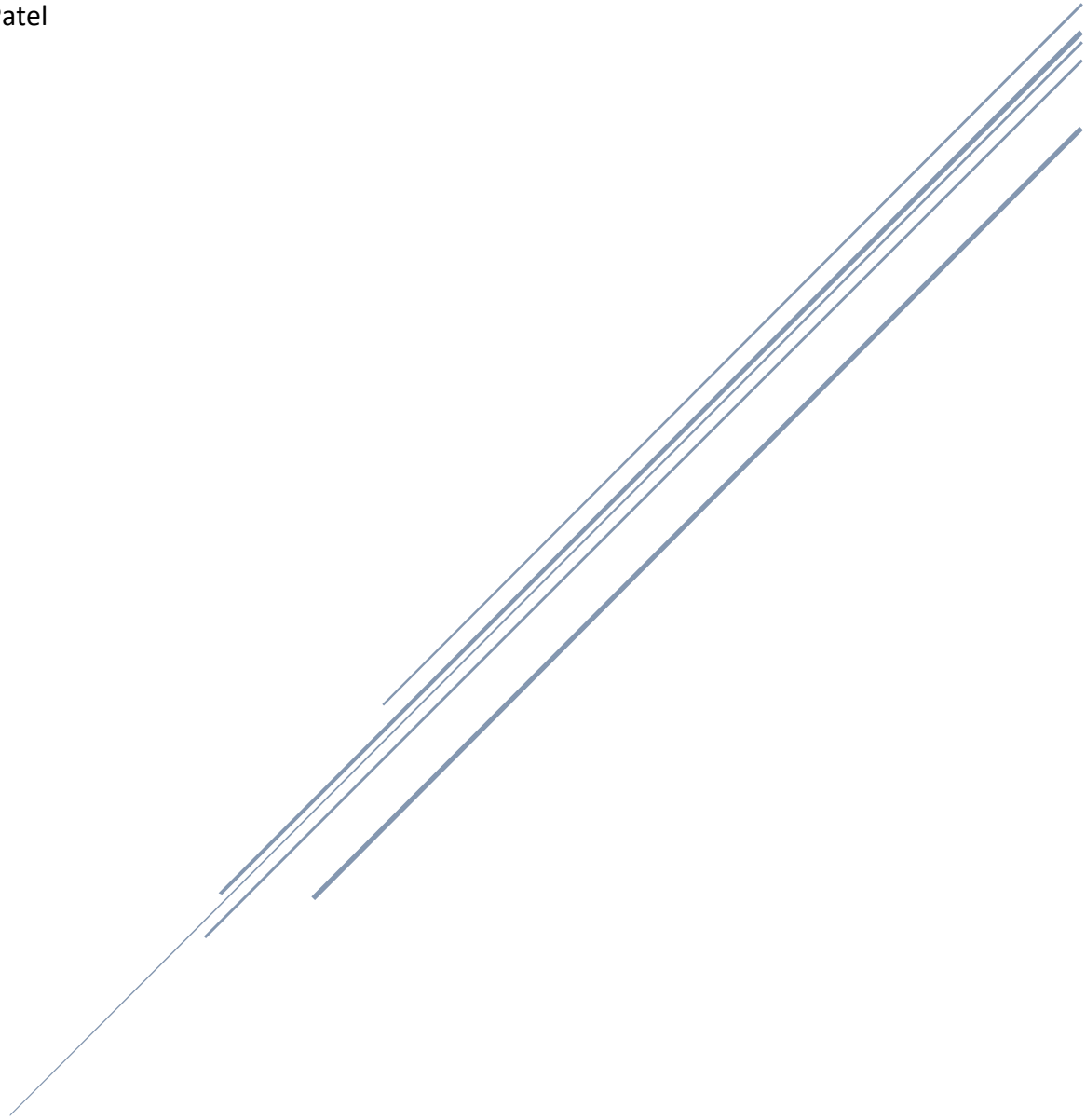
SENTIMENT ANALYSIS OF KAGGLE MOVIE REVIEW

Final Project: Natural Language Processing

By: Parin Patel



Introduction:

Project Overview:

The goal of the project is to complete classification tasks, where we develop features for the task and demonstrate that we can carry out experiments that show which sets of features are the best for that data.

We have chosen to work with the Kaggle competition move review phrase data, which has been labeled for sentiment. This dataset was produced for the Kaggle competition (https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews). This data uses sentiment analysis provided by Socher et al, detailed at this web site: http://nlp.stanford.edu/sentiment/.

Additionally, the data was taken from the original Pang and Lee movie review corpus, which is based on reviews from the Rotten Tomatoes website. Socher's group used crowdsourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: "negative", "somewhat negative", "neutral", "somewhat positive", "positive".

The following table shows the numbers associated with the sentiment labels in the dataset:

0	Negative
1	Somewhat Negative
2	Neutral
3	Somewhat positive
4	Positive

Data Review:

We used training data "train.tsv" and testing "test.tsv" datasets for our experiments. The train.tsv dataset contained the reviews with their related sentiment labels. In addition, each phrase is given a SentenceId that corresponds to each sentence. This allows us to track which sentences belong to which reviews. The test.tsv dataset contains only the reviews and their associated SentenceId. We will seek to assign a sentiment label to each of these reviews.

Below, we will first review our text classification tasks. This includes the steps taken to process the text, tokenize it, and if we chose to complete some pre-processing or additional filtering steps. We will then, for Phase Two, produced "bag of word" features in the notation of the NLTK. This function will be written in Python. For Phase Three, we will complete several experiments where we will use two different sets of features and compare their results. We have chosen to first use the Unigram feature as a baseline and see if we can improve accuracy after pre-processing. Our second experiment will be a POS tag features. After, we will use sentiment lexicon and count the subjectivity features (positive count and negative count). Our fourth experiment will be representing negation of opinions. Finally, our fifth experiment will use Bigram features in addition to Unigram features. For Phase Four of our project, we will use the NLTK Naïve Bayes Classifier to train and test a classifier on each of our feature sets. We will analyze the results by using cross-validation to compare their precision, recall, and F-measure

scores. Additionally, for our advanced task, we trained the classifier on the whole training set and tested it on a separate test set. We chose to use Weka and Sklearn to accomplish this task.

We have chosen to remove all mentions of python code from our report. For specific technical questions, please see the appendix or attached python code.

