Report

Assignment 1

CS 565(Intelligent System and Interfaces)

Abhinav Ravi (120101003)

Pintu Kumar (120101050)

PART 1

Tool 1:

(Abhinav Ravi)

Apache Opennlp and Stanford NLP.

1. Corpora source: nltk_data/abc/rural.txt(Web source)

2. Dictionary is present in “\Supplementary \Dictionary” folder.

3. The unigrams in monotonically decreasing order of frequency is present with respective names as uni.csv files in “Supplementary\Tool1_ApacheOpennlp\Lemmatized_Results” along with lemmatized corpus and in “Supplementary\Tool1_ApacheOpennlp\unlemmatized_Results” along with unlemmatized corpus.

4. The bigrams in monotonically decreasing order of frequency is present with respective names as bi.csv files in “Supplementary\Tool1_ApacheOpennlp\Lemmatized_Results” along with lemmatized corpus and in “Supplementary\Tool1_ApacheOpennlp\unlemmatized_Results” along with unlemmatized corpus.

5. The trigrams in monotonically decreasing order of frequency is present with respective names as tri.csv files in “Supplementary\Tool1_ApacheOpennlp\Lemmatized_Results” along with lemmatized corpus and in “Supplementary\Tool1_ApacheOpennlp\unlemmatized_Results” along with unlemmatized corpus.

6. Explored tool : Apache Opennlp and Stanford NLP.

7. I have used Apache Opennlp for

1. Sentence segmentation and
2. Word Tokenization

Apache opennlp uses ***Maximum Entropy principle(MAxEnt)***. MaxEnt uses multinimial logistic regression to find best model fitting our data. It comes under **Supervised Machine Learning**.

In formal words MaxEnt algo can be stated as : Given a collection of facts, choose a model which is consistent with all the facts, being as uniform as possible. The same principle is used by the opennlp algo. From all the models that fit our training data, select the one which has largest entropy.

To create a model, one needs (of course) the training data, and then implementations of two interfaces in the opennlp.maxent package (all available as jar).

A)**. Sentence segmentation:**

Training Dataset: en-sent.bin (available from website)

Custom training dataset can also be prepared using training API.

For custom training : train file should contain data as shown below:

```
<START> Pierre Vinken, 61 years old, will join the board as a nonexecutive
director Nov. 29.<End>
<START> Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
 <End>…..
```

And so on.

B)**. Word Tokenization**

Training Dataset: en-token.bin (available from website)

Custom training dataset can also be prepared using training API.

OpenNLP offers multiple tokenizer implementations:

- Whitespace Tokenizer - A whitespace tokenizer, non whitespace sequences are identified as tokens.
- Simple Tokenizer - A character class tokenizer, sequences of the same character class are tokens.
- Learnable Tokenizer - A maximum entropy tokenizer, detects token boundaries based on probability model, I have used this tool only due to high range of accuracy obtained.

With OpenNLP, tokenization is a two-stage process: first, sentence boundaries are identified, then tokens within each sentence are identified.

C). **Lemmatization**

Since no tools are provided in apache opennlp for lemmatization and stemming, hence I have used **Stanford NLP Lemmatization tool**.

It also includes **Penn Tree Bank**. It'll split the joint words like: I've => I have.

It generates the word lemmas for all tokens in the corpus.

It uses a pipeline like: `props.put("annotators", "tokenize, ssplit, pos, lemma");`

Pipeline takes in a string and returns various analyzed linguistic forms. The String is tokenized via a tokenizer (such as PTBTokenizerAnnotator), and then other sequence model style annotation can be used to add things like lemmas, POS tags, and named entities. These are returned as a list of CoreLabels. Other analysis components build and store parse trees, dependency graphs, etc.
This class is designed to apply multiple Annotators to an Annotation. The idea is that we first build up the pipeline by adding Annotators, and then we take the objects you wish to annotate and pass them in and get in return a fully annotated object.

Tool 2:

Pintu Kumar (120101050)

Natural Language Toolkit (NLTK)

1.Corpus source: nltk_data/abc/rural.txt

2. Dictionary is present in "\Supplementary \Dictionary" folder.

3. The unigrams in monotonically decreasing order of frequency is present with respective names as unigram.txt files in "Supplementary\Tool2_NLTK\Lemmatized_Results" along with lemmatized corpus and in "Supplementary\Tool2_NLTK\unlemmatized_Results" along with unlemmatized corpus.

