Report

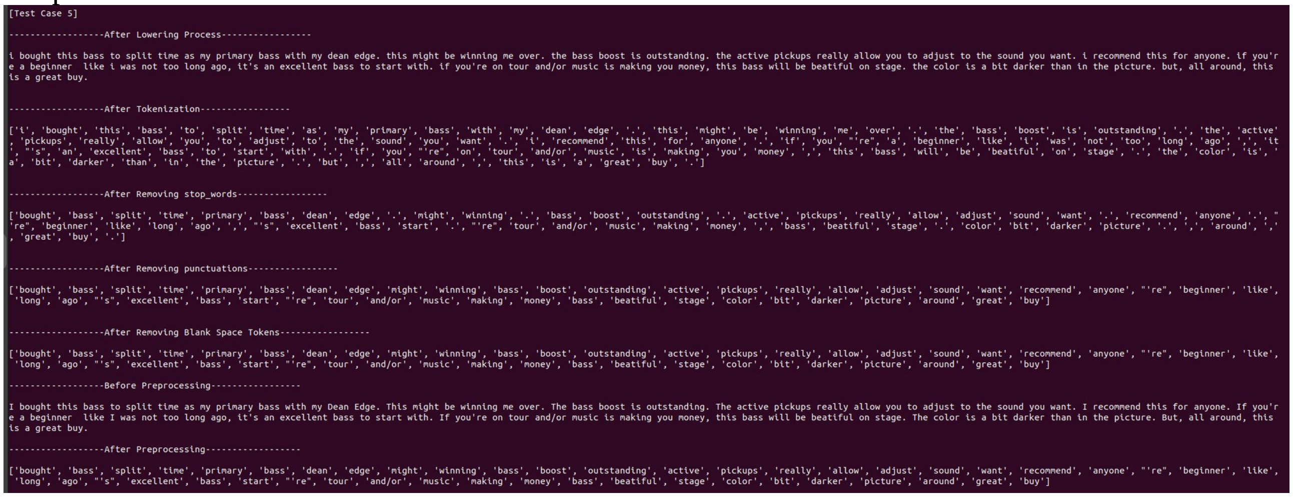
Data Preprocessing:

For data preprocessing I used NLTK python library to tokenize the files and then remove stopwords, punctuations, and blank space tokens.

Assumption: I didnot undergo the task of stemming or lemmatization as it was not asked in the assignment. This doesn't restrict us in efficient search since the search phrase also undergoes same steps mentioned for text files.

For part2 I immediately printed contents of first 5 files after and before each operation.

