Zane Alderfer

Professor Larche

IST 664

December 19, 2023

Final Project

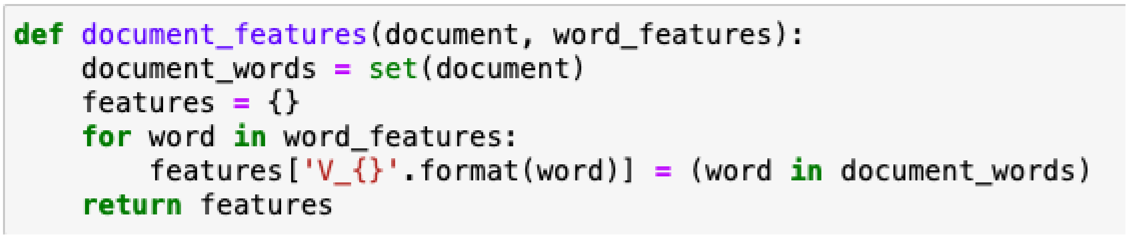
I chose the movie review tsv files to investigate for my final project. This one seemed the most interesting of the three choices and I also love movies so there was a subjective choice involved here as well.

Text Processing

In terms of text processing, I just used the functions from the finaldataproject files that broke down the train.tsv file line by line and then randomly shuffling these phrases. From there the words in each phrase were tokenized, and then changed to all lower case words. A word items list of the 1500 most common words was created and then a label list called “docs” was made with 5 folds. The label list contains 5 features of “negative”, “somewhat negative”, “neutral”, “somewhat positive”, and “positive.” They are represented by numbers 0-4 in the same order as above. This function can be seen in the python file attached.

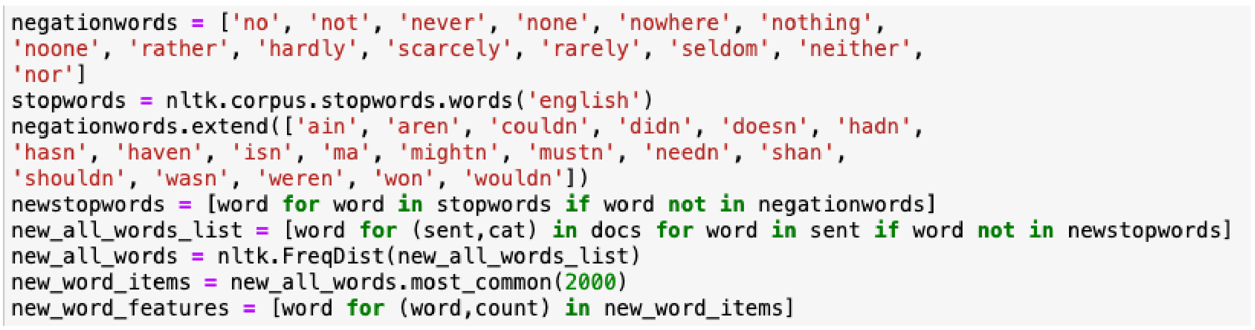
Feature Sets Created

Using functions to create the feature sets, I was able to create 9 feature sets. They all stem from the baseline function that uses all the phrases and its connotation attached to it without any modifications to the dataset at all as seen below.



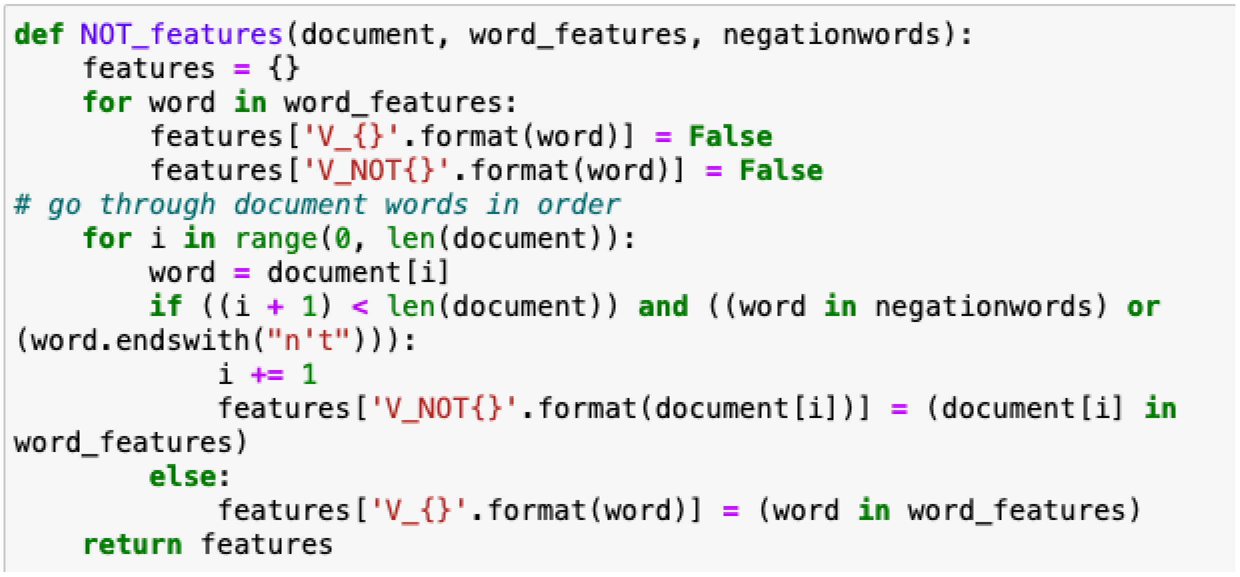
```
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features
```

Before making more featuresets I made a second words list to build featuresets that remove stopwords from the list to ignore common words and tokenized punctuation. I then built two separate featuresets for every modified featureset created. The new word list is seen below. Along with the new word list is a created variable that includes negation words for one of the modified feature sets.

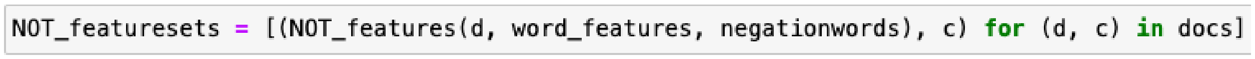


```
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing',
'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither',
'nor']
stopwords = nltk.corpus.stopwords.words('english')
negationwords.extend(['ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn',
'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan',
'shouldn', 'wasn', 'weren', 'won', 'wouldn'])
newstopwords = [word for word in stopwords if word not in negationwords]
new_all_words_list = [word for (sent,cat) in docs for word in sent if word not in newstopwords]
new_all_words = nltk.FreqDist(new_all_words_list)
new_word_items = new_all_words.most_common(2000)
new_word_features = [word for (word,count) in new_word_items]
```

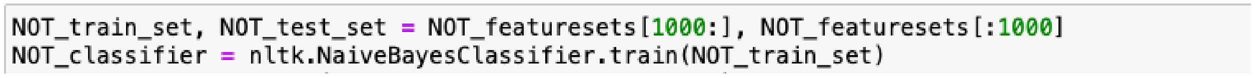
The first featureset removed words after negation words to hopefully narrow down on the meaningful words. To even further narrow down the list, I removed stopped words as well to find the most relevant words possible. Both featuresets were generated using the same function and then from those featuresets, a training and testing variable was made followed by a classifier. The function and classifier results can be seen below. The featuresets are called "NOT_featursets" and "new_NOT_featuresets" respectively.



