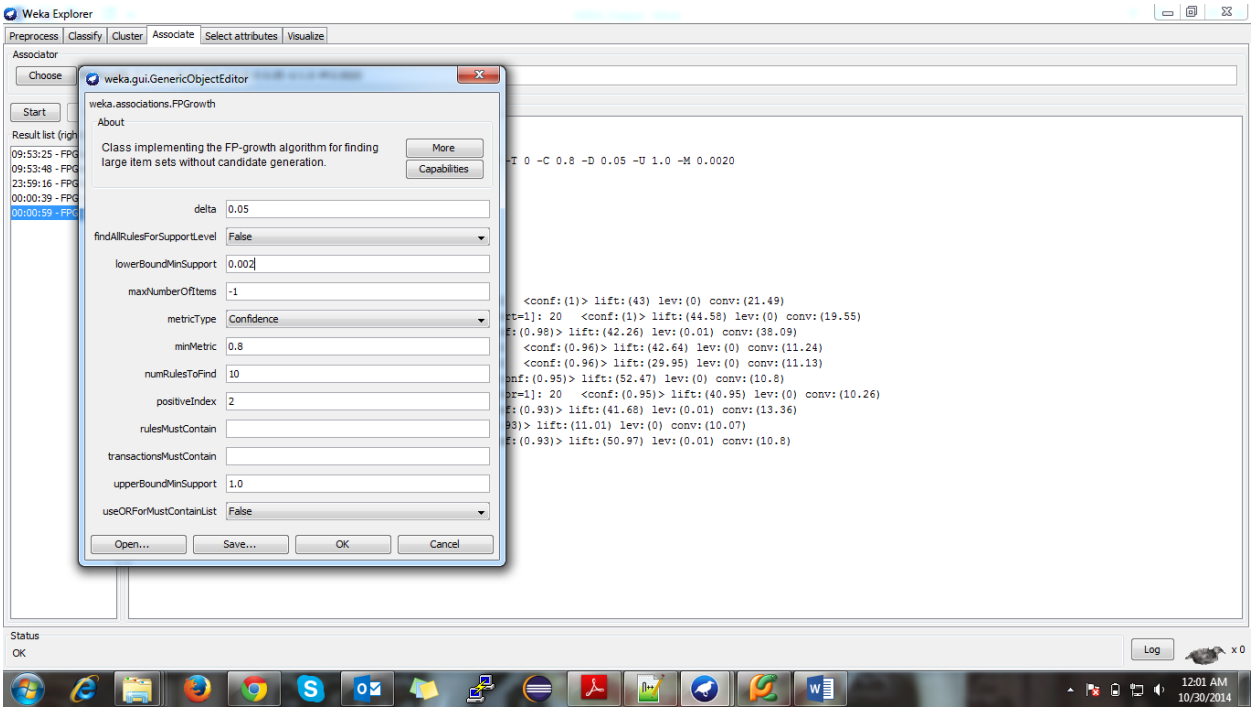
Preprocess →



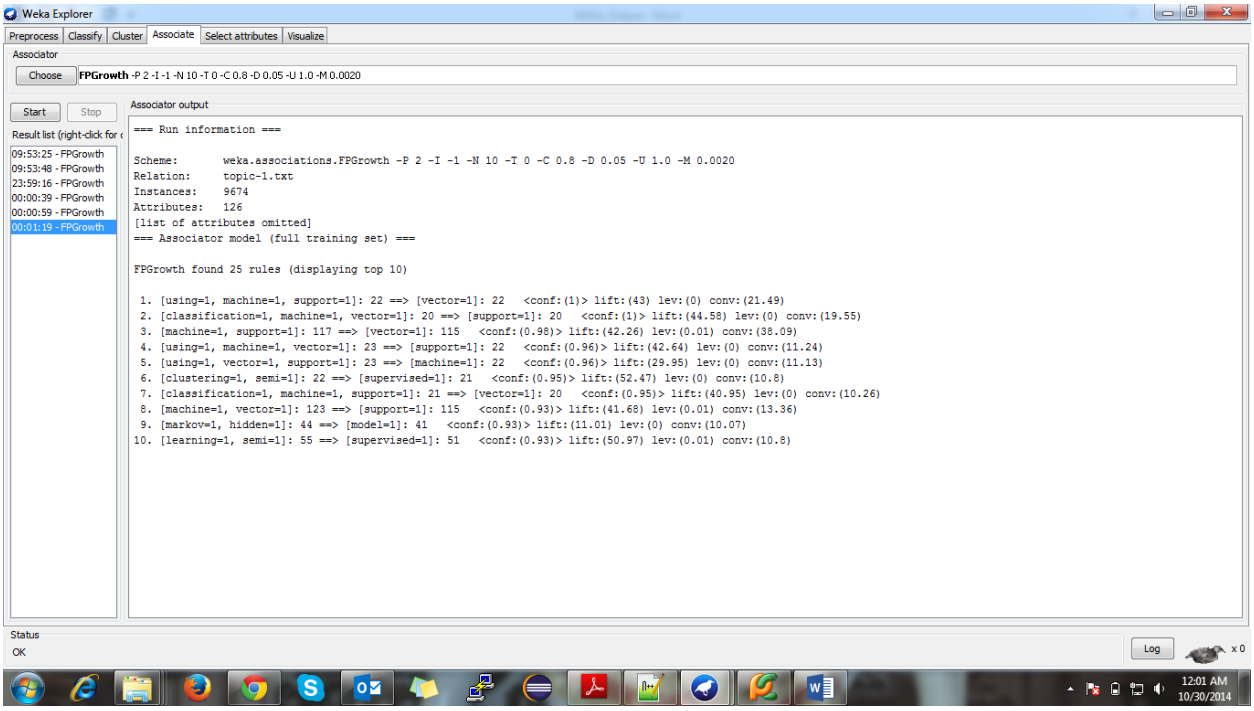
Parameter Settings →

lowerBoundMinSupport → 0.002

minMetric → 0.8

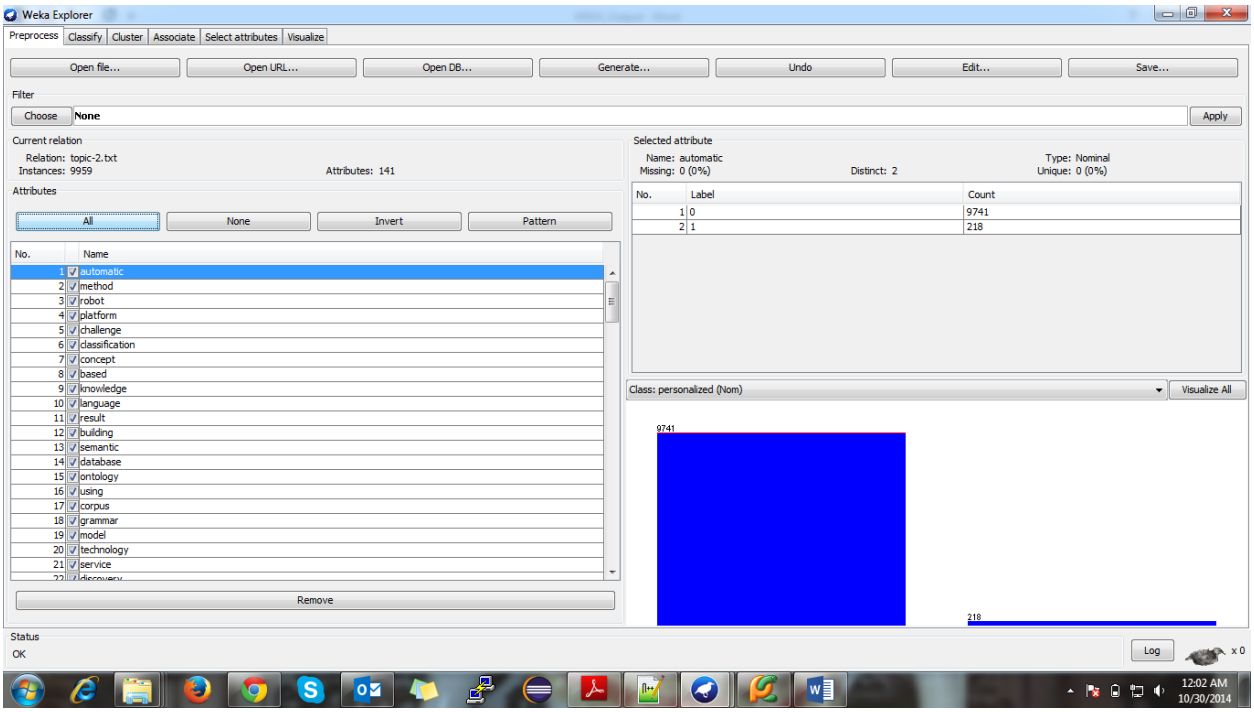


Association Rules →



Filename: - topic-2.arff