Phase One: Data Import and Cleaning

First, we start by importing our training dataset. After ignoring the first line, we start looping through the rows of the data and passing the first limit of each as an argument a method called processkaggle. This method is starts with the first review. Each line has four columns separated by tab. We chose to only keep phrase and sentiment data in the training set and remove their ID's because we have no need to link the sentences back to the phrases they came from.

Next, we decided to start cleaning and pre-processing our data. The list below describes the steps we took when cleaning our dataset:

1. **Lower Case:** First, we decided to convert all the text to lower-case by removing the upper case and lower-case sensitivity. This was done using the re.split function and all words were placed in the list called "word_list"
2. **Remove Punctuation Symbols and/or Numbers:** We chose to then use the "re.compile" function to remove any symbols or numbers in the corpus. For our case, it is important to remove them because they will cause noise in our dataset and cause us to likely lose valuable insights from our later results and analysis.
3. **Remove Stop Words:** We chose to remove words that do not help us find context or meaning in our sentence. To do this, we removed stop words using the NLTK English stop words list, in addition to the more-specific list we created for our projects purpose. This list includes negationwords and partial forms of a word that we found. For example, we also chose to remove words and parts of words like "Can","not" ,"t" , and "ll".
4. **Tokenization:** We decided to two types of tokens lists for this project to help us with the feature functions and experiments conducted below. The first list of tokens contains only words from the training data that are longer than three characters. The second one requires the minimum and considers all tokens regardless of character length. We call this "get_words_from_phasedocs_normal" list.
    a. This first tokens list requires more pre-processing steps than the second. We actually created this list later, and then re-ran our code, to see if it could help with increasing the accuracy of our experiments by reducing some of the noise. Our other option was to remove more stop words from our data, however, we tested it out, and created a more pre-processed tokens list was the better outcome.

Phase Two: Feature Definition:

We then had to write a feature function in Python. We chose to write a "bag-of-words" feature to collect any words in the corpus. We chose to then collect the 300 most frequently occurring words. We have listed some of the most frequently occurring ones below.

A Few of the Most Frequently Occurring Words:

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

Phase Three: Feature Experiments:

In the results section later, we will use Weka and SkLearn to compare how pre-processing our feature sets effects the accuracy of our classifier. In addition, we will compare the feature sets for word count and size of their vocabulary. Finally, we will run a cross-validation on the sets to obtain a precision score, recall score, and f-measure. The last part of our experiment involves running a logistic regression classifier on all feature sets with 10,000 records

3.1 Unigram Feature Experiment:

For our first feature experiment, we chose to run an experiment to define a unigram feature for each document. We wanted to filter by pre-processing method. In the above, cleaning section, we discussed our two methods of tokenization. Additionally, for this feature experiment, our feature label is "contains(*keyword*)". And for each *keyword* (word in the unigram feature) the value of feature's value has been set to Boolean based on if the word is found within the documents. In the figure below we can see the output difference from when we pre-process the Unigram feature vs when we do not.

Output Prior to Pre-Processing Unigram Feature:

({'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)

Output After Pre-Processing Unigram Feature:

({'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)

3.2 POS Tag Feature Experiment:

For our second experiment, we chose to conduct a Part-of-Speech (POS) tag feature. We initially tried to use a. POS tag function within the NLTK library. However, we ran into some major limitations. We found that this method was very CPU-extensive and took a long duration to runover our dataset. Therefore, we completed this feature task by defining a POS tag feature to help with our classification. This is method is very useful when reviewing short phrase data like tweets or reviews with character limits. Our feature is likely to have some problems when parsing over datasets with longer strings, like analysis and classification of long poems or novels. But for our purpose, to minimize overloading the CPU, we chose to take a subset of our data (2000 training sentences) and run the counts of the different determined tags. We made sure our feature was able to count the part of speech tags. Like nouns, verbs, adjectives, and adverbs. To see this code, please review the attached python file. We will discuss the results in a later part of this report.

3.3 Sentiment Lexicon & Count of Subjectivity Features

For our third experiment, we chose to read the subjectivity words from the given subjectivity lexicon file. This module was created by the Wiebe et al, as part of the Multiple Perspective QA project, at the University of Pittsburg. The file has the format that each line is formatted to either be feature themselves or in combination with another set of data. We used this file to create two features that count if the subjectivity of a word in a document is positive or negative. This code was not created by us, as mentioned above, therefore, we copy and pasted the function for readSubjectivity from the sentiment_read_subjectivity.py file. The output of this file creates a Subjectivity Lexicon in the form of a mapped dictionary where the words are mapped to a list that states their resulting strength and if they are positive or negative. From sentiment_read_subjectivity.py file chose to only use the function that defines readSubjectivity.

For our feature extraction function, we chose make sure it includes all the word features, in addition to “positivecount” and “negativecount”. These two features would count either the positive or negative subjectivity of the words in the document. The feature makes sure that the subjectivity is counted based on the word’s polarity, or level of subjectivity. Therefore, a weekly subjective word is counted once while a strongly subjective (like a strong positive or negative word) is counted double. We will discuss the results in a later part of this report.

3.4 Negation of Opinions Feature

Our fourth feature defines negation words. This is an important part of the sentiment classification task because the whole computational technique is for categorizing the sentiment of the text, based on if it is positive or negative. By defining and handling negation modifiers or words, like “not” , “never”, and “no” we are able to better understand the sentiment conveyed by the associated words. We also chose to include negation words that appeared in our phrases as contractions “don’t” or “doesn’t” . We found this by reviewing our documents, we have listed below some of the words found in the first document

Words from First Document – Example:

if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself

We chose to negate the word following the negation word instead of negating all the words between the punctuation mark and the end of the negation word. We then made sure to parse the document to account for adding the word features. However, when we found word after the negation word, we changed the feature to the negated word.Below we have listed an example of the feature set output. We will discuss the results in a later part of this report. To see this code, please review the attached python file

Feature Set Output- Example:

({'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)

3.5 New Feature Function - Combining Unigrams and Bigrams.

For our final feature function, we chose to run a unique feature. We decided to combine using a bigram feature alongside using a unigram feature. To get the most frequent bigrams, we first chose to remove any special characters. They then filtered by frequency, with the most frequently occurring bigrams listed at the top. We were able to use the nbest function to return any bigrams that were the highest scoring frequency on the list. We used this number in both measures.

