Final Project –CIS 668/IST 664

Classification of Kaggle Movie Reviews

Student: *Professor:* Michael Larche

Midhuneshwar Kandasamy Kanivalavan

SUID: 351564889

Mail: mkandasa@syr.edu

Table of Contents

Introduction

The primary objective of this project is to classify and predict the sentiment of move reviews. The scale of sentiment classification is 0-4 with equal intervals. The details of scale for sentiment follows,

Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

Based on the classification intervals, it can be said that the scaling is biased towards negative and positive reviews leaving neutral sentiment scoring a single interval classification.

The dataset was produced for the Kaggle competition, described here data. This dataset data from the sentiment analysis by Socher et al, detailed at this web site: http://nlp.stanford.edu/sentiment/.
The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site. Socher's group used crowd-sourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: "negative", "somewhat negative", "neutral", "somewhat positive", "positive".

Data Description

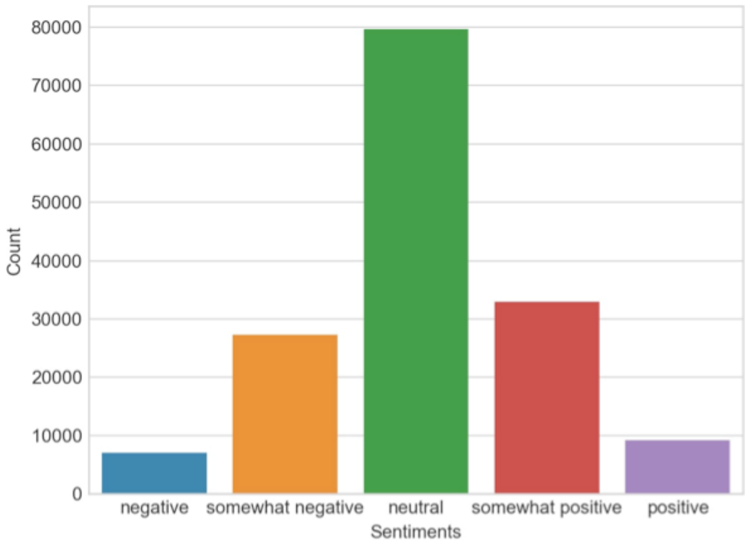
The data provided for this project has two data files namely 'train.csv' and 'test.csv'. The details of train data are,

- 1,56,060 rows of movie reviews. Each row represents review for a movie
- 4 columns, these are

Column_name	Description
PhraseId	Represents id number of a phrase
SentenceId	Represents id for each sentence

Column_name	Description
Phrase	Represents the review for movie
Phrase Sentiment	Represents the sentiment for each review

Histogram of count of sentiments for train data,



The details of test data are ,

- 66,292 rows of movie reviews. Each row represents review for a movie
- 3 columns, these are

Column_name	Description
PhraseId	Represents id number of a phrase
SentenceId	Represents id for each sentence
Phrase	Represents the review for movie

It must be noted that test data has no sentiment, which means that sentiment column is the outcome/target label which must be predicted. Therefore, for all of our analysis Phraseid, SentenceID and Phrase will be features and Sentiment will be target label.

Reading the Train Data

The first step is to read the data into python environment. This is done using he code below,

```
# function to read kaggle training file, train and test a classifier
def processkaggle(dirPath,limitStr):
    # convert the limit argument from a string to an int
    limit = int(limitStr)

    os.chdir(dirPath)

    f = open('./train.tsv', 'r')
    # loop over lines in the file and use the first limit of them
    phrasedata = []
    for line in f:
        # ignore the first line starting with Phrase and read all lines
        if (not line.startswith('Phrase')):
            # remove final end of line character
            line = line.strip()
            # each line has 4 items separated by tabs
            # ignore the phrase and sentence ids, and keep the phrase and sentiment
            phrasedata.append(line.split('\t')[2:4])
```

Here, we are writing a function which will loop through each line of the csv line and create a list called 'phrasedata'. Further, we have a option to limit the number of rows we can include in the loaded data. This limiting the loaded data is helpful when we want to run the analysis on a limited number of the instances. Since we are trying to classify and then predict, the better approach would be to load all the available data, however, the constraint is the computing power. Though, we have a constraint we tried to run our analysis on entire dataset.

Tokenization and Filtering

Once we have loaded the entire train dataset, we need to tokenize each word of the a phrase to proceed with our analysis. Using these tokenized phrases, we can create multiple different feature sets. The code developed is below, run_sklearn_model_performance.py × run_sklearn_model_performance_Decision_Tree.py × modified_classifyKaggle.py ×

```python
    # pick a random sample of length limit because of phrase overlapping sequences
    random.shuffle(phrasedata)
    phraselist = phrasedata[:limit]

    print('Read', len(phrasedata), 'phrases, using', len(phraselist), 'random phrases')
    #for phrase in phraselist[:10]:
      #print (phrase)

    # create list of phrase documents as (list of words, label)
    phrasedocs = []
    phrasedocs_without = []
    # add all the phrases
    for phrase in phraselist:

      #without preprocessing
      tokens = nltk.word_tokenize(phrase[0])
      phrasedocs_without.append((tokens, int(phrase[1])))

      # with pre processing
      tokenizer = RegexpTokenizer(r'\w+')
      phrase[0] = pre_processing_documents(phrase[0])
      tokens = tokenizer.tokenize(phrase[0])
      phrasedocs.append((tokens, int(phrase[1])))

    # possibly filter tokens
    normaltokens = get_words_from_phasedocs_normal(phrasedocs_without)
    preprocessedTokens = get_words_from_phasedocs(phrasedocs)

    # continue as usual to get all words and create word features
    word_features = get_word_features(normaltokens)
    featuresets_without_preprocessing = [(normal_features(d, word_features), s) for (d, s) in phrasedocs_without]
```

processkaggle()