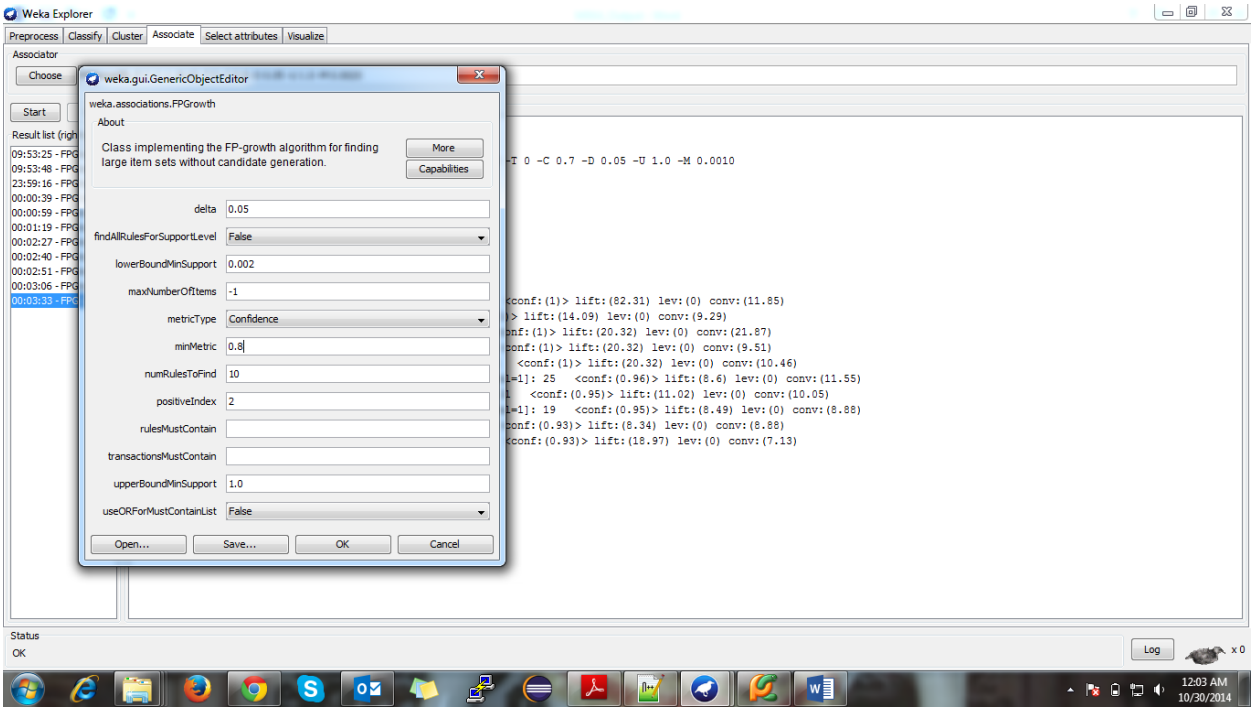
Preprocess →



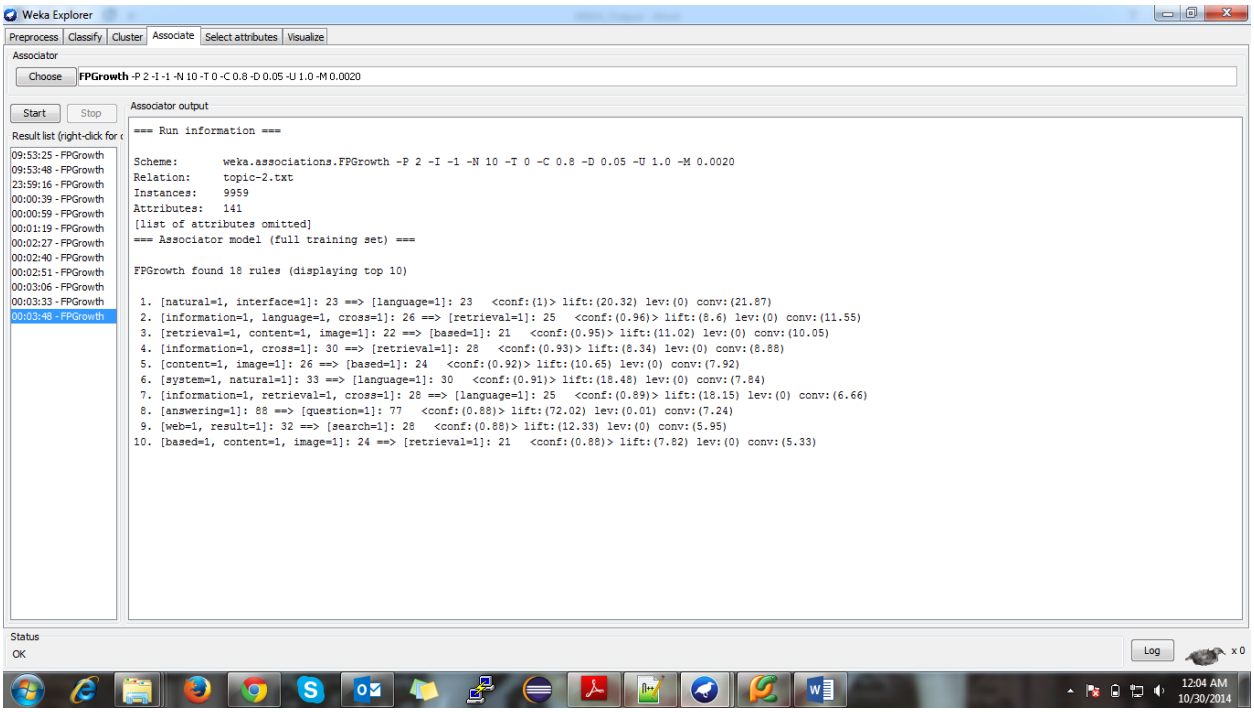
Parameter Settings →

lowerBoundMinSupport → 0.002

minMetric → 0.8

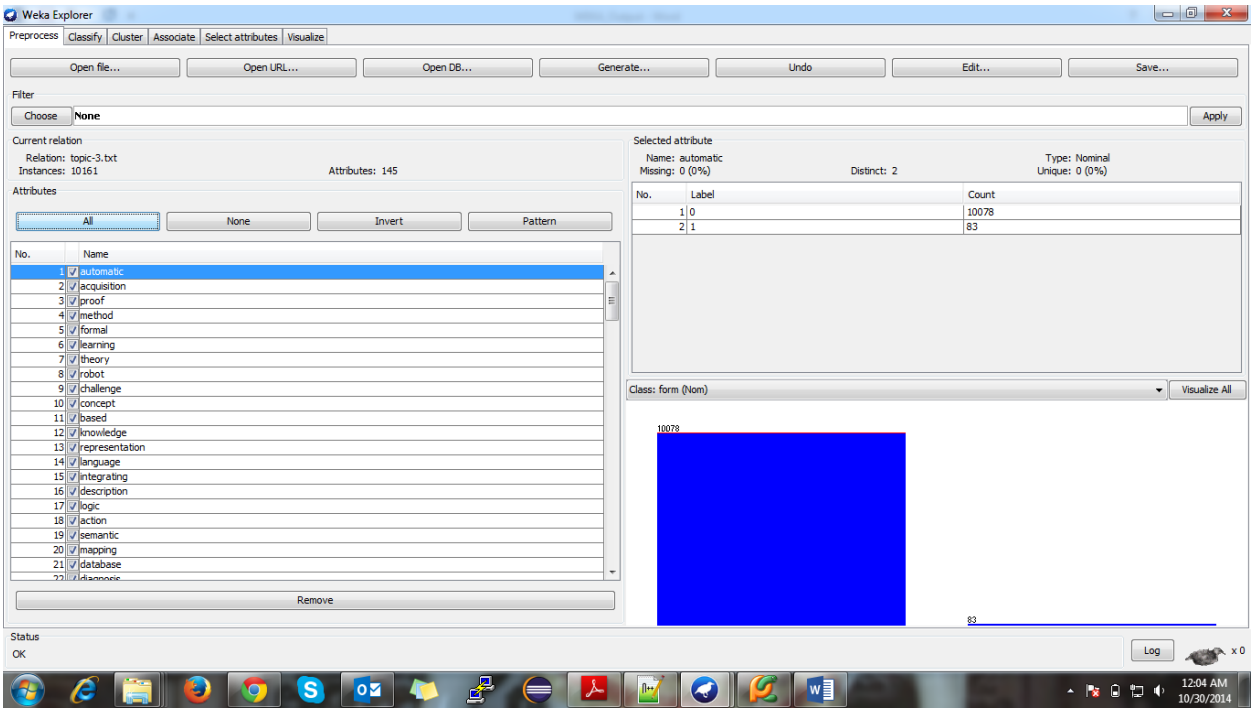


Association Rules →



Filename: - topic-3.arff