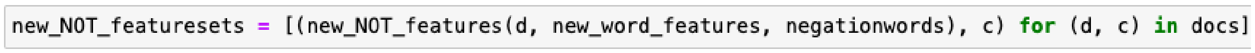
```
def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
# go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or
(word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in
word_features)
        else:
            features['V_{}'.format(word)] = (word in word_features)
    return features
```



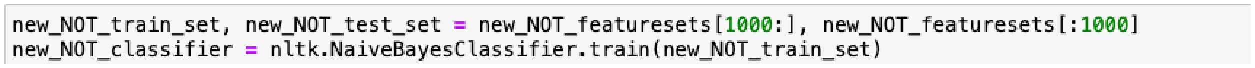
```
NOT_featuresets = [(NOT_features(d, word_features, negationwords), c) for (d, c) in docs]
```



```
NOT_train_set, NOT_test_set = NOT_featuresets[1000:], NOT_featuresets[:1000]
NOT_classifier = nltk.NaiveBayesClassifier.train(NOT_train_set)
```

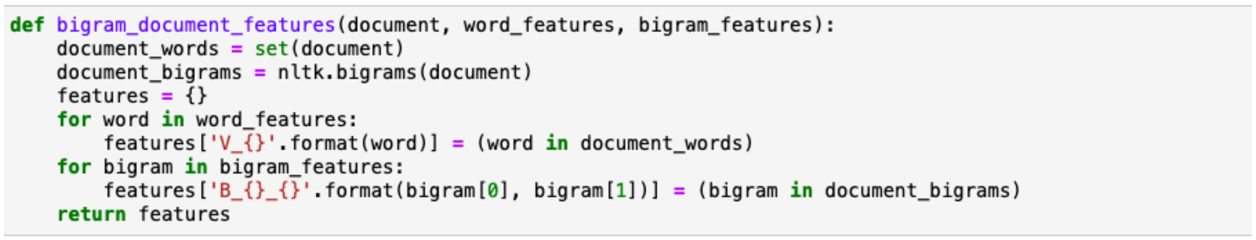


```
new_NOT_featuresets = [(new_NOT_features(d, new_word_features, negationwords), c) for (d, c) in docs]
```

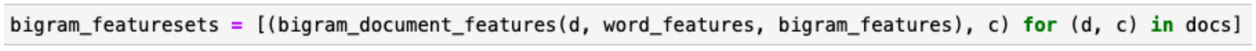


```
new_NOT_train_set, new_NOT_test_set = new_NOT_featuresets[1000:], new_NOT_featuresets[:1000]
new_NOT_classifier = nltk.NaiveBayesClassifier.train(new_NOT_train_set)
```

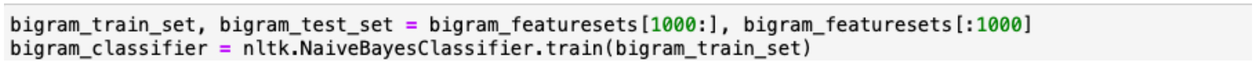
A bigram featureset with and without the stopped words was created. I removed stopped words in variations of featursets to gauge how much of an impact removing the stopped words will have on the accuracy of the classifiers and cross validations. The function and featuresets for bigrams can be seen below. Whenever the featuresets have "new" in the name, you can imply this one is using the new word list without stop words.



```
def bigram_document_features(document, word_features, bigram_features):
    document_words = set(document)
    document_bigrams = nltk.bigrams(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    for bigram in bigram_features:
        features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
    return features
```



```
bigram_featuresets = [(bigram_document_features(d, word_features, bigram_features), c) for (d, c) in docs]
```



```
bigram_train_set, bigram_test_set = bigram_featuresets[1000:], bigram_featuresets[:1000]
bigram_classifier = nltk.NaiveBayesClassifier.train(bigram_train_set)
```

```
new_bigram_featuresets = [(bigram_document_features(d, new_word_features, bigram_features), c) for (d, c) in docs]
```

```
new_bigram_train_set, new_bigram_test_set = new_bigram_featuresets[1000:], new_bigram_featuresets[:1000]
new_bigram_classifier = nltk.NaiveBayesClassifier.train(new_bigram_train_set)
```

The next two featuresets use a part of speech tagging function. This will distinguish the nouns, verbs, adjectives, and adverbs in the document. The function used to create this featureset and the corresponding featureset without stop words can be seen below.

```
def POS_features(document, word_features):
    document_words = set(document)
    tagged_words = nltk.pos_tag(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features
```

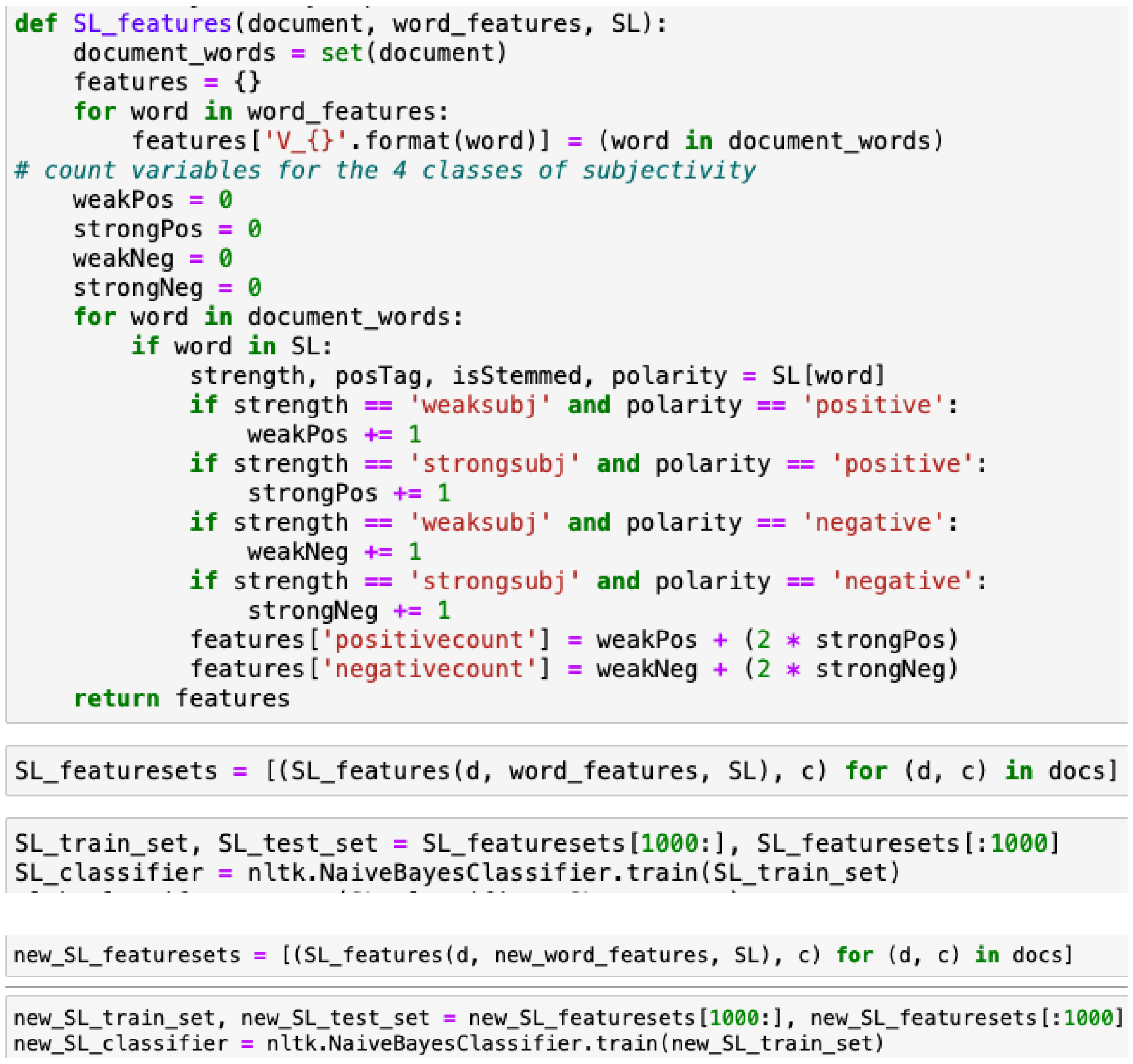
```
POS_featuresets = [(POS_features(d, word_features), c) for (d, c) in docs]
```

```
POS_train_set, POS_test_set = POS_featuresets[1000:], POS_featuresets[:1000]
POS_classifier = nltk.NaiveBayesClassifier.train(POS_train_set)
```

```
new_POS_featuresets = [(POS_features(d, new_word_features), c) for (d, c) in docs]
```

```
new_POS_train_set, new_POS_test_set = new_POS_featuresets[1000:], new_POS_featuresets[:1000]
new_POS_classifier = nltk.NaiveBayesClassifier.train(new_POS_train_set)
```

