Text

Description automatically generated

**Course:**2022-0706 IST 664 Natural Language Processing

**Name:** Jacob N. Conard

**Assignment:** Final Project

**Date due:** 15 September 2022

**Date submitted:**12 September 2022

**Number of Pages:** 13

**Final Project: NLP Final Project**

**Background:**

The final project will be a classification task, where you will develop features for the task and demonstrate that you can carry out experiments that show which sets of features are the best for that data.

**Data Description:**

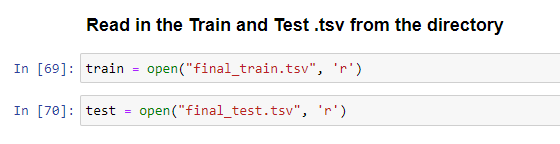
This dataset was produced for the Kaggle competition, described [here](https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews), and which uses data from the sentiment analysis by Socher et al, detailed at this web [site](http://nlp.stanford.edu/sentiment). The data was taken from the original Pang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

Although the actual Kaggle competition is over, the data is still available [here](https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data). We are going to use the training data “train.tsv”, and some test data is also available “test.tsv”. There appear to be 156,060 phrases in the training data file, and one of the challenges will be to choose an appropriate subset for processing and training.

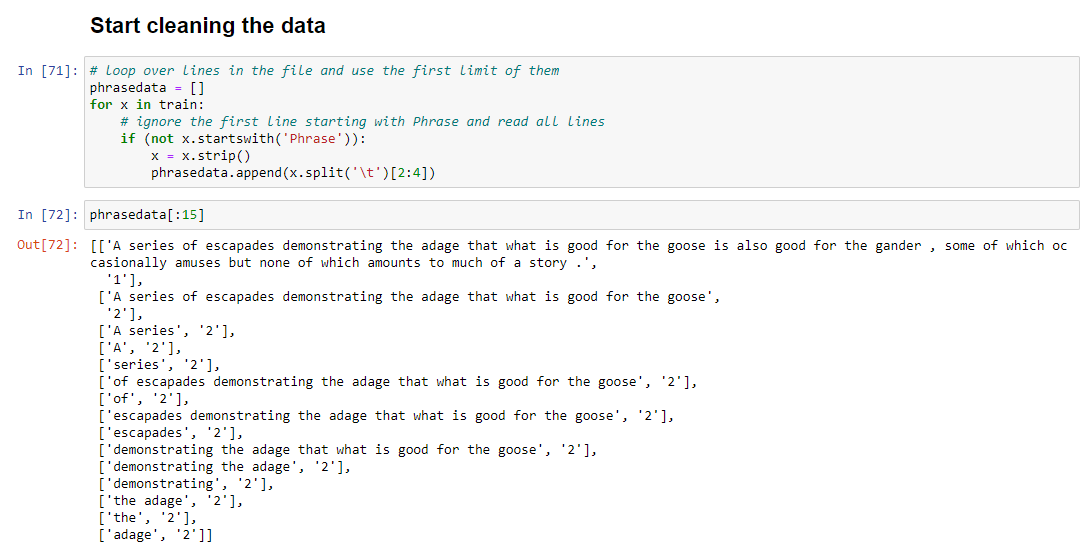
**Questions:**

***Step:1 Data Cleaning and Data Preparation***

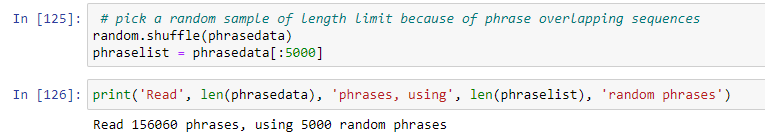
For the final project, I have two different corpora one is a training dataset related to the Rotten Tomatoes reviews and the other corpus is a testing dataset that is also related to the Rotten Tomatoes reviews. The first step that needs to be accomplished is to read the datasets in to memory for processing by the Jupyter Notebook.