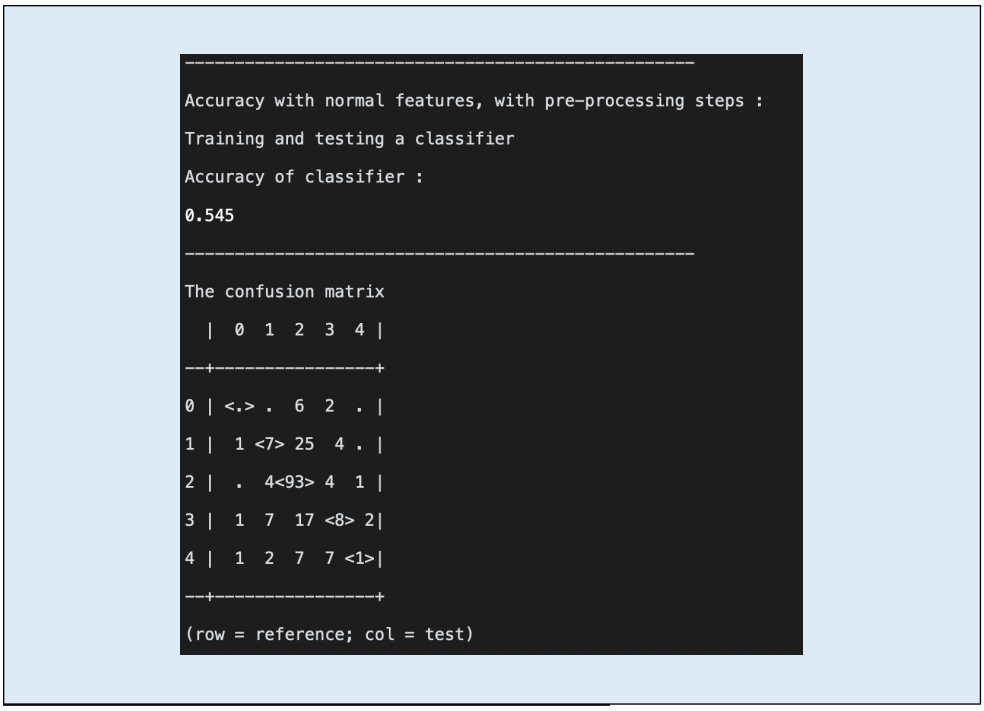
Feature Set Combining Unigrams/Bigram- Example:

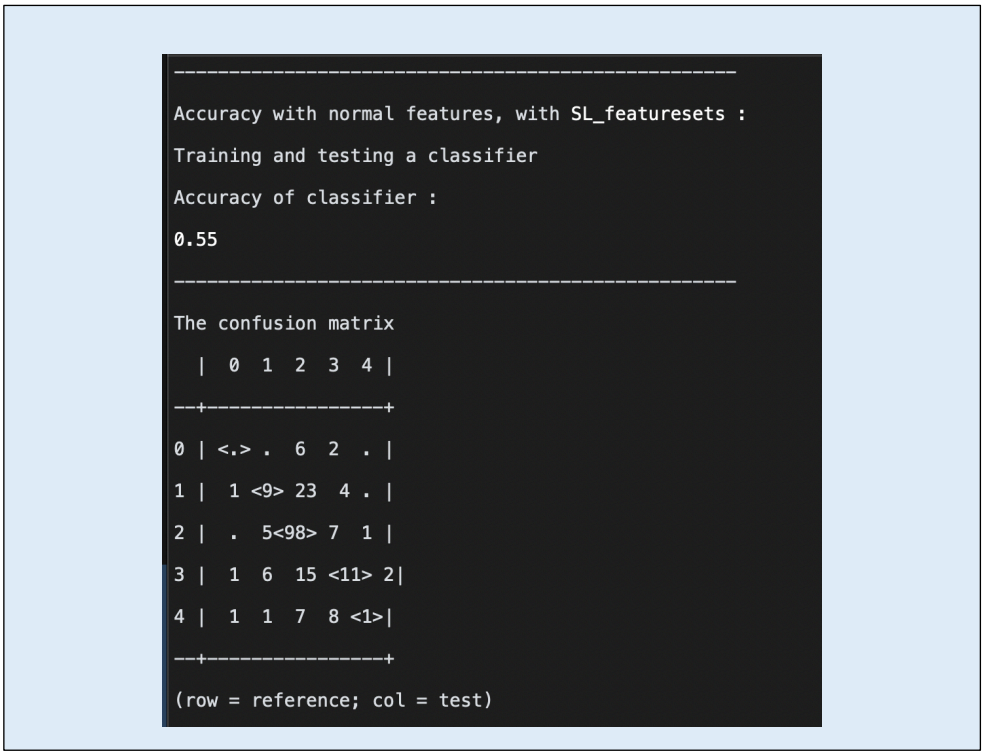
({'bigram(denying hardscrabble)': False, 'bigram(downsizing rock)': False, 'bigram(earn cheesy)': False, 'contains(seems)': False, 'contains(movies)': False, 'contains(lot)': False, 'contains(genuine)': False, 'bigram(concentrate city)': False, 'contains(deeply)': False,
…..
…….
……..
'bigram(carried bits)': False, 'bigram(earnest core)': False, 'bigram(circles crimeland)': False, 'bigram(clothed excess)': False, 'bigram(city ensemble)': False, 'bigram(credits metaphor)': False, 'bigram(ballistic ecks)': False}, 2)

Phase Four: Running NLTK Naïve Bayes Classifier:

In this phase, we use the NLTK Naïve Bayes classifier on the train and test data. We are split our data as 90% training and 10% test. We were able to use cross-validation methods to obtain a precision, recall, and F-measure score, which can be seen later. In addition ,we printed out a confusion matrix, that displayed the results of the test data. The matrix showed how many of the actual labels were able to match the predicted labels. We have listed our confusion matrices below. To see the full output, please reference Appendix A. It's important to note that the feature files labeled "without" below were created using "normal" bag of words (unigram), and therefore, we labeled as such.

```
--------------------------------------------------

Accuracy with normal features, without pre-processing steps :

Training and testing a classifier

Accuracy of classifier :

0.51

--------------------------------------------------

The confusion matrix

   |  0  1  2  3  4 |
---+----------------+
0 | <.> 1  5  2  . |
1 |  3 <8> 22  3  1 |
2 |  2  8<83> 8  1 |
3 |  4  4  17 <9> 1 |
4 |  2  4  5  5 <2>|
---+----------------+
(row = reference; col = test)
```