Finally a subjectivity featureset was created by reading in the HLTEMNLP05.tff file to tag the words in the document with “strength” and “polarity.” A function was then created that distinguishes a list between positive and negative. This function is particularly ideal for distinguishing the movie reviews for their inherent polarity. The function and featuresets can be seen below.



```
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
# count variables for the 4 classes of subjectivity
    weakPos = 0
    strongPos = 0
    weakNeg = 0
    strongNeg = 0
    for word in document_words:
        if word in SL:
            strength, posTag, isStemmed, polarity = SL[word]
            if strength == 'weaksubj' and polarity == 'positive':
                weakPos += 1
            if strength == 'strongsubj' and polarity == 'positive':
                strongPos += 1
            if strength == 'weaksubj' and polarity == 'negative':
                weakNeg += 1
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
```

```
SL_featuresets = [(SL_features(d, word_features, SL), c) for (d, c) in docs]
```

```
SL_train_set, SL_test_set = SL_featuresets[1000:], SL_featuresets[:1000]
SL_classifier = nltk.NaiveBayesClassifier.train(SL_train_set)
```

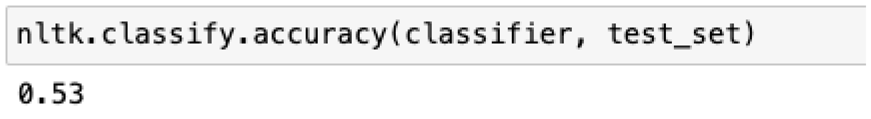
```
new_SL_featuresets = [(SL_features(d, new_word_features, SL), c) for (d, c) in docs]
```

```
new_SL_train_set, new_SL_test_set = new_SL_featuresets[1000:], new_SL_featuresets[:1000]
new_SL_classifier = nltk.NaiveBayesClassifier.train(new_SL_train_set)
```

Results of Feature Sets

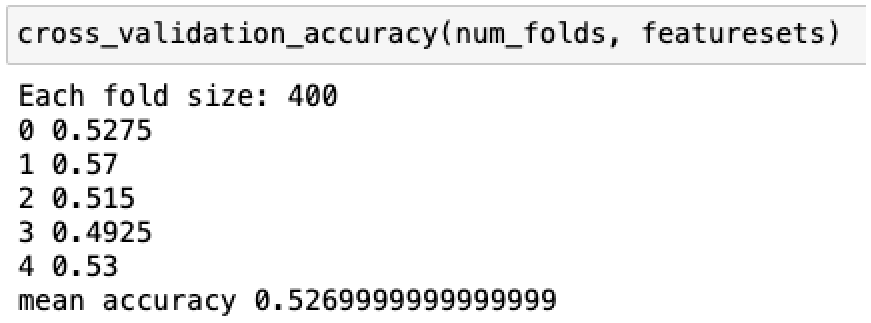
The results can be seen below of the classifiers, cross validation, confusion matrix, and precision, recall and F1 measures for each featureset below. A review of the results to follow.

Unmodified Featureset



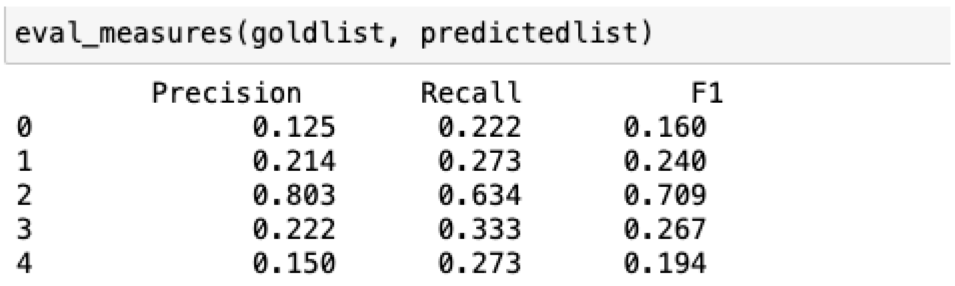
```
nltk.classify.accuracy(classifier, test_set)
```

```
0.53
```



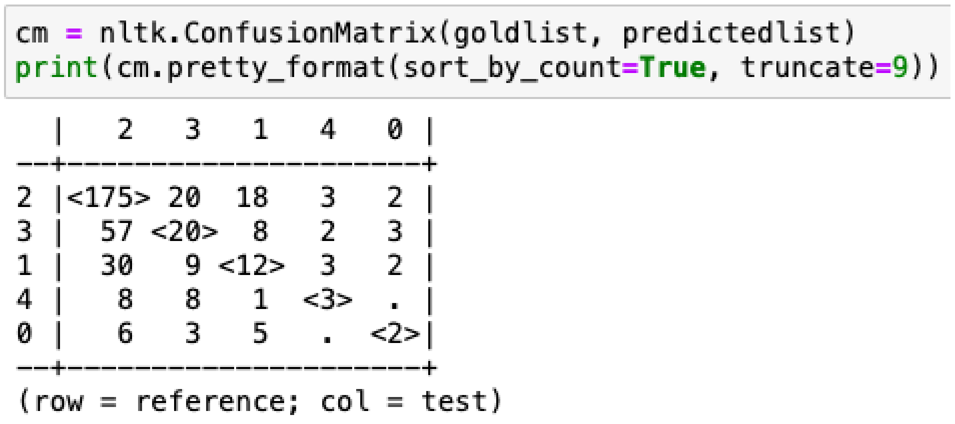
```
cross_validation_accuracy(num_folds, featuresets)
```

```
Each fold size: 400
0 0.5275
1 0.57
2 0.515
3 0.4925
4 0.53
mean accuracy 0.5269999999999999
```



```
eval_measures(goldlist, predictedlist)
```

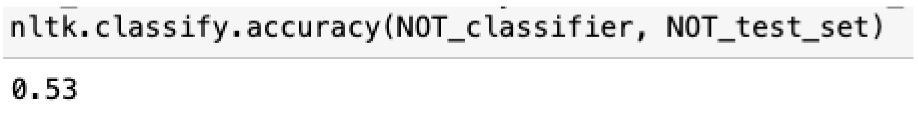
```
        Precision       Recall          F1
0             0.125      0.222      0.160
1             0.214      0.273      0.240
2             0.803      0.634      0.709
3             0.222      0.333      0.267
4             0.150      0.273      0.194
```



```
cm = nltk.ConfusionMatrix(goldlist, predictedlist)
print(cm.pretty_format(sort_by_count=True, truncate=9))
```