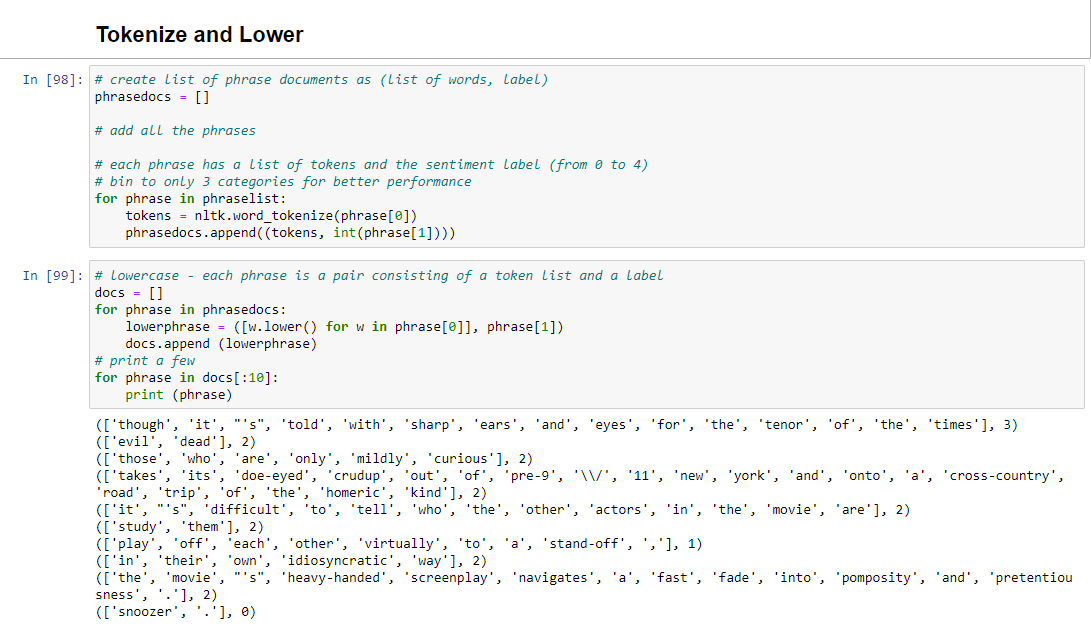
The data is tab delineated and needs some more cleaning to only process the data that we need and make the processing faster. The label data starts with the word “phrase” so we need to skip over that, and the skip the first couple columns which are the phrase and the sentence identifiers which are not needed for processing.



The next step is to take a random sample of a defined number because of phrase overlapping sentences. For this example, we selected 5,000.

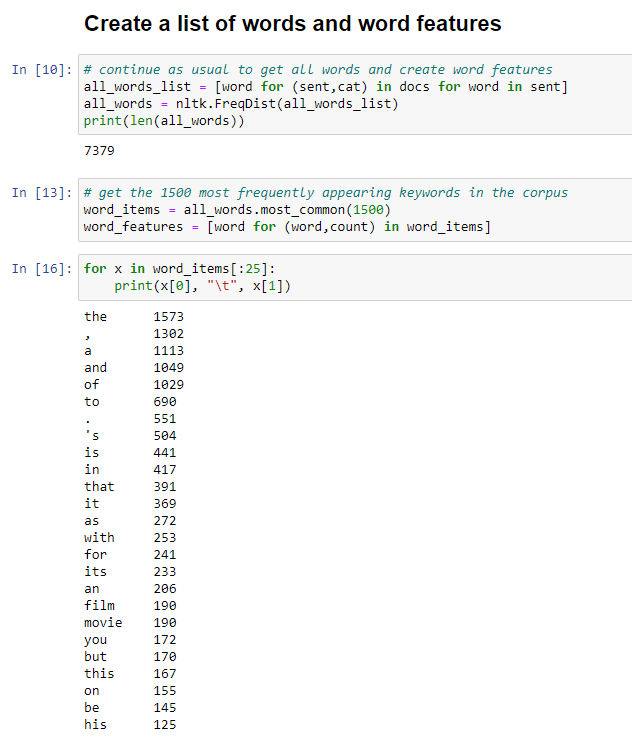


The next data cleaning step that we need to accomplish is to tokenize the sentences and convert all of the text to lowercase. This is accomplished with just a few lines of code.

