Data Mining

LAB 1

Jai Jaswani

High Level Algorithm:

1. Read Each file into memory
2. Extract out one REUTER document.
3. From this document extract the TEXT contents.
4. Parse the text and store in a hash table (this hash table is flushed for every document)
5. Parse the body, if the word in the body also appears in the title add +5 to the weight of the word
6. Store each word in a hash-table (cumulative hash table for the entire data set)
7. Calculate TF IDF based on the body word hash table for the entire data set.
8. Use the top 2000 words in the feature vector list.

Explanation for some of the decisions:

1. Reading file into memory: Each file in the data set is around 1.3Mb in size, given that computing devices today have RAM in Gbs, it is efficient to read the entire file at once into the memory rather than chopping the file into blocks. This also increases the performance of regex operations that are done on the file.
2. Parsing of title and body tags involves getting rid of digits and several special characters. All characters were converted to lowercase for homogeneity. Also stemming was used to extract common roots of the word.
3. Storage of title words was flushed for every document to make sure weightage is only given to those words which belong to the title and body of the **same** document.
4. Storage for body words has to be cumulative to calculate TFIDF for each word. However, filtering while parsing ensures we only end up with around 30k words in the body. So the burden on memory for a hash table with just 30k words and a few other integers is not heavy.

5. Using the top 2000 words in the list was just to have reasonable size document when printing the sparse matrix feature vector.

6. TFIDF: There were two approaches discussed in class to select relevant words out of the data set parsed. One of the approaches was to build the histogram of all the words in the data set and throw away the highest and lowest frequency terms. This approach is totally heuristic based and relies to selecting a good set from the histogram.

   On the other hand TFIDF is another approach which takes into account not only the frequency of term in the document but all the number of documents in which the term appears, this approach is considered generally superior than the histogram approach.

   While working on the Lab I started with histogram approach, but moved to the TFIDF approach as it was recommended in class. In this method, I pick the top 2000 words that yield to me the highest TFIDF value. Based on these values I create my feature vector. However the results still indicate that a lot of documents do not possess words with these values. This suggests further tuning might be needed to get better matches in the documents. But at the moment we are just preprocessing the data, and I believe there will be a better definition of accuracy during future labs and code is easily tunable to achieve the same.

Results:

Sample Vector List:

```
 1 Docid =  21578  max_word_count =  53359
 2 BODY
 3 baltek [11, 1, 1.0, 9.979429556111285]
 4 augat [6, 1, 1.0, 9.979429556111285]
 5 proga [6, 1, 1.0, 9.979429556111285]
 6 corken [6, 1, 1.0, 9.979429556111285]
 7 omc [6, 1, 1.0, 9.979429556111285]
 8 beld [6, 1, 1.0, 9.979429556111285]
 9 vol [6, 1, 1.0, 9.979429556111285]
10 harlyn [6, 1, 1.0, 9.979429556111285]
11 pacificar [6, 1, 1.0, 9.979429556111285]
12 slough [6, 1, 1.0, 9.979429556111285]
13 micom [6, 1, 1.0, 9.979429556111285]
```

14 mrt [11, 1, 1.0, 9.979429556111285]

15 epsco [6, 1, 1.0, 9.979429556111285]

Adjacency List:

1 <DOCID: 1> {} <TOPIC> <D>cocoa</D>

2 <DOCID: 2> {} <TOPIC>

3 <DOCID: 3> {} <TOPIC>

4 <DOCID: 4> {} <TOPIC>

5 <DOCID: 5> {} <TOPIC> <D>grain</D><D>wheat</D><D>corn</D><D>barley</D><D>oat</D><D>sorghum</D>

6 <DOCID: 6> {} <TOPIC> <D>veg-oil</D><D>linseed</D><D>lin-oil</D><D>soy-oil</D><D>sun-oil</D><D>soybean</D><D>oilseed</D><D>corn</D><D>sunseed</D><D>grain</D><D>sorghum</D><D>wheat</D>

7 <DOCID: 7> {} <TOPIC>

8 <DOCID: 8> {} <TOPIC>

9 <DOCID: 9> {} <TOPIC> <D>earn</D>

10 <DOCID: 10> {} <TOPIC> <D>acq</D>

11 <DOCID: 11> {} <TOPIC> <D>earn</D>

12 <DOCID: 12> {} <TOPIC> <D>earn</D><D>acq</D>

13 <DOCID: 13> {} <TOPIC> <D>earn</D>

14 <DOCID: 14> {} <TOPIC> <D>earn</D>

15 <DOCID: 15> {} <TOPIC>

16 <DOCID: 16> {} <TOPIC>

17 <DOCID: 17> {} <TOPIC>

18 <DOCID: 18> {'larsen': 1} <TOPIC> <D>earn</D>

Sparce Vector:

1 <DOCID: XXXXXX> aar abbott abn abram aca accugraph accuray aceto acmex activis acton acustar addison adia admar advest advo ael aep aerospatial aetna afghan afp aground ahe aidc airba

2 <DOCID: 1> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

3 <DOCID: 2> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

4 <DOCID: 3> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] <TOPIC>