From the code it can be observed that we are leveraging tokenize function of nltk to tokenize the phrases. Before tokenization, we are preprocessing for all those tokens using a regular expression '\w+', this regex pattern matches any alphanumeric with underscore with one or more occurrences. Once we have tokenized the data, we create two different lists *'normaltokens'* which includes word phrases without filtering and the second list is *'preprocessedTokens'* which includes word phrases with filtering.

To perform the above said tokenization and filtering we have defined the following functions,

a. Pre processing documents

This function splits each phrase into individual lines and converts them into lower case. Later, this lower case lines are run through regular expressions wherein any word with punctuation marks is removed and stored in a list called *'word_list'*. Further this list is checked for any stop words i.e. all the stop words are removed from the list and a list called *'final_word_list'* is created.

```python
def pre_processing_documents(document):
    # "Pre_processing_documents"
    # "create list of lower case words"
    word_list = re.split('\s+', document.lower())
    # punctuation and numbers to be removed
    punctuation = re.compile(r'[-.?!/\%@,":;()|0-9]')
    word_list = [punctuation.sub("", word) for word in word_list]
    final_word_list = []
    for word in word_list:
        if word not in newstopwords:
            final_word_list.append(word)
    line = " ".join(final_word_list)
    return line
```

b. Retrieving words/tokens

There are three functions defined in this code. The first function returns list which contains tokens/words from the documents where the length is greater than 3. The second function returns a list of all words with corresponding sentiment. The third function returns all lines from the tokens.

```python
def get_words_from_phasedocs(docs):
    all_words = []
    for (words, sentiment) in docs:
        # more than 3 length
        possible_words = [x for x in words if len(x) >= 3]
        all_words.extend(possible_words)
    return all_words

def get_words_from_phasedocs_normal(docs):
    all_words = []
    for (words, sentiment) in docs:
        all_words.extend(words)
    return all_words

# get all words from tokens
def get_words_from_test(lines):
    all_words = []
    for id,words in lines:
        all_words.extend(words)
    return all_words
```

Creating Feature Sets

The next key step in classification tasks in NLP is to create features from raw tokens. We had to define multiple features generating functions such as bag of words(BOW), bigrams, sentiment lexicons, negation words, POS tag features etc. Let us try to understand each of these functions,

a. Bag of Words (BOW)/Unigram features

For creating unigrams we defined two different functions. The first function creates a list of most repeated 200 words from the *'wordlist'* which has processed tokens. The second function creates unique list of words from the documents and returns them as features.

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


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

b. Bigram Features

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

```
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['contains({})'.format(word)] = (word in document_words)
  for bigram in bigram_features:
    features['bigram({} {})'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
  return features
def get_biagram_features(tokens):
  bigram_measures = nltk.collocations.BigramAssocMeasures()
  finder = BigramCollocationFinder.from_words(tokens,window_size=3)
  #finder.apply_freq_filter(6)
  bigram_features = finder.nbest(bigram_measures.chi_sq, 3000)
  return bigram_features[:500]
```

c. Sentiment Lexicons

We will first read in the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh in the MPQA project. Although these words are often used as features themselves or in conjunction with other information, we will create two features that involve counting the positive and negative subjectivity words present in each document. I copy and pasted the definition of the readSubjectivity function from the Subjectivity.py module which is provided by Professor. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength and polarity. A feature extraction function that has all the word features as before, but also has two features 'positivecount' and 'negativecount'. These features contain counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice.

d. Negation word features:

Negation of opinions is an important part of sentimental classification. Here I tried a simple strategy which professor explained in Lab-10. I look for negation words "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t". For example, my first document has the following words: if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself', One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation. I followed the first strategy here, and I go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.

```
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather',
                 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']
def NOT_features(document, word_features, negationwords):
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = False
    features['contains(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):
      i += 1
      features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
    else:
      if ((i + 3) < len(document)) and (word.endswith('n') and document[i+1] == "'" and document[i+2] == 't'):
        i += 3
        features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
      else:
        features['contains({})'.format(word)] = (word in word_features)
  return features
```

e. POS features

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

```
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
```