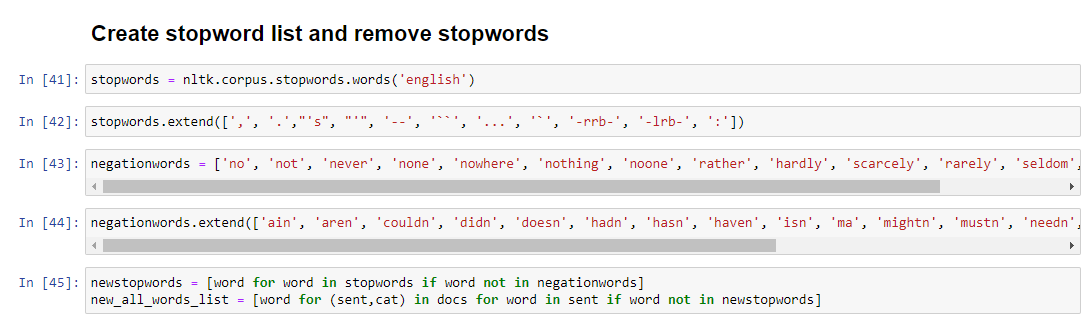


***Step:2 Feature Functions and NLTK Naïve Bayes***

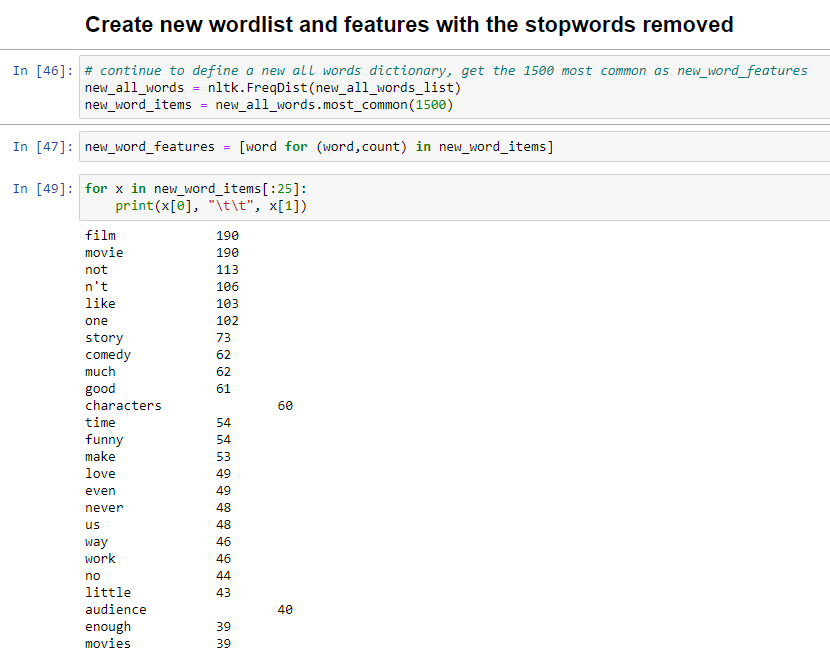
Now that the training corpus is “clean” the next step is to produce the features in the notation of the NLTK. For this we need to write feature functions in Python as we have been doing in the lab. We need to start with the “bag-of-words” features where we collect all the words in the corpus and select some number of most frequent words to be the word features. For this example, we are going to use 1,500 of the most frequent words in the training corpus. We will print out the first 25 most frequent words from the “bag-of-words” features to get an idea of processing that may need to be completed.



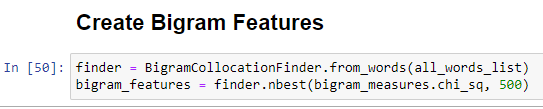
It looks like we will need to do some stop word processing in this corpus to get rid of some of the stop words, but you can already see towards the end of the list the words “file” and “movie” are fairly common in this corpus. That is what we would expect to see considering that this corpus is related to Rotten Tomatoes reviews. The next step in the processing is to create the stop-word list and remove those stop-words from the corpus.



