**Part 1: Dataset:**

For this assignment, we are given dataset of Amazon Product data provided by Julian McAuley. We need to analyze the contents of this dataset. This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 – July 2014. It includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price,brand, and image features), and links (also viewed/also bought graphs).

In this assignment to perform analysis on the given dataset we will focus on only users and items which have atleast 5 reviews in dataset i.e 5-core subsets of the following categories: Clothing,Shoes and Jewelry.

The unzipped and modified file that is provided to us for analysis is:

**clothing\_shoes\_jewellery.txt.**

The data in this dataset can be categorized into 9 fields. Data associated with each field is significant to uniquely identify every review. Those fields are:

* reviewerId: which helps to uniquely identify the reviewer
* asin: which represents the id of the product
* reviewerName
* helpful- which indicates how useful was the review for the other users as well
* reviewText : this is the most prominent field because it represent about quality of product ,how satisfied the buyer is and how highly will they recommend the same.
* overall: it is the rating of product. Amazon use 5 star rating model with one star representing the lowest rating and five stars representing the highest.
* summary :a short description of product review
* unixReviewTime: time of review.This data can also be used to extract yearly market trends.(unix time)
* reviewTime: time of review (raw)

The final objective of this analysis is to find the similarities or dissimilarities of the most common words and bigrams for the contents of amazon product review text.

The code is developed in Jupyter Notebook and it is developed in Python Language.

**Part 2: Data Pre-processing**

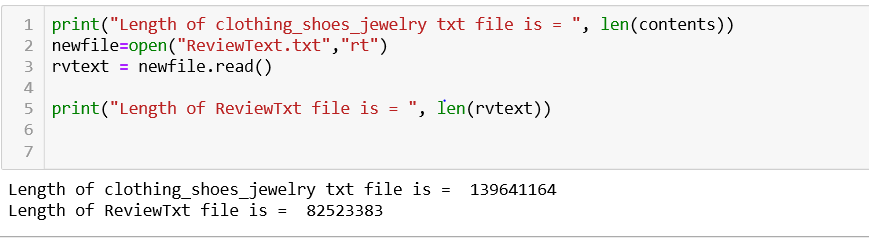
Since the dataset is huge and it consists various valuable information for analysis .For this assignment we will extract only the reviewText field.

First we will load the clothing\_shoes\_jewellery.txt using nltk package and open file command in read mode.We will then read the contents of dataset using by using read command.

Now to extract the contents of reviewText field and to write in a new file ‘ReviewText.txt’ we iterate through the lines of the file.

Code screenshot:



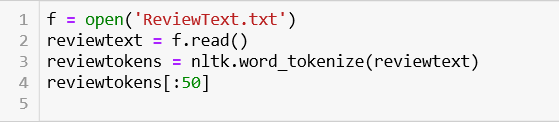


Now when the ReviewText file is ready for analysis we will perform the following steps for data pre-processing:

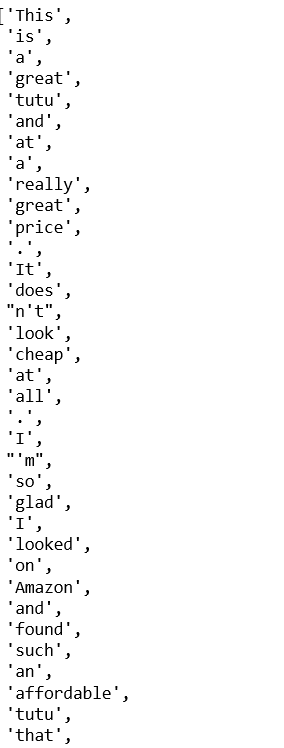
**∗Perform tokenization using NLTK tokenizer**

The string got from the Plaintext Corpus Reader is a single string and for the analysis this needs to be tokenized and hence tokenization is performed over the raw text got from the input file.

