**NLP Final Project -Baskar Dakshinamoorthy**

**Processing and Classification of Sentiment on Movie Reviews:**

This dataset was produced for the Kaggle competition, described here: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews, and which uses data from the sentiment analysis by Socher et al, detailed at this web site: http://nlp.stanford.edu/sentiment/.

The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site. Socher’s group used crowd-sourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: “negative”, “somewhat negative”, “neutral”, “somewhat positive”, “positive”.

The train/test split has been preserved for the purposes of benchmarking, but the sentences have been shuffled from their original order. Each Sentence has been parsed into many phrases by the Stanford parser. Each phrase has a PhraseId. Each sentence has a SentenceId. Phrases that are repeated (such as short/common words) are only included once in the data.

**train.tsv** contains the phrases and their associated sentiment labels. We have additionally provided a SentenceId so that you can track which phrases belong to a single sentence. **test.tsv** contains just phrases. You must assign a sentiment label to each phrase.

The sentiment labels are:

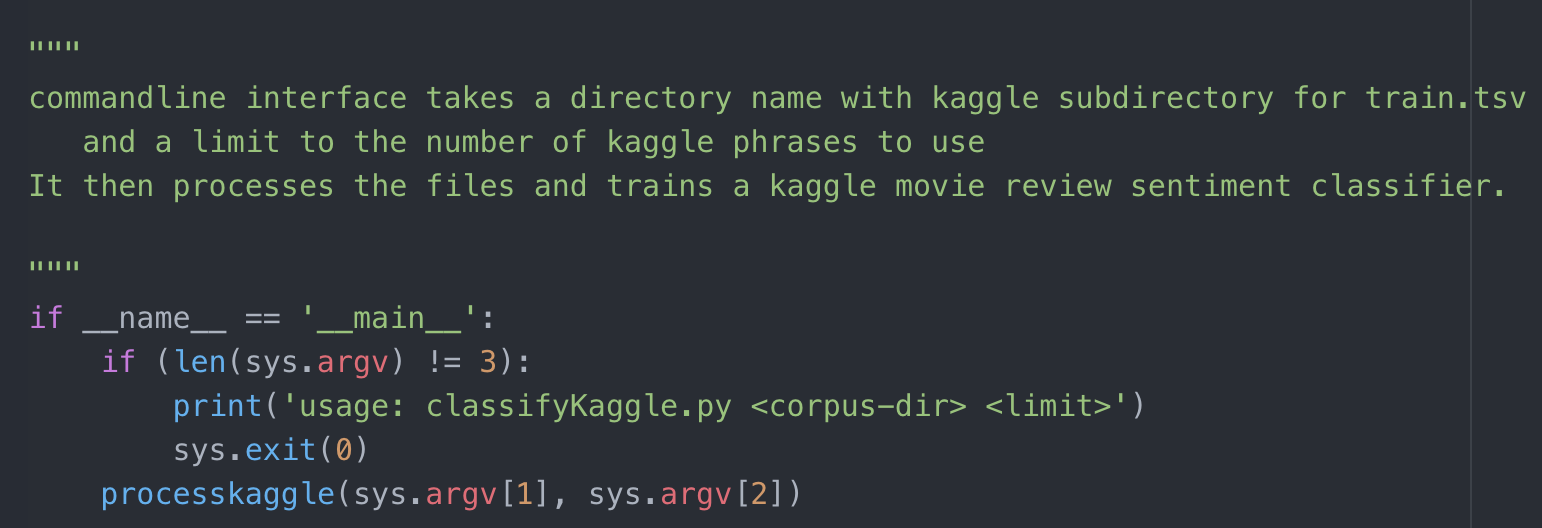
0 - negative  
1 - somewhat negative 2 - neutral  
3 - somewhat positive 4 – positive

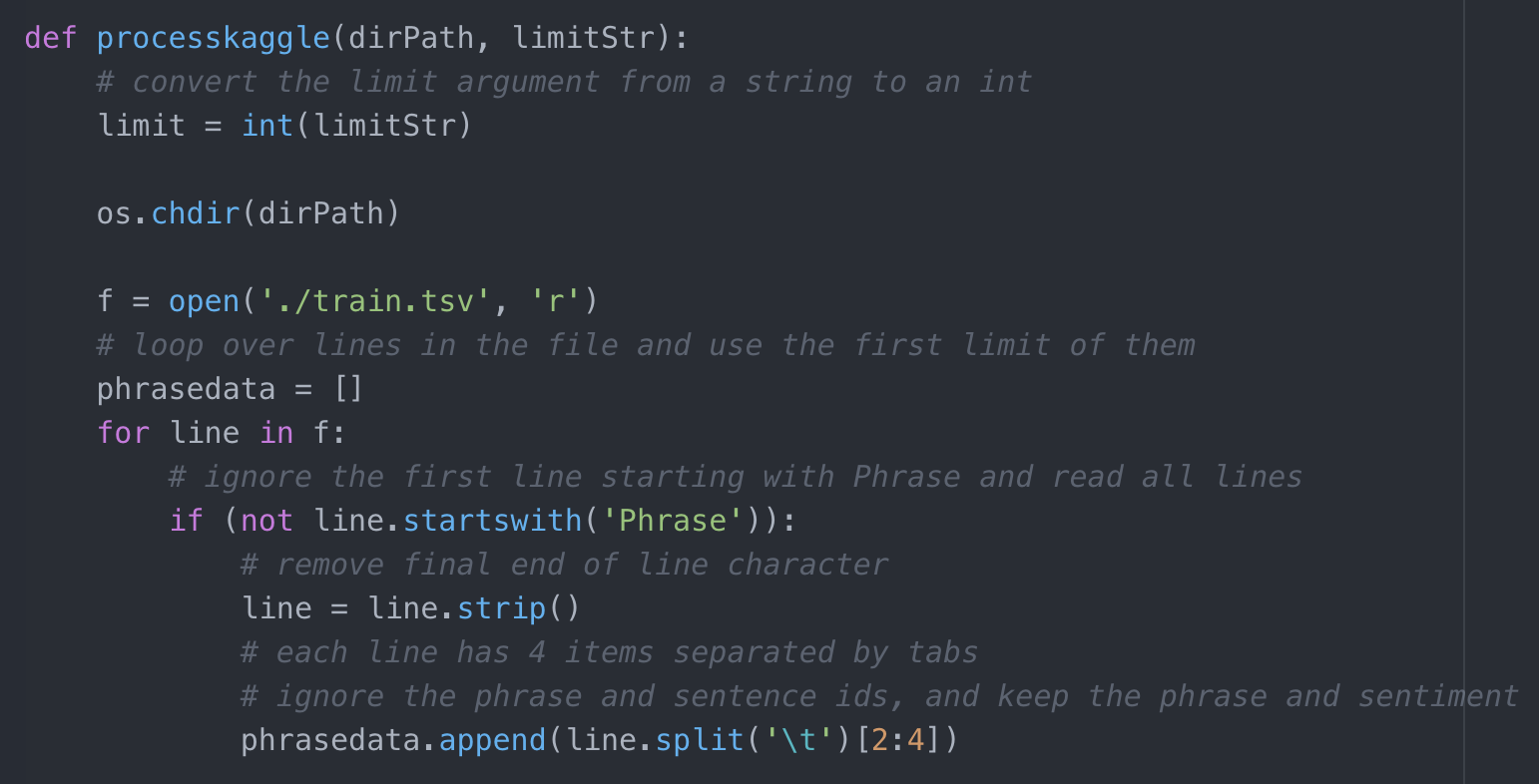
**Following are steps for phrase sentimental classification:**