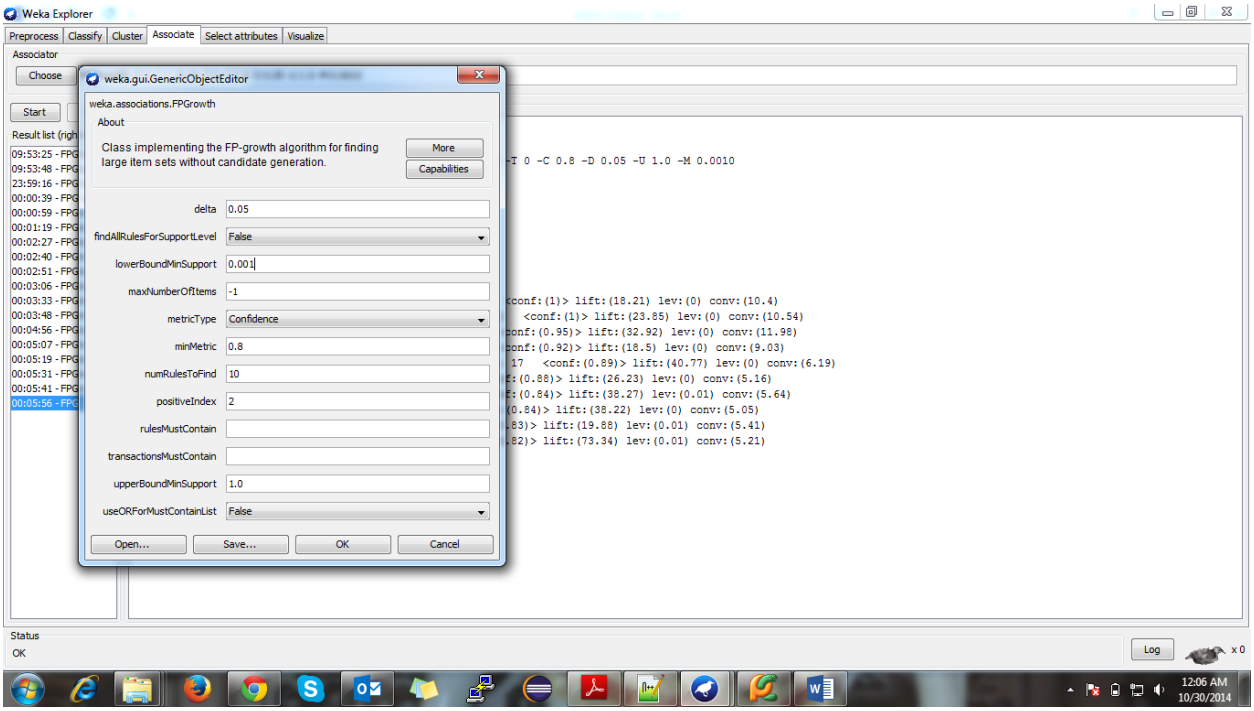
Preprocess →



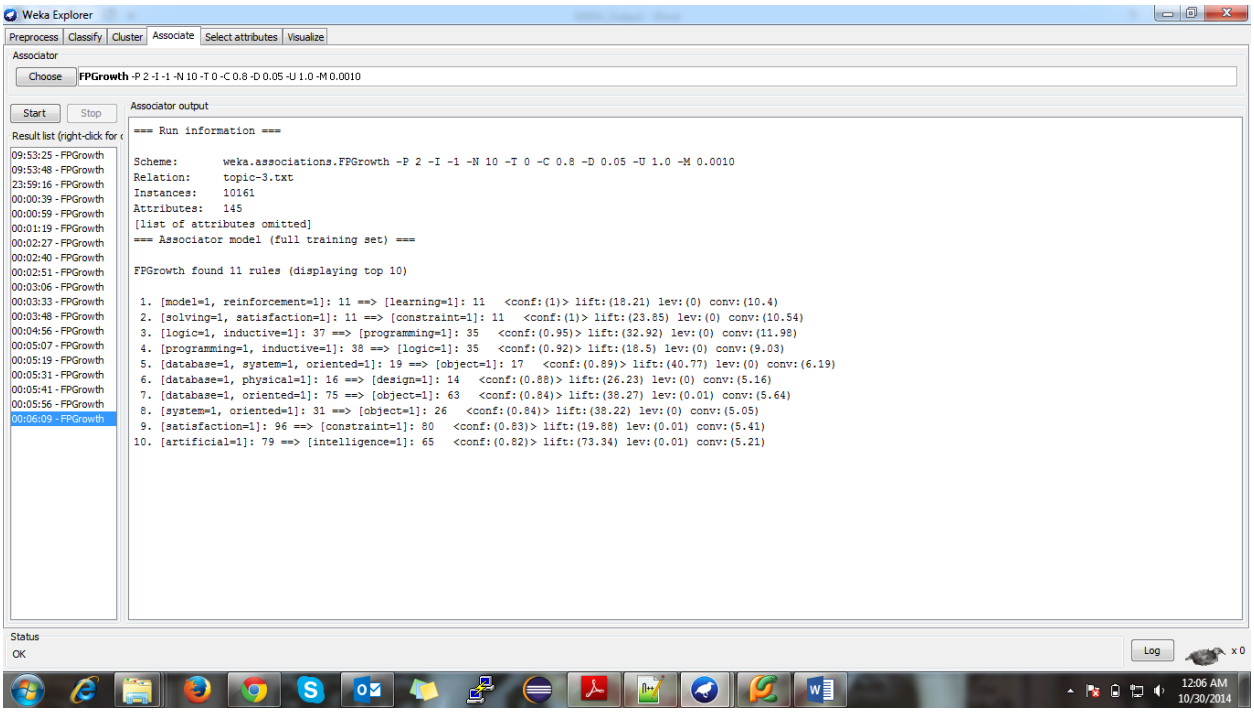
Parameter Settings →

lowerBoundMinSupport → 0.001

minMetric → 0.8

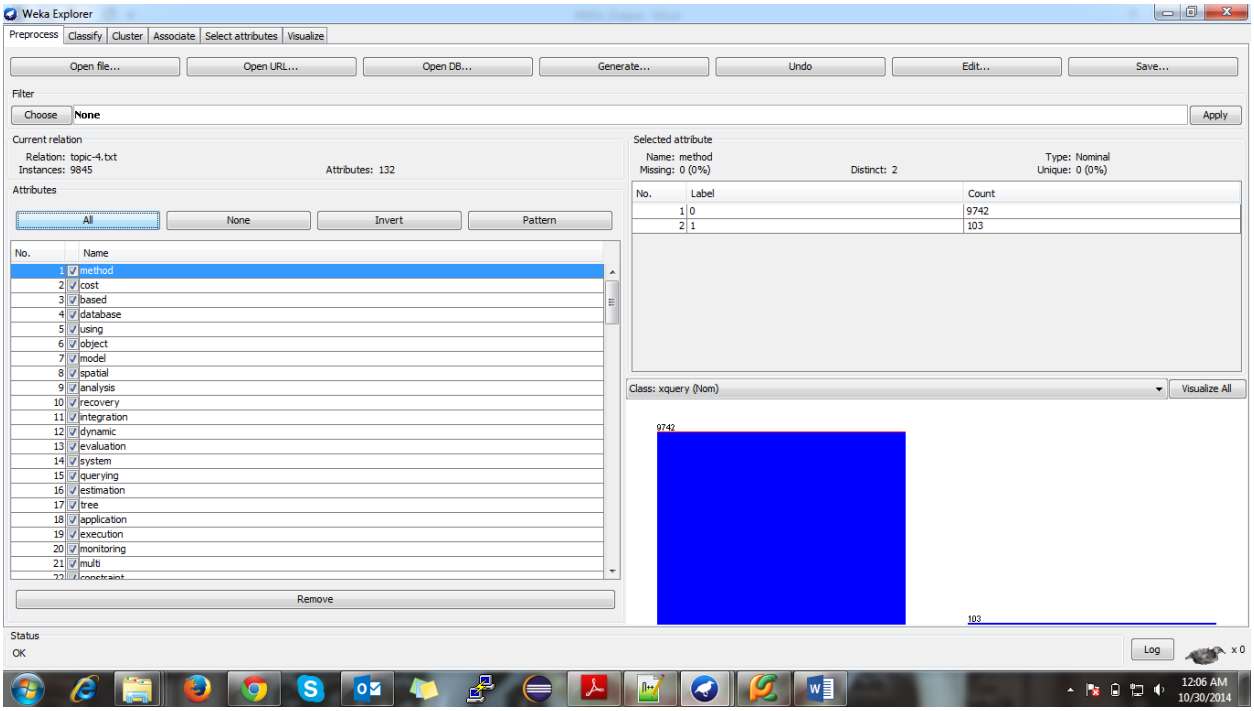


Association Rules →



