Final Project

Step1 : Preprocessing and Filtering

For this project, we decided to work on the classification model on movie reviews dataset. By using processkaggle() function and reading raw training data, we wanted to use stop-words from nltk corpus and negation words to remove unnecessary words. Overall, the preprocessing step includes tokenizing movie reviews into words, transferring all words into lowercase, removing all non-alphabets and stopwords (screenshot 1). We defined a function preProcessdata that first split the document using white space, and by using lower() function we then transferred all words in the word list to lowercase. We then used regex expressions to remove all the non-alphabets and numbers. After we removed all unnecessary words, we then created an empty list named final_word_list, in which we will append all the words in the word list but remove the ones that also appear in the stopword list we defined. The preprocessing function would provide us with all the necessary words we need for future analysis. For further predictive model, we will do a comparison between the accuracy from model with preprocessing steps and model without.

```
stopwords = nltk.corpus.stopwords.words('english')
modStopwords = [word for word in stopwords if word not in ['not', 'no', 'can', 'don', 't']]

def preProcessdata(data):
    wordList = re.split('\s+', data.lower())
    punctuation = re.compile(r'[-.?!/\%@,":;()|0-9]')
    wordList = [punctuation.sub("", word) for word in wordList]
    finalWordList = []
    for word in wordList:
        if word not in modStopwords:
            finalWordList.append(word)
    res = " ".join(finalWordList)
    return res
```

Step 2: Featuresets

For this project, we used bag-of-word to do the feature engineering for Naive Bayes algorithm. Specifically, we created a function named bagofWords() and defined a set of words , in which we selected the top 200 most frequent words to be used for features, by using FreqDist(). Then we used normal_feature() function to define a unigram feature, and label each keyword in the word_feature set with 'V_keyword'. In the next section, we will use the unigram model as the baseline for model accuracy.

```
def bagofWords(wordList):
    wordlist = nltk.FreqDist(wordList)
    wordFeatures = [w for (w, c) in wordlist.most_common(200)]
    return wordFeatures


def unigram_features(data, wordFeatures):
    documentWords = set(data)
    features = {}
    for word in wordFeatures:
        features['V_{}'.format(word)] = (word in documentWords)
    return features
```

We also defined a bigram feature to compare if adding more features can help increase predictive model accuracy. We first used Bigram CollocationFinder to find and rank all sets of bigrams, then we used the chi-square method to select the most informative bigrams. We then defined a bigram feature extraction function that contains both unigram feature V_{} and bigram feature B_{}_{}.

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


def bigram_finder(wordList):
    bigram_measures = nltk.collocations.BigramAssocMeasures()
    finder = BigramCollocationFinder.from_words(wordList, window_size=3)
    bigram_features = finder.nbest(bigram_measures.chi_sq, 3000)
    return bigram_features[:500]
```

Apart from using unigram and bigram features, we also wanted to test out if sentiment lexicon features can help generate a model with higher accuracy. We basically used the subjectivity lexicon file provided by professors, in which case, all words in the document will be classified by their strength, POS tag, true or false to indicate if the word is stemmed, and 'positive' or 'negative' to categorize the word. We then created a feature set for Naive Bayes model, using positive and negative counts (see detailed screenshots in reference).

```
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)
    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)
    if 'positivecount' not in features:
        features['positivecount'] = 0
    if 'negativecount' not in features:
        features['negativecount'] = 0
    return features
```

For combined feature sets we have created a function that will combine different feature sets such as unigram features, bigram features, sentiment lexicons, the result of this function is a single feature set which is a combination of bigram and sentiment lexicons. One of the main constraints running this function is the execution time taken to generate the feature sets.