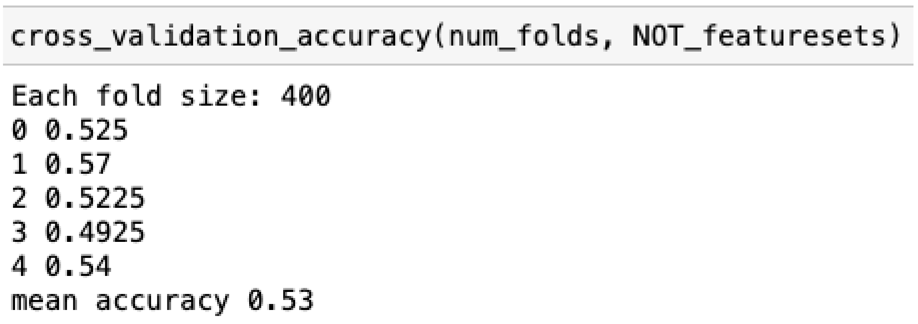
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<175> 20  18   3   2 |
3 |  57 <20>  8   2   3 |
1 |  30   9 <12>  3   2 |
4 |   8   8   1  <3>  . |
0 |   6   3   5   .  <2>|
--+---------------------+
(row = reference; col = test)
```

Words following negation words removed



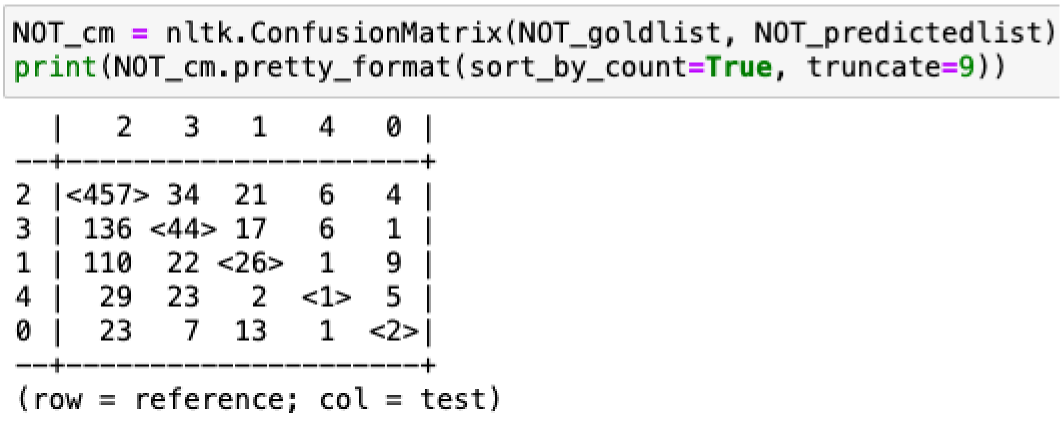
```
nltk.classify.accuracy(NOT_classifier, NOT_test_set)
```

```
0.53
```



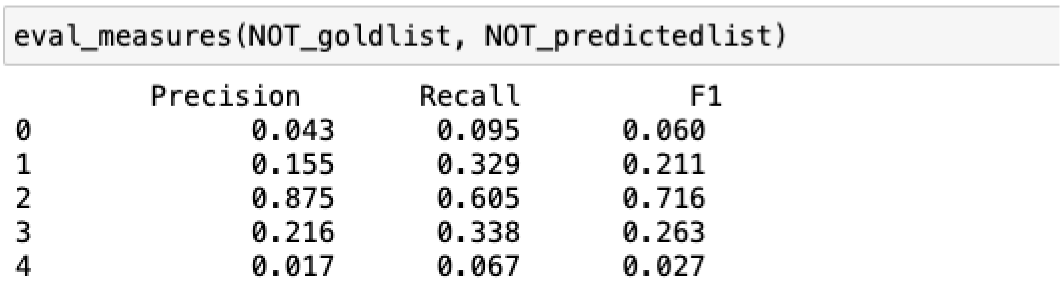
```
cross_validation_accuracy(num_folds, NOT_featuresets)
```

```
Each fold size: 400
0 0.525
1 0.57
2 0.5225
3 0.4925
4 0.54
mean accuracy 0.53
```



```
NOT_cm = nltk.ConfusionMatrix(NOT_goldlist, NOT_predictedlist)
print(NOT_cm.pretty_format(sort_by_count=True, truncate=9))
```

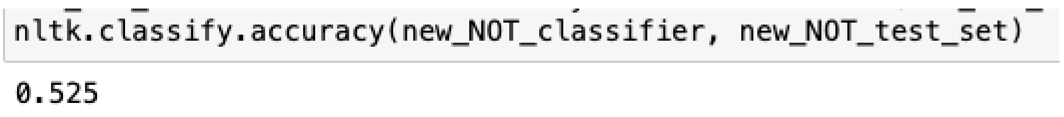
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<457> 34  21   6   4 |
3 | 136 <44> 17   6   1 |
1 | 110  22 <26>  1   9 |
4 |  29  23   2  <1>  5 |
0 |  23   7  13   1  <2>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(NOT_goldlist, NOT_predictedlist)
```

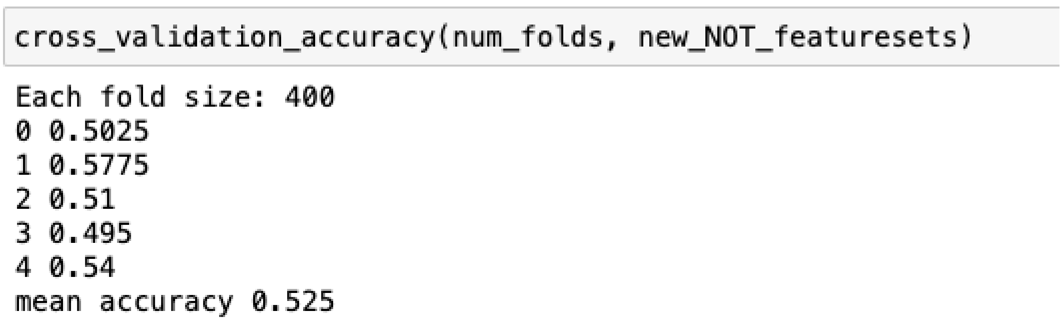
```
        Precision       Recall          F1
0             0.043      0.095      0.060
1             0.155      0.329      0.211
2             0.875      0.605      0.716
3             0.216      0.338      0.263
4             0.017      0.067      0.027
```

Words following negation words and stopped words removed



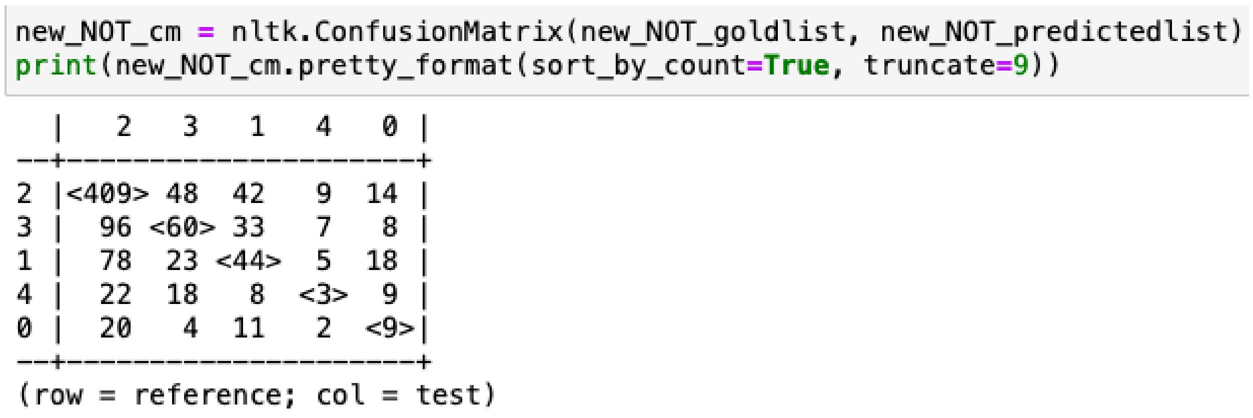
```
nltk.classify.accuracy(new_NOT_classifier, new_NOT_test_set)
```

```
0.525
```



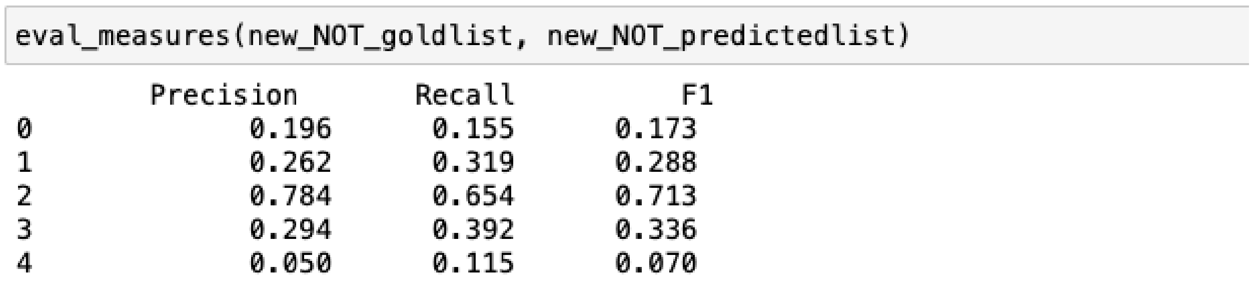
```
cross_validation_accuracy(num_folds, new_NOT_featuresets)
```

```
Each fold size: 400
0 0.5025
1 0.5775
2 0.51
3 0.495
4 0.54
mean accuracy 0.525
```



```
new_NOT_cm = nltk.ConfusionMatrix(new_NOT_goldlist, new_NOT_predictedlist)
print(new_NOT_cm.pretty_format(sort_by_count=True, truncate=9))
```