Filename: - topic-4.arff

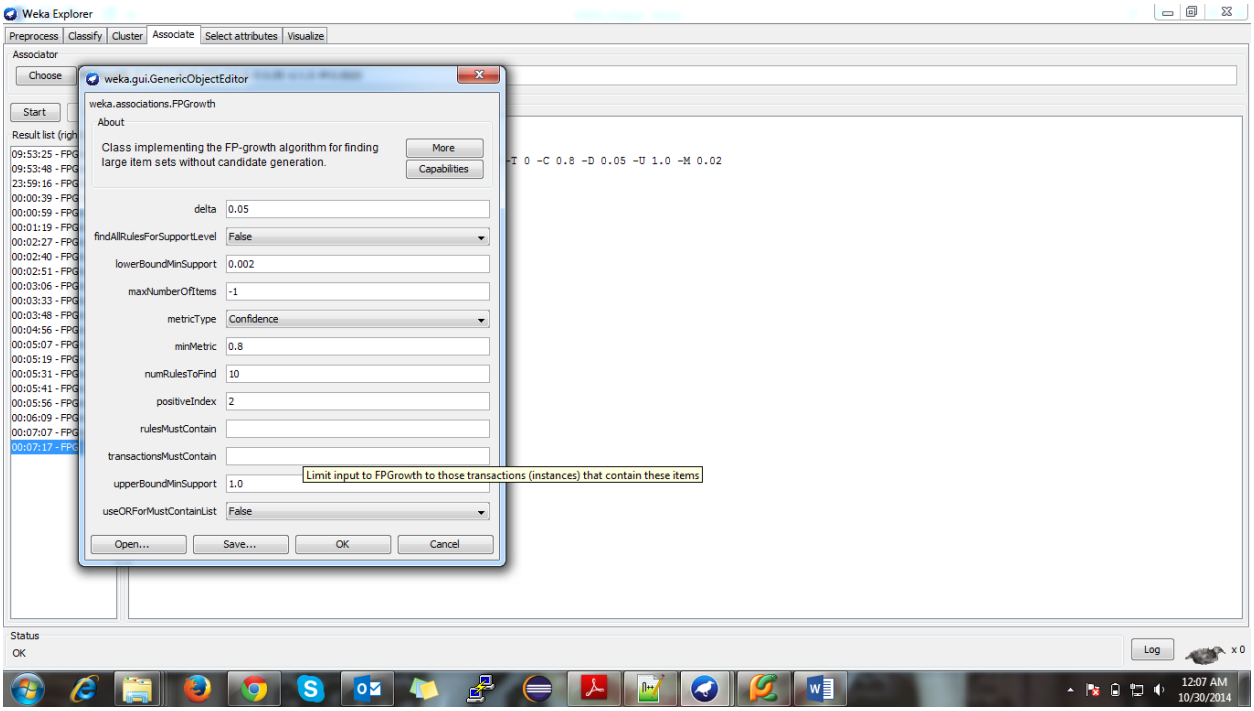
Preprocess →



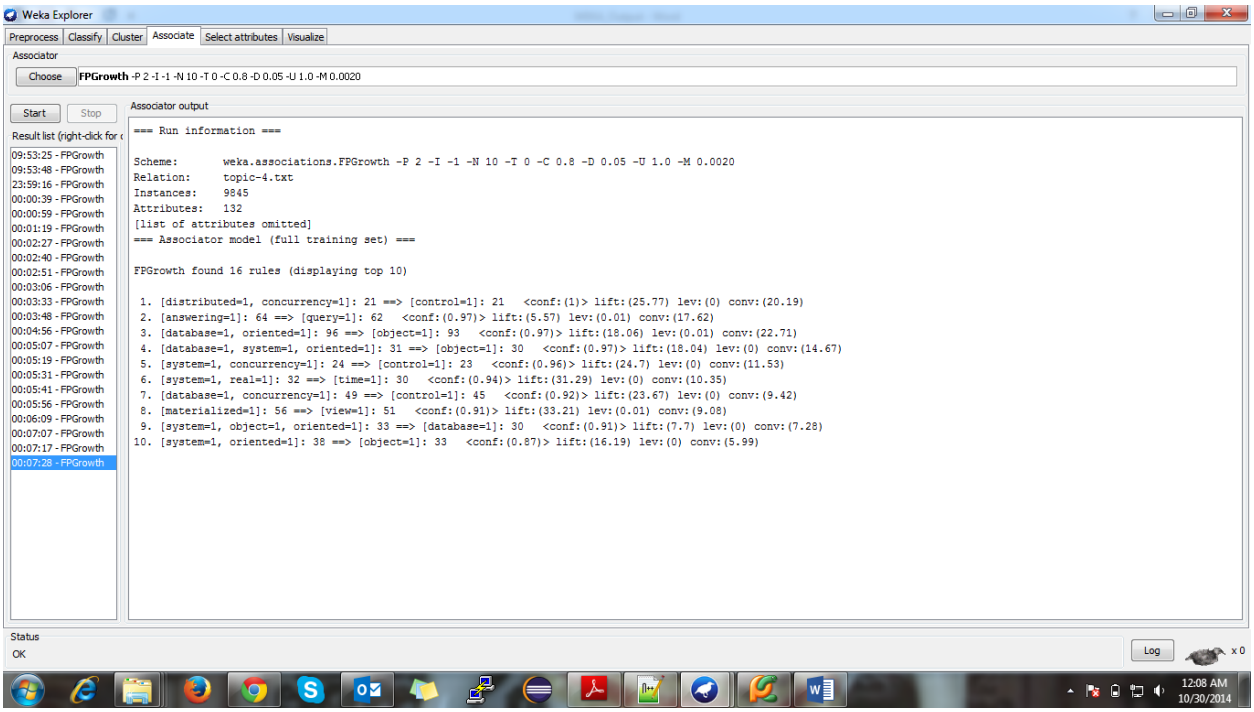
Parameter Settings →

lowerBoundMinSupport → 0.002

minMetric → 0.8



Association Rules →



Step 4: Re-rank by Purity of Patterns

1. We have use Purity as a ranking measures for frequent patterns. It is used to determine the distinctness of frequent pattern in a topic-I file.
2. A phrase is pure in topic t if it is only frequent in titles i.e. topic t and not frequent in documents about other topics.
3. Think of it as sort of a basal noise that you remove from the supports of each pattern in all the topics and you get how much more frequent the pattern is in
4. We have use below formula to calculate the purity of each frequent patterns