We needed to extend the stop-words list to include some odd examples that were present in the Rotten Tomatoes corpus. We also created a list of negation words and extended that list. We are going to use this negation list later on to conduct experiments and test out what Naïve Bayes model works the best. Now we will check out the “bag-of-words” with the stop-words removed.



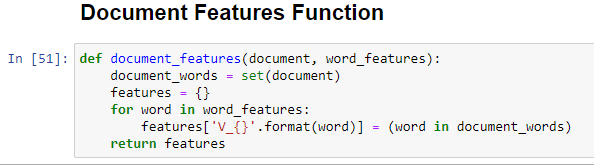
The next step is to create the bigram features.



The next thing that we need to do is create a list of functions to conduct the experiments in Step 3. These functions will be for Negation Features, Part-of-Speech Features, Original Document Features, and Bigram Document Features. We will provide screenshots of the code snippets for the functions.

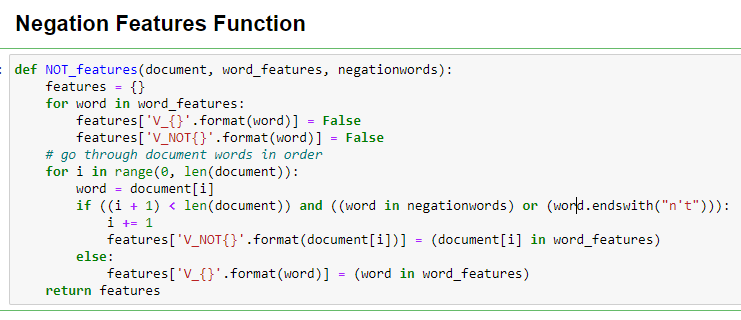
***Document Features Function***

This function is going to take two parameters to include the document and the word features that we created previously. It is going to iterate through the all the words and make a determination if the words are present in the word features list (top 1,500).



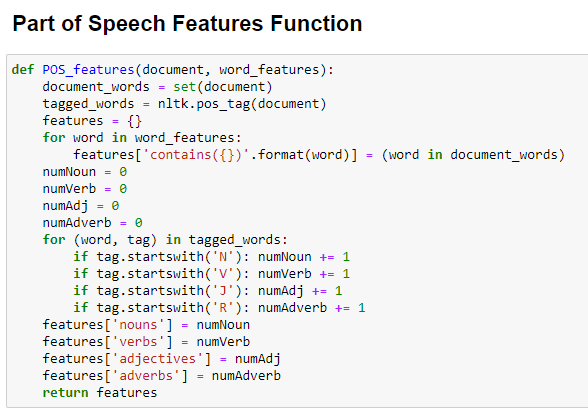
***Negation Features Function***

This function is very similar to the Document Features Function mentioned above; however, it is also going to include the negation words.



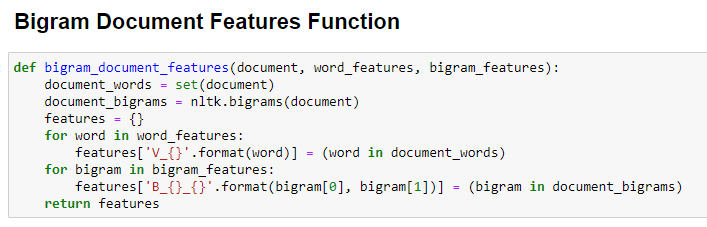
***Part-of-Speech Features Function***

This function is very similar to the Document Features Function mentioned above; however, it is also going to do POS tagging for the words as well.