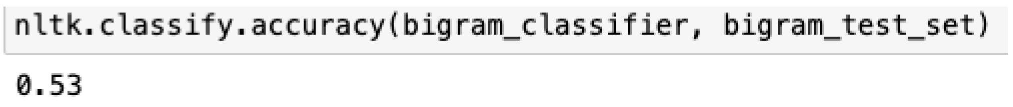
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<409> 48  42   9  14 |
3 |  96 <60> 33   7   8 |
1 |  78  23 <44>  5  18 |
4 |  22  18   8  <3>  9 |
0 |  20   4  11   2  <9>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(new_NOT_goldlist, new_NOT_predictedlist)
```

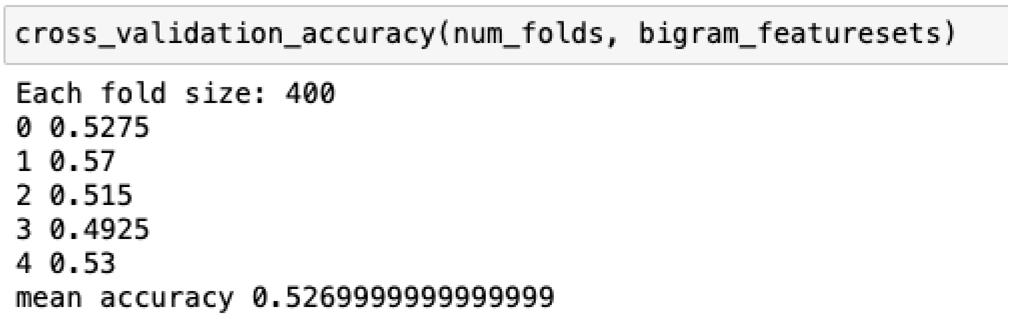
```
        Precision       Recall          F1
0             0.196      0.155      0.173
1             0.262      0.319      0.288
2             0.784      0.654      0.713
3             0.294      0.392      0.336
4             0.050      0.115      0.070
```

Bigram featureset



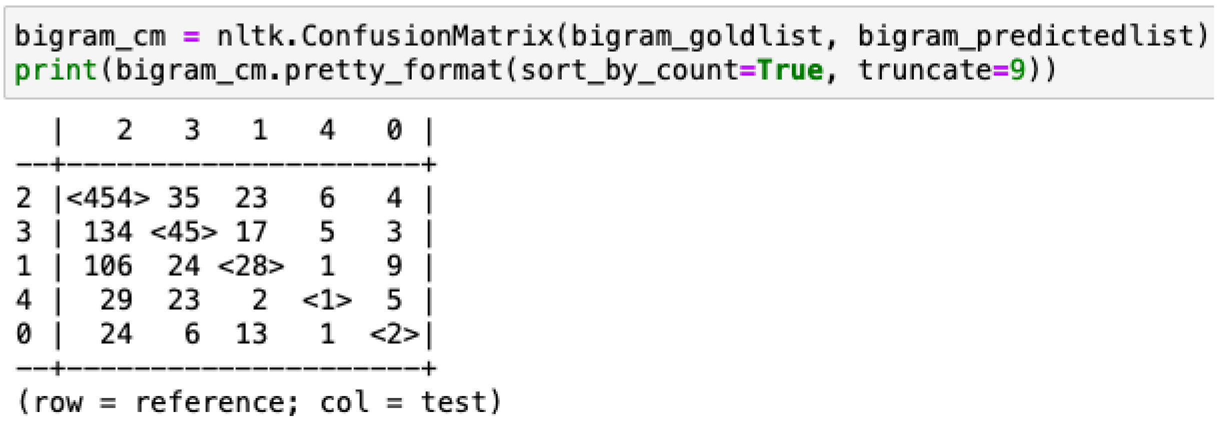
```
nltk.classify.accuracy(bigram_classifier, bigram_test_set)
```

```
0.53
```



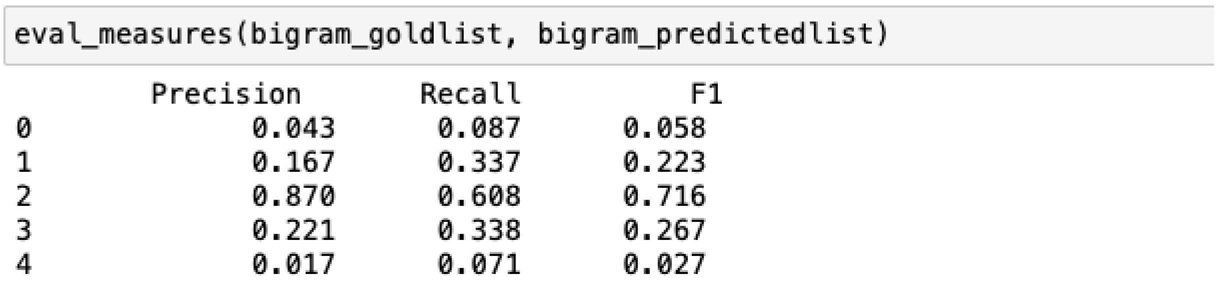
```
cross_validation_accuracy(num_folds, bigram_featuresets)
```

```
Each fold size: 400
0 0.5275
1 0.57
2 0.515
3 0.4925
4 0.53
mean accuracy 0.5269999999999999
```



```
bigram_cm = nltk.ConfusionMatrix(bigram_goldlist, bigram_predictedlist)
print(bigram_cm.pretty_format(sort_by_count=True, truncate=9))
```

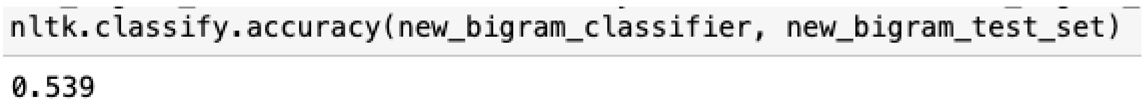
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<454> 35  23   6   4 |
3 | 134 <45> 17   5   3 |
1 | 106  24 <28>  1   9 |
4 |  29  23   2  <1>  5 |
0 |  24   6  13   1  <2>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(bigram_goldlist, bigram_predictedlist)
```

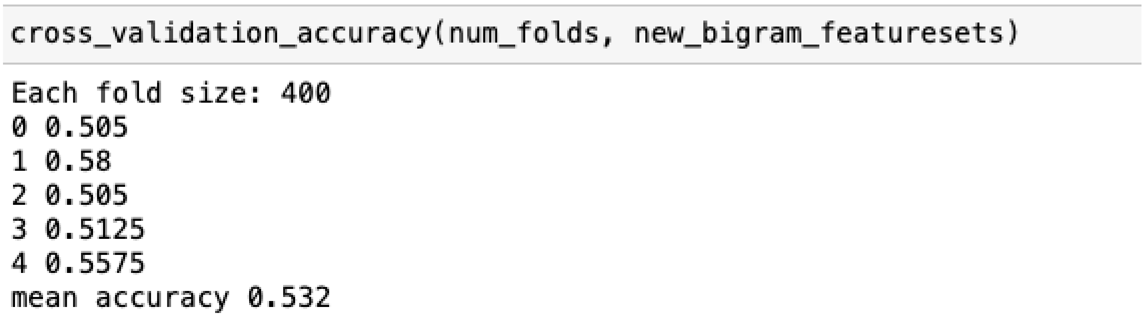
```
        Precision       Recall          F1
0             0.043      0.087      0.058
1             0.167      0.337      0.223
2             0.870      0.608      0.716
3             0.221      0.338      0.267
4             0.017      0.071      0.027
```