Code :



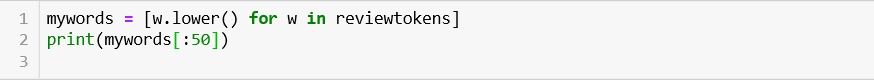
Output:



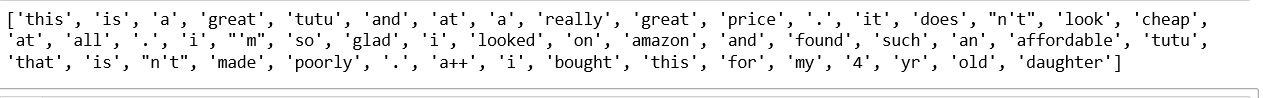
**∗Convert tokenized words to lower case**

The tokenized words in then converted to its respective lower-case words since our analysis over text is case sensitive and same words with different case should be considered as one single word.

Code:



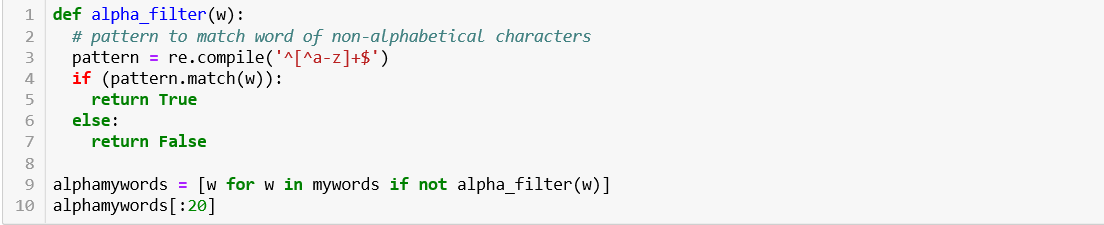
Output:



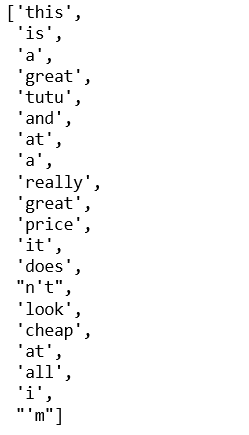
**∗Filter the words using the isalpha function**

Many punctuations would have been considered as a word which doesn’t help in the analysis of the document and this can be removed by applying the isalpha filter.

Code:



Output:

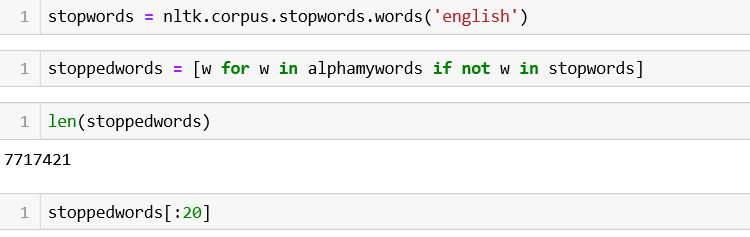


**∗Remove the stopwords from the list**

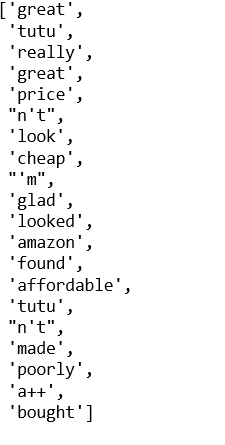
Many most common words used to correct the grammar which has less information to analyze the document is called stop words. There is no single universal list of stop words used by all-natural language processing tools, and indeed not all tools even use such a list. Some tools avoid removing stop words to support phrase search. The list of extended stop words are use to remove words like "'s","b","n't" etc which have no significant meaning.

Code:

Removing universal list of stop words

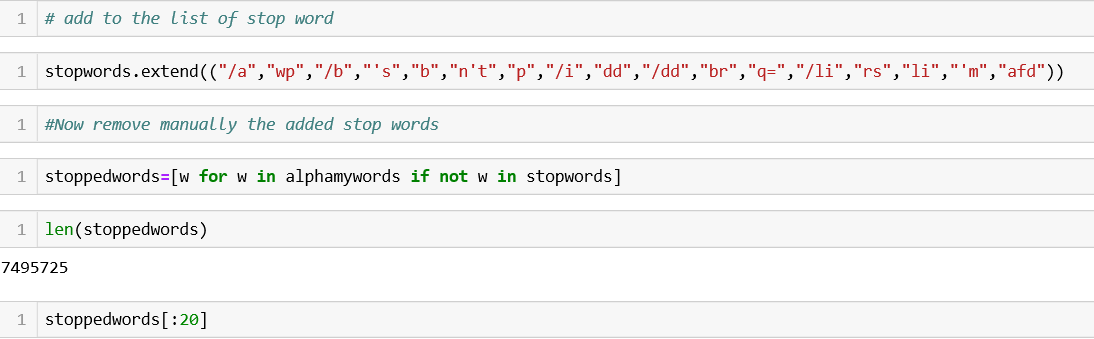


Output:

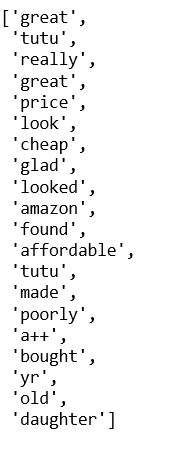


Extending the list of stop words by adding smart stop words filter.

Code:



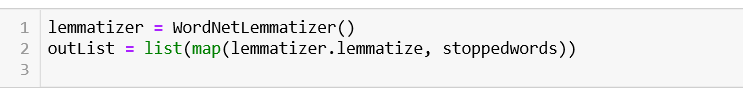
Output:



**∗Perform Lemmatization over the list of words**

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set.

Code:



**Part 3: Data Analysis**

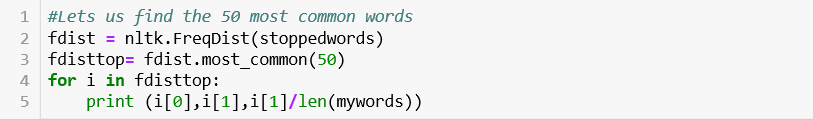
After pre-processing of dataset, to analyze dataset we will perform three tasks they are:

**a) List of top 50 words by frequency (normalized by the length of the document)**

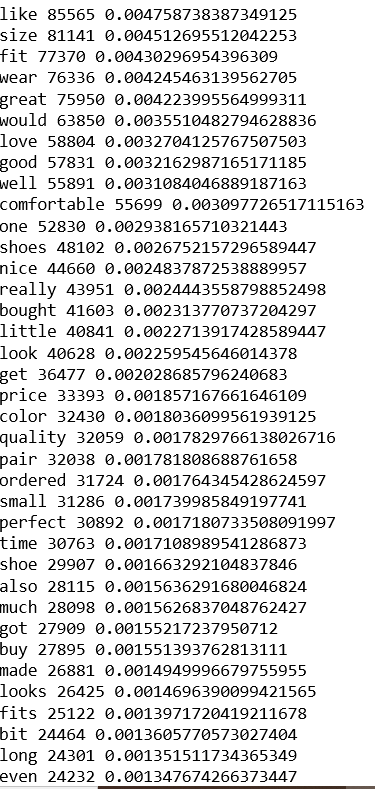
∗Calculate the frequency distribution of the updated word list

Frequency Distribution tells us the frequency of each vocabulary item in the text. It is a "distribution" because it tells us how the total number of word tokens in the text are distributed across the vocabulary items.

Code:



Output:



**b) List of top 50 bigrams by frequencies**

∗Find and display the top 50 words based on the frequency of the words.

The steps followed by to get the top 50 bigrams by frequencies are as follows,

∗ Import collocation finder package from nltk to calculate bigram measure

∗Consider the list of words got after the tokenization and the lower case words filtering for the bigram frequency analysis.

If we consider the words after the alpha filtering or the stop words filtering then the bigram which is got might not be the bigrams in the original text. For filtering the stopwords or filtering only the alpha words we use finder function with the filter.

∗Create a finder by the Bigram Collocation finder package

Finder will help to apply the word filter for the list of bigrams after the bigrams and its frequency is found.

∗Apply the alpha filter using the regex by python package

For any finder, we can also apply various filter functions. First let’s apply our alpha\_filter that we created earlier. It uses a filter that is applied to the individual words. Note that the function apply\_word\_filter changes the bigram collocation in the variable “finder”. Any function which takes a word parameter and returns True or False as a result can be used as the passing parameter in the apply\_word\_filter function.