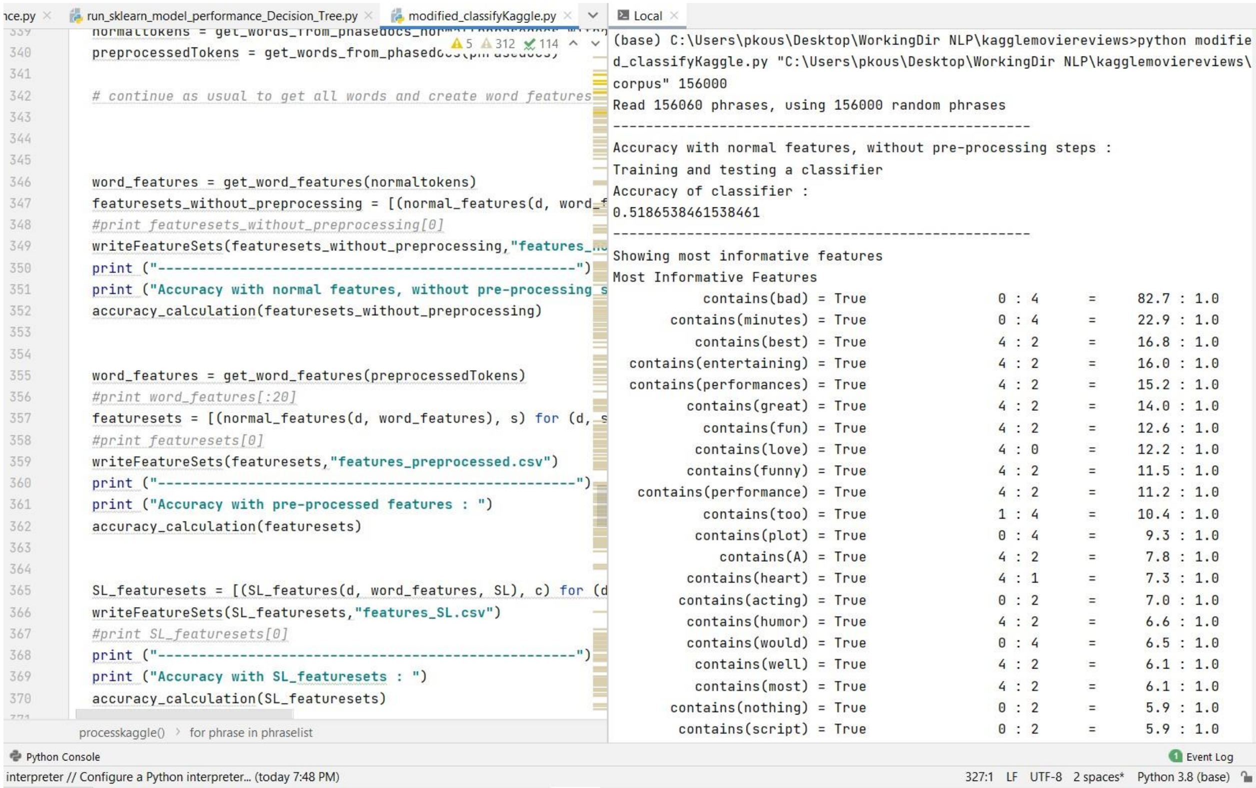
Classification: Naïve Bayes Classifier

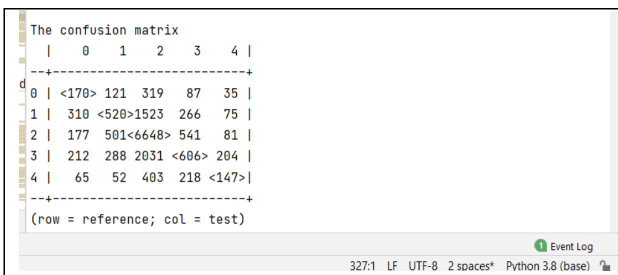
In this section we will discuss the classifier obtained for each of the feature sets created by running the above functions. Before delving into details of outputs for each of the feature sets, let's look at the accuracies obtained for each of them,

Feature Set	Accuracy achieved
Normal Features without preprocessing	51.86%
Bigram Features	53.75%
Negation word Features	55.47%
Preprocessed Features	53.75%
Sentiment Lexicon features	55.08%

Since, we have established the accuracies for each of the feature sets, let us now focus on the confusion matrix and make some definitive conclusions.