**1. Read text from train.tsv file:**

Read the kaggle training file (train.tsv), loop over lines in the file and use the first limit of them as passed as an argument to processkaggle method, ignore the first line starting with Phrase and only use lines up to the limit, each line has 4 items separated by tabs thus ignore the phrase and sentence ids and keep the phrase and sentiment.

**Code:**





**2. Tokenization:**

Performed two types of tokenization, for one I have not applied pre-processing on sentence and then created pairs of (tokensOf(sentence), label) list for future classification. In second one I have used few pre-processing steps before start features selection and classification. Following steps are carried out for pre-processing steps.

**Lower case:**

Converted text to lower case words to remove upper case and lowercase sensitivity. word\_list = re.split('\s+', document.lower())

**Remove punctuation and numbers:**

Remove punctuation and numbers from text as that will deviate our result from correct analysis.

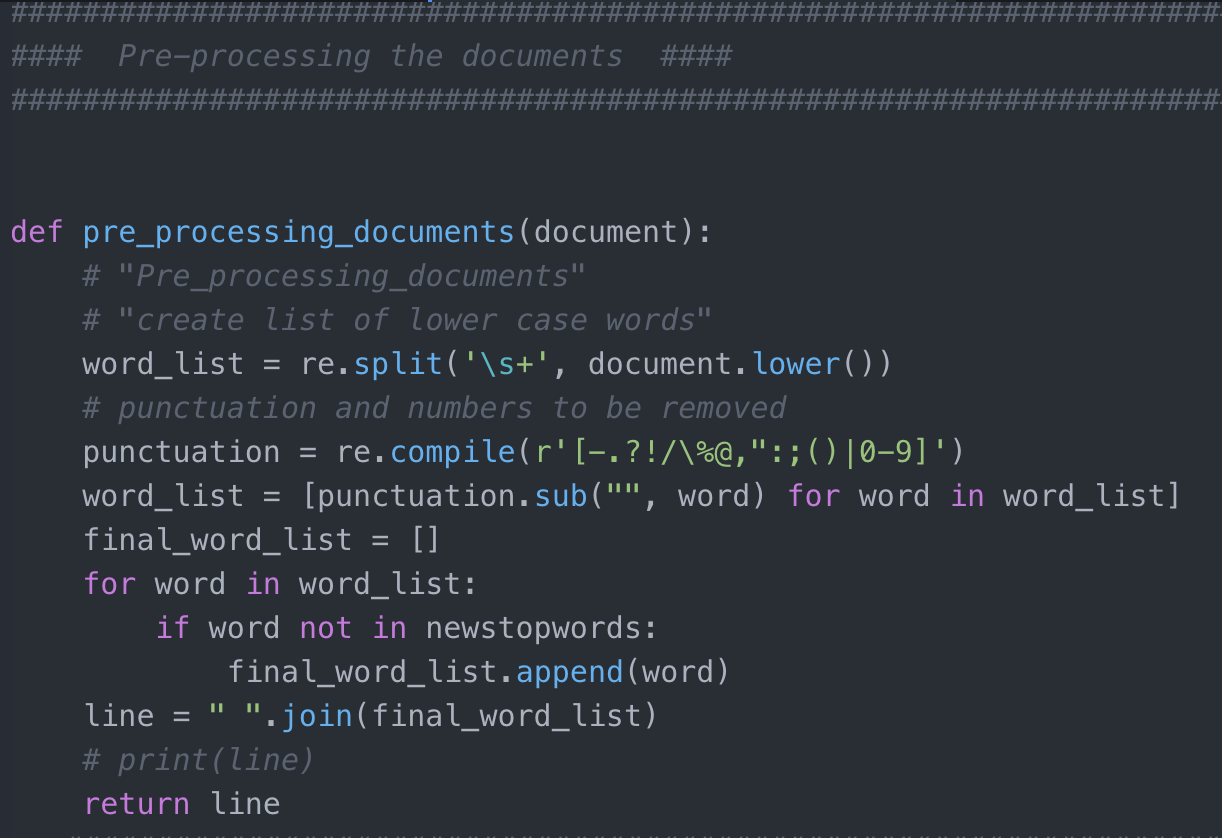
punctuation = re.compile(r'[-.?!,":;()|0-9]')  
word\_list = [punctuation.sub("", word) for word in word\_list]

**Removed stop words:**

Started with the NLTK english stop word list, but I have removed some of the negationwords, or parts of words. As we know that “not”, “can not” these words are important when you are working on Classification of Sentiment task.

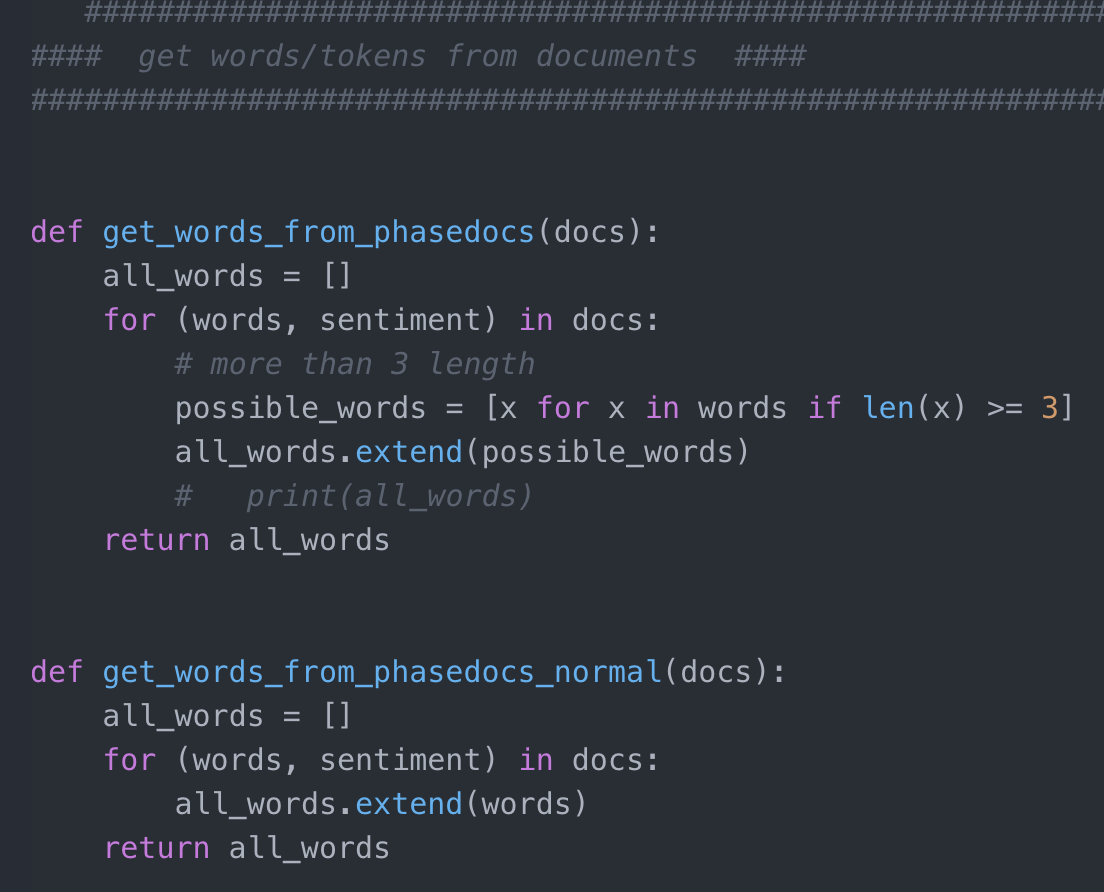
if word not in newstopwords: final\_word\_list.append(word)