Accuracy with normal features, with pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.545
The confusion matrix
| 0 1 2 3 4 |
0 | <.> . 6 2 . |
1 | 1 <7> 25 4 . |
2 | . 4<93> 4 1 |
3 | 1 7 17 <8> 2|
4 | 1 2 7 7 <1>|
(row = reference; col = test)


Accuracy with normal features, with SL_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.55
The confusion matrix
| 0 1 2 3 4 |
0 | <.> . 6 2 . |
1 | 1 <9> 23 4 . |
2 | . 5<98> 7 1 |
3 | 1 6 15 <11> 2|
4 | 1 1 7 8 <1>|
(row = reference; col = test)

```
--------------------------------------------------

Accuracy with normal features, with NOT_featuresets :

Training and testing a classifier

Accuracy of classifier :

0.51

--------------------------------------------------

The confusion matrix

  |  0  1  2  3  4 |

--+----------------+

0 | <.> .  5  3  . |

1 |  5 <7> 22  3 . |

2 |  2  10<79> 8 3 |

3 |  4  7  12 <12> .|

4 |  2  1  4  7 <4>|

--+----------------+

(row = reference; col = test)
```

```
--------------------------------------------------

Accuracy with normal features, with bigram featuresets:

Training and testing a classifier

Accuracy of classifier :

0.545

--------------------------------------------------

The confusion matrix

  |  0  1  2  3  4 |

--+----------------+

0 | <.> .  6  2  . |

1 |  1 <7> 25 4  . |

2 |  .  4<93> 5  1 |

3 |  1  7  17 <8> 2|

4 |  1  2  7  7 <1>|

--+----------------+

(row = reference; col = test)
```

Phase Five: Feature Results:

We then chose to determine our 30 most informative features. To do this, we used the function “show_most_informative_features()”. This function was associated with the “with pre-processing step” feature. The results of the most informative feature are below.

```
--------------------------------------------------
Accuracy with normal features, with pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.545

--------------------------------------------------
Showing most informative features
Most Informative Features
          contains(good) = True              4 : 2      =     20.0 : 1.
        contains(script) = True              0 : 2      =     17.8 : 1.0
           contains(can) = True              0 : 2      =     17.8 : 1.0
          contains(make) = True              4 : 2      =     13.8 : 1.0
         contains(whole) = True              4 : 2      =     12.8 : 1.0
        contains(actors) = True              4 : 2      =     12.8 : 1.0
          contains(well) = True              4 : 2      =     12.8 : 1.0
        contains(cinema) = True              4 : 2      =     12.8 : 1.0
          contains(gets) = True              4 : 2      =     12.8 : 1.0
           contains(man) = True              4 : 2      =     12.8 : 1.0
        contains(brings) = True              4 : 2      =     12.8 : 1.0
  contains(entertaining) = True              0 : 2      =     10.7 : 1.0
          contains(back) = True              0 : 2      =     10.7 : 1.0
         contains(great) = True              0 : 2      =     10.7 : 1.0
         contains(thing) = True              0 : 2      =     10.7 : 1.0
          contains(rise) = True              0 : 2      =     10.7 : 1.0
       contains(neither) = True              0 : 2      =     10.7 : 1.0
         contains(story) = True              0 : 2      =     10.7 : 1.0
         contains(three) = True              0 : 2      =     10.7 : 1.0
          contains(know) = True              0 : 2      =     10.7 : 1.0
         contains(seems) = True              0 : 2      =     10.7 : 1.0
      contains(material) = True              0 : 2      =     10.7 : 1.0
        contains(series) = True              0 : 2      =     10.7 : 1.0
           contains(guy) = True              0 : 2      =     10.7 : 1.0
         contains(place) = True              0 : 2      =     10.7 : 1.0
        contains(school) = True              0 : 2      =     10.7 : 1.0
         contains(women) = True              0 : 2      =     10.7 : 1.0
        contains(nature) = True              0 : 2      =     10.7 : 1.0
           contains(end) = True              0 : 3      =     10.3 : 1.0
       contains(nothing) = True              1 : 2      =      8.0 : 1.0
None
```

Phase Six : Weka/Sklearn Testing:

For the next phase of the project, we are going to use Weka to train and test our classifier. We will first create a .csv file to use in the Weka Program, and use cross-validation to obtain a precision, recall, and F-measure score. Below, we have listed the files we created for every feature. In addition, will run them through Weka. In addition to the following files, we created additional training datasets and ran classifiers on the datasets when they were 1000, 2000, 10000 phrases. It’s important to note that the feature files labeled “normal” below were

created using “normal” bag of words (unigram), and therefore, we labeled as such. We also ran it on the entire training dataset. Due to the constraints of online uploads on the 2SU site, we will not include those in attached zip file.

Feature files created:

features_normal.csv
features_preprocessed.csv
features_SL.csv
features_NOT.csv
features_biagram.csv

In the below “Results of Our Experiments” section, we highlight the outputs.

Phase Seven: Results of our Experiments:

Pre-Processing vs Accuracy:

For our first experiment we sought to compare the accuracy of the classifier before and after we applied a filter based on the stop words and other pre-processing steps:

Number of records	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	POS Tag Features (1000 Records)	SL Features	Not Features	Bigram Features
(1000 Records) Accuracy	0.48	0.55	0.54	0.56	0.5	0.55
(10000 Records) Accuracy	0.504	0.535	NA	0.537	0.46	0.536
(156060 Records) Accuracy	0.524093297	0.542547738	NA	0.551967192	0.558951685	0.542547738

Results: Based on our results, we can see that there is a higher accuracy after we applied pre-processing and stop words in our classifier. This is likely because pre-processing and stop-word removalare necessary to reduce noise and words that are not needed. Both the unigram and bigram features have higher accuracy marks as well.

Word/Vocabulary Comparison:

We then ran our classifer over our different feature sets to compare them words used. Specifically, we wanted to see how the size of the vocabulary differed across the different calssifiers, and how this effected accuracy.

Number of Words	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	POS Tag Features (1000 Records)	SL Features	Not Features	Bigram Features
300	0.48	0.55	NA	0.56	0.5	0.55
500	0.51	0.545	NA	0.55	0.51	0.545
1000	0.539	0.551	NA	0.556	0.498	0.551
2000	0.572	0.556	NA	0.575	0.556	0.585

Results: from the output table, we were able to see that the larger feature sets (like Before Preprocessing and SL feature) had larger words, and therefore larger accuracy. This is likely because the larger feature sets have more data, therefore words, to classify.