Output for File #5



```
[Test Case 5]

-----------------After Lowering Process----------------

i bought this bass to split time as my primary bass with my dean edge. this might be winning me over. the bass boost is outstanding. the active pickups really allow you to adjust to the sound you want. i recommend this for anyone. if you'r
e a beginner  like i was not too long ago, it's an excellent bass to start with. if you're on tour and/or music is making you money, this bass will be beatiful on stage. the color is a bit darker than in the picture. but, all around, this
is a great buy.


-----------------After Tokenization----------------

['i', 'bought', 'this', 'bass', 'to', 'split', 'time', 'as', 'my', 'primary', 'bass', 'with', 'my', 'dean', 'edge', '.', 'this', 'might', 'be', 'winning', 'me', 'over', '.', 'the', 'bass', 'boost', 'is', 'outstanding', '.', 'the', 'active'
, 'pickups', 'really', 'allow', 'you', 'to', 'adjust', 'to', 'the', 'sound', 'you', 'want', '.', 'i', 'recommend', 'this', 'for', 'anyone', '.', 'if', 'you', "'re", 'a', 'beginner', 'like', 'i', 'was', 'not', 'too', 'long', 'ago', ',', 'it
', "'s", 'an', 'excellent', 'bass', 'to', 'start', 'with', '.', 'if', 'you', "'re", 'on', 'tour', 'and/or', 'music', 'is', 'making', 'you', 'money', ',', 'this', 'bass', 'will', 'be', 'beatiful', 'on', 'stage', '.', 'the', 'color', 'is', '
a', 'bit', 'darker', 'than', 'in', 'the', 'picture', '.', 'but', ',', 'all', 'around', ',', 'this', 'is', 'a', 'great', 'buy', '.']


-----------------After Removing stop_words----------------

['bought', 'bass', 'split', 'time', 'primary', 'bass', 'dean', 'edge', '.', 'might', 'winning', '.', 'bass', 'boost', 'outstanding', '.', 'active', 'pickups', 'really', 'allow', 'adjust', 'sound', 'want', '.', 'recommend', 'anyone', '.', "
're", 'beginner', 'like', 'long', 'ago', ',', "'s", 'excellent', 'bass', 'start', '.', "'re", 'tour', 'and/or', 'music', 'making', 'money', ',', 'bass', 'beatiful', 'stage', '.', 'color', 'bit', 'darker', 'picture', '.', ',', 'around', ','
, 'great', 'buy', '.']


-----------------After Removing punctuations----------------

['bought', 'bass', 'split', 'time', 'primary', 'bass', 'dean', 'edge', 'might', 'winning', 'bass', 'boost', 'outstanding', 'active', 'pickups', 'really', 'allow', 'adjust', 'sound', 'want', 'recommend', 'anyone', "'re", 'beginner', 'like',
 'long', 'ago', "'s", 'excellent', 'bass', 'start', "'re", 'tour', 'and/or', 'music', 'making', 'money', 'bass', 'beatiful', 'stage', 'color', 'bit', 'darker', 'picture', 'around', 'great', 'buy']


-----------------After Removing Blank Space Tokens----------------

['bought', 'bass', 'split', 'time', 'primary', 'bass', 'dean', 'edge', 'might', 'winning', 'bass', 'boost', 'outstanding', 'active', 'pickups', 'really', 'allow', 'adjust', 'sound', 'want', 'recommend', 'anyone', "'re", 'beginner', 'like',
 'long', 'ago', "'s", 'excellent', 'bass', 'start', "'re", 'tour', 'and/or', 'music', 'making', 'money', 'bass', 'beatiful', 'stage', 'color', 'bit', 'darker', 'picture', 'around', 'great', 'buy']
-----------------Before Preprocessing----------------

I bought this bass to split time as my primary bass with my Dean Edge. This might be winning me over. The bass boost is outstanding. The active pickups really allow you to adjust to the sound you want. I recommend this for anyone. If you'r
e a beginner  like I was not too long ago, it's an excellent bass to start with. If you're on tour and/or music is making you money, this bass will be beatiful on stage. The color is a bit darker than in the picture. But, all around, this
is a great buy.

-----------------After Preprocessing----------------

['bought', 'bass', 'split', 'time', 'primary', 'bass', 'dean', 'edge', 'might', 'winning', 'bass', 'boost', 'outstanding', 'active', 'pickups', 'really', 'allow', 'adjust', 'sound', 'want', 'recommend', 'anyone', "'re", 'beginner', 'like',
 'long', 'ago', "'s", 'excellent', 'bass', 'start', "'re", 'tour', 'and/or', 'music', 'making', 'money', 'bass', 'beatiful', 'stage', 'color', 'bit', 'darker', 'picture', 'around', 'great', 'buy']
```

Inverted Index And Boolean Queries:

I created Inverted Indicies from scratch without using any libraries.

Procedure:

Each word obtained after tokenization was checked if it was already encountered if it was:
Then the document id of which we are currently processing is appended to the list assosciated with the word.
Else a new key value pair is added to dictionary with key as word and doc id as first element of the list as value.

Operation:
All operations are written form scratch as were given in lecture thereby acheiving a TC of O(n) on every operation to maximize search procedure.

AND
Since the documents are already sorted in ascending order (Because of the way we process docs) AND can be computed with two pointers wherein we increment the ptr pointing to lower value and add to result if they both point to same value.

OR
OR operation is also performed in O(max(x, y)) where x and y are length of lists. We increment both the ptr if they point to same value and increment the smaller one always adding the value in our resultant array.

NOT
NOT operation is also a linear function depending upon # of documents.

AND NOT
AND NOT is computed using the NOT and then AND functionality as described earlier.

OR NOT
OR NOT is computed using the NOT and then OR functionality as described earlier.

Results are stored in a list and returned.

Positional Index and Phrase Queries

I created Positional Indicies from scratch without using any libraries.

Now instead of a list containg doc id I store the position of the word where it appears in the document too.

Again the dictionary maintains a list for each word. Each element of the list is a list with first element denoting the doc id and subsequent elements denoting position of the word in the doc. (Beacuse of the way we preprocess data again all the results are already sorted).

Procedure:

I took a basic approach of tackling the problem of Phrase queries where in I search for consecutive bigrams and *and* the results obtained.

For bigrams I first find the document containing both of the words. Then I process that doccument and with the help of positional indicies determine if they appear together. If they appear together positional index of first word will be one less than the next word in bigram.

After computing for all the docs. I check for next bigram and *and* the results obtained here with the previously obtained results.

Exp:
apple tastes good.

First I search for **apple tastes** bigram and say the resultant documents are [1, 2, 4, 6, 8].
Then I will search for **tastes good** bigram taking only those locations in consideration which have word *apple* 1 index before tastes and say now the results are [2, 4, 8]

So taking AND will give the required results of [2, 4, 8].