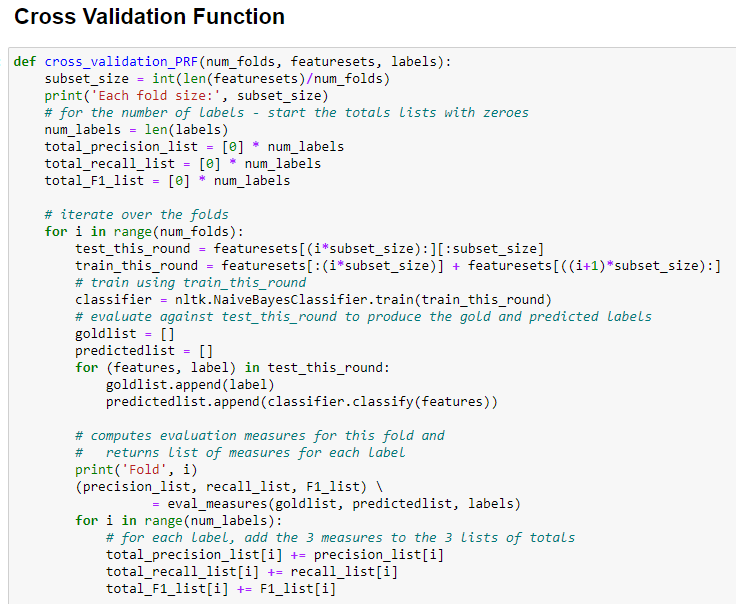
***Bigram Features Function***

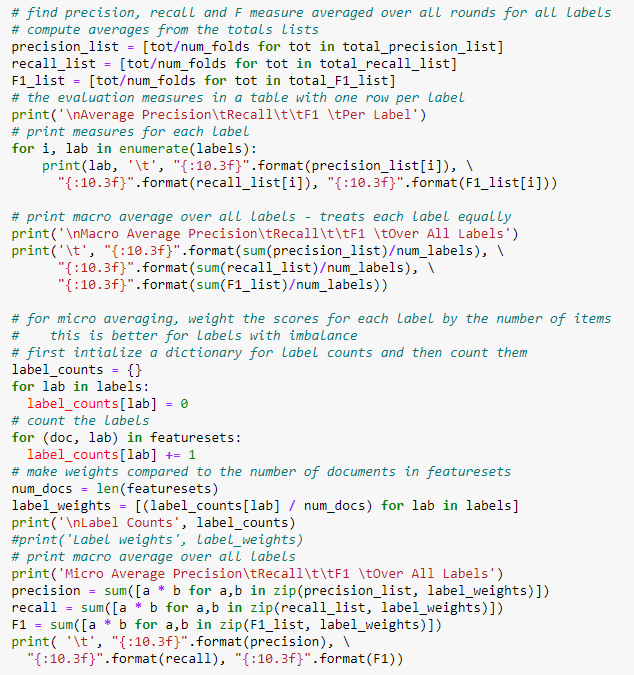
This function is very similar to the Document Features Function mentioned above; however, it is also going to also utilize the bigram features list that we created above.



***Cross Validation Function***

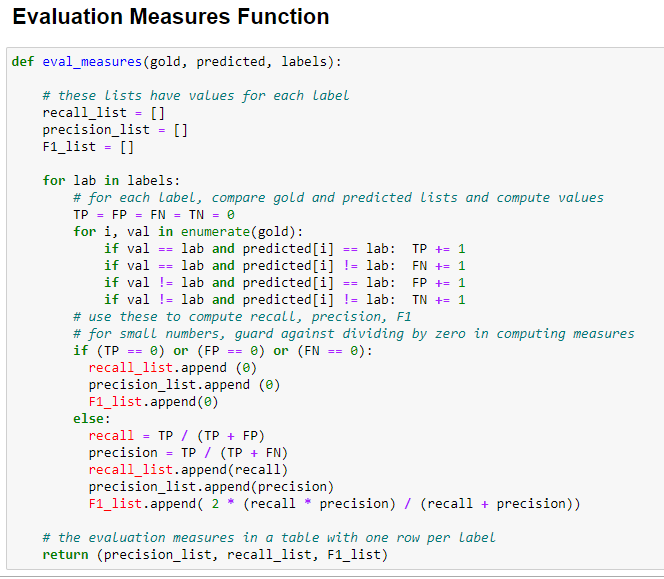
This function is going to create the NLTK Naïve Bayes classifiers to be utilized to test the performance of the different models in Step 3 during the experiments. We are going to leverage this function to test how well each of the different document features performs, to include, Bigram, Negation, POS, and the Original feature set. The function sets the number of folds and utilizes a specific feature set and labels. The function will also be utilized to calculate and display the precision metrics, recall metrics, and F1 scores of each of the classifiers.