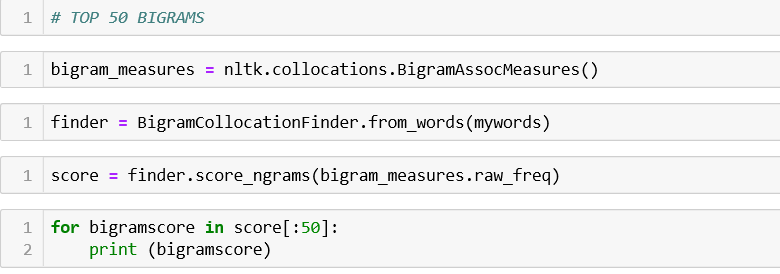
∗Apply the stop words filter

Using the same technique as the previous step we can apply the Stopwords filter to remove all the bigrams which consists of the stopwords.

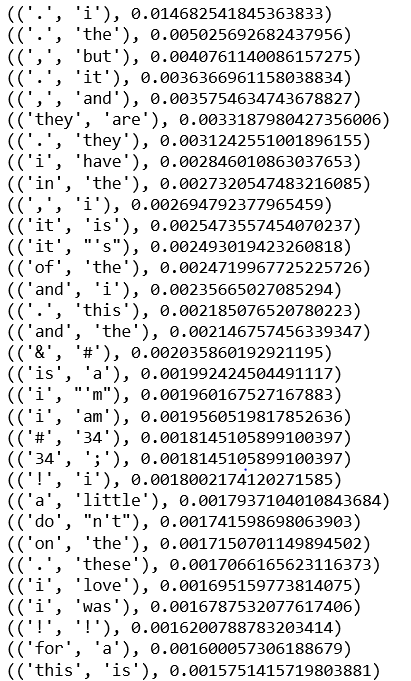
\* Finally, it is often important to remove low frequency candidates, as we lack sufficient evidence about their significance as collocations

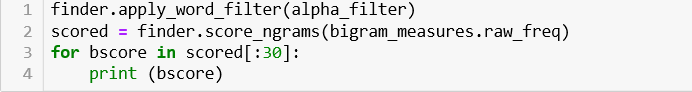
∗Find and display the top 50 bigrams by its frequency

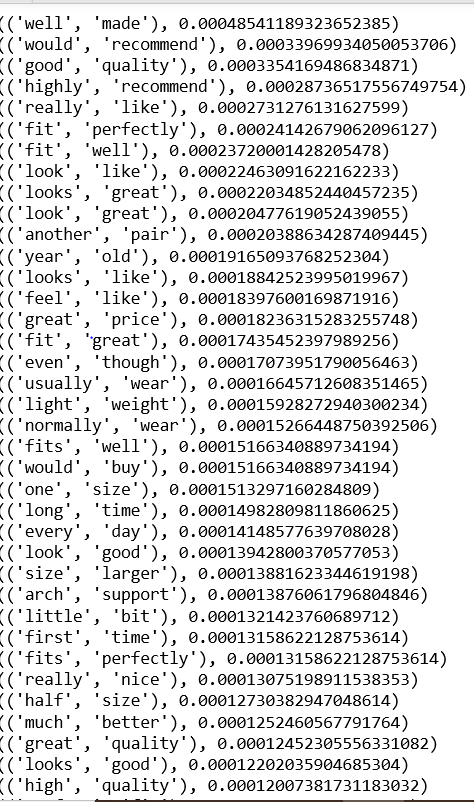
Code:

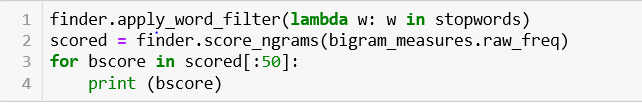


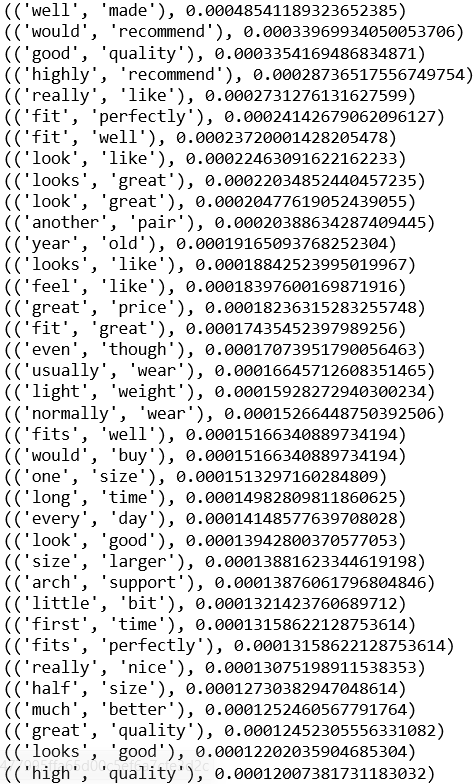
Output:











**c) List of top 50 bigrams by their Mutual Information scores (using min frequency 5)**

The steps followed by to get the top 50 bigrams by their mutual information scores,

∗ Import collocation finder package from nltk to calculate bigram measure

∗Consider the list of words got after the tokenization and the lower case words filtering for the bigram freuqency analysis.

If we consider the words after the alpha filtering or the stop words filtering then the bigram which is got might not be the bigrams in the original text. For filtering the stopwords or filtering only the alpha words we use finder function with the filter.

∗Apply the minimum frequency of 5

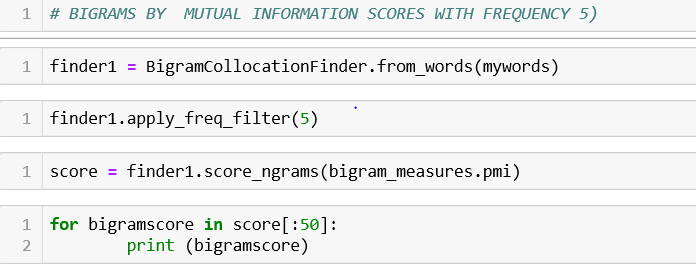
If you apply the Mutual Information score to all the bigrams, the results don’t really make sense, because uniquely occurring pairs of words get high scores. It is recommended to run the PMI scorer with a minimum frequency of 5, which will make more sense on very large documents. The Church and Hanks paper has more discussion of this

∗Find the Mutual Information Scores

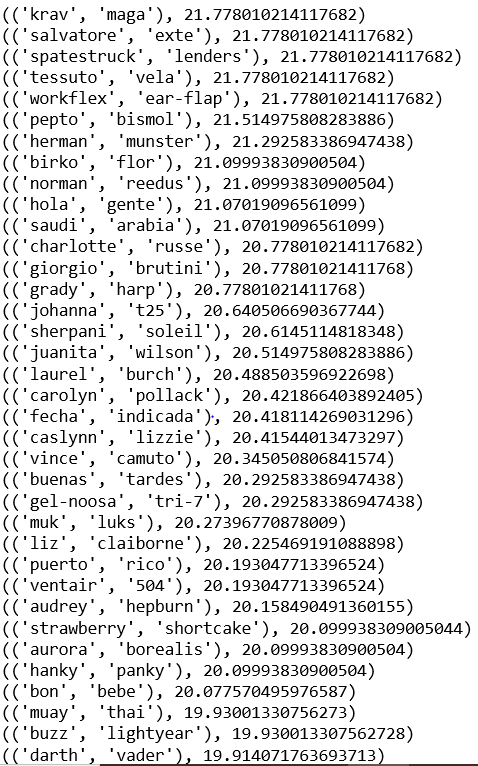
Use the score\_ngrams function to find the mutual information scores which is provided by the nltk libraries.

∗Find and display the top 50 bigrams by its Mutual information scores.

Code:



Output:



**Part 4: Interpretation of Results**

**a)Interpretation of results based on list of top 50 words by frequency:**

We can see below the most frequent word is “like”. We can infer from the words that most of the review texts of products are positive and are verbs or adjective used most commonly to express people’s opinion about any product.

The 10 common words in the top 50 words by frequency are as below,

like 85565 0.001036857638276899

size 81141 0.0009832485902813751

fit 77370 0.0009375524510428759

wear 76336 0.0009250226690294556

great 75950 0.0009203452068851807

would 63850 0.0007737200982174955

love 58804 0.0007125737925697981

good 57831 0.000700783194988504

well 55891 0.0006772747064913711

comfortable 55699 0.0006749480931968094

one 52830 0.0006401821893317195

shoes 48102 0.0005828893369531397

nice 44660 0.0005411799465370924

From the above result it also ambiguous to understand the meaning of words would and well. Since it can be used in both positive and negative form sentence say for example “would not recommend” or “would avoid” but it can also be like “would buy again” or “would suggest”

Hence using unigrams its difficult to perform to infer the context of the review.