Find attached below screenshot for pre processing document function from python script.



**Possibly filter tokens:**

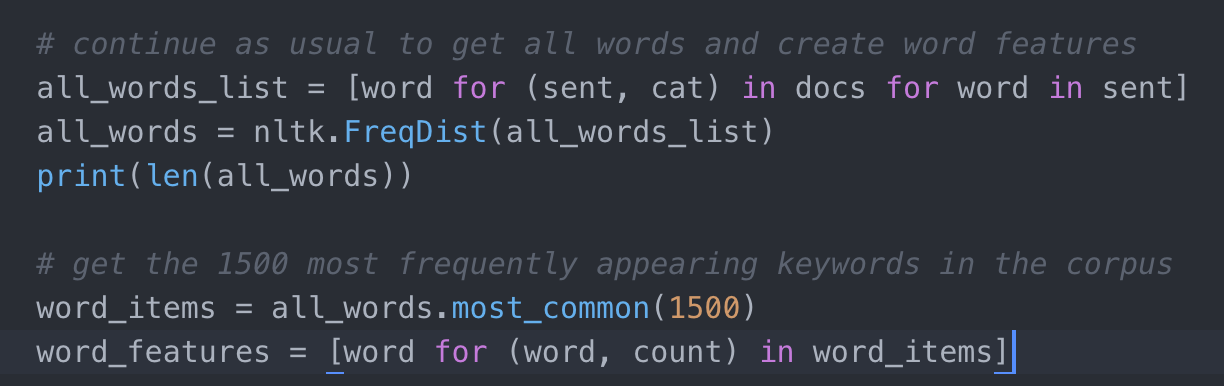
I used two types of tokens list, in one I considered only words from train data which length is more than 3. In one normal case I considered all tokens.



**3. Features Selection:**

“bag-of-words” features

Used “bag-of-words” features to collect all the words in the corpus and select 300 number of most frequent words to be the word features. I have changed this number to analyze classification. I will discuss more in experiment section.



**Example:**

['movie', 'story', 'one', 'like', 'manipulative', 'film', 'lrb', 'rrb', 'makes', 'melodrama', 'disparate', 'not', 'found', 'really', 'pleasant', 'consequence', 'feel', 'service', 'script', 'suspense']

**4. Features Functions:**

**4.1 Unigram features as baseline features**

I defined the features for each document. The feature label will be ‘contains(keyword)’ for each keyword (aka word) in the word features set, and the value of the feature will be Boolean, according to whether the word is contained in that document.

**Example of feature set item without pre-processing:**

({'contains(of)': False, 'contains(movies)': False, 'contains(allows)': False, "contains(')": False, 'contains(woman)': False, 'contains(company)': False, 'contains(actorish)': False, 'contains(that)': False, 'contains(deeply)': True, 'contains(on)': False,

........ ........

'contains(pokepie)': False, 'contains(sit)': False, 'contains(amount)': False}, 3)

**Example of feature set item after pre-processing:**

({'contains(terrifying)': False, 'contains(actorish)': False, 'contains(movies)': False, 'contains(allows)': False, 'contains(woman)': False, 'contains(stewart)': False, 'contains(company)': False, 'contains(deeply)': True, 'contains(movie)': False, 'contains(introduce)': False,

........  
........  
........  
'contains(tense)': False, 'contains(amount)': False}, 3)

**4.2 Sentiment Lexicon: Subjectivity Count features**

We will first read in the subjectivity words from the subjectivity lexicon file in the project folder. Although these words are often used as features themselves or in conjunction with other information, we will create two features that involve counting the positive and negative subjectivity words present in each document. I copy and pasted the definition of the readSubjectivity function from the Subjectivity.py module which is provided by Professor. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength and polarity.

A feature extraction function that has all the word features as before, but also has two features ‘**positivecount’** and ‘**negativecount’**. These features contain counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice.



**4.3 Negation features**

Negation of opinions is an important part of sentimental classification. I look for negation words "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t".

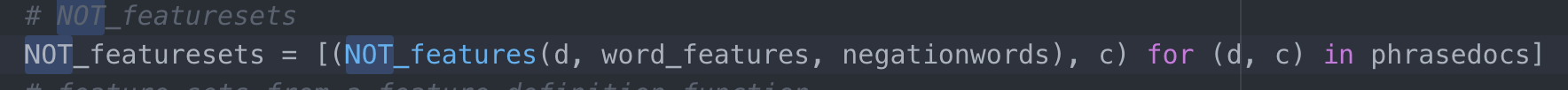
For example, my first document has the following words: if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself', One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation.

I followed the first strategy here, and I go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.

