**Main steps in the project.**

**Step 0: Preparation of input file for sentence splitter script:**

* + Python module ***MakeFileName.py***
  + Input: Takes Directory Path of the folder that stores the data files (texts of the medical records) and input.
  + Output: Gets the names of all the files in the directory and writes the full path of the files to an output text file : *filename.txt*

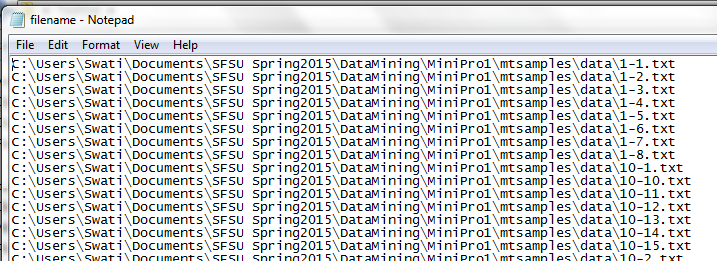


Figure 1: Snippet of the filename.txt

**Step 1: Selecting the Candidate sentences from the Medical Record files:**

* Python module ***MakeSentencesWithId.py*** [[1]](#footnote-1)
* Method: extractSent(filepaths) :
  + - Takes the filename.txt created in above step as input.
    - It opens each file -> reads the text of the files -> tokenizes the sentences using sentence detector from nltk .
    - The sentence is selected if it contained family members.
    - The sentences are then processed to produce 2 different files:

File 1: *sentences\_allwords.txt* -Contains the sentences (stop words included). Non-alpha numeric characters such as “, (, {, [, ‘, - etc. are stripped off.

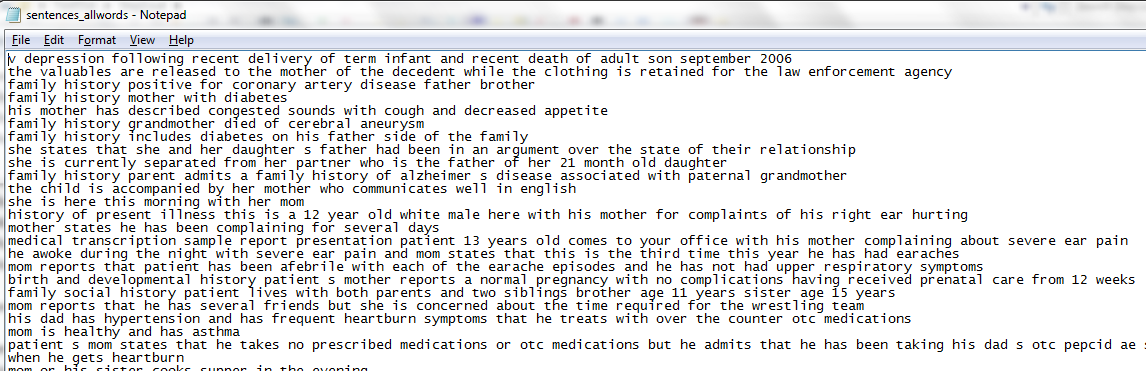


Figure 2: sentences\_allwords.txt

File 2: sentences\_nostopwords.txt

* This file contains sentences after removal of stop words.
* Stop words removed are based on nltk stop words corpus, which contains 127 stop words.
* Single letter words and pure numbers are also removed.
* Every sentences has the report id in the beginning of the sentence, this id number will be used to count the support in Apriori algorithm.