a. Normal Features without preprocessing

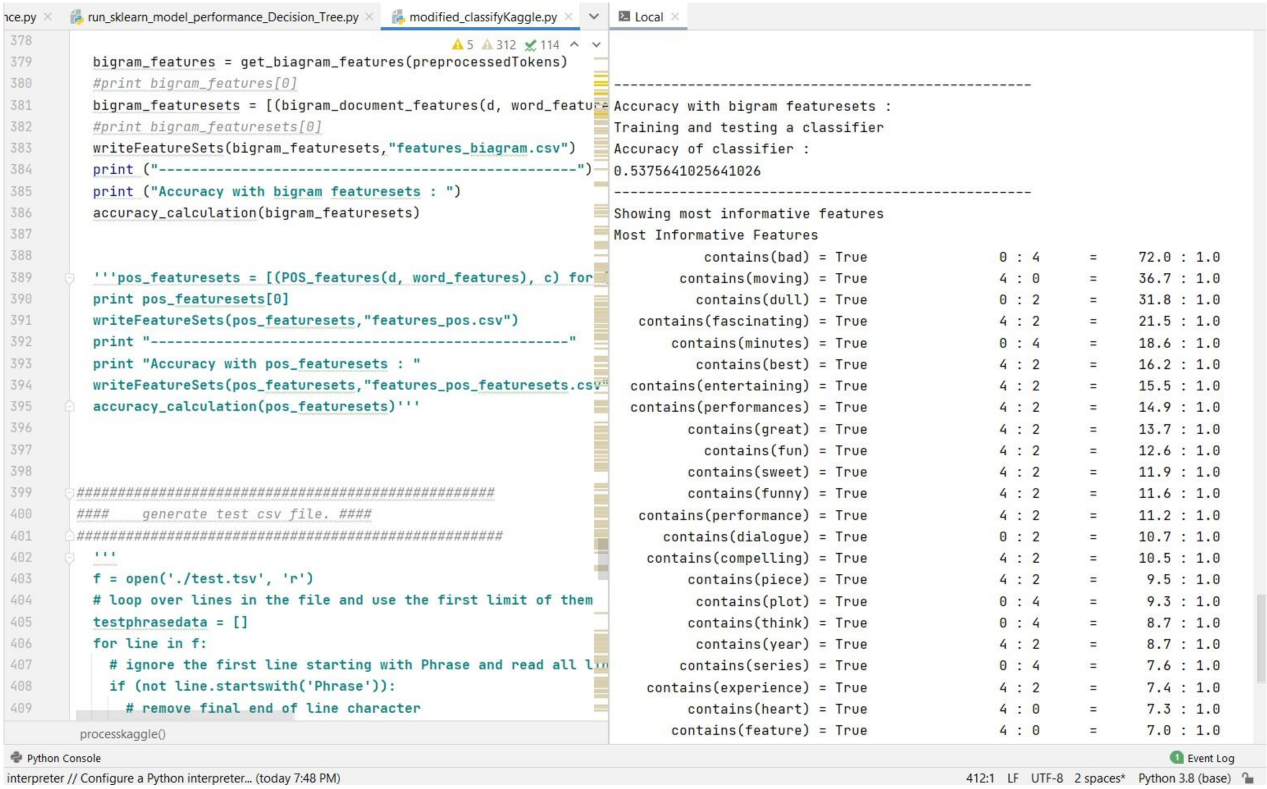




Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

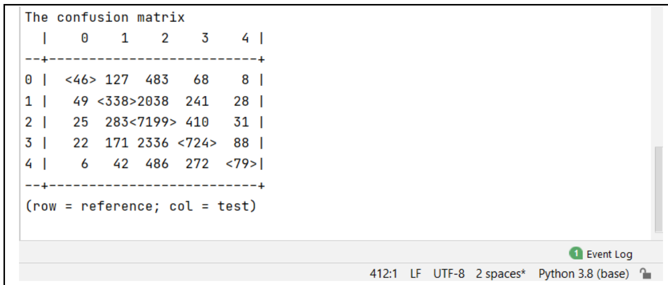
From the above confusion matrix, it can be said that most of neutral sentiments are predicted accurately and the least accurately predicted label is positive sentiment labels.

b. Bigram Features



```
bigram_features = get_biagram_features(preprocessedTokens)
#print bigram_features[0]
bigram_featuresets = [(bigram_document_features(d, word_featu
#print bigram_featuresets[0]
writeFeatureSets(bigram_featuresets,"features_biagram.csv")
print ("--------------------------------------------------")
print ("Accuracy with bigram featuresets : ")
accuracy_calculation(bigram_featuresets)


'''pos_featuresets = [(POS_features(d, word_features), c) for
print pos_featuresets[0]
writeFeatureSets(pos_featuresets,"features_pos.csv")
print "--------------------------------------------------"
print "Accuracy with pos_featuresets : "
writeFeatureSets(pos_featuresets,"features_pos_featuresets.csv"
accuracy_calculation(pos_featuresets)'''


####################################################
####    generate test csv file. ####
####################################################
'''
f = open('./test.tsv', 'r')
# loop over lines in the file and use the first limit of them
testphrasedata = []
for line in f:
  # ignore the first line starting with Phrase and read all li
  if (not line.startswith('Phrase')):
    # remove final end of line character
```

```
--------------------------------------------------
Accuracy with bigram featuresets :
Training and testing a classifier
Accuracy of classifier :
0.5375641025641026
--------------------------------------------------
Showing most informative features
Most Informative Features
           contains(bad) = True          0 : 4    =     72.0 : 1.0
        contains(moving) = True          4 : 0    =     36.7 : 1.0
          contains(dull) = True          0 : 2    =     31.8 : 1.0
   contains(fascinating) = True          4 : 2    =     21.5 : 1.0
       contains(minutes) = True          0 : 4    =     18.6 : 1.0
          contains(best) = True          4 : 2    =     16.2 : 1.0
  contains(entertaining) = True          4 : 2    =     15.5 : 1.0
  contains(performances) = True          4 : 2    =     14.9 : 1.0
         contains(great) = True          4 : 2    =     13.7 : 1.0
           contains(fun) = True          4 : 2    =     12.6 : 1.0
         contains(sweet) = True          4 : 2    =     11.9 : 1.0
         contains(funny) = True          4 : 2    =     11.6 : 1.0
   contains(performance) = True          4 : 2    =     11.2 : 1.0
      contains(dialogue) = True          0 : 2    =     10.7 : 1.0
    contains(compelling) = True          4 : 2    =     10.5 : 1.0
         contains(piece) = True          4 : 2    =      9.5 : 1.0
          contains(plot) = True          0 : 4    =      9.3 : 1.0
         contains(think) = True          0 : 4    =      8.7 : 1.0
          contains(year) = True          4 : 2    =      8.7 : 1.0
        contains(series) = True          0 : 4    =      7.6 : 1.0
    contains(experience) = True          4 : 2    =      7.4 : 1.0
         contains(heart) = True          4 : 0    =      7.3 : 1.0
       contains(feature) = True          4 : 0    =      7.0 : 1.0
```