Cross-Validation: Precision, Recall and F- Measures Score

We then used the Ski-Learn python script on all of our csv files of features. We compared them using cross-validation for Precision, Recall and F- measures scores. Overall, the precision score quantifies the number of positive class predictions that actually belong to the positive class.The recall score is the number of positive class predictions made out of all positive examples in the dataset. Finally, the F-Measure score provides a single score that balances both the concerns of precision and recall in one number. The lower scores, the. Poorer the result.

Cross-Validation Score	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	POS Tag Features (1000 Records)	SL Features	Not Features	Bigram Features
Precision score	0.53	0.51	NA	0.55	0.55	0.51
Recall score	0.51	0.49	NA	0.52	0.52	0.49
F-measure score	0.52	0.5	NA	0.53	0.53	0.5

Results: From our output table, we can see that SL Feature has a better overall performance measure than the other classifiers (with the exception of Not-Features). This is likely because the feature contains a token that are found within the Lexicon dictionary we imported. These words or tokens are therefore, found within the training dataset, and classified. We can see, also, that the features recall score is less than that of the F-measure score .These two scores

are less than the precision score. However, overall, all scores are above 50%, so they are not poor results.

Weka Output: Logistic Regression

Bigram Features

We ran a logisitcregression classifier on the csv file with 10,000 records for the bigram feature (10000features_biagram.csv). We used 10-fold cross validation. We can see from our results that the dataset has a slightly higher precision of 0.51. Unfortunately, this is still quite low. Interesting that the big

	precision	recall	f1-score	support
neg	0.23	0.39	0.29	497
neu	0.68	0.63	0.65	5100
pos	0.24	0.42	0.31	557
sneg	0.32	0.33	0.32	1725
spos	0.42	0.33	0.37	2121
avg / total	0.51	0.49	0.50	10000

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	194	106	29	124	44	497
neu	285	3208	266	760	581	5100
pos	21	81	235	46	174	557
sneg	244	636	100	564	181	1725
spos	103	706	349	256	707	2121
All	847	4737	979	1750	1687	10000

Unigram Features

We ran a logisitcregression classifier on the csv file with 10,000 records for the normal, unigram feature (10000features_normal.csv). We used 10-fold cross validation. From our below output, we can see that that this feature has a slightly better overall score than the Bigram feature.

	precision	recall	f1-score	support
neg	0.24	0.40	0.30	497
neu	0.69	0.66	0.68	5100
pos	0.25	0.43	0.32	557
sneg	0.35	0.33	0.34	1725
spos	0.41	0.34	0.37	2121
avg / total	0.53	0.51	0.52	10000

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	201	92	39	124	41	497
neu	238	3381	229	650	602	5100
pos	31	77	240	38	171	557
sneg	248	637	80	566	194	1725
spos	113	695	360	238	715	2121
All	831	4882	948	1616	1723	10000

Negation Feature:

We ran a logisitcregression classifier on the csv file with 10,000 records for the not (negation) feature (10000features_NOT.csv). We used 10-fold cross validation. From our below output, we can see that that this feature has a slightly better overall score is a lot better than that for the Bigram feature and normal unigram features.

	precision	recall	f1-score	support
neg	0.24	0.42	0.31	497
neu	0.72	0.66	0.69	5100
pos	0.26	0.46	0.33	557
sneg	0.36	0.36	0.36	1725
spos	0.45	0.37	0.41	2121
avg / total	0.55	0.52	0.53	10000

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	211	73	26	147	40	497
neu	262	3342	213	697	586	5100
pos	22	51	254	45	185	557
sneg	289	560	87	614	175	1725
spos	101	627	382	216	795	2121
All	885	4653	962	1719	1781	10000

Pre-Processed Unigram Feature:

We ran a logisitcregression classifier on the csv file with 10,000 records for the feature where we pre-processed the tokens using our bag-of-words feature and our get_words_from_phasedocs list (10000features_preprocessed.csv). We used 10-fold cross validation. From our below output, we can see that that this feature has a slightly better overall score is a lot better than that for the Bigram feature and normal unigram features.

	precision	recall	f1-score	support
neg	0.23	0.39	0.29	497
neu	0.68	0.63	0.65	5100
pos	0.24	0.42	0.31	557
sneg	0.32	0.33	0.32	1725
spos	0.42	0.33	0.37	2121
avg / total	0.51	0.49	0.50	10000

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	194	106	29	125	43	497
neu	285	3209	266	760	580	5100
pos	21	81	235	46	174	557
sneg	244	636	100	564	181	1725
spos	104	704	349	256	708	2121
All	848	4736	979	1751	1686	10000

3.3 Sentiment Lexicon & Count of Subjectivity Features

We ran a logisitcregression classifier on the csv file with 10,000 records for the sentiment lexicon feature (10000features_SL.csv). We used 10-fold cross validation. From our below output, we can see that that this feature has a slightly better overall score is a lot better than that for the Bigram feature and normal unigram features. Overall, the SL_feature has the highest precision score. This is likely because the feature contains a token that are found within the Lexicon dictionary, we imported

	precision	recall	f1-score	support
neg	0.24	0.42	0.31	497
neu	0.72	0.66	0.69	5100
pos	0.26	0.46	0.33	557
sneg	0.36	0.36	0.36	1725
spos	0.45	0.37	0.41	2121
avg / total	0.55	0.52	0.53	10000

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	211	73	26	147	40	497
neu	262	3342	213	697	586	5100
pos	22	51	254	45	185	557
sneg	289	560	87	614	175	1725
spos	101	627	382	216	795	2121
All	885	4653	962	1719	1781	10000

Conclusion:

Throughout our project, we sought to determine which features are the best for sentiment analysis of the Kaggle movie review dataset. Based on our analysis and feature experiment results, we have determined that Sentiment Lexicon & Count of Subjectivity Features yield the highest and best classified results. In addition, these features (like all of the others) increase in accuracy after pre-processing. We have determined that this is likely because the feature contains a token that are found within the Lexicon dictionary we imported. These words or tokens are therefore, found within the training dataset, and classified.