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

   Term →
   f(t,p) → frequency of pattern p appearing in topic t.
   D(t) → 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 }
   D(t,t') → 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

5. Now after finding the Purity, We observe that pure patterns are rank higher than less pure patterns. Now to combine Purity and Support. I have used below formula

   purity'(p) = purity(p) * log10 ft(p)

   Analysis → There are many techniques to combine the Support and Purity.
   1. ***purity'(p) = purity(p) + ft(p)***

   But since the value of purity is generally small by an/several order of magnitude compared to the frequency or support count, directly summing up the purity value with support would lead to elimination of the notion of purity and the ranking would become a pure measure of frequency. Thus, items with higher support would rank higher in this case and purity would go for a toss. Hence this approach is inappropriate.

   2. ***purity'(p) = purity(p) * ft(p)***

   But if we directly multiply the purity and frequency then it will clearly distinguish between Strong pure items (items having positive values) and remaining items (items having negative values). But items which have low frequency and also occur frequently in other topics will have less positive values. So applying above mentioned measures will result into inaccuracy for the items which are less frequent in other topic and have high frequency.

   3. ***purity'(p) = purity(p) * log10 ft(p)***

   So in order to minimize/smoothen the effect I have used log factor. After using log I observed that effect is minimized and accuracy of ranking increased. After performing sort on this values

the strongly pure items with higher frequency/support are ranked higher compared to the ones having a lower frequency/support.

Final support → purity(p) * log10 ft(p)
Arrange by Final support in decreasing order.

Source Code →
1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***func_Purity()***

```
# Define a func_Purity() function which purify the frequent pattern
def func_Purity(allTopicFreqPattern,indx,dtList,outputPurityFileName,dictVocab):

    purityDict = {}
    d_t = dtList[indx][0]
    maxLogTerm = 0.0
    pattern2bsearch = []
    f_t_p = 0.0
    topicIndex = 0
    f_tdash_p = 0.0
    d_t_tdashdict = []
    d_t_tdash = 1.0
    tempMaxLogTerm = 0.0
    purity = 0.0
    tempTopicPurityList = []
    sortbysupport = []
    finalpuritylist = []
    puritylist = []
    isAppend = False
    purtiy_support_combine = 0.0

    for items in allTopicFreqPattern:
        maxLogTerm = 0.0
        firstfactor = 0.0
        purity = 0.0
        if (items[0] == indx):
            pattern2bsearch = items[2]
            f_t_p = items[1]
            for allItems in allTopicFreqPattern:
                if (allItems[0] != indx):
                    topicIndex = allItems[0]
                    if ((len(frozenset(pattern2bsearch).intersection(frozenset(allItems[2])))) == len(pattern2bsearch)):
                        f_tdash_p = allItems[1]
                    else:
                        f_tdash_p = 0.0
```

Step5: Bonus
1. I have implemented Phraseness and Completeness as filtering/ranking measures.

Phraseness
1. Phraseness is defined as group of words 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.
    a. Example: ‘active learning’ is a better phrase than ‘learning classification’ in the Machine Learning topic.
2. I have used probability module to calculate Phraseness. Below is the formula

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

3. Idea is to calculate the probability of pattern ‘p’ and subtract with the summation of probability of all words belongs to that pattern

Observation: -

1. Consider the phrase **['series', 'time']** and **['rule', 'mining', 'association']**. From all the outputs, I clearly figured out that **['series', 'time']** is not a meaningful phrase and the words '**series**' and '**time**' co-occur only because two words are independently popular in the topic Data Mining and they together do not form a meaningful phrase.
2. However, **['rule', 'mining', 'association']** the phrase co-occurs more frequently than the expected co-occurrence frequency given that each word in the phrase occurs independently i.e. '**association**' is not so popular when compared to others but it is very important in the above phrase. It is required to make the phrase meaningful.
3. Therefore, by implementing this ranking measure, the quality of mined phrase list is clearly improved as seen in the above output. Also, 1-frequent itemsets which are ranked 0 since we are looking for meaningful phrases helps us to focus more on the exact phrase lists from frequent patterns. I personally think that this is a good result in text mining.

Source Code →
1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***func_Phraseness()***

Completeness

1. Completeness is defined as phrase which has no superset i.e. A phrase is not complete if it is a subset of a longer phrase, in the sense that it rarely occurs in a title without the presence of the longer phrase.
    a. Example: ‘support vector machines’ is a complete phrase, whereas ‘vector machines’ is not because ‘vector machines’ is almost always accompanied by ‘support’.
2. I have used superset subset relationship to calculate the completeness.