**b) Interpretation of results based on the bigrams using their frequency:**

To look for interesting characterizations of a product review is to look at pairs of words that are frequently collocated, that is, they occur in a sequence called a bigram.

From below we can see the most frequent bigram pair is “well” “made”. This shows the high satisfaction of amazon customers. We can infer from the pairs value that the first word of pairs are verbs or adjective like looks, feel etc. whereas the second word is mainly adjective. It also be inferred that the pair of words can be used interchangeably and would not change the meaning much. For example, look good or good look and recommend highly or highly recommend have similar meanings .

(('well', 'made'), 0.00048541189323652385)

(('would', 'recommend'), 0.00033969934050053706)

(('good', 'quality'), 0.0003354169486834871)

(('highly', 'recommend'), 0.00028736517556749754)

(('really', 'like'), 0.0002731276131627599)

(('fit', 'perfectly'), 0.00024142679062096127)

(('fit', 'well'), 0.00023720001428205478)

(('look', 'like'), 0.00022463091622162233)

(('looks', 'great'), 0.00022034852440457235)

(('look', 'great'), 0.00020477619052439055)

(('another', 'pair'), 0.00020388634287409445)

(('year', 'old'), 0.00019165093768252304)

(('looks', 'like'), 0.00018842523995019967)

(('feel', 'like'), 0.00018397600169871916)

Bigrams are thought to potentially hold more sentimental meaning than unigrams, as they can include adjectives or other terms. For instance, the word bad could be used in context of ”very bad” or ”not bad.” A drawback of bigrams is however the fact there are significantly more of them and much fewer usages across documents.

**c) Interpretation results based on the bigrams using mutual information scores:**

Technically the original information theoretic definition of mutual information allows the two words to be in either order, but that the association ratio defined by Church and Hanks requires the words to be in order from left to right wherever they appear in the window. The pointwise mutual score is a measure of association. Unlike the bigrams by frequency the PMI measure is symmetric (pmi(x;y) = pmi(y;x)).

If you apply the Mutual Information score to all the bigrams, the results don’t really make sense, because uniquely occurring pairs of words get high scores. Hence as per recommendation we run the PMI scorer with a minimum frequency of 5, which will make more sense on very large documents.

('krav', 'maga'), 21.778010214117682)

(('salvatore', 'exte'), 21.778010214117682)

(('spatestruck', 'lenders'), 21.778010214117682)

(('tessuto', 'vela'), 21.778010214117682)

(('workflex', 'ear-flap'), 21.778010214117682)

(('pepto', 'bismol'), 21.514975808283886)

(('herman', 'munster'), 21.292583386947438)

(('birko', 'flor'), 21.09993830900504

(('norman', 'reedus'), 21.09993830900504)

(('hola', 'gente'), 21.07019096561099)

(('saudi', 'arabia'), 21.07019096561099)

(('charlotte', 'russe'), 20.778010214117682)

Since we know less frequent words will have high PMI. Hence we can country names or not common words appear in our output.

**Problems with the word or bigram lists found and possible solutions**

• After the analysis of the words and the bigrams the noticed problems are as below,

If the word has the apostrophe, the bigram was not properly split.

Example: (('wo', "n't"), 1.9191905570728073e-10)

**Solution**: This can be solved by using the custom-made tokenizer which will not split the words which has apostrophe.

• If there are any minor spelling mistakes in the text then the words or bigrams might have duplicate values with the wrong values.

Example: (('looks', 'good'), 2.8408353603174803e-05)

(('luks', 'good' ), 2.84083536031748e-05)

**Solution:** This can be solved by checking for spelling mistake using the dictionary of all possible words.

• If there are any many single letter relation in the bigrams then the inference got from it is very less.

Example: (('b.', 'johnson'), 3.055987443686704e-05)

((''caro, 's.'), 2.9371372869471536e-05)

**Solution:** This can be solved by filtering the bigrams by the bigrams length and considering some threshold.

• If there are particular bigram pairs in the document whose frequency is more, and we cannot infer more information from those words.

Example: ('still',’be’)

(‘still’,’now’)

**Solution:** Those words needs to be considered as stop words.