4. The bigrams in monotonically decreasing order of frequency is present with respective names as bigram.txt files in "Supplementary\Tool2_NLTK\Lemmatized_Results" along with lemmatized corpus and in "Supplementary\Tool2_NLTK\unlemmatized_Results" along with unlemmatized corpus.

5. The trigrams in monotonically decreasing order of frequency is present with respective names as trigram.txt files in "Supplementary\Tool2_NLTK\Lemmatized_Results" along with lemmatized corpus and in "Supplementary\Tool2_NLTK\unlemmatized_Results" along with unlemmatized corpus.

6. Explored tool : NLTK.

7. Working principle for NLTK :

1. The Natural Language Toolkit (NLTK) is an open source Python library for Natural Language Processing.
2. Sentence segmenter : sent_tokenize from nltk
   : sent_tokenize is one of instances of PunktSentenceTokenizer (machine learning based) from the nltk.tokenizer.punkt module.This tokenizer divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used.
   The NLTK data package includes a pre-trained Punkt tokenizer for English.

3. Word Tokenizers: word_tokenize is a wrapper function that calls tokenize by the TreebankWordTokenizer . Except the TreebankWordTokenizer, there are other alternative word tokenizers, such as PunktWordTokenizer and WordPunktTokenizer.
   The Treebank tokenizer (rule based) uses regular expressions to tokenize text as in Penn Treebank. This tokenizer performs the following steps:
   - Split standard contractions, e. g. ``don't`` -> ``do n't`` and ``they'll`` -> ``they 'll``
   - Treat most punctuation characters as separate tokens
   - Split off commas and single quotes, when followed by whitespace
   - Separate periods that appear at the end of line

   PunktTokenizer splits on punctuation, but keeps it with the word. Punkt knows that the periods in Mr. Smith and Johann S. Bach do not mark sentence boundaries. And sometimes sentences can start with non-capitalized words. E.g.   i is a good variable name. WordPunktTokenizer splits all punctuations into separate tokens.

PART 2.

(Abhinav Ravi)

Apache Opennlp and Stanford NLP.

Un-lemmatized version: Total Token Count: 337569

1. How many (most frequent) words are required for 90% coverage of the selected corpus?

Ans: 11600

2. How many (most frequent) bigrams are required for 80% coverage of the corpus?

Ans: 124265

3. How many (most frequent) trigrams are required for 70% coverage of the corpus?

Ans: 214327

4. Repeat the above after performing lemmatization.

Ans:

   Lemmatized Corpora + Post-Processing (Removing inverted commas):

   Token Count: 333084

   4.1. How many (most frequent) words are required for 90% coverage of the selected corpus?

   Ans: 11005

   4.2. How many (most frequent) bigrams are required for 80% coverage of the corpus?

   Ans: 66420

4.3. How many (most frequent) trigrams are required for 70% coverage of the corpus?

Ans: 221257

5. Compare the statistics of the two cases.

Apache Opennlp	90% of uni	80% of bi	70% of tri
Lemmatized	11005	66420	221257
UnLemmatized	11600	124265	214327

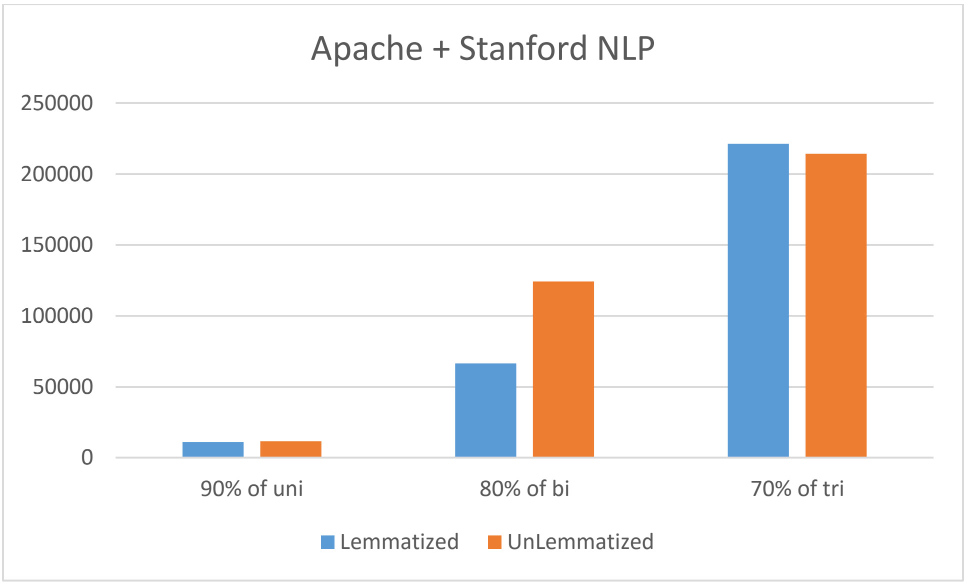
6. Summarize the results in the report.