Observation: -

1. Completeness helps to rank high for the frequent patterns which are superset of some other patterns. i.e ['rule', 'mining', 'association'] is rank higher than ['mining'], ['rule'], ['association'], ['rule', 'association'], ['mining', 'association'] and ['rule', 'mining'] in topic-0 frequent pattern

Source Code →
1. Python File → ***rudani2_apriori.py***
2. Method Name → a. ***func_Completeness()***

```
# Define a func_Completeness() function which finds the completeness of the pattern.
def func_Completeness(listItemwithFreq,dt,indx,outputCompletenessFileName,dictVocab):...

# Define a func_Phraseness() function which re-rank the frequent pattern based on phraseness.
def func_Phraseness(allTopicFreqPattern,indx,dtList,outputPhrasenessFileName,dictVocab):

    d_t = dtList[indx][0]
    f_t_w = 0.0
    firstfactor = 0.0
    secondfactor = 0.0
    sumsecondfactor = 0.0
    result = 0.0
    f_t_p = 0.0
    f_t_w = 0.0
    phraseness = []
    tempphraseness = []
    sortbysupport = []
    phrasenessDict = {}
    for items in allTopicFreqPattern:
        firstfactor = 0.0
        if (items[0] == indx):
            pattern2bsearch = items[2]
            f_t_p = items[1]
            for word in pattern2bsearch:
                word2bsearch = [word]
                for trans in allTopicFreqPattern:
                    if (trans[0] == indx):
                        if ((len(frozenset(trans[2]).difference(frozenset(word2bsearch)))) == 0):
                            f_t_w = trans[1]
                secondfactor = math.log((float(f_t_w)/float(d_t)))
                sumsecondfactor = float(float(sumsecondfactor) + float(secondfactor))
            firstfactor = math.log((float(f_t_p)/float(d_t)))
            result = round(float(firstfactor) - float(sumsecondfactor),4)
```

Jigar Rudani (rudani2)
CS 412 Data Mining

Answers to Questions

***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.*

Ans A)

1. I chose the min_support to be 0.01
2. We need to choose min_support in such a way that we balance not thinning down too much (very low support generating too many frequent patterns) as well as not increasing so much so that we can't distinguish between frequent and closed/maximal patterns
3. I tried various support values (0.01,0.002,0.5,0.005 and 0.001) and 0.01 seemed to work best for the given data set
4. 0.01 would mean for a transaction list of 10047 transactions, a pattern is frequent if it appears at least 100 times or more which seems to be a good estimate

***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.*

Ans B)

1. Based on the analysis of *.txtphrase files in frequent pattern folder. I was able to decipher the following
2. These observations are based on the examining the mined patterns closely across all the topics. Let us consider pattern-0, a number of patterns like 'rule association mining', 'dimensionality reduction', 'decision tree', 'sequential pattern' etc. were found. Based on my knowledge, I can safely conclude that these patterns belong to the domain of 'Data Mining' (DM). Similarly other frequent pattern files for files belonging to every domain were observed. In our observation we found it challenging to distinguish between patterns of topic 3 and 4 as there were overlapping phrases from two domains occurring in both and it was hard to narrow down them to one of the topics. On observing all the mined patterns, I found patterns like 'buffer', 'client', 'shared', etc. and thus find it appropriate to bucket these patterns to domain 'Theory' (TH).

    a. Phrase 0 → Associated to Data Mining (DM) Domain
    b. Phrase 1 → Associated to Machine Learning(ML) Domain
    c. Phrase 2 → Associated to Information Retrieval(IR) Domain
    d. Phrase 3 → Associated to Database (DB) Domain
    e. Phrase 4 → Associated to Theory (TH) Domain

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

Ans C)
I have compared the results of Frequent Patterns, Maximal Patterns and Closed Patterns. I have found below observations

1. The results are quite satisfying. The Frequent Patterns in pattern-i.txt files clearly shows all the words that were frequent in the paper titles.
2. Around 75 patterns per topic for min_support 0.01 and approximately 75-85 Closed Pattern and 56-65 Maximal Pattern are found. So there are no proper supersets with the same count as that of the pattern in every pattern-i file. However, when I tried support as 0.002, consider pattern-0 and closed-0. I found that there are 580 frequent patterns and 576 closed patterns. So every frequent pattern is not closed pattern.
3. I also found that all of my Maximal patterns are also Closed.

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.*

Ans D)
I have compared the results of Frequent Patterns and Association of Phrases that satisfies both min_sup and min_conf. I have found below observations

1. Consider the phrase **['rule', 'association']** obtained as a strong association rule from Weka. The association rule is as follows:

   [rule=1]: 416 ==> [association=1]: 233 <conf:(0.56)> lift:(16.75) lev:(0.02) conv:(2.19)

2. I can clearly see that both patterns satisfy the support of 0.01 i.e. support count of 100 and and minimum confidence of 30%. It is meaningful phrase because both terms together represent a technique in the domain data mining.

3. When I compare this results to that obtained in step1, I found that in step 1, '**rule**' independently occurs 416 times and '**association**' independently occurs 336 times. When I looked for the pattern **['rule', 'association']**, it was interesting to see that both terms '**rule**' and '**association**' together occurs 233 times as one pattern.
4. So as per confidence rule, it says

   Conf (A==>B) → support (A U B)/support (A)
   Conf (rule ==> association) → support (rule U association)/support (rule)

5. So if we calculate confidence we get (233/416 → 0.56).
6. It means that '**rule**' occurs 416 times as an item in the transaction and every time it occurs about 56% of same time both '**rule**' and '**association**' occurs together i.e. 56% of 416 → ~233. Therefore, the above mentioned rule is a strong association rule.