```python
def combined_document_features(document, word_features, bigram_features, SL):
    document_words = set(document)
    document_bigrams = nltk.bigrams(document)
    features = {}
    for word in document_words:
        posword = 0
        neutword = 0
        negword = 0
        for word in document_words:
            if word in SL[0]:
                posword += 1
            if word in SL[1]:
                neutword += 1
            if word in SL[2]:
                negword += 1
            features['positivecount'] = posword
            features['neutralcount'] = neutword
            features['negativecount'] = negword
        for word in word_features:
            features['V_{}'.format(word)] = False
            features['V_NOT{}'.format(word)] = False
        for bigram in bigram_features:
            features['B_{}_{}'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
        return features
```

Step 3 Experiments

3.1 Naive Bayes Classifier

In general, we used Naive Bayes classifier to train and test movie reviews data with different sets of features. Basically, each review data from the movie review dataset was being categorized by sentiment scale ("negative"-0, "somewhat negative"-1, "neutral"-2, "somewhat positive"-3, "positive"- 4), we aimed to compare our results generated by our model to the original labels given, so as to evaluate the model accuracy. We defined the size of training data and test data based on the length of each feature set. Specially, we set the training size to equal the length of the featureset * 0.1 and used int() to get an integer. The size of test data is intuitively the rest of data in each feature set. To inspect the output of each model, we also designed a function generateMatrix() that put our results into a confusion matrix based on sentiment classification scale. More specifically, within the function, we started out with two empty lists, goldlist and predictlist, in which goldlist would contain the original label and predictlist would have all the predictive results we generate from test data. We would also use precision, F1 and recall to evaluate the performance of different models for each label.

3.1.1 Unigram feature set result

As mentioned in step 2, we would use the unigram feature set as the baseline for model accuracy and apply additional features to see if we can improve the accuracy. From the screenshots below, it is indicated that preprocessing steps helped to increase model accuracy. Moreover, based on the confusion matrix, 'neutral'-2 can be categorized more accurately compared to other labels. 'Somewhat positive'-3 has the second highest accuracy.

```
Accuracy without pre-processing unigram features :
Accuracy :
0.522
Confusion matrix
  |   0   1   2   3   4 |
--+---------------------+
0 |  <6>  5  11   5   5 |
1 |  18 <40>104  15   9 |
2 |  15  36<442> 27   2 |
3 |  16  17 132 <29> 12 |
4 |   9   4  22  14  <5>|
--+---------------------+
(row = reference; col = test)

Confusion matrix in percentage
  |      2      3      1      4      0 |
--+-----------------------------------+
2 | <44.2%>  2.7%   3.6%   0.2%   1.5% |
3 |  13.2%  <2.9%>  1.7%   1.2%   1.6% |
1 |  10.4%   1.5%  <4.0%>  0.9%   1.8% |
4 |   2.2%   1.4%   0.4%  <0.5%>  0.9% |
0 |   1.1%   0.5%   0.5%   0.5%  <0.6%>|
--+-----------------------------------+
(row = reference; col = test)

Model Evaluation
        Precision       Recall          F1
0             0.188      0.094       0.125
1             0.215      0.392       0.278
2             0.847      0.622       0.717
3             0.141      0.322       0.196
4             0.093      0.152       0.115
```

```
Accuracy with pre-processed unigram features :
Accuracy :
0.545
Confusion matrix
  |   0   1   2   3   4 |
--+---------------------+
0 |  <3>  7  19   3   . |
1 |   4 <30>135  13   4 |
2 |   3  25<473> 18   3 |
3 |   3  12 146 <35> 10 |
4 |   3   3  25  19  <4>|
--+---------------------+
(row = reference; col = test)

Confusion matrix in percentage
  |      2      3      1      4      0 |
--+-----------------------------------+
2 | <47.3%>  1.8%   2.5%   0.3%   0.3% |
3 |  14.6%  <3.5%>  1.2%   1.0%   0.3% |
1 |  13.5%   1.3%  <3.0%>  0.4%   0.4% |
4 |   2.5%   1.9%   0.3%  <0.4%>  0.3% |
0 |   1.9%   0.3%   0.7%      .  <0.3%>|
--+-----------------------------------+
(row = reference; col = test)

Model Evaluation
        Precision       Recall          F1
0             0.094      0.188       0.125
1             0.161      0.390       0.228
2             0.906      0.593       0.717
3             0.170      0.398       0.238
4             0.074      0.190       0.107
```

3.1.2 Bigram Features result

The overall model accuracy generated by using bigram feature is 0.528, which is even lower than using unigram feature model. From the below confusion matrix, ‘neutral’-2 is still being predicted more accurately than the rest labels.