Example of feature set item:

({'contains(NOTeasy)': False, 'contains(movies)': False, 'contains(NOTfour)':

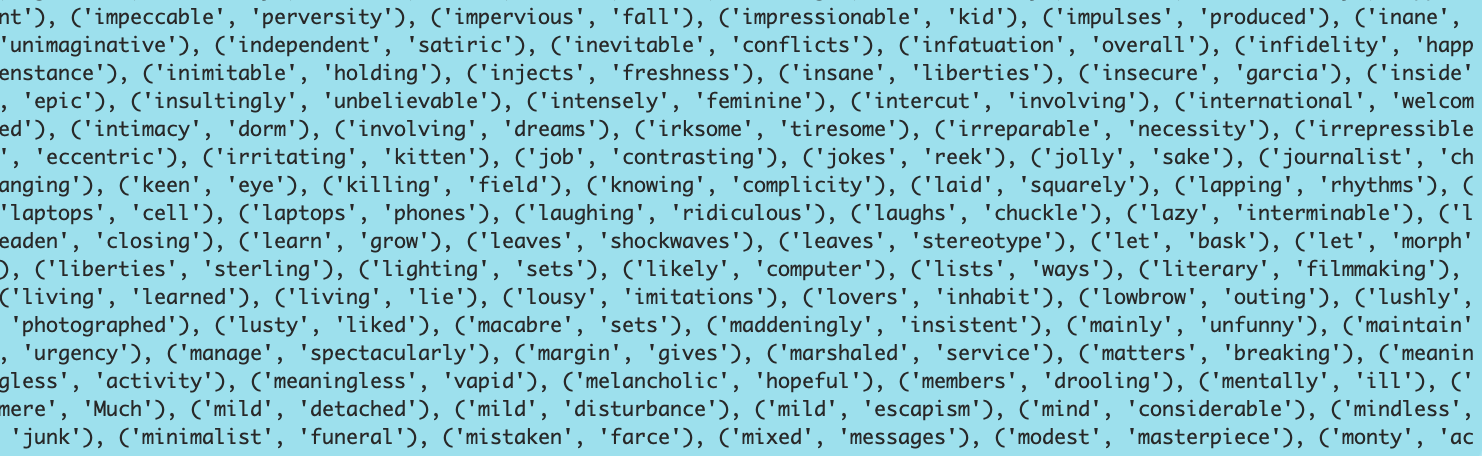
False, 'contains(deeply)': True, 'contains(NOTgripping)': False, 'contains(introduce)': False, 'contains(short)': False, 'contains(consequence)': False, 'contains(NOTcalm)': False, 'contains(drama)': ........  
........  
........  
False, 'contains(NOTtill)': False}, 3)

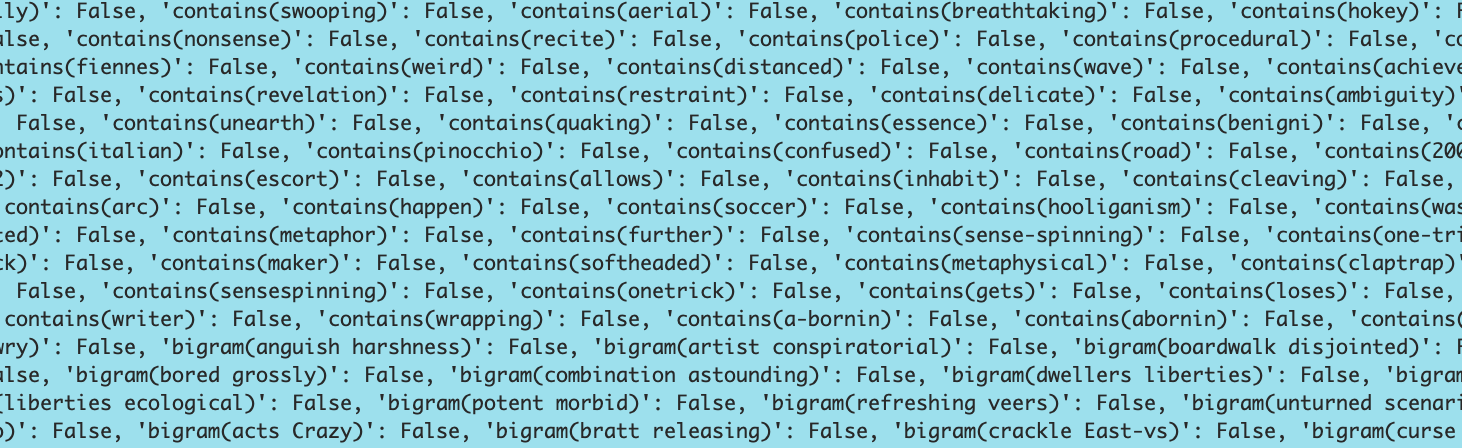


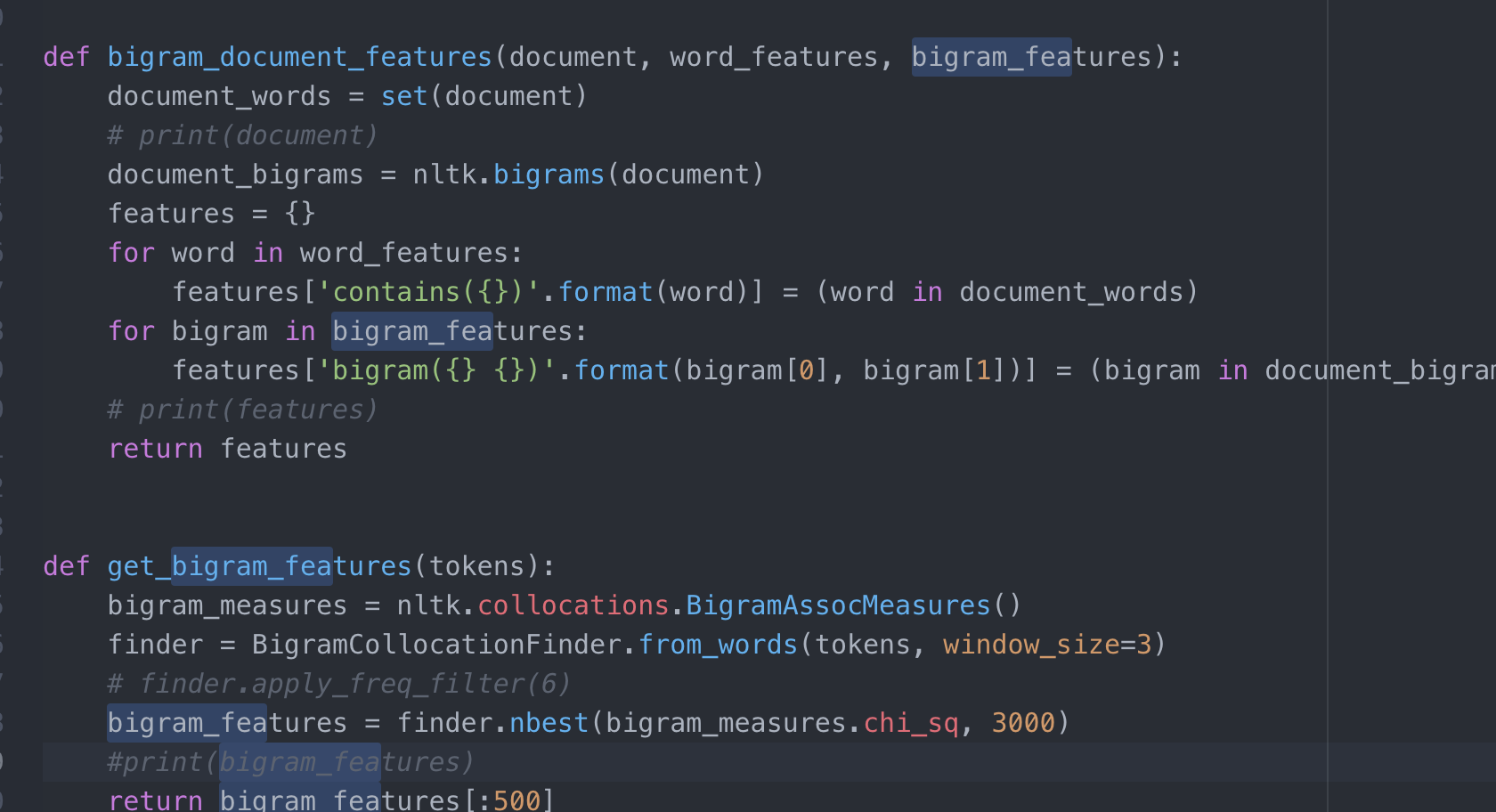
**4.4 Using Bigram features along with unigram features**