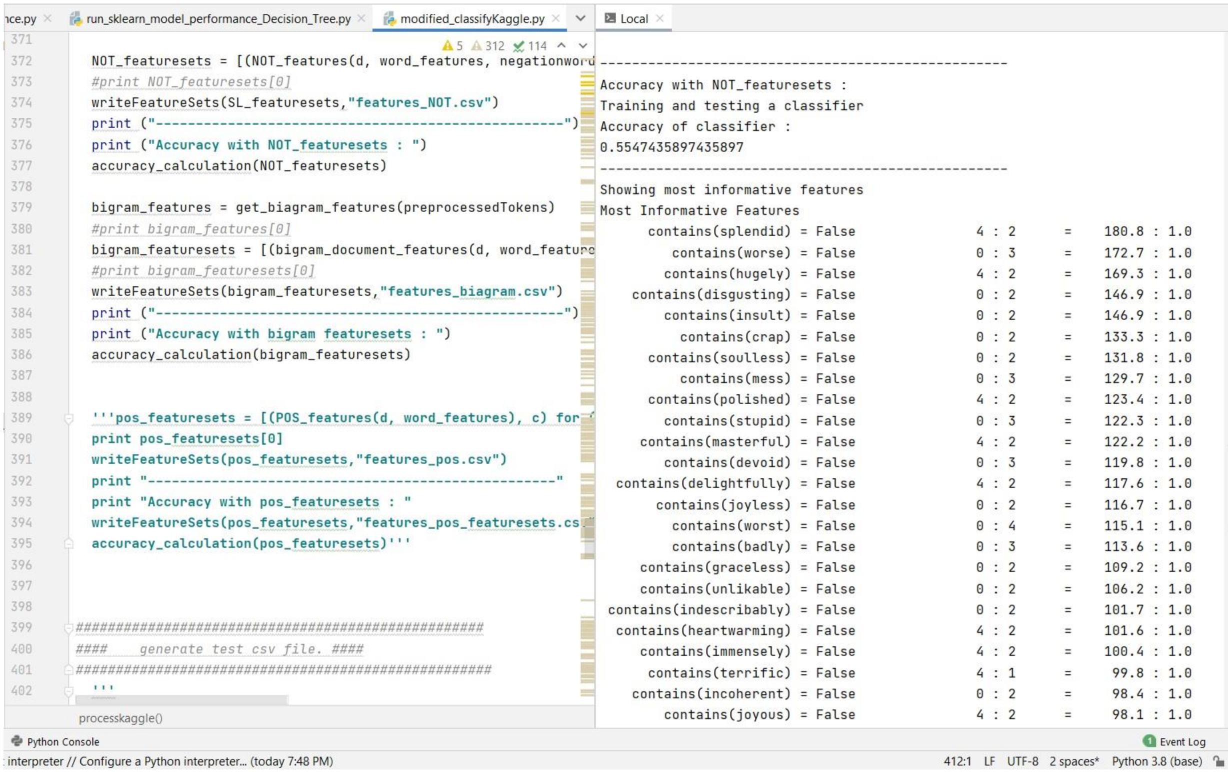


```
The confusion matrix
  |    0    1    2    3    4 |
--+-------------------------+
0 |  <46> 127  483   68    8 |
1 |   49 <338>2038  241   28 |
2 |   25  283<7199> 410   31 |
3 |   22  171 2336 <724>  88 |
4 |    6   42  486  272  <79>|
--+-------------------------+
(row = reference; col = test)
```

Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

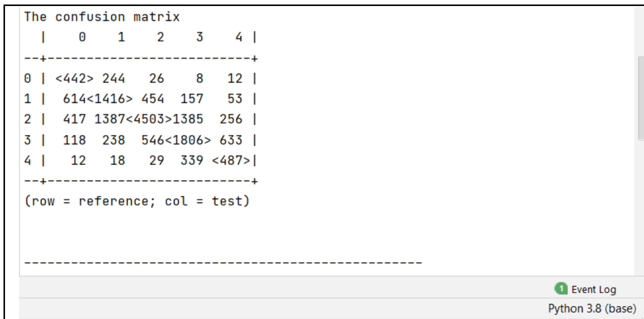
From the above confusion matrix, it can be said that most of neutral sentiments and strong negative labels are predicted accurately for most of the training data and the least accurately predicted label is positive sentiment labels.