References:

https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/
https://en.wikipedia.org/wiki/Sentiment_analysis
https://www.nltk.org

Appendix:

Appendix A: Naïve Bayes Output 10,000

```
Parins-MacBook-Pro:kagglemoviereviews parinppatel$ python classifyKaggle.py corpus/ 10000

('Read', 156060, 'phrases, using', 10000, 'random phrases')
--------------------------------------------------
Accuracy with normal features, without pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.51
--------------------------------------------------
Showing most informative features

Most Informative Features
          contains(make) = True                4 : 2      =     23.1 : 1.0
          contains(good) = True                4 : 2      =     20.0 : 1.0
            contains(up) = True                4 : 2      =     17.9 : 1.0
        contains(script) = True                0 : 2      =     17.8 : 1.0
            contains(we) = True                0 : 2      =     17.8 : 1.0
           contains(was) = True                0 : 2      =     17.8 : 1.0
           contains(did) = True                0 : 2      =     17.8 : 1.0
           contains(can) = True                0 : 2      =     17.8 : 1.0
          contains(here) = True                0 : 2      =     17.8 : 1.0
           contains(too) = True                0 : 2      =     14.9 : 1.0
          contains(well) = True                4 : 2      =     12.8 : 1.0
        contains(cinema) = True                4 : 2      =     12.8 : 1.0
          contains(gets) = True                4 : 2      =     12.8 : 1.0
        contains(brings) = True                4 : 2      =     12.8 : 1.0
          contains(with) = True                4 : 1      =     12.7 : 1.0
             contains(A) = True                4 : 2      =     10.8 : 1.0
         contains(there) = True                0 : 2      =     10.7 : 1.0
          contains(back) = True                0 : 2      =     10.7 : 1.0
       contains(neither) = True                0 : 2      =     10.7 : 1.0
         contains(story) = True                0 : 2      =     10.7 : 1.0
         contains(three) = True                0 : 2      =     10.7 : 1.0
           contains(has) = True                0 : 2      =     10.7 : 1.0
         contains(seems) = True                0 : 2      =     10.7 : 1.0
          contains(know) = True                0 : 2      =     10.7 : 1.0
        contains(school) = True                0 : 2      =     10.7 : 1.0
           contains(...) = True                0 : 2      =     10.7 : 1.0
         contains(would) = True                0 : 2      =     10.7 : 1.0
          contains(just) = True                0 : 3      =     10.3 : 1.0
          contains(like) = True                0 : 3      =     10.3 : 1.0
           contains(end) = True                0 : 3      =     10.3 : 1.0
```

```
None
The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> 1  5  2  . |
1 |  3 <8> 22  3  1 |
2 |  2  8<83> 8  1 |
3 |  4  4  17 <9> 1 |
4 |  2  4  5  5 <2>|
--+----------------+
(row = reference; col = test)




--------------------------------------------------
Accuracy with normal features, with pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.545

--------------------------------------------------
Showing most informative features
Most Informative Features
          contains(good) = True              4 : 2      =     20.0 : 1.
        contains(script) = True              0 : 2      =     17.8 : 1.0
           contains(can) = True              0 : 2      =     17.8 : 1.0
          contains(make) = True              4 : 2      =     13.8 : 1.0
         contains(whole) = True              4 : 2      =     12.8 : 1.0
        contains(actors) = True              4 : 2      =     12.8 : 1.0
          contains(well) = True              4 : 2      =     12.8 : 1.0
        contains(cinema) = True              4 : 2      =     12.8 : 1.0
          contains(gets) = True              4 : 2      =     12.8 : 1.0
           contains(man) = True              4 : 2      =     12.8 : 1.0
        contains(brings) = True              4 : 2      =     12.8 : 1.0
  contains(entertaining) = True              0 : 2      =     10.7 : 1.0
          contains(back) = True              0 : 2      =     10.7 : 1.0
         contains(great) = True              0 : 2      =     10.7 : 1.0
         contains(thing) = True              0 : 2      =     10.7 : 1.0
          contains(rise) = True              0 : 2      =     10.7 : 1.0
       contains(neither) = True              0 : 2      =     10.7 : 1.0
         contains(story) = True              0 : 2      =     10.7 : 1.0
         contains(three) = True              0 : 2      =     10.7 : 1.0
          contains(know) = True              0 : 2      =     10.7 : 1.0
```

```
       contains(story) = True                0 : 2      =     10.7 : 1.0
       contains(three) = True                0 : 2      =     10.7 : 1.0
        contains(know) = True                0 : 2      =     10.7 : 1.0
       contains(seems) = True                0 : 2      =     10.7 : 1.0
    contains(material) = True                0 : 2      =     10.7 : 1.0
      contains(series) = True                0 : 2      =     10.7 : 1.0
         contains(guy) = True                0 : 2      =     10.7 : 1.0
       contains(place) = True                0 : 2      =     10.7 : 1.0
      contains(school) = True                0 : 2      =     10.7 : 1.0
       contains(women) = True                0 : 2      =     10.7 : 1.0
      contains(nature) = True                0 : 2      =     10.7 : 1.0
         contains(end) = True                0 : 3      =     10.3 : 1.0
     contains(nothing) = True                1 : 2      =      8.0 : 1.0
one

he confusion matrix
  |   0   1   2   3   4 |
--+---------------------+
  | <.> .   6   2   . |
  |   1 <7> 25  4   . |
  |   .  4<93> 4   1 |
  |   1   7  17 <8> 2 |
  |   1   2   7   7 <1>|
--+---------------------+

row = reference; col = test)



----------------------------------------------------
ccuracy with normal features, with SL_featuresets :  :
raining and testing a classifier
ccuracy of classifier :
.55
----------------------------------------------------
howing most informative features

ost Informative Features
         negativecount = 3                 0 : 2      =     24.2 : 1.0
        contains(good) = True              4 : 2      =     20.0 : 1.0
      contains(script) = True              0 : 2      =     17.8 : 1.0
         contains(can) = True              0 : 2      =     17.8 : 1.0
        contains(make) = True              4 : 2      =     13.8 : 1.0
         positivecount = 3                 4 : 2      =     13.1 : 1.0
         positivecount = 5                 4 : 2      =     13.1 : 1.0
       contains(whole) = True              4 : 2      =     12.8 : 1.0
      contains(actors) = True              4 : 2      =     12.8 : 1.0
        contains(well) = True              4 : 2      =     12.8 : 1.0
      contains(cinema) = True              4 : 2      =     12.8 : 1.0
        contains(gets) = True              4 : 2      =     12.8 : 1.0
```