I have worked on generating bigram feature from documents. To get high frequent bigrams, I have filter our special characters as well as filter by frequency. I have used the nbest function which just returns the highest scoring bigrams, using the number specified in both the measures.

Example of Bigram Featuresets:

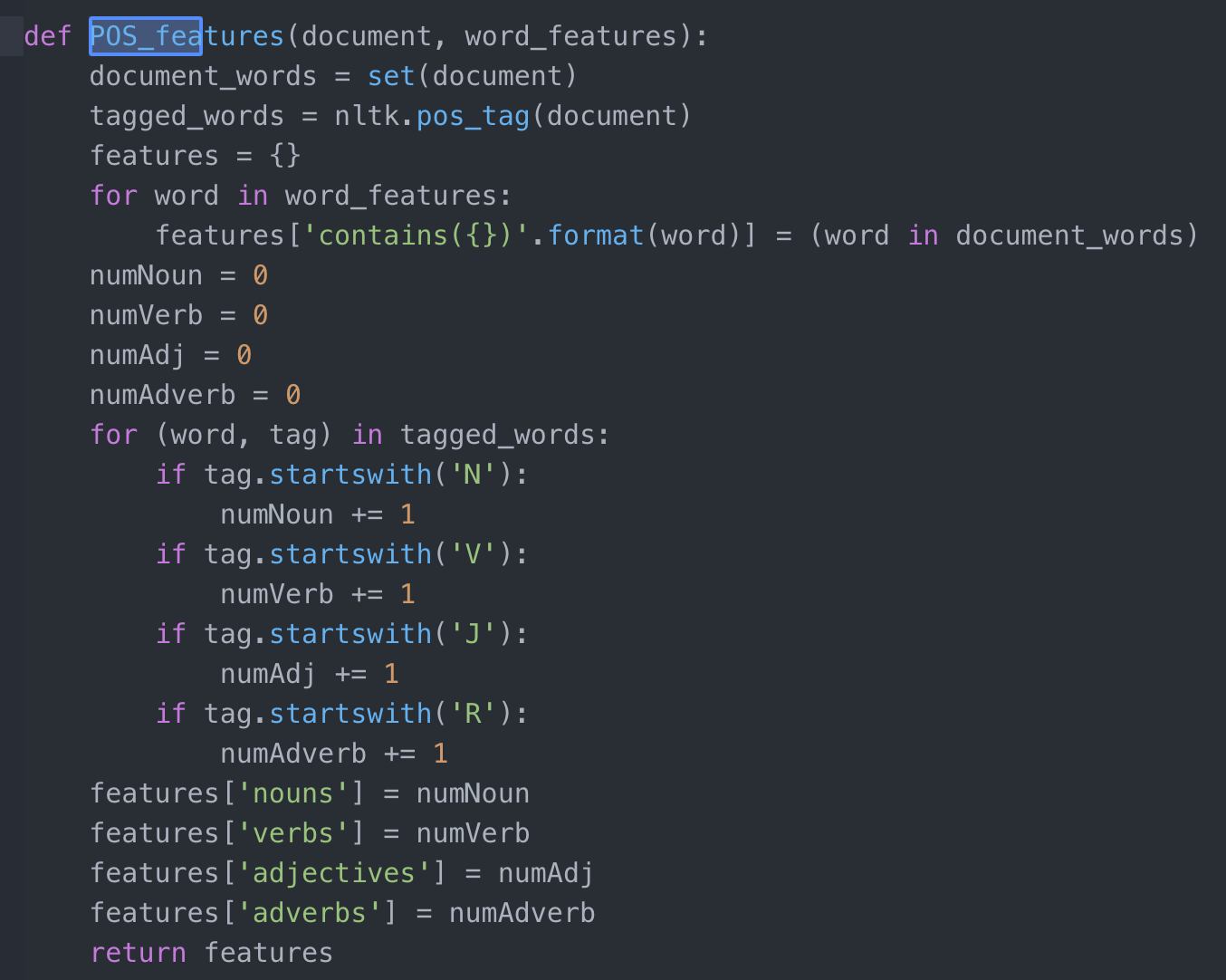






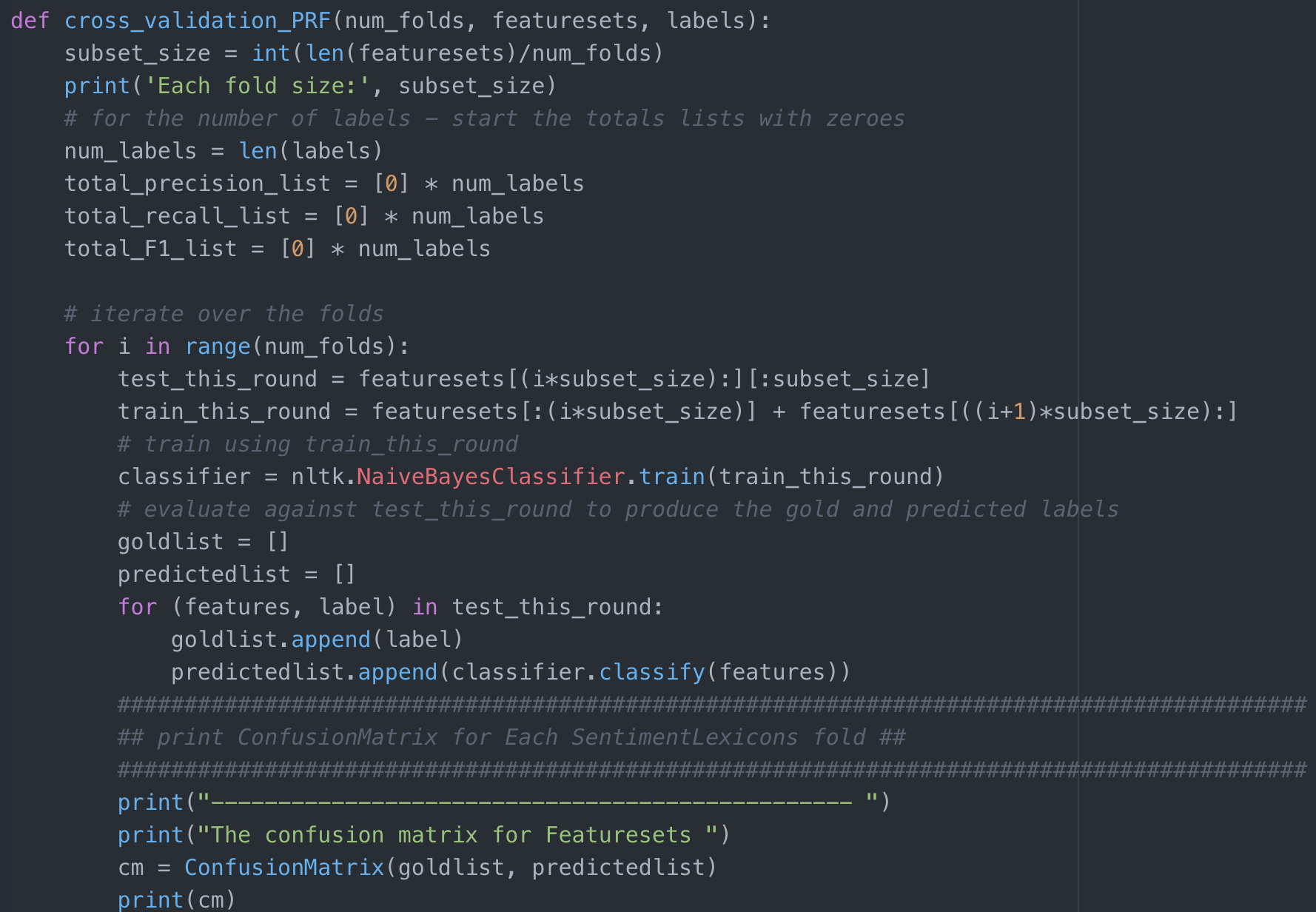
**4.5 POS tag features**

For this classification task used part-of-speech tag features. This is more likely for shorter units of classification; such as sentence level classification or shorter social media such as tweets. In this dataset, we have large training dataset and moreover, in the NLTK, this is difficult to demonstrate, since on computer, it takes the default NLTK POS tagger too much time. Because of this limitation I tested on only 2000 training sentences. The most common way to use POS