Bigrams without stopped words



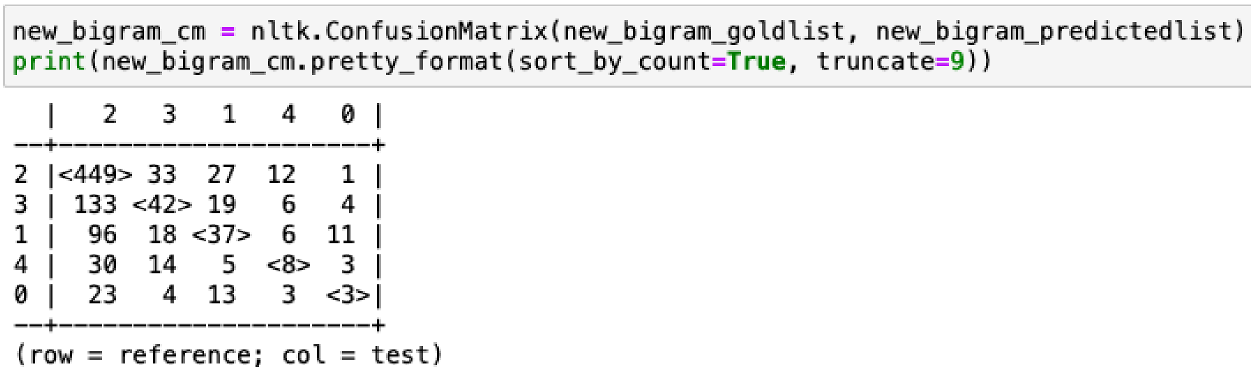
```
nltk.classify.accuracy(new_bigram_classifier, new_bigram_test_set)
```

```
0.539
```



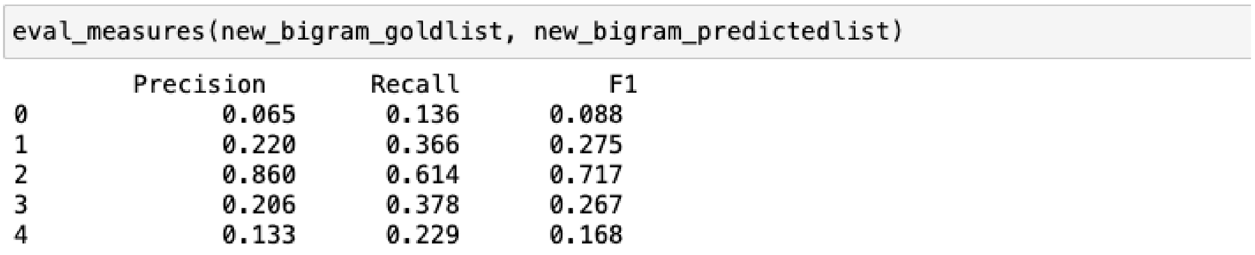
```
cross_validation_accuracy(num_folds, new_bigram_featuresets)
```

```
Each fold size: 400
0 0.505
1 0.58
2 0.505
3 0.5125
4 0.5575
mean accuracy 0.532
```



```
new_bigram_cm = nltk.ConfusionMatrix(new_bigram_goldlist, new_bigram_predictedlist)
print(new_bigram_cm.pretty_format(sort_by_count=True, truncate=9))
```

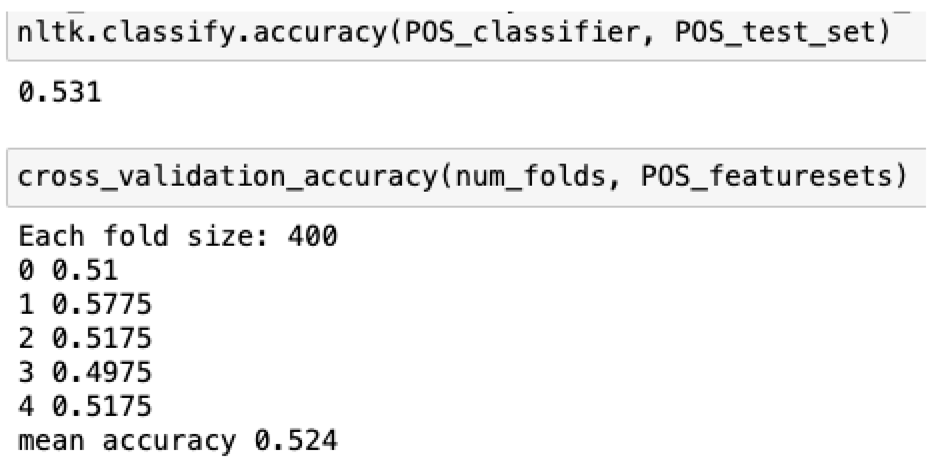
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<449> 33  27  12   1 |
3 | 133 <42> 19   6   4 |
1 |  96  18 <37>  6  11 |
4 |  30  14   5  <8>  3 |
0 |  23   4  13   3  <3>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(new_bigram_goldlist, new_bigram_predictedlist)
```

```
        Precision       Recall          F1
0           0.065        0.136       0.088
1           0.220        0.366       0.275
2           0.860        0.614       0.717
3           0.206        0.378       0.267
4           0.133        0.229       0.168
```

Part of Speech featureset

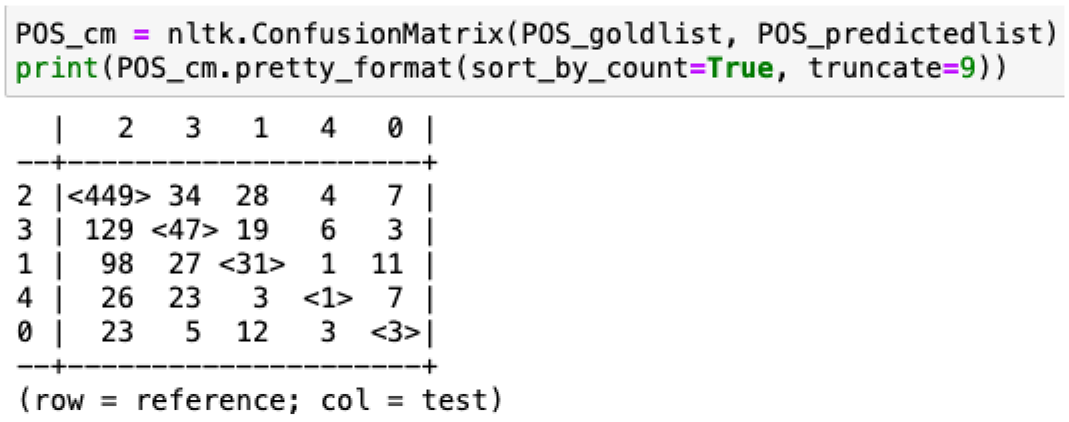


```
nltk.classify.accuracy(POS_classifier, POS_test_set)
```

```
0.531
```

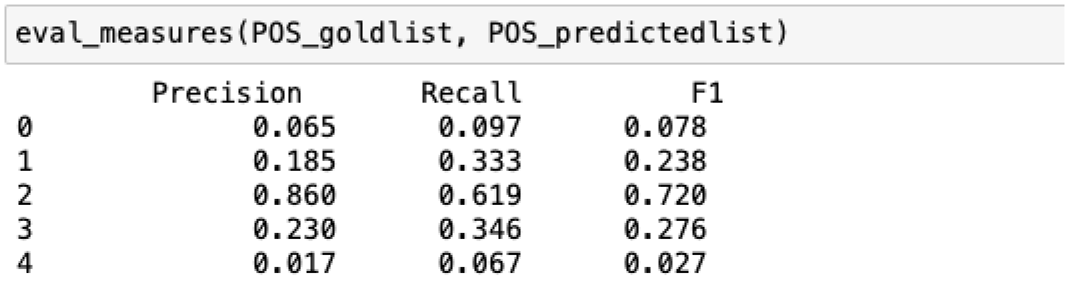
```
cross_validation_accuracy(num_folds, POS_featuresets)
```