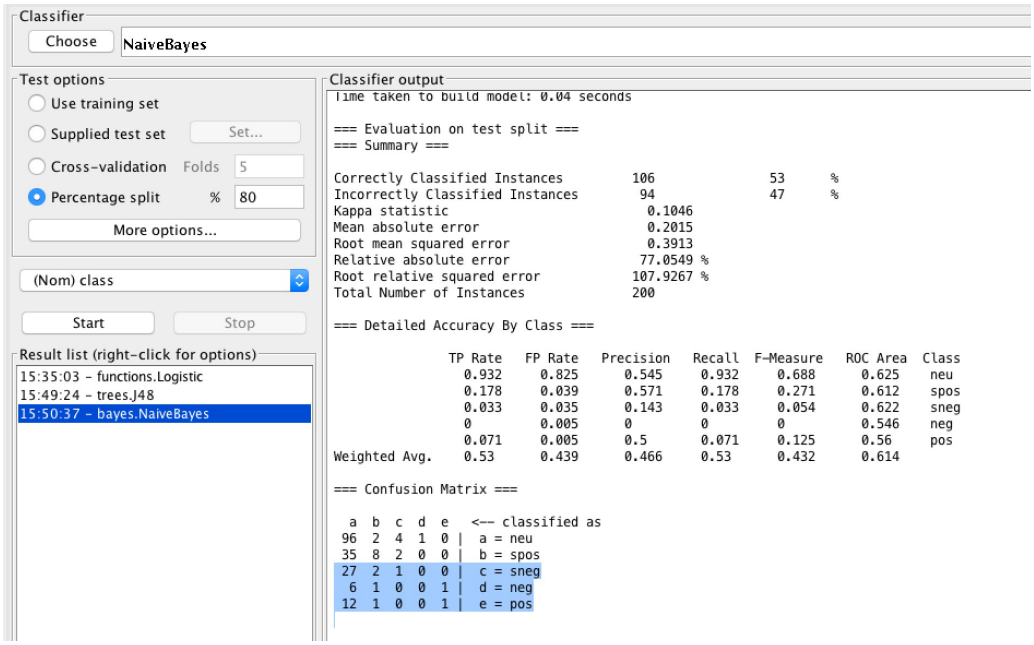
```
         contains(whole) = True                4 : 2      =     12.8 : 1.0
        contains(actors) = True                4 : 2      =     12.8 : 1.0
          contains(well) = True                4 : 2      =     12.8 : 1.0
        contains(cinema) = True                4 : 2      =     12.8 : 1.0
          contains(gets) = True                4 : 2      =     12.8 : 1.0
           contains(man) = True                4 : 2      =     12.8 : 1.0
        contains(brings) = True                4 : 2      =     12.8 : 1.0
           negativecount = 5                   4 : 2      =     12.2 : 1.0
           positivecount = 6                   4 : 2      =     12.1 : 1.0
  contains(entertaining) = True                0 : 2      =     10.7 : 1.0
          contains(back) = True                0 : 2      =     10.7 : 1.0
         contains(great) = True                0 : 2      =     10.7 : 1.0
         contains(thing) = True                0 : 2      =     10.7 : 1.0
          contains(rise) = True                0 : 2      =     10.7 : 1.0
       contains(neither) = True                0 : 2      =     10.7 : 1.0
         contains(story) = True                0 : 2      =     10.7 : 1.0
         contains(three) = True                0 : 2      =     10.7 : 1.0
         contains(seems) = True                0 : 2      =     10.7 : 1.0
      contains(material) = True                0 : 2      =     10.7 : 1.0
          contains(know) = True                0 : 2      =     10.7 : 1.0
        contains(series) = True                0 : 2      =     10.7 : 1.0
           contains(guy) = True                0 : 2      =     10.7 : 1.0
         contains(place) = True                0 : 2      =     10.7 : 1.0
None

The confusion matrix
  |  0  1  2  3  4 |
--+-----------------+
0 | <.> .  6  2  . |
1 |  1 <9> 23  4 . |
2 |  .  5<98> 7  1 |
3 |  1  6  15 <11> 2|
4 |  1  1  7  8 <1>|
--+-----------------+
(row = reference; col = test)



--------------------------------------------------
Accuracy with normal features, with NOT_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.51
--------------------------------------------------
Showing most informative features

Most Informative Features
          contains(good) = True                4 : 2      =     20.0 : 1.0
        contains(script) = True                0 : 2      =     17.8 : 1.0
           contains(can) = True                0 : 2      =     17.8 : 1.0
          contains(make) = True                4 : 2      =     13.8 : 1.0
```

```
0.51
--------------------------------------------------
Showing most informative features

Most Informative Features
          contains(good) = True               4 : 2      =     20.0 : 1.0
        contains(script) = True               0 : 2      =     17.8 : 1.0
           contains(can) = True               0 : 2      =     17.8 : 1.0
          contains(make) = True               4 : 2      =     13.8 : 1.0
         contains(whole) = True               4 : 2      =     12.8 : 1.0
        contains(cinema) = True               4 : 2      =     12.8 : 1.0
          contains(gets) = True               4 : 2      =     12.8 : 1.0
  contains(intelligence) = False              4 : 2      =     12.8 : 1.0
        contains(brings) = True               4 : 2      =     12.8 : 1.0
        contains(actors) = True               4 : 2      =     12.8 : 1.0
          contains(well) = True               4 : 2      =     12.8 : 1.0
           contains(man) = True               4 : 2      =     12.8 : 1.0
           contains(put) = False              0 : 2      =     10.7 : 1.0
         contains(thing) = True               0 : 2      =     10.7 : 1.0
       contains(strange) = False              0 : 2      =     10.7 : 1.0
      contains(material) = True               0 : 2      =     10.7 : 1.0
          contains(know) = True               0 : 2      =     10.7 : 1.0
       contains(already) = False              0 : 2      =     10.7 : 1.0
          contains(dish) = False              0 : 2      =     10.7 : 1.0
          contains(hack) = False              0 : 2      =     10.7 : 1.0
          contains(teen) = False              0 : 2      =     10.7 : 1.0
  contains(entertaining) = True               0 : 2      =     10.7 : 1.0
      contains(realized) = False              0 : 2      =     10.7 : 1.0
         contains(women) = True               0 : 2      =     10.7 : 1.0
         contains(three) = True               0 : 2      =     10.7 : 1.0
        contains(nature) = True               0 : 2      =     10.7 : 1.0
    contains(disgusting) = False              0 : 2      =     10.7 : 1.0
       contains(maudlin) = False              0 : 2      =     10.7 : 1.0
       contains(thought) = False              0 : 2      =     10.7 : 1.0
        contains(source) = False              0 : 2      =     10.7 : 1.0
None


The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  5  3  . |
1 |  5 <7> 22  3 . |
2 |  2  10<79> 8 3 |
3 |  4  7  12 <12> .|
4 |  2  1  4  7 <4>|
--+----------------+
(row = reference; col = test)
```

```
--------------------------------------------------
Accuracy with normal features, with bigram featuresets :
Training and testing a classifier
Accuracy of classifier :
0.545
--------------------------------------------------
 Showing most informative features
Most Informative Features
          contains(good) = True                4 : 2      =     20.0 : 1.0
        contains(script) = True                0 : 2      =     17.8 : 1.0
           contains(can) = True                0 : 2      =     17.8 : 1.0
          contains(make) = True                4 : 2      =     13.8 : 1.0
         contains(whole) = True                4 : 2      =     12.8 : 1.0
        contains(actors) = True                4 : 2      =     12.8 : 1.0
          contains(well) = True                4 : 2      =     12.8 : 1.0
        contains(cinema) = True                4 : 2      =     12.8 : 1.0
          contains(gets) = True                4 : 2      =     12.8 : 1.0
           contains(man) = True                4 : 2      =     12.8 : 1.0
        contains(brings) = True                4 : 2      =     12.8 : 1.0
  contains(entertaining) = True                0 : 2      =     10.7 : 1.0
          contains(back) = True                0 : 2      =     10.7 : 1.0
         contains(great) = True                0 : 2      =     10.7 : 1.0
         contains(women) = True                0 : 2      =     10.7 : 1.0
         contains(thing) = True                0 : 2      =     10.7 : 1.0
          contains(rise) = True                0 : 2      =     10.7 : 1.0
        contains(nature) = True                0 : 2      =     10.7 : 1.0
         contains(story) = True                0 : 2      =     10.7 : 1.0
         contains(three) = True                0 : 2      =     10.7 : 1.0
         contains(seems) = True                0 : 2      =     10.7 : 1.0
      contains(material) = True                0 : 2      =     10.7 : 1.0
          contains(know) = True                0 : 2      =     10.7 : 1.0
        contains(series) = True                0 : 2      =     10.7 : 1.0
           contains(guy) = True                0 : 2      =     10.7 : 1.0
         contains(place) = True                0 : 2      =     10.7 : 1.0
        contains(school) = True                0 : 2      =     10.7 : 1.0
       contains(neither) = True                0 : 2      =     10.7 : 1.0
           contains(end) = True                0 : 3      =     10.3 : 1.0
       contains(nothing) = True                1 : 2      =      8.0 : 1.0
None

The confusion matrix
  |  0  1  2  3  4 |
--+-----------------+
0 | <.> .  6  2  . |
1 |  1 <7> 25 4 . |
2 |  . 4<93> 4 1 |
3 |  1  7  17 <8> 2|
4 |  1  2  7  7 <1>|
--+-----------------+
```