a). After lemmatizing the corpora most of the joint words are broken like I've=>I have etc. which increases the no. of unigrams as well as frequency of existing unigrams are increased due to word splitting.

b).Since the derived words are transformed into their root words hence frequency of already existing unigrams and bigrams increases.

c).Hence less no. of unigrams and bigrams will cover 90% and 80% of corpora respectively.

d). **Also most of the times most frequent unigrams and bigrams are articles, prepositions supporting verbs and stop words**. On lemmatization their counts further increases, hence to cover 90% and 80% of corpora we need less no. of unigrams and bigrams as shown in the chart below.



(Pintu Kumar)

NLTK.

Un-lemmatized version: Total Token Count: 337919

1. How many (most frequent) words are required for 90% coverage of the selected corpus?

Ans: 11758

2. How many (most frequent) bigrams are required for 80% coverage of the corpus?

Ans: 61018

3. How many (most frequent) trigrams are required for 70% coverage of the corpus?

Ans: 175139

4. Repeat the above after performing lemmatization.

Ans:

Lemmatized Corpora + Post-Processing (Removing inverted commas):

Token Count: 337964

4.1. How many (most frequent) words are required for 90% coverage of the selected corpus?

Ans: 11917

4.2. How many (most frequent) bigrams are required for 80% coverage of the corpus?

Ans: 109850

4.3. How many (most frequent) trigrams are required for 70% coverage of the corpus?

Ans: 210749

5. Compare the statistics of the two cases.

Ans: After comparing the results we see that word_tokenize for unlemmatized corpus results in lesser number of words in the dictionary as compared to that for lemmatized corpus. This can be due to following facts:

1. After lemmatization words like I've become I have so where I've was being counted as one in unlemmatized whereas two in the lemmatized one.

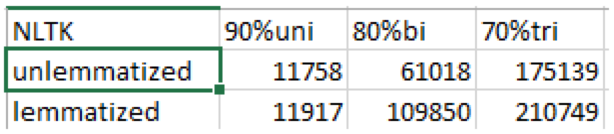
Such words contain verbs, prepositions and articles which are already in majority in the corpus. After this splitting this number increases more.

Thus less number of such words in the corpus constitute majority of the corpus.

This is why lemmatization reduces the number of most frequent words which constitute 90% of the corpus.

2. Lemmatization does converts a word into its stem/root word based on context. To determine context PoS tagging is used. Without lemmatization the words go, going, gone, goes, went were being

treated as separate tokens, but after lemmatization they all are represented by their stem i.e. go. Thus frequency of go becomes sum of the frequencies of these all derived words.



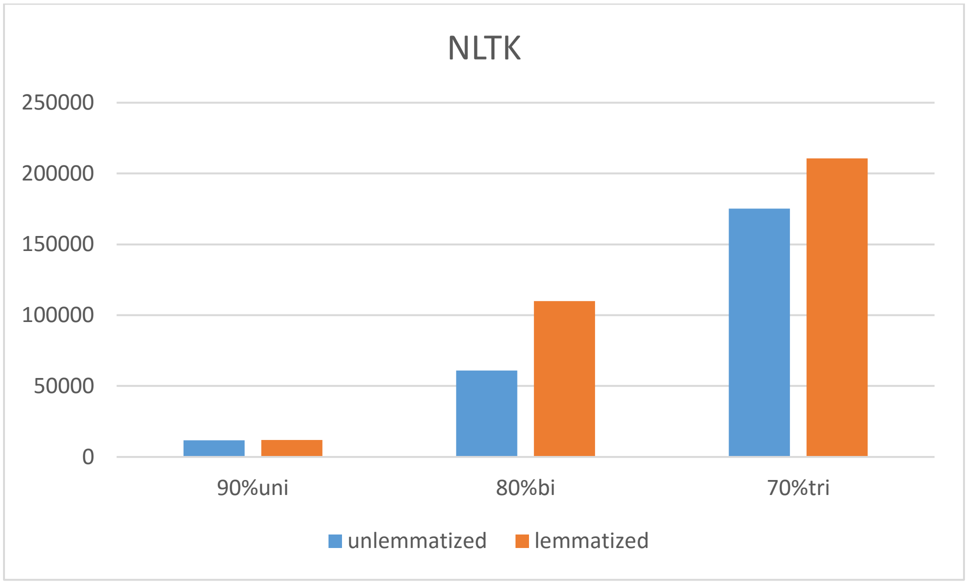
NLTK	90%uni	80%bi	70%tri
unlemmatized	11758	61018	175139
lemmatized	11917	109850	210749