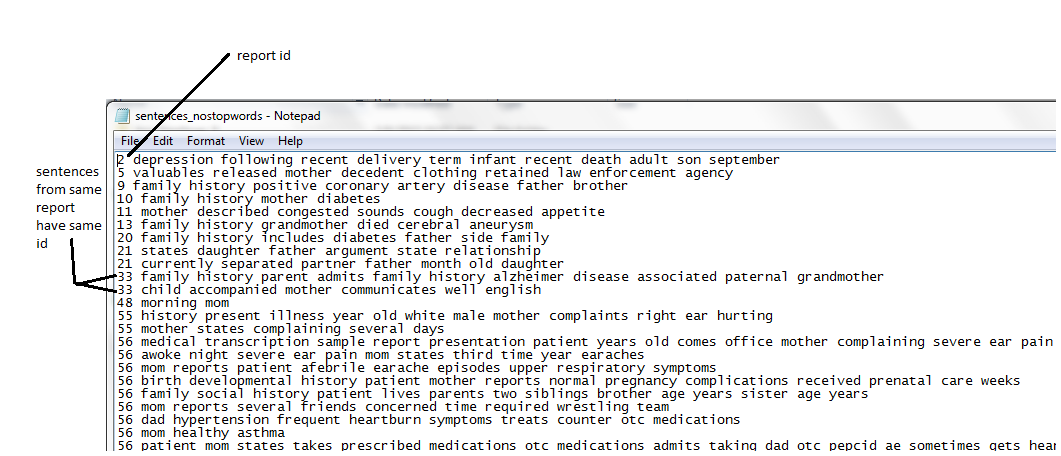
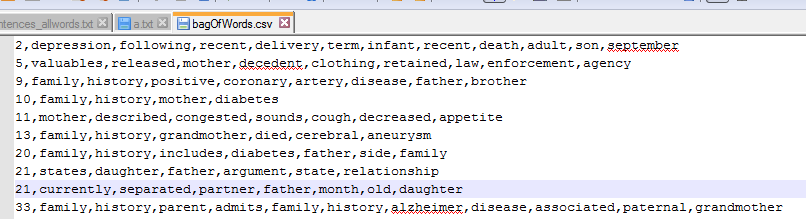


Figure 3: sentencs\_nostopwords.txt: file after removing stop words and adding report ids.

**Step 2: Making a bag of words**

* Python module ***MakeSentencesWithId.py***
* Method: makeBagOfWords(filename):
* Input : filepath of the text file with sentences without the stop words created in the step above i.e. sentencs\_nostopwords.txt
* Creates the csv file *bagOfWords.csv*, splitting sentences into bag of words.
* The first item in each transaction is a report-id



**Step 3: Finding word associations with minimum support – 5**

* Python module ***Apriori.py*** [[2]](#footnote-2)
* ***Algorithm keeps the track of Record-id while counting support and increments the support count only once for each report. So if the report contains multiple sentences having a particular word association, support is counted only once.\****
* Output: gives word associations and support count. (*output.csv*)

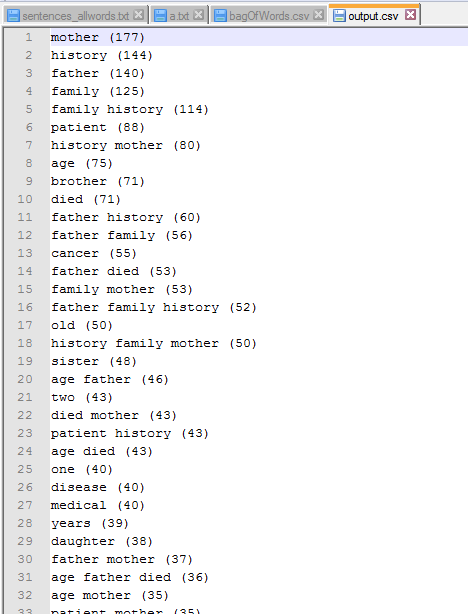


Figure 4: Snippet of output.csv

**Step 4: Finding word associations with that are found with span of k words (3, 5, 10)**

* Python module ***CheckSpan.py***
* Method: checkSpan(sentenceFile, mysetfile, outpath, k )
* Input :
  + - sentencesFile: the file that contains all candidate sentences created in step 1.
    - Mysetfile : Association set with support file created in step 3.
    - Outpath: directory where output file will be stores.
    - K : span within which words should be found in sentence to be considered.

**Concept of the algorithm:**

1. Algorithm picks word association which have more than one word and less than or equal to k words, and loops through all sentences (complete sentences with stop words).
2. If the word association is present in the sentences, indices (positions) of the words in the sentences are obtained.
3. The index of last word is subtracted from first word’s index and if the result is <= k the association is counted.
4. If >=40% of sentences had association within the span, they were included in consideration. (Thus words that were not within the span in > 60% sentences were discarded)

* Output: The algorithm is called with k-values 3, 5 and 10, output files are created for each with included word associations and their supports.