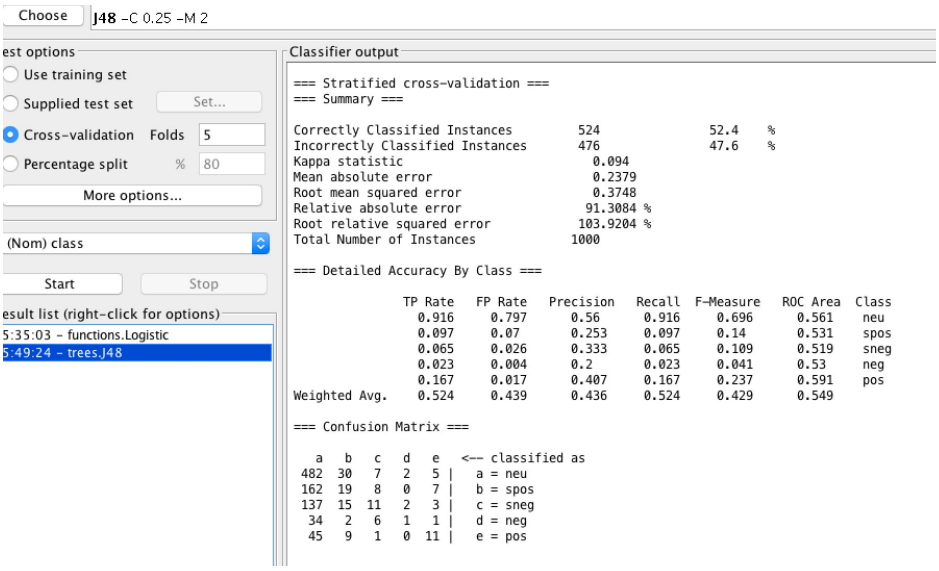
Appendix B:

We ran the Naiive Bayes Classifer in Weka as test with a percentage split of 80%. We can see that about 53% of the instances were correctly classified. Not much better than the Logistic Regression



Appendix C:

We ran a J48 Decision tree classifier with a 5 folds' cross validation in Weka. Output is Below. We can see that about 52% of the instances were correctly classified. Not much better than the Logistic Regression or Naiive Bayes.


Choose
J48 -C 0.25 -M 2
est options
Use training set
Supplied test set
Set...
Cross-validation
Folds
5
Percentage split
%
80
More options...
(Nom) class
Start
Stop
esult list (right-click for options)
5:35:03 - functions.Logistic
5:49:24 - trees.J48
Classifier output
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 524 52.4 %
Incorrectly Classified Instances 476 47.6 %
Kappa statistic 0.094
Mean absolute error 0.2379
Root mean squared error 0.3748
Relative absolute error 91.3084 %
Root relative squared error 103.9204 %
Total Number of Instances 1000
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
0.916 0.797 0.56 0.916 0.696 0.561 neu
0.097 0.07 0.253 0.097 0.14 0.531 spos
0.065 0.026 0.333 0.065 0.109 0.519 sneg
0.023 0.004 0.2 0.023 0.041 0.53 neg
0.167 0.017 0.407 0.167 0.237 0.591 pos
Weighted Avg. 0.524 0.439 0.436 0.524 0.429 0.549
=== Confusion Matrix ===
a b c d e <-- classified as
482 30 7 2 5 | a = neu
162 19 8 0 7 | b = spos
137 15 11 2 3 | c = sneg
34 2 6 1 1 | d = neg
45 9 1 0 11 | e = pos