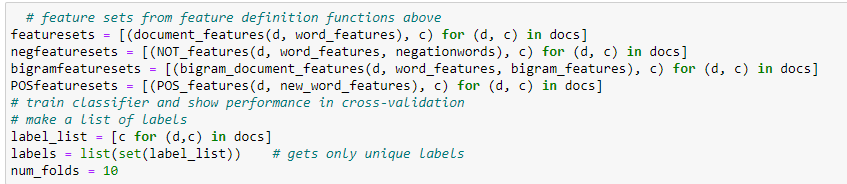
***Evaluation Measures Function***

This function is going to allow for the creation and presentation of the Naïve Bayes classifier metrics to include the recall metrics, precision metrics, and F1 scores for each model.

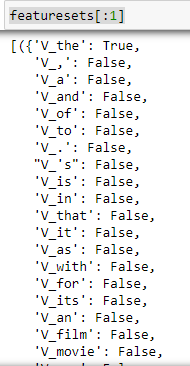


***Step:3 Experiments***

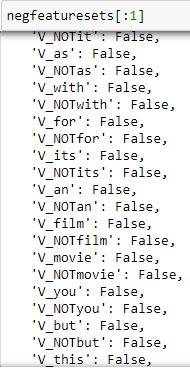
In this portion we are going to conduct experiments where we use different sets of features and compare the results of each of the different Naïve Bayes classification models to determine which one performs the best. We are going to then compare the different features we designed to determine which model performs the best in the classification tasks. We will do this with Negation, POS, Bigrams, and the Original feature set.