c. Negation word Features



```
NOT_featuresets = [(NOT_features(d, word_features, negationword
#print NOT_featuresets[0]
writeFeatureSets(SL_featuresets,"features_NOT.csv")
print ("--------------------------------------------------")
print ("Accuracy with NOT_featuresets : ")
accuracy_calculation(NOT_featuresets)

bigram_features = get_biagram_features(preprocessedTokens)
#print bigram_features[0]
bigram_featuresets = [(bigram_document_features(d, word_features
#print bigram_featuresets[0]
writeFeatureSets(bigram_featuresets,"features_biagram.csv")
print ("--------------------------------------------------")
print ("Accuracy with bigram featuresets : ")
accuracy_calculation(bigram_featuresets)


'''pos_featuresets = [(POS_features(d, word_features), c) for
print pos_featuresets[0]
writeFeatureSets(pos_featuresets,"features_pos.csv")
print "--------------------------------------------------"
print "Accuracy with pos_featuresets : "
writeFeatureSets(pos_featuresets,"features_pos_featuresets.cs
accuracy_calculation(pos_featuresets)'''



#####################################################
####    generate test csv file. ####
#####################################################
'''
```

```
--------------------------------------------------
Accuracy with NOT_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.5547435897435897
--------------------------------------------------
Showing most informative features
Most Informative Features
      contains(splendid) = False          4 : 2    =    180.8 : 1.0
         contains(worse) = False          0 : 3    =    172.7 : 1.0
        contains(hugely) = False          4 : 2    =    169.3 : 1.0
    contains(disgusting) = False          0 : 2    =    146.9 : 1.0
        contains(insult) = False          0 : 2    =    146.9 : 1.0
          contains(crap) = False          0 : 2    =    133.3 : 1.0
      contains(soulless) = False          0 : 2    =    131.8 : 1.0
          contains(mess) = False          0 : 3    =    129.7 : 1.0
      contains(polished) = False          4 : 2    =    123.4 : 1.0
        contains(stupid) = False          0 : 3    =    122.3 : 1.0
     contains(masterful) = False          4 : 2    =    122.2 : 1.0
        contains(devoid) = False          0 : 3    =    119.8 : 1.0
  contains(delightfully) = False          4 : 2    =    117.6 : 1.0
       contains(joyless) = False          0 : 2    =    116.7 : 1.0
         contains(worst) = False          0 : 4    =    115.1 : 1.0
         contains(badly) = False          0 : 3    =    113.6 : 1.0
     contains(graceless) = False          0 : 2    =    109.2 : 1.0
     contains(unlikable) = False          0 : 2    =    106.2 : 1.0
 contains(indescribably) = False          0 : 2    =    101.7 : 1.0
  contains(heartwarming) = False          4 : 2    =    101.6 : 1.0
     contains(immensely) = False          4 : 2    =    100.4 : 1.0
      contains(terrific) = False          4 : 1    =     99.8 : 1.0
    contains(incoherent) = False          0 : 2    =     98.4 : 1.0
        contains(joyous) = False          4 : 2    =     98.1 : 1.0
```

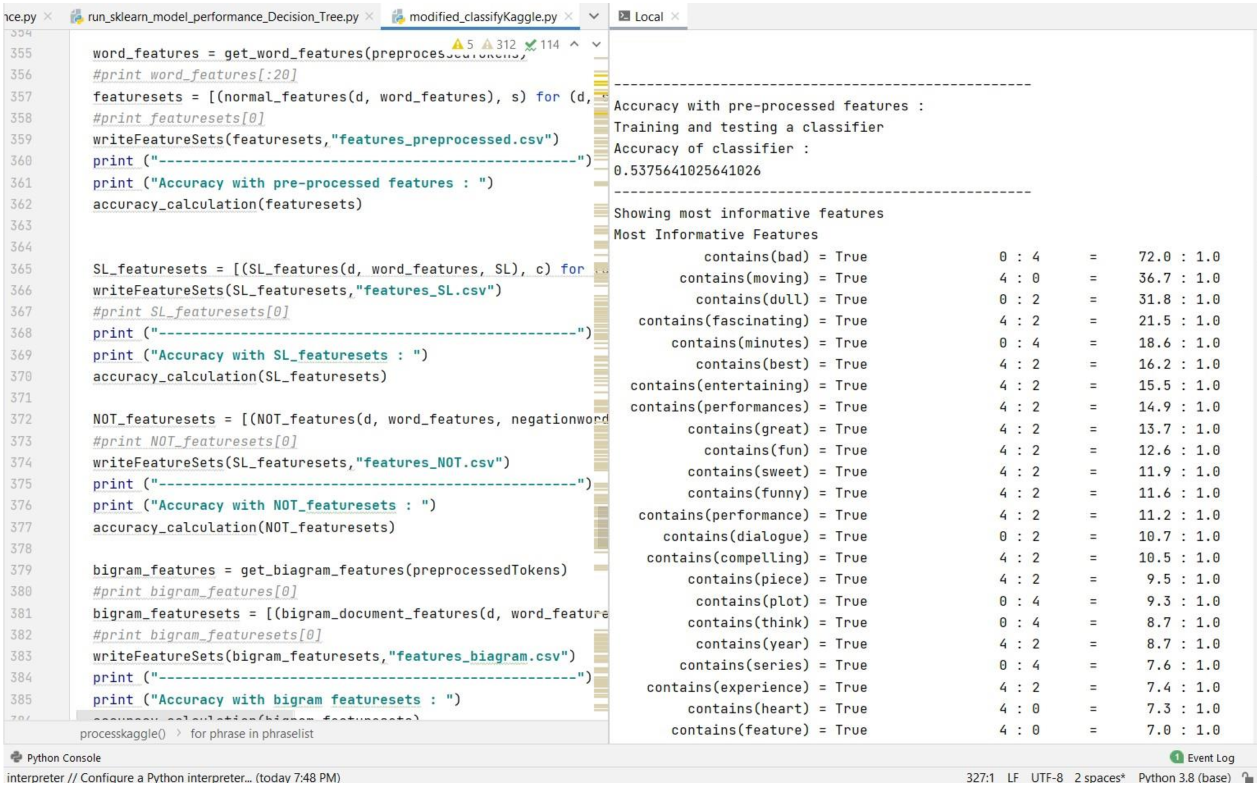


```
The confusion matrix
  |    0    1    2    3    4 |
--+--------------------------+
0 | <442> 244   26    8   12 |
1 |  614<1416> 454  157   53 |
2 |  417 1387<4503>1385  256 |
3 |  118  238  546<1806> 633 |
4 |   12   18   29  339 <487>|
--+--------------------------+
(row = reference; col = test)

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

Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

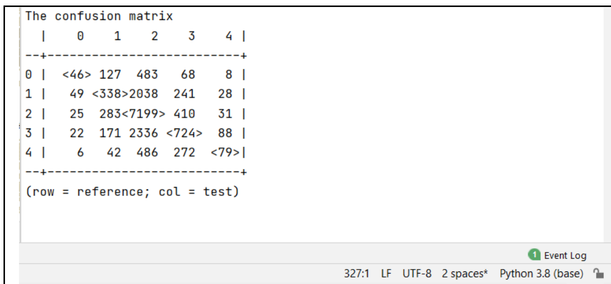
From the above confusion matrix, it can be said that most of neutral and negative labels are predicted accurately.

d. Preprocessed Features



