Algorithmic steps

- Construct the hash of the words of the paper titles
- Construct topic-I files which have the above words segregated by the the topic allocated by the LDA package in word_assignments.dat
- **Step 2:** We generate a dictionary of words versus integers. Integer value for words is merely the corresponding line number in the vocab.txt
- **Step 3:** After assigning topics via LDA ,we reorganize words according to their topics into the topic-i.txt files
- **Step 4: Apriori pattern mining**
    - We generate k-Item candidate sets with k starting from 1 and increasing towards higher tuple values and then
    - As we generate these K-Item sets we also evaluate their count and generate their support based on # of items/total count
    - If this metric satisfies our min support criteria then we consider its frequent pattern
    - The min support chosen is 0.005 choice of which I would explain later in this document
    - We can use pruning to drastically reduce our search space and optimize our algorithm
    - Which Python file? raguram2-apriori.py
    - Which Function ? aprioriFrequentPatternMining
- **Step 5: Maximal Pattern Mining**
    - Evaluate a frequent pattern against subsets of all other frequent patterns
    - If a frequent pattern is found as a subset, It is not maximal
    - Which Python file? raguram2-apriori.py
    - Which Function? findMaximalPatterns
- **Step 6: Closed Pattern Mining**
    - One step less stringent that Max Patterns. Subsets are allowed but they shouldn't have the same support.
    - All Maximal patterns are expected to be closed as well. We hope to get some closed non maximal patterns
    - Which Python file? raguram2-apriori.py
    - Which Function? findClosedPatterns
- **Step 7: Purity**
    - Determines the distinctness of frequent pattern in a topic-I fie.
    - To determine this we calculate a purity metric which takes into account how prevalent the frequent pattern is in other topic files
    - 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
    - The metric we used to combine support and purity is
        - Final support = purity*(1+support)
    - Note: Purity is a term that is un damped because of the negative Logarithm(small float values) while support is not hence mixing them is quite difficult
    - Also purity in itself encompasses support, that was another area of concern for me regarding what kind of mixing should I do so that I don't weigh support twice (duplicate counting)

(1) *How you choose min sup for this task? Explain how you choose the min_sup in your report. Any reasonable choice will be fine.*

- I chose the min_support to be 0.005
- 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 cant distinguish between frequent and closed/maximal patterns
- I tried various support values (0.5, 0.1 , 0.05, 0.02, 0.01 , 0.005 ,0.002 & 0.001) and 0.005 seemed to work best for the given data set
- 0.005 would mean for a transaction list of 10000 transactions, a pattern is frequent if it appears at least 50 times or more which seems to be a good estimate

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

- Based on the analysis of the *. phrase files in frequent patterns I was able to decipher the following
  - pattern-0 seems to be associated with Data Mining
  - pattern -1 seems to be associated with Information retrieval
  - pattern -2 seems to be associated with Database Systems
  - pattern -3 seems to be a bit difficult to classify and I would put my best bet on Machine Learning and Data Clustering
- In general I was able to decipher some amount of topic-based coherency.

(3) *Compare the result of frequent patterns, maximal patterns and closed patterns, is the result satisfying? Write down your analysis.*

- The results are quite satisfying and patterns-i.txt.phrase files would tell you what words were frequent in the paper titles. Around 190-200 patterns per topic for support 0.005
- One issue which I faced was all of my closed patterns were also maximal that is I was unable to generate any non-maximal closed pattern. One particular reason might be that the support values which we compare are numerically small float values hence checking for equality in small values might have yield incorrect numerical results
- Hence only those terms were retained that never had a superset i.e. were maximal whilst all (subset->superset) relation frequent patterns were knocked off

Bonus Question

- While analyzing the frequent patterns (lets say pattern-0.txt) we see words like
    - Using
    - Based
    - Query

  That are ranked really high but really do not serve to give any meaning or act a key-determining factor in identifying the topic.

- The reason possibly being several words in the parts of speech are necessary to establish syntactic sense but add very little to identifying what the sentence is about
- The purity measure we built tries partially to remove this effect but there can be a case that parts of speech that were used in one topic were never used in another which nullifies the effect of pattern mining based on purity
- Instead along with topics we call also identify the word based on POS Tagging techniques and give them weights
- For example

  "Locating passage using based excerpt count"

    - Here both using and count could be words that might be frequent but add very little to the identification of what this is about
    - Passage and excerpt add more to the knowledge
    - Locating passage and excerpt together add even more knowledge
    - I would try to give ranks to words as well

      So "locating[0][1] passage[0][1] using[0][4] based[1][4] excerpt[0][1] count[1][4]"

      using[0][4] –meaning using could be frequent but its ranked less so "using " gets damped

Hence we could add a penalty based on the rank to the final support value

So we modify the apriori step where in before storing support, we penalize based on POS Tagging and hence we dampen the support

- **List your source file names and their corresponding Steps**

(4) dictGenerator.py

- Generates your dictionary of (words, IDs) based on the vocab.txt file
- It also encodes/tokenizes the paper.txt into numeric data reverse mapped to words in the paper title by the vocab.txt
- Usage: **python dictGenerator.py**

(5) lda package

- Topic Modeling Latent Dirichlet Allocation package which allocates topics to each word of the paper titles. Output of which gets organized into word_assignments.dat

(6) reorganizeTopics.py(works upon word_assignments.dat and organizes transactions according to the

- reorganizes word_assignments.dat into the five topic-i.txt files stripping words relevant to each topic from each line of the title.txt and adding them in topic-i.txt
- Usage: **python reorganizeTopics.py**

(7) raguram2-apriori.py (finds frequent patterns, closed patterns, maximal patterns and purity values)

- Runs Apriori and generates frequent patterns. Also generates closed, maximal and purity patterns and places them in their respective folders(note the folders need to be there prior .Will try appending a shell script that does that)
- Usage : **python raguram2-apriori.py 0.005**

(8) vocabReverseMapper.py (reverse maps the integer values to their word identities as per vocab.txt)

- Maps created patterns in all of the above categories back to their word form as dictated by the vocab.txt
- Usage: **python vocabReverseMapper.py**