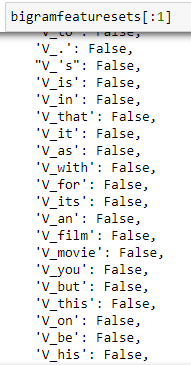
***Original Feature Set***



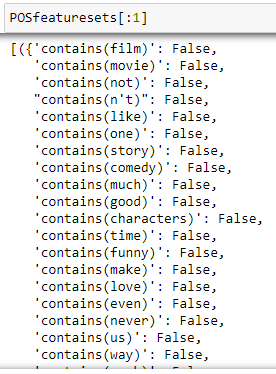
***Negation Feature Set***



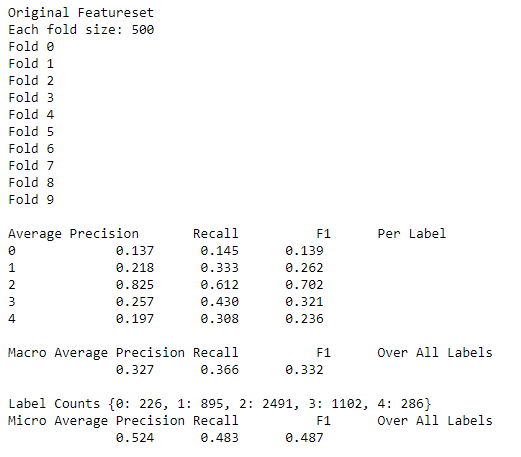
***Bigram Feature Set***



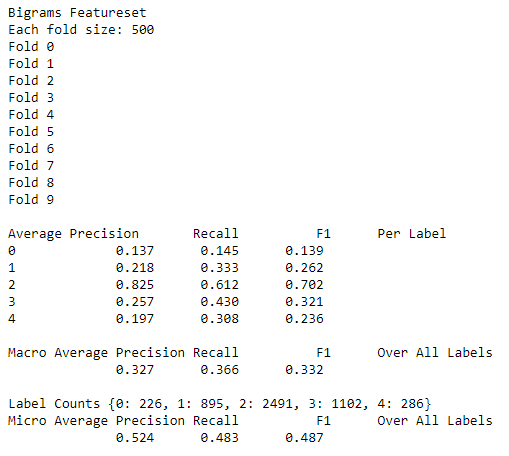
***Part-of-Speech Feature Set***



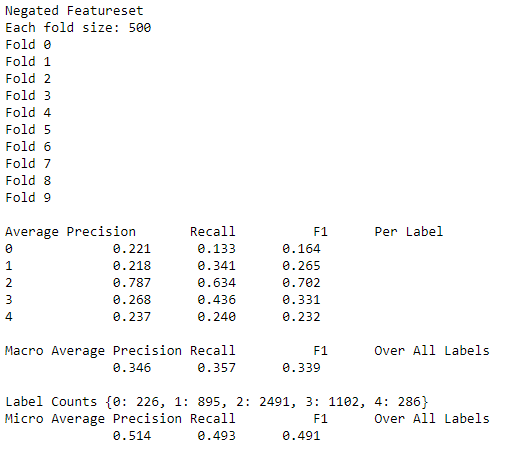
***Cross Validation Results for Original Feature Set***



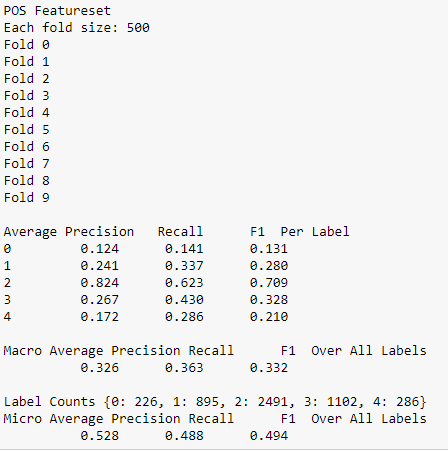
***Cross Validation Results for Bigram Feature Set***



***Cross Validation Results for Negation Feature Set***



***Cross Validation Results for Part-of-Speech Feature Set***



***Step:4 Interpretation of Results***

The final step of the assignment is to take the analysis of the four Naïve Bayes classification models and interpret the results. The first step in doing an interpretation of the results is to establish a baseline of the precision, recall, and F1 measurements and what they actually are.

***Precision Scores***

The model precision score measures the proportion of positively predicted labels that are actually correct. Precision is also known as the positive predictive value. Precision is used in conjunction with the recall to trade-off false positives and false negatives. Precision is affected by the class distribution. If there are more samples in the minority class, then precision will be lower. Precision can be thought of as a measure of exactness or quality. (<https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/>)

***Recall Scores***

Model recall score represents the model’s ability to correctly predict the positives out of actual positives. This is unlike precision which measures how many predictions made by models are actually positive out of all positive predictions made. For example: If your machine learning model is trying to identify positive reviews, the recall score would be what percent of those positive reviews did your machine learning model correctly predict as a positive. (<https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/>)

***F1 Scores***

Model F1 score represents the model score as a function of precision and recall score. F-score is a machine learning model performance metric that gives equal weight to both the Precision and Recall for measuring its performance in terms of accuracy, making it an alternative to Accuracy metrics (it doesn’t require us to know the total number of observations). It’s often used as a single value that provides high-level information about the model’s output quality. (<https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/>)

***Interpretation of Results***

Since the F1 score is often used as a single value that provides high-level information about the model’s output quality this is the score that we will utilize to determine the model that performed the best during our experiments. The F1 score is actually a score of the precision and the recall scores, so this metric will be a great metric to test the performance of the models. We will start with the performance of the Original Feature Set, this model had a precision average of 52%, recall of 48%, and an overall F1 score of 48%. The Bigram Feature Set had a precision average of 52%, recall of 48%, and an overall F1 score of 48%. The Negation Feature Set had a precision of 51%, recall of 49%, and an overall F1 score of 49%. The Part-of-Speech Feature Set had a precision of 52.8%, recall of 48.8%, and an overall F1 score of 49.4%. The best preforming model was the Part-of-Speech Feature Set.