```
word_features = get_word_features(preprocessedTokens)
#print word_features[:20]
featuresets = [(normal_features(d, word_features), s) for (d, ...
#print featuresets[0]
writeFeatureSets(featuresets,"features_preprocessed.csv")
print ("----------------------------------------------------")
print ("Accuracy with pre-processed features : ")
accuracy_calculation(featuresets)


SL_featuresets = [(SL_features(d, word_features, SL), c) for ...
writeFeatureSets(SL_featuresets,"features_SL.csv")
#print SL_featuresets[0]
print ("----------------------------------------------------")
print ("Accuracy with SL_featuresets : ")
accuracy_calculation(SL_featuresets)

NOT_featuresets = [(NOT_features(d, word_features, negationword...
#print NOT_featuresets[0]
writeFeatureSets(SL_featuresets,"features_NOT.csv")
print ("----------------------------------------------------")
print ("Accuracy with NOT_featuresets : ")
accuracy_calculation(NOT_featuresets)

bigram_features = get_biagram_features(preprocessedTokens)
#print bigram_features[0]
bigram_featuresets = [(bigram_document_features(d, word_feature...
#print bigram_featuresets[0]
writeFeatureSets(bigram_featuresets,"features_biagram.csv")
print ("----------------------------------------------------")
print ("Accuracy with bigram featuresets : ")
```

```
--------------------------------------------------
Accuracy with pre-processed features :
Training and testing a classifier
Accuracy of classifier :
0.5375641025641026
--------------------------------------------------
Showing most informative features
Most Informative Features
          contains(bad) = True              0 : 4      =     72.0 : 1.0
       contains(moving) = True              4 : 0      =     36.7 : 1.0
         contains(dull) = True              0 : 2      =     31.8 : 1.0
  contains(fascinating) = True              4 : 2      =     21.5 : 1.0
      contains(minutes) = True              0 : 4      =     18.6 : 1.0
         contains(best) = True              4 : 2      =     16.2 : 1.0
 contains(entertaining) = True              4 : 2      =     15.5 : 1.0
 contains(performances) = True              4 : 2      =     14.9 : 1.0
        contains(great) = True              4 : 2      =     13.7 : 1.0
          contains(fun) = True              4 : 2      =     12.6 : 1.0
        contains(sweet) = True              4 : 2      =     11.9 : 1.0
        contains(funny) = True              4 : 2      =     11.6 : 1.0
  contains(performance) = True              4 : 2      =     11.2 : 1.0
     contains(dialogue) = True              0 : 2      =     10.7 : 1.0
   contains(compelling) = True              4 : 2      =     10.5 : 1.0
        contains(piece) = True              4 : 2      =      9.5 : 1.0
         contains(plot) = True              0 : 4      =      9.3 : 1.0
        contains(think) = True              0 : 4      =      8.7 : 1.0
         contains(year) = True              4 : 2      =      8.7 : 1.0
       contains(series) = True              0 : 4      =      7.6 : 1.0
   contains(experience) = True              4 : 2      =      7.4 : 1.0
        contains(heart) = True              4 : 0      =      7.3 : 1.0
      contains(feature) = True              4 : 0      =      7.0 : 1.0
```