tagging information is to include counts of various types of word tags. Here is an example feature function that counts nouns, verbs, adjectives and adverbs for features.



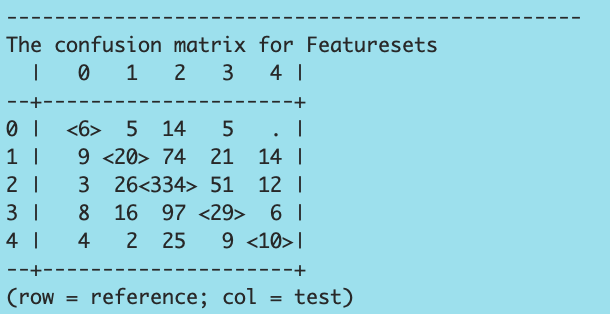
**5. NLTK Naive Bayes classifier:**

NLTK Naïve Bayes classifier was used to train and test data. Initially taken 90 % of data as training set and 10% as test set. Printed also confusion matrix with it. We start by looking at the confusion matrix, which shows the results of a test for how many of the actual class labels (the gold standard labels) match with the predicted labels.

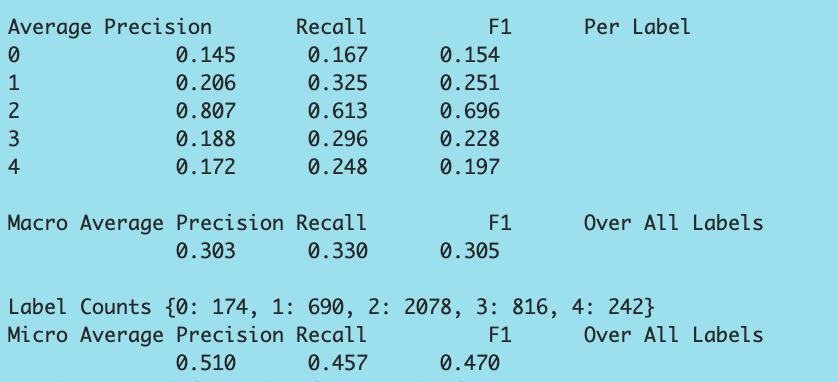


**Sample Output, with confusion matrix ,Average Precision, Recall and F1 Measures**

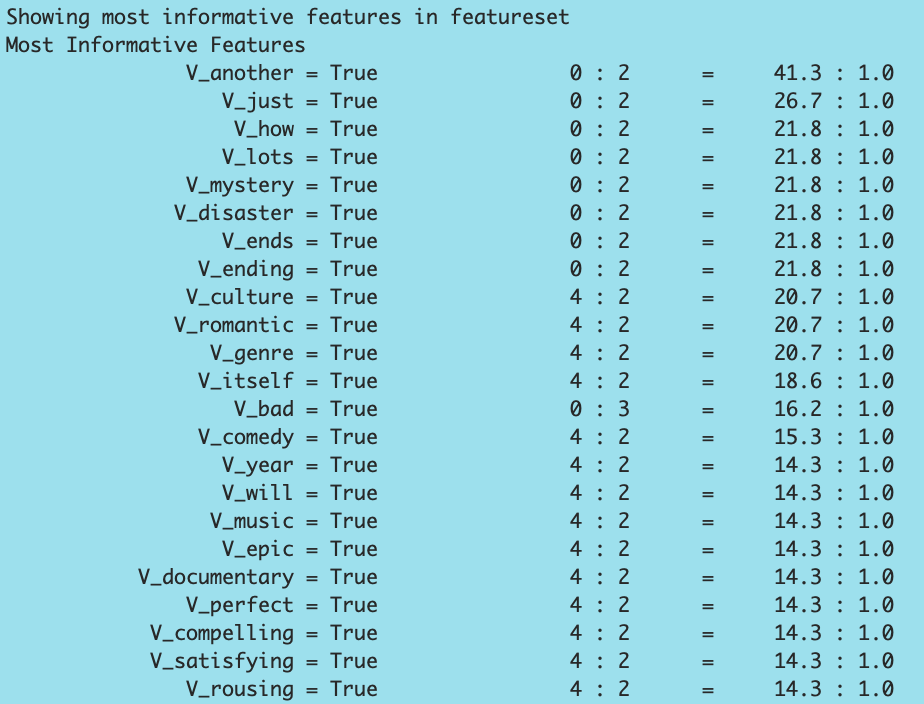
**Confusion Matrix for Featuresets:**



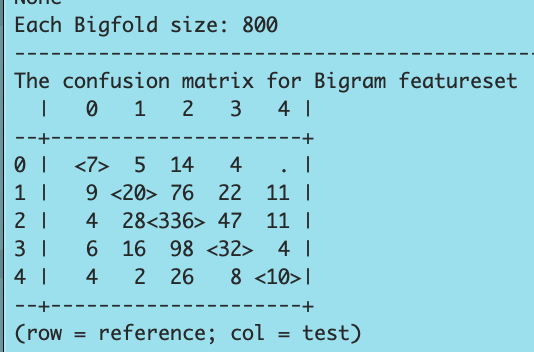
Average Precision ,Recall and F1 Measures for featuresets:

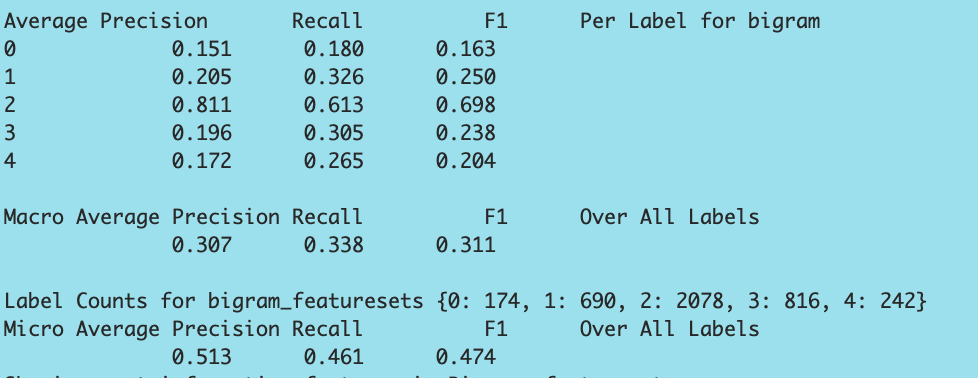


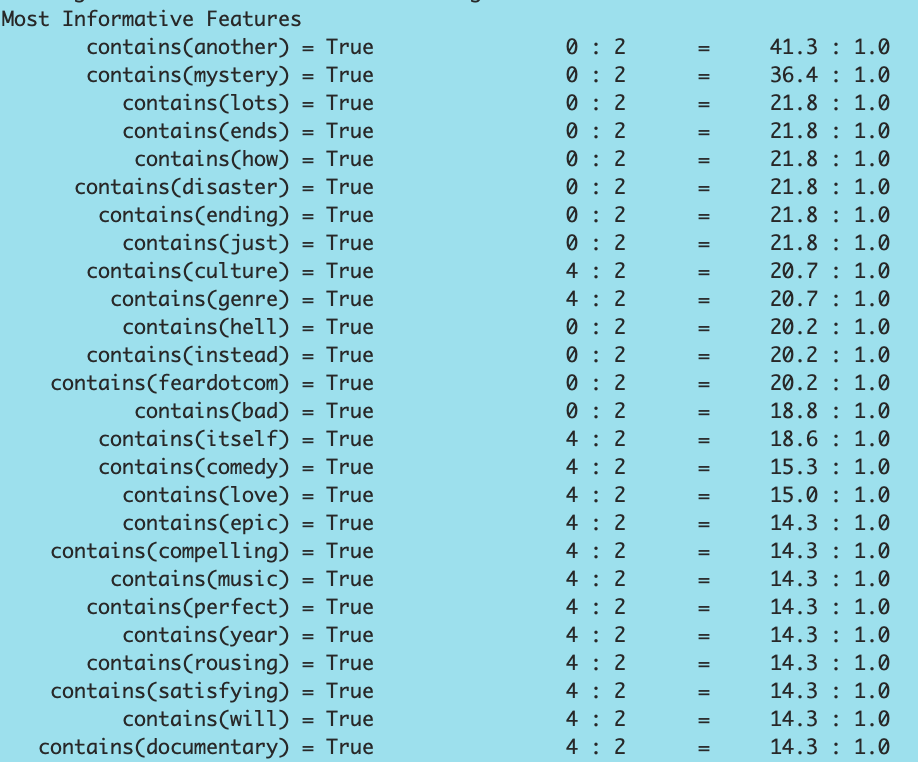
Displayed top 30 most informative features by using **“show\_most\_informative\_features()”**



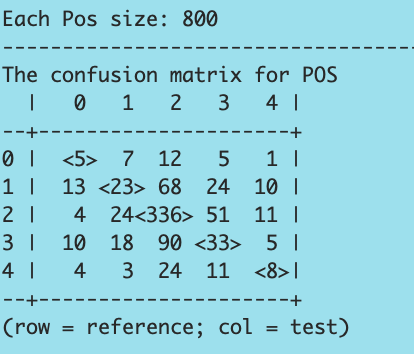
**Confusion Matrix for Bigram Featuresets:**