6. Summarize the results in the report.

Ans:

a). Since after lemmatization, Abhinav Ravi also post processed the data like removal of inverted commas(“”) which had a lot of frequency hence results obtained from NLTK (un processed)and Apache opennlp are non-synchronous especially in the case of no. of bigrams needed to cover 80% of corpus.

b). The graph obtained is as plotted below:



PART 3.

1. Repeat sections 1 and 2 after implementing discussed heuristics in the class for sentence segmenter and word tokenizer.
Ans:

Heuristics applied are:

a).Sentence boundaries are marked by dot(.) ,!,?.

b).Exception to above rule is made only if the preceding characters are “Dr”, “Mr”, “Mrs”,”vs”,”etc”.

c).Word boundaries are detected by “space” or “\n”.

d).Only those words are counted as legitimate which were present in the dictionary.

e).Words are converted into lower cases only if they are not at the beginning of the sentence and are not one of “Dr”, “Mr”, “Mrs” and words following them as they symbolize the names.

3.1.1. How many (most frequent) words are required for 90% coverage of the selected corpus?

Ans: 16366

3.1.2. How many (most frequent) bigrams are required for 80% coverage of the corpus?

Ans: 107541

3.1.3. How many (most frequent) trigrams are required for 70% coverage of the corpus?

Ans: 162467

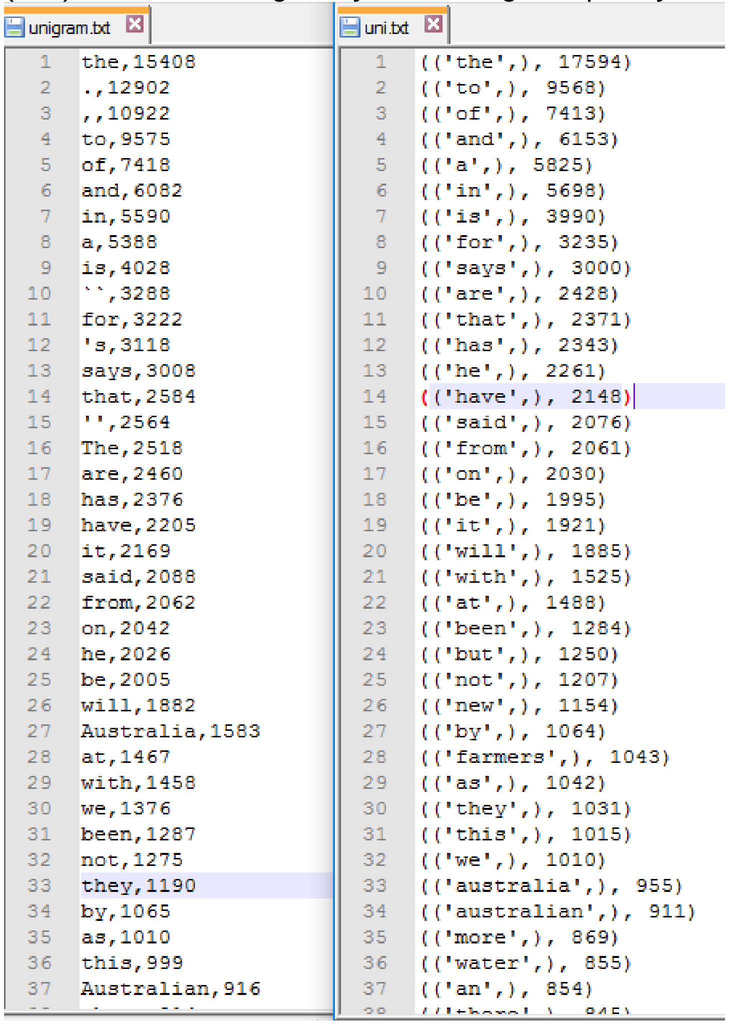
2. Implement Pearson's Chi-Square Test for finding all bigram (contiguous or discontiguous) collocations in your chosen corpus.
Ans:

Chi-square Test is implemented and all the bigrams (contiguous or discontiguous) is present in folder “supplementary \Part3\Chi_square\chisq_bigram.txt”.