```
Accuracy with bigram featuresets :
Accuracy :
0.528
Confusion matrix
  |   0   1   2   3   4 |
--+---------------------+
0 |  <4>  3  31   2   . |
1 |   5 <20>138  11   6 |
2 |   5  18<464> 26   9 |
3 |   2   4 141 <34> 11 |
4 |   1   2  45  12  <6>|
--+---------------------+
(row = reference; col = test)

Confusion matrix in percentage
  |      2      3      1      4      0 |
--+-----------------------------------+
2 | <46.4%>  2.6%   1.8%   0.9%   0.5% |
3 |  14.1%  <3.4%>  0.4%   1.1%   0.2% |
1 |  13.8%   1.1%  <2.0%>  0.6%   0.5% |
4 |   4.5%   1.2%   0.2%  <0.6%>  0.1% |
0 |   3.1%   0.2%   0.3%      .  <0.4%>|
--+-----------------------------------+
(row = reference; col = test)

Model Evaluation
        Precision       Recall          F1
0             0.100      0.235       0.140
1             0.111      0.426       0.176
2             0.889      0.567       0.692
3             0.177      0.400       0.245
4             0.091      0.188       0.122
```

3.1.3 Sentiment Lexicon Features result

Model accuracy generated by using sentiment lexicon is the highest among four models, which is 0.558.

```
Accuracy with SL_featuresets :
Accuracy :
0.558
Confusion matrix
  |   0   1   2   3   4 |
--+---------------------+
0 |  <5> 10  11   6   . |
1 |   6 <34>116  24   6 |
2 |   5  25<453> 32   7 |
3 |   4  14 113 <60> 15 |
4 |   4   4  23  17  <6>|
--+---------------------+
(row = reference; col = test)

Confusion matrix in percentage
  |      2      3      1      4      0 |
--+------------------------------------+
2 | <45.3%>  3.2%   2.5%   0.7%   0.5% |
3 |  11.3%  <6.0%>  1.4%   1.5%   0.4% |
1 |  11.6%   2.4%  <3.4%>  0.6%   0.6% |
4 |   2.3%   1.7%   0.4%  <0.6%>  0.4% |
0 |   1.1%   0.6%   1.0%      .  <0.5%>|
--+------------------------------------+
(row = reference; col = test)

Model Evaluation
        Precision       Recall          F1
0             0.156      0.208       0.179
1             0.183      0.391       0.249
2             0.868      0.633       0.732
3             0.291      0.432       0.348
4             0.111      0.176       0.136
```

3.2 Combined Features result

```
Accuracy with combined featuresets :
Training and testing a classifier
Accuracy of classifier :
0.5
---------------------------------------------------
Showing most informative features

The confusion matrix
  | 1 2 3 |
--+-------+
1 |<.>1 . |
2 | .<3>. |
3 | . 2<.>|
--+-------+
(row = reference; col = test)
```

4. SciKit Learn Classifier

In this section, we used Decision Tree classifier rather than Naive Bayes classifier with all the features we created above. For the decision tree classifier, we defined the maximum depth of tree to be 12 (max_depth=12), and minimum number of samples with a node to be 2 (mini_samples_split=2). Since we have already known that unigram without any preprocessing feature will always give the lowest accuracy among all other features. We would only use Decision Tree classifier for unigram with preprocess feature, bigram feature and sentiment lexicon feature.

4.1 Unigram Feature Result

```
C:\Users\raksh\Downloads\FinalProject_Data\FinalProjectData\kagglemoviereviews>python sk_learn_decision_tree.py "C:\Users\raksh\Downloads\FinalProject_Data
\FinalProjectData\kagglemoviereviews\corpus\features.csv"
Shape of feature data - num instances with num features + class label
(15000, 201)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.21      0.01      0.02       712
         neu       0.52      0.97      0.68      7548
         pos       0.28      0.01      0.02       901
        sneg       0.39      0.06      0.10      2636
        spos       0.44      0.04      0.08      3203

    accuracy                           0.51     15000
   macro avg       0.37      0.22      0.18     15000
weighted avg       0.45      0.51      0.38     15000



Predicted  neg    neu  pos  sneg  spos    All
Actual
neg          7    621    3    67    14    712
neu          7   7353    7   114    67   7548
```

```
pos          2    818    7    15    59    901
sneg        15   2427    2   157    35   2636
spos         2   3005    6    53   137   3203
All         33  14224   25   406   312  15000
```

4.2 Bigram Feature Result

```
C:\Users\raksh\Downloads\FinalProject_Data\FinalProjectData\kagglemoviereviews>python sk_learn_decision_tree.py "C:\Users\raksh\Downloads\FinalProject_Data
\FinalProjectData\kagglemoviereviews\corpus\features_biagram.csv"
Shape of feature data - num instances with num features + class label
(15000, 701)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.41      0.02      0.05       686
         neu       0.52      0.98      0.68      7635
         pos       0.51      0.04      0.07       896
        sneg       0.30      0.01      0.02      2603
        spos       0.42      0.06      0.10      3180

    accuracy                           0.52     15000
   macro avg       0.43      0.22      0.18     15000
weighted avg       0.46      0.52      0.38     15000