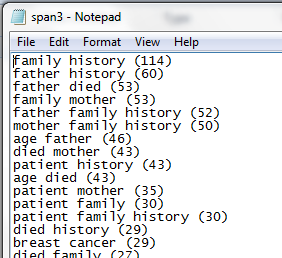


Figure 5: span3.txt snippet of word associations satisfying the k-span

**Step 5: Finding the Frequent Order of word associations :**

* Python module ***OrderCheck.py***
* Method: checkOrder(sentenceFile, mysetfile, mydir, k ):
* Input :
  + - sentencesFile: the file that contains all candidate sentences created in step 1.
    - Mysetfile : Association set with support file created in step 3.
    - mydir: directory where output file will be stores.
    - K : span within which words should be found in sentence to be considered. This is used just to name the file .

**Concept of the algorithm:**

1. From each word association , its permutations are obtained: eg (mother, diabetes) and (diabetes, mother)
2. Each ordered list is then checked in sentences, if the words are found in that order.
3. Order check - if ordered word association was found in sentences words: index of first word was found, the list of all-words (sentence words) was then reduced to include only the words subsequent to the found word. The next word was checked in the shortened list and its index found to again shorten the all-words. If all words could be found in subsequently shortened list, in that order, the ordered word association’s frequency was increased by 1.
4. The count of all ordered word association was stored and written to a file.

* Output : Two files are written for each input set 1) that contains all ordered sets created and their frequencies 2) that contains reverse sorted list of ordered sets (0 frequency sets are discarded)

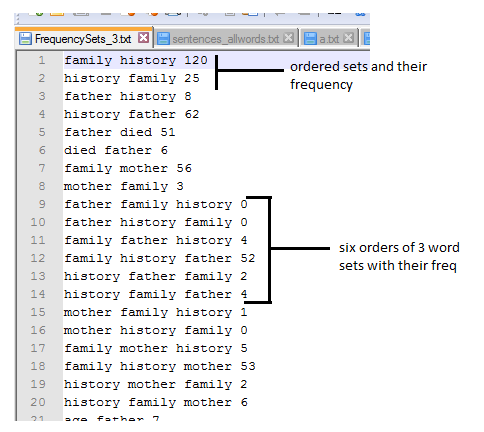


Figure 6: Ordered word sets and their frequencies

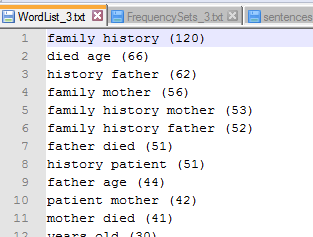


Figure 7: The Wordlist (ordered sets arranged in decreasing order of frequency)

**Step 5: Characterization of top N List: answers to questions**

* Python module ***TopWordLists.py***
* Method: getTopLists(filename, k , mydir):
* Input :
  + - filename: the file that contains WordList ( this project check 3 files – 3, 5, and 10 spanned wordlists)
    - k : Used to create directory and filename
    - mydir: directory where result folders need to be stored
    - The algorithm takes DiseasesList.txt file which is included in the source package.
* Output :
  + - Ordered association sets with their frequencies are written to respective files based on the conditions they satisfy:
    - All files are stores in separate folders, based on input files.
    1. a.txt contain at least one family member?
    2. b.txt contain at least one of the following diseases?
    3. c.txt one family member but no disease?
    4. d.txt both a family member and a disease?
    5. e.txt neither a family nor a disease?
  + Individual family member wise file created, to study the family member wise association.

1. Modified from sentenceSplitter.py written by Neal Lewis to include stop word removal and generate multiple files. [↑](#footnote-ref-1)
2. Modifies from Everaldo Aguiar & Reid Johnson’s Apriori algorithm : [link](http://nbviewer.ipython.org/github/cse40647/cse40647/blob/sp.14/10%20-%20Apriori.ipynb) [↑](#footnote-ref-2)