```
Each fold size: 400
0 0.51
1 0.5775
2 0.5175
3 0.4975
4 0.5175
mean accuracy 0.524
```



```
POS_cm = nltk.ConfusionMatrix(POS_goldlist, POS_predictedlist)
print(POS_cm.pretty_format(sort_by_count=True, truncate=9))
```

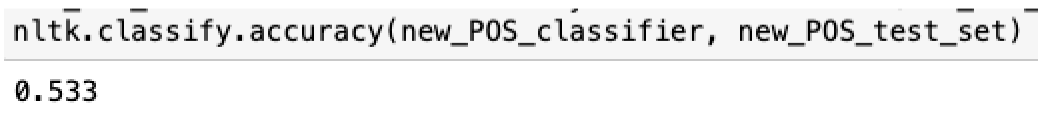
```
  |   2   3   1   4   0 |
--+---------------------+
2 |<449> 34  28   4   7 |
3 | 129 <47> 19   6   3 |
1 |  98  27 <31>  1  11 |
4 |  26  23   3  <1>  7 |
0 |  23   5  12   3  <3>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(POS_goldlist, POS_predictedlist)
```

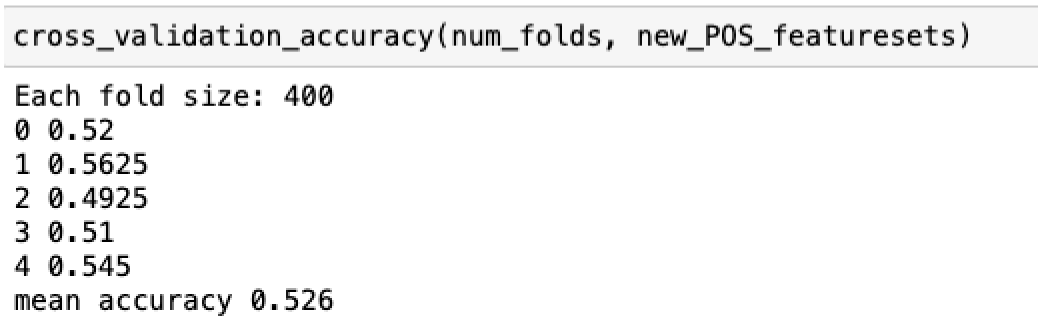
```
        Precision       Recall          F1
0             0.065      0.097       0.078
1             0.185      0.333       0.238
2             0.860      0.619       0.720
3             0.230      0.346       0.276
4             0.017      0.067       0.027
```

Part of Speech without stopped words



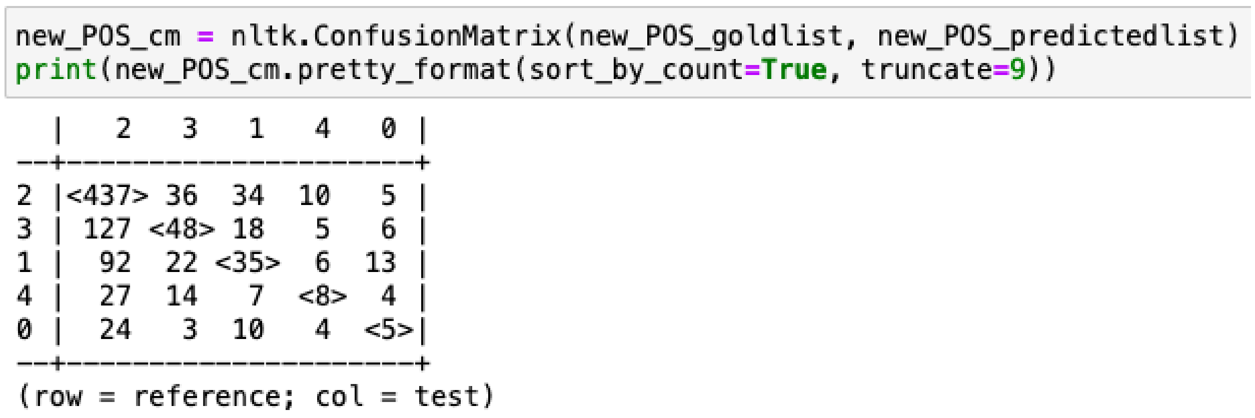
```
nltk.classify.accuracy(new_POS_classifier, new_POS_test_set)
```

```
0.533
```



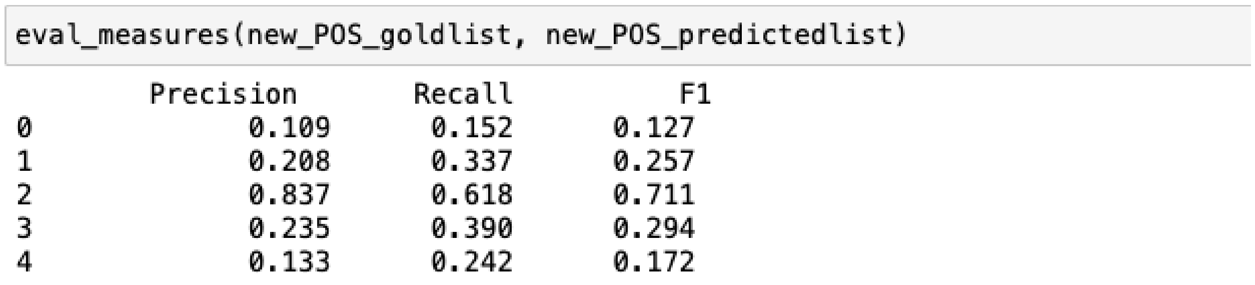
```
cross_validation_accuracy(num_folds, new_POS_featuresets)
```

```
Each fold size: 400
0 0.52
1 0.5625
2 0.4925
3 0.51
4 0.545
mean accuracy 0.526
```



```
new_POS_cm = nltk.ConfusionMatrix(new_POS_goldlist, new_POS_predictedlist)
print(new_POS_cm.pretty_format(sort_by_count=True, truncate=9))
```

```
  |   2   3   1   4   0 |
--+---------------------+
2 |<437> 36  34  10   5 |
3 | 127 <48> 18   5   6 |
1 |  92  22 <35>  6  13 |
4 |  27  14   7  <8>  4 |
0 |  24   3  10   4  <5>|
--+---------------------+
(row = reference; col = test)
```



```
eval_measures(new_POS_goldlist, new_POS_predictedlist)
```

```
        Precision       Recall          F1
0            0.109       0.152       0.127
1            0.208       0.337       0.257
2            0.837       0.618       0.711
3            0.235       0.390       0.294
4            0.133       0.242       0.172
```

Subjectivity featureset

```
nltk.classify.accuracy(SL_classifier, SL_test_set)
```

```
0.543
```

```
cross_validation_accuracy(num_folds, SL_featuresets)
```

```
Each fold size: 400
0 0.535
1 0.6025
2 0.51
3 0.5125
4 0.545
mean accuracy 0.541
```

```
SL_cm = nltk.ConfusionMatrix(SL_goldlist, SL_predictedlist)
print(SL_cm.pretty_format(sort_by_count=True, truncate=9))
```

```
  |   2   3   1   4   0 |
--+---------------------+
2 |<444> 38  29   5   6 |
3 | 116 <60> 17   7   4 |
1 |  96  28 <34>  1   9 |
4 |  22  29   3  <2>  4 |
0 |  25   4  13   1  <3>|
--+---------------------+
(row = reference; col = test)
```

```
eval_measures(SL_goldlist, SL_predictedlist)
```

```
        Precision        Recall          F1
0             0.065       0.115       0.083
1             0.202       0.354       0.258
2             0.851       0.632       0.725
3             0.294       0.377       0.331
4             0.033       0.125       0.053
```

Subjectivity without stopped words