Code for it’s implementation can be found in folder “supplementary Part3\Chi_square\chisquare_impl.py”.

3. In your report, please summarize your findings by comparing the results obtained after using your heuristics and after using tools.
Ans:

a).While applying the heuristic, we didn’t consider (comma),(space),(Inverted commas) and (dot) as tokens though they were in high frequency.



```
the,15408
.,12902
,,10922
to,9575
of,7418
and,6082
in,5590
a,5388
is,4028
``,3288
for,3222
's,3118
says,3008
that,2584
'',2564
The,2518
are,2460
has,2376
have,2205
it,2169
said,2088
from,2062
on,2042
he,2026
be,2005
will,1882
Australia,1583
at,1467
with,1458
we,1376
been,1287
not,1275
they,1190
by,1065
as,1010
this,999
Australian,916
```

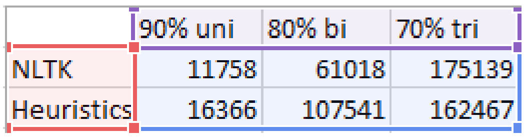
```
(('the',), 17594)
(('to',), 9568)
(('of',), 7413)
(('and',), 6153)
(('a',), 5825)
(('in',), 5698)
(('is',), 3990)
(('for',), 3235)
(('says',), 3000)
(('are',), 2428)
(('that',), 2371)
(('has',), 2343)
(('he',), 2261)
(('have',), 2148)
(('said',), 2076)
(('from',), 2061)
(('on',), 2030)
(('be',), 1995)
(('it',), 1921)
(('will',), 1885)
(('with',), 1525)
(('at',), 1488)
(('been',), 1284)
(('but',), 1250)
(('not',), 1207)
(('new',), 1154)
(('by',), 1064)
(('farmers',), 1043)
(('as',), 1042)
(('they',), 1031)
(('this',), 1015)
(('we',), 1010)
(('australia',), 955)
(('australian',), 911)
(('more',), 869)
(('water',), 855)
(('an',), 854)
```

Above picture shows the comparisons of the frequent unigrams.
unigram.txt contains unigrams obtained by NLTK tool while uni.txt contains unigrams obtained after applying the heuristics.

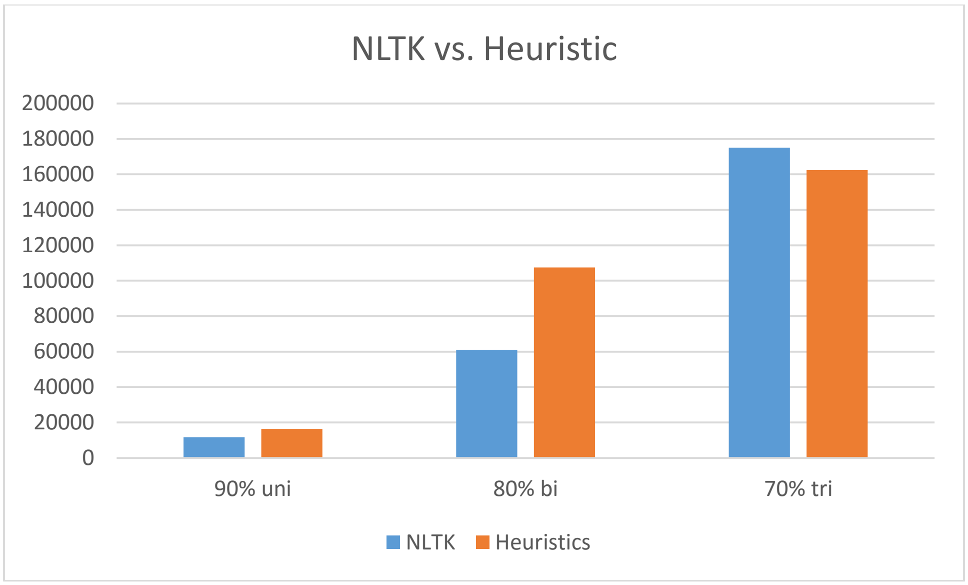
It is clear that after applying the heuristics we didn’t get (comma),(space),(Inverted commas) and (dot) as unigrams in uni.txt and hence more no. of unigrams were needed to cover the 90% of the corpus.

Same is the case with the bigrams which suffer due to absence of (comma),(space),(Inverted commas) and (dot) in bigrams.

While we have approximately equal count for trigrams due to various combinations of other unigrams.