Predicted  neg    neu  pos  sneg  spos    All
Actual
neg         17    626    2    25    16    686
neu          7   7515    6    18    89   7635
```

```
pos          1    775   32     2    86    896
sneg        15   2499    4    24    61   2603
spos         1   2968   19    11   181   3180
All         41  14383   63    80   433  15000
```

4.3 Sentiment Lexicon Feature Result

```
C:\Users\raksh\Downloads\FinalProject_Data\FinalProjectData\kagglemoviereviews>python sk_learn_decision_tree.py "C:\Users\raksh\Downloads\FinalProject_Data
\FinalProjectData\kagglemoviereviews\corpus\features_SL.csv"
Shape of feature data - num instances with num features + class label
(15000, 203)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.32      0.06      0.10       686
         neu       0.62      0.83      0.71      7635
         pos       0.43      0.05      0.08       896
        sneg       0.35      0.10      0.15      2603
        spos       0.39      0.47      0.43      3180

    accuracy                           0.54     15000
   macro avg       0.42      0.30      0.29     15000
weighted avg       0.50      0.54      0.49     15000


Predicted  neg    neu  pos  sneg  spos    All
Actual
neg         39    385    2   122   138    686
neu         24   6343   17   220  1031   7635
pos          2    232   41    22   599    896
sneg        46   1783    7   253   514   2603
spos        10   1557   29   103  1481   3180
All        121  10300   96   720  3763  15000
```

5. Comparison and Analysis

	Summary Method	Measurement	Naïve Bayes	Decision Tree
Unigram Features with Preprocessing	Weighted Average	precision	0.2968	0.45
		Recall	0.3518	0.51
		F1-score	0.283	0.38
Bigram Features	Weighted Average	precision	0.2748	0.46
		Recall	0.3632	0.52
		F1-score	0.275	0.38
Sentiment Lexicon	Weighted Average	precision	0.3206	0.5
		Recall	0.368	0.54
		F1-score	0.3288	0.49

Table 1. Summary Table

Comparing Naive Bayes and Decision Tree model (Table 1), it is obvious that Decision Tree generated higher accuracies regardless of the type of feature set we used. In this case, if we compare the precision values, accuracies generated by decision tree classifiers are almost 51.61% higher than the ones generated by Naive Bayes.

Comparing among three different models, we can see that sentiment lexicon method generated highest precision accuracy out of three different feature sets. If we compare the precision values for three models, it suggests that sentiment lexicon generated had 8% improvement in precision from other two models.

In terms of the three different measurements, we can see that the recall index gives the highest output for both classifiers and all three feature sets. Given the fact that recall measurement is calculated by TP/(TP+FP), our output suggests that the model has captured more True Positive results out of all the True Positive and False Positive results. Also one of the reasons that our model generated high TP results is due to the large amount of 'neutral'-2 movie reviews in the data. Among all 15,000 movie reviews we used for the training model, 7,635 reviews were labeled as 'neutral-2', which made our model distorted and not evenly distributed among different labels, even though we had randomly selected the data.

6. Lesson Learned

- One of the challenges we first encountered when doing the project was to understand how to implement all the processes such as defining feature sets, preprocessing, etc, and combine all these functions and process through the processkaggle() function. We think working on this project really helped us to learn more useful coding techniques for NLP projects.
- Learned how to use classifiers other than Naive Bayes, specifically we learned the utilization of Sciki learn. Even though we only included a decision tree classifier for this project, we also learned other classifiers such as logistic regression classifier when we were processing the sklearn_model_performance.py file.
- By doing this project, we were able to connect and use everything we learn from this semester, rather than having shattered pieces of knowledge. We think we were able perceive nltk tools from a broader scope and used it wisely by doing this project.
- Work distribution:
    - Zhongwei Chen: Data Preprocessing, generate unigram(with/without preprocessing), bigram, and naive bayes classifier, model evaluation for NB classifier, write report
    - Rakshith: Data Preprocessing, combined features, tokenization, sciki learn classifier, model evaluation for sciki classifier, write report

Reference

Screenshots for sentiment lexicon code:

```
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)
    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)
    if 'positivecount' not in features:
        features['positivecount'] = 0
    if 'negativecount' not in features:
        features['negativecount'] = 0
    return features
```