```
nltk.classify.accuracy(new_SL_classifier, new_SL_test_set)
```

```
0.554
```

```
cross_validation_accuracy(num_folds, new_SL_featuresets)
```

```
Each fold size: 400
0 0.53
1 0.5825
2 0.5225
3 0.51
4 0.555
mean accuracy 0.54
```

```
new_SL_cm = nltk.ConfusionMatrix(new_SL_goldlist, new_SL_predictedlist)
print(new_SL_cm.pretty_format(sort_by_count=True, truncate=9))
```

```
  |   2   3   1   4   0 |
--+---------------------+
2 |<430> 42  34  10   6 |
3 | 110 <65> 16   8   5 |
1 |  90  20 <41>  4  13 |
4 |  20  23   3 <11>  3 |
0 |  22   4  13   .  <7>|
--+---------------------+
(row = reference; col = test)
```

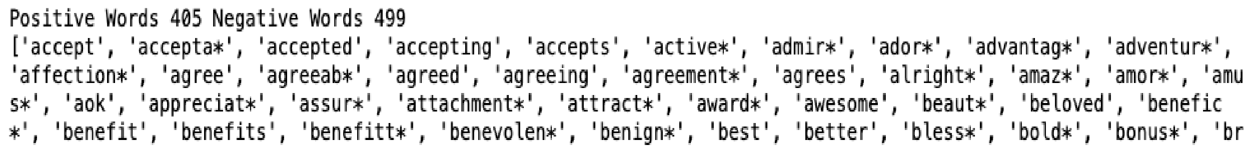
```
eval_measures(new_SL_goldlist, new_SL_predictedlist)
```

```
        Precision       Recall          F1
0            0.152       0.206      0.175
1            0.244       0.383      0.298
2            0.824       0.640      0.720
3            0.319       0.422      0.363
4            0.183       0.333      0.237
```

The number two is representative of “neutral” for movie reviews. Looking at the confusion matrices, the large majority of the words are neutral in nature. Some notable takeaways from the results above would be that just simply removing words after negation words and stopped words from the unmodified word list. In fact, the featureset that removed both is actually less accurate than the unmodified featureset. Bigrams and subjectivity featuresets promoted the highest level of accuracy. Bigrams without stopped words almost reaches 54% accuracy and relative to the other featuresets, subjectivity featuresets takes a big jump by half a percent with the stopped words and then without the stopped words, jumps an impressive 1 more percent to reach 55.4%. In conclusion, taking polarity into account for polarity based reviews, allows us to reach a more desirable percent of accuracy.

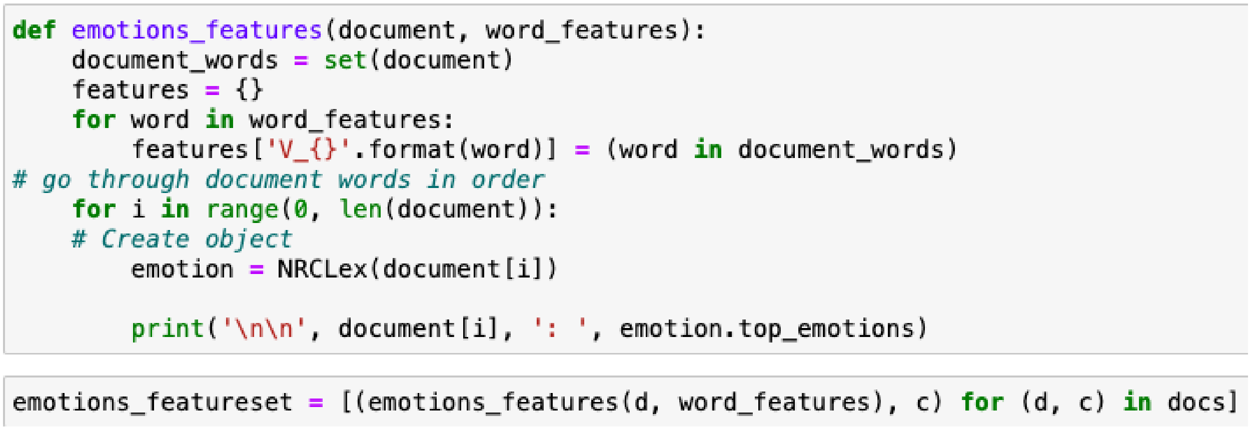
Further Testing and Results

I was playing around with the LIWC dictionary and was able to develop some lists of positive and negative words from a subset of the document. A small portion of these results can be seen below. If the words were considered neutral or not strongly positive or strongly negative, they were removed from the list. The functions and code to develop the list is in the attached python file.



```
Positive Words 405 Negative Words 499
['accept', 'accepta*', 'accepted', 'accepting', 'accepts', 'active*', 'admir*', 'ador*', 'advantag*', 'adventur*',
'affection*', 'agree', 'agreeab*', 'agreed', 'agreeing', 'agreement*', 'agrees', 'alright*', 'amaz*', 'amor*', 'amu
s*', 'aok', 'appreciat*', 'assur*', 'attachment*', 'attract*', 'award*', 'awesome', 'beaut*', 'beloved', 'benefic
*', 'benefit', 'benefits', 'benefitt*', 'benevolen*', 'benign*', 'best', 'better', 'bless*', 'bold*', 'bonus*', 'br
```

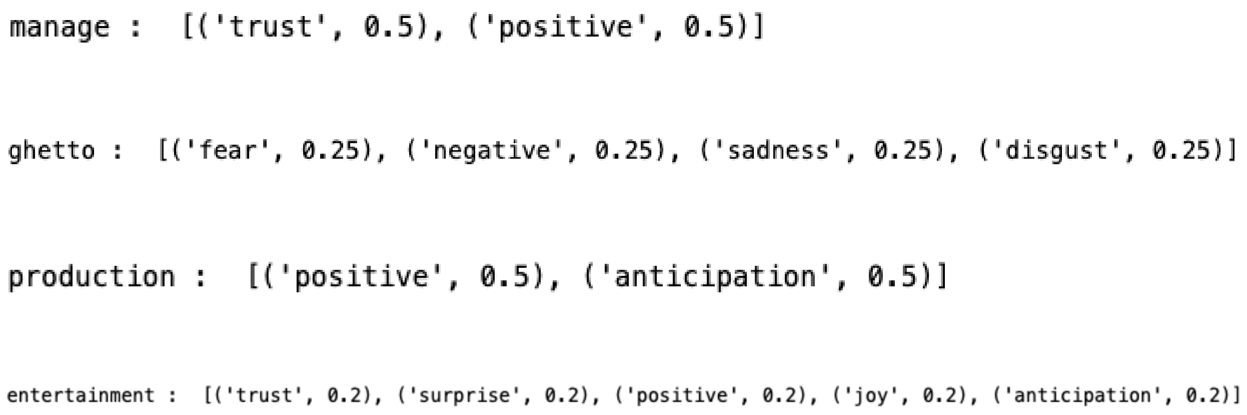
The last result involved the NRCLex library. From this I was able to distinguish the emotions that encapsulate each word in the document. If they were neutral, they had a score of 0 for all the”emotion describing” words. The function and featureset created to establish the list is seen below along with some examples of words and emotion scores.



```
def emotions_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
# go through document words in order
    for i in range(0, len(document)):
    # Create object
        emotion = NRCLex(document[i])

        print('\n\n', document[i], ': ', emotion.top_emotions)
```

```
emotions_featureset = [(emotions_features(d, word_features), c) for (d, c) in docs]
```



```
manage :  [('trust', 0.5), ('positive', 0.5)]


ghetto :  [('fear', 0.25), ('negative', 0.25), ('sadness', 0.25), ('disgust', 0.25)]


production :  [('positive', 0.5), ('anticipation', 0.5)]


entertainment :  [('trust', 0.2), ('surprise', 0.2), ('positive', 0.2), ('joy', 0.2), ('anticipation', 0.2)]
```