	90% uni	80% bi	70% tri
NLTK	11758	61018	175139
Heuristics	16366	107541	162467



4. For collocations, please discuss if you have made any interesting observation.

Ans: a). Most of the times **stop words come as collocations** if frequency of contiguous bigrams are given priority.

b).In case of corpus containing **Narrative sentences** which include a lot of inverted commas(“”), **removing the inverted commas** results in better bigrams as inverted commas only increase the noise in the data without any actual inference.

c). **Also most of the times most frequent contiguous collocations are articles, prepositions, supporting verbs, stop words and punctuation marks.**

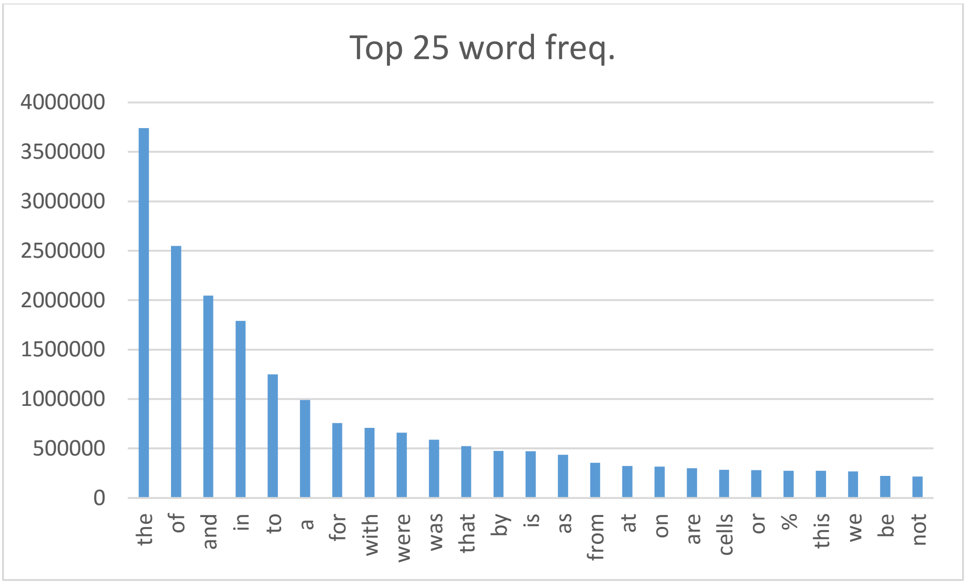
PART 4 (Bonus Problem).

Sub-corpus assigned = pubmed_2.

1. Plot word frequency distribution.

Ans: The word frequency distribution is present in folder “Supplementary\Bonus Problem\uni.csv”.

Note: I have removed _OPAD,_CPAD markers, Abbreviations for numbers like _FN . _DS,_NUM etc. if they are present in both the terms of the bigrams. This is done because they didn’t signify useful unigrams.



2. Find all bigram contiguous collocations in the sub-corpus. Discuss your method and results in the report. Mention only top 20 collocations in your report. Put the complete list of collocations in a supplementary file.

Ans: All bigram contiguous collocations in the sub-corpus is present in folder "Supplementary\Bonus Problem\bi.csv".

Top 20 collocations are as shown:

Note: I have removed _OPAD,_CPAD markers, Abbreviations for numbers like _FN and _DS etc. if they are present in both the terms of the bigrams. This is done because they didn't signify useful collocations.

1.of the,528023
2.in the,426989
3._OPAD the,413006
4., and,292219
5.- ,,291688
6., the,218338
7.to the,184192
8.- and,176899nd
9._OPAD in,159915
10.and the,132544
11., we,116324
12.for the,115151
13.with the,104141
14.on the,102829
15.from the,91088
16._OPAD we,86661

17.that the,83038
18., which,82926
19._OPAD this,82089
20.by the,76357

Method Used:

Increase java Heap Space:

For reading such a large file (500 Mb) and creating the map for counting the word frequency , a large amount of java heap space is required.
Hence we have first **increased the java heap space to 3 Gb** using following configuration in eclipse:

In the run configuration, open the tab `Arguments` and add `-Xmx2048m`in the VM arguments section.

Intelligent read of file:

Instead of reading the whole file in one go, I have **read the file line by line**. This tactics saved a lot of space.

Have Patience:

On my PC with 4gb RAM and i5 core it took 1 hour and 15 minutes to achieve the results.
Hence have patience while running this.

Complete list of collocations is present in folder “Supplementary\Bonus Problem\bi.csv”.