```
The confusion matrix
  |    0    1    2    3    4 |
--+--------------------------+
0 |  <46> 127  483   68    8 |
1 |   49 <338>2038  241   28 |
2 |   25  283<7199> 410   31 |
3 |   22  171 2336 <724>  88 |
4 |    6   42  486  272  <79>|
--+--------------------------+
(row = reference; col = test)
```

Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

From the above confusion matrix, neutral sentiment labels are predicted accurately.

e. Sentiment Lexicon features

```
SL_featuresets = [(SL_features(d, word_features, SL), c) for (d
writeFeatureSets(SL_featuresets,"features_SL.csv")
#print SL_featuresets[0]
print ("------------------------------------------------------")
print ("Accuracy with SL_featuresets : ")
accuracy_calculation(SL_featuresets)

NOT_featuresets = [(NOT_features(d, word_features, negationword
#print NOT_featuresets[0]
writeFeatureSets(SL_featuresets,"features_NOT.csv")
print ("------------------------------------------------------")
print ("Accuracy with NOT_featuresets : ")
accuracy_calculation(NOT_featuresets)

bigram_features = get_biagram_features(preprocessedTokens)
#print bigram_features[0]
bigram_featuresets = [(bigram_document_features(d, word_feature
#print bigram_featuresets[0]
writeFeatureSets(bigram_featuresets,"features_biagram.csv")
print ("------------------------------------------------------")
print ("Accuracy with bigram featuresets : ")
accuracy_calculation(bigram_featuresets)


'''pos_featuresets = [(POS_features(d, word_features), c) for (
print pos_featuresets[0]
writeFeatureSets(pos_featuresets,"features_pos.csv")
print "------------------------------------------------------"
print "Accuracy with pos_featuresets : "
writeFeatureSets(pos_featuresets,"features_pos_featuresets.csv
```

```
------------------------------------------------------
Accuracy with SL_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.5508974358974359
------------------------------------------------------
Showing most informative features
Most Informative Features
           contains(bad) = True           0 : 4    =     72.0 : 1.0
           positivecount = 11             4 : 2    =     71.7 : 1.0
           positivecount = 10             4 : 2    =     44.9 : 1.0
           negativecount = 11             0 : 2    =     41.4 : 1.0
        contains(moving) = True           4 : 0    =     36.7 : 1.0
          contains(dull) = True           0 : 2    =     31.8 : 1.0
           negativecount = 9              0 : 2    =     29.6 : 1.0
           positivecount = 9              4 : 1    =     28.7 : 1.0
           negativecount = 8              0 : 2    =     27.3 : 1.0
   contains(fascinating) = True           4 : 2    =     21.5 : 1.0
       contains(minutes) = True           0 : 4    =     18.6 : 1.0
           negativecount = 10             0 : 2    =     17.9 : 1.0
           positivecount = 12             4 : 2    =     17.8 : 1.0
          contains(best) = True           4 : 2    =     16.2 : 1.0
           positivecount = 8              4 : 2    =     16.1 : 1.0
           positivecount = 6              4 : 2    =     15.7 : 1.0
           positivecount = 7              4 : 2    =     15.5 : 1.0
  contains(entertaining) = True           4 : 2    =     15.5 : 1.0
  contains(performances) = True           4 : 2    =     14.9 : 1.0
           negativecount = 7              0 : 2    =     14.8 : 1.0
         contains(great) = True           4 : 2    =     13.7 : 1.0
           contains(fun) = True           4 : 2    =     12.6 : 1.0
           positivecount = 5              4 : 2    =     12.6 : 1.0
         contains(sweet) = True           4 : 2    =     11.9 : 1.0
         contains(funny) = True           4 : 2    =     11.6 : 1.0
```

```
The confusion matrix
  |    0    1    2    3    4 |
--+--------------------------+
0 |  <90> 162  370   89   21 |
1 |  108 <478>1692  357   59 |
2 |   50  389<6732> 719   58 |
3 |   29  196 1781<1138> 197 |
4 |   10   40  288  391 <156>|
--+--------------------------+
(row = reference; col = test)
```

Classification Scale	Sentiment
0	Negative
1	Strong Negative
2	Neutral
3	Positive
4	Strong Positive

From the above confusion matrix, neutral and positive sentiment labels are accurately predicted.

Combination of features sets:

In this section, we will try to create a new function which will combine different feature sets such as Sentiment Lexicons, Bigram features and unigram features. The code follows,

```
def combined_document_features(document, word_features, SL, bigram_features):
  document_words = set(document)
  document_bigrams = nltk.bigrams(document)
  features = {}

  for word in document_words:
        # features object
    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
```

From the above code we tried to create one single feature set which is a combination of bigrams and sentiment lexicons.

```
Accuracy with combined featuresets :
Training and testing a classifier
Accuracy of classifier :
0.8
-----------------------------------------------
```

The accuracy we obtained for combined features is 80%. One of possible reasons for such high accuracy is because we are capturing the non-linearity in the data by creating and combining multiple features because of which the algorithm is maximizing learning.

The main constraint running this function is that the processing time to generate feature sets was time taking i.e. the execution time was high.