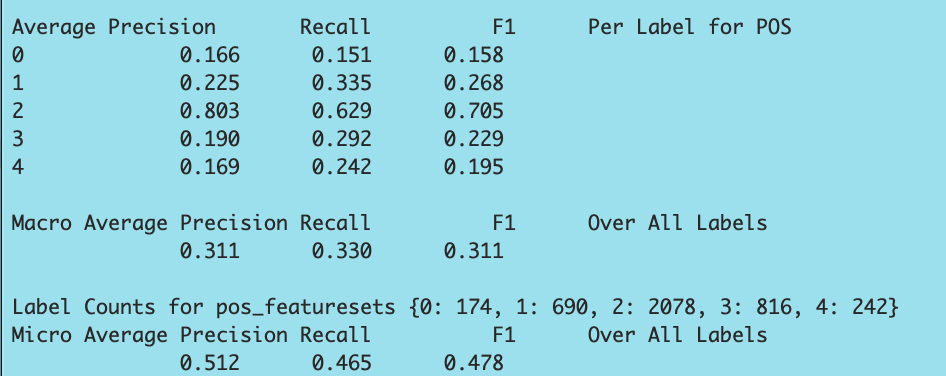
****

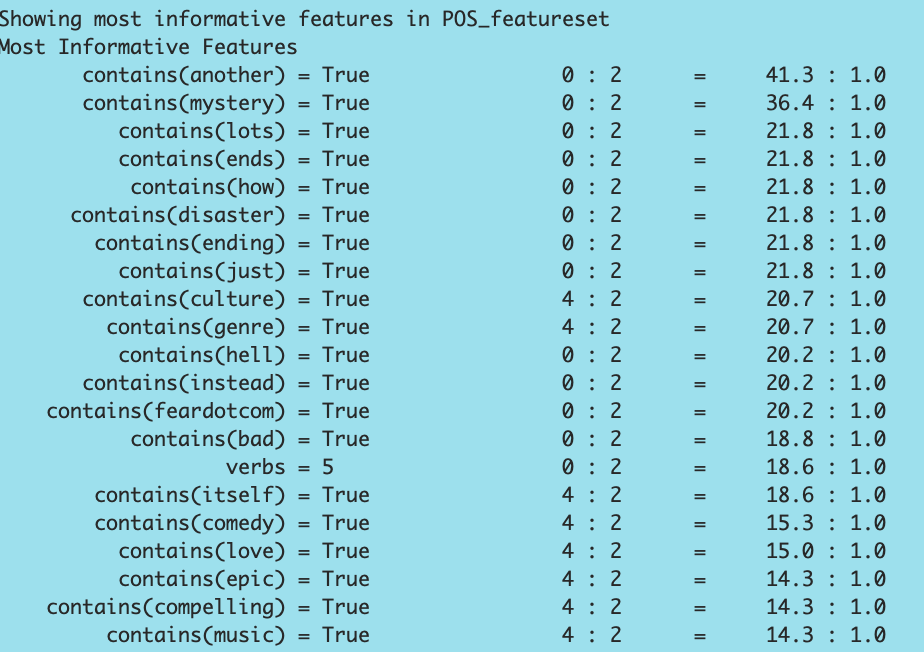
****

****

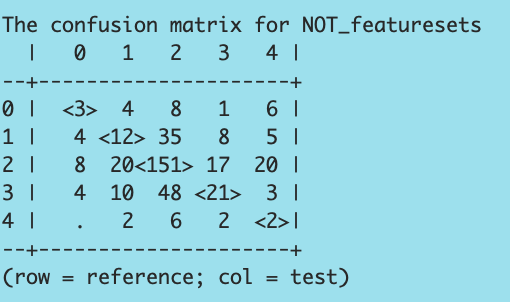
**Confusion Matrix for Part of speech Featuresets:**

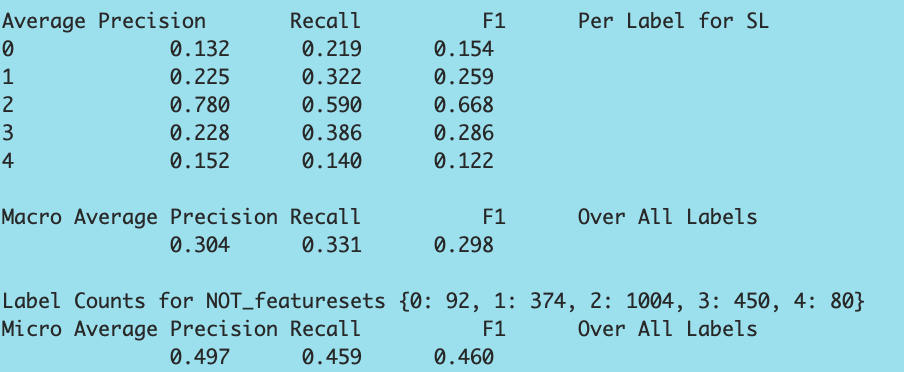
****

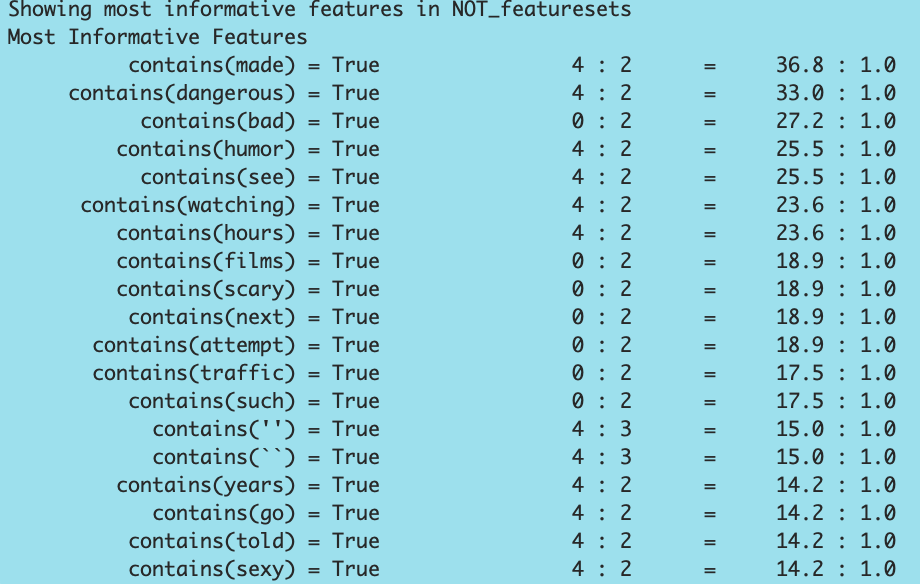
****

****

**Confusion Matrix for Negation Featuresets**

****

****

****

**8. Results:**

. Comparison on different sizes of Corpuses.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Corpus | page15image59314880page15image59315264  Default Featuresets | POS  Features | page15image59316416  SL Features | page15image59317184  Not Features | page15image40583728  Bigram Features |
| 300 | page15image59320256page15image593206400.518page15image59321600 | 0.529 | page15image59322176  0.530  page15image59322752 | page15image593231360.516page15image59324096 | page15image593248640.518page15image59325440 |
| 500 | 0.482 | 0.485 | 0.503 | 0.481 | 0.485 |
| 1000 | page15image59295616page15image592960000.510 page15image59321216 | 0.512 | 0.522 | page15image592984960.497 | page15image593002240.513 |
| 2000 | 0.517 | 0.517 | 0.529 | 0.515 | 0.517 |

Observation:

Increase in size of vocabularies show increase in precision and once the corpus reaches 2000 words almost all the featuresets showed the same precision except Sentiment Lexicons.

3. cross- validation, Precision, Recall and F- measures score in all three classifier

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of records | page15image59314880page15image59315264Default Featuresets | POSFeatures | page15image59316416SL Features | page15image59317184Not Features | page15image40583728Bigram Features |
| Precision score | page15image59320256page15image59320640  0.510 page15image59321216page15image59321600 | 0.512 | page15image59322176  0.522  page15image59322752 | page15image59323136  0.497  page15image59324096 | page15image59324864  0.513 page15image59325440 |
| Recall score | 0.457 | 0.465 | 0.478 | 0.459 | 0.461 |
| F-measure score | page15image59295616page15image59296000  0.470 page15image59296576page15image59296960 | 0.478 | 0.489 | page15image59298496  0.460 page15image59299456 | page15image59300224  0.474 page15image40590672 |

SL features is performance better in classification other features functions because of few words are unseen in train data as features. Those words or tokens covers on Lexicon dictionary. Plus, we observed recall score lesser than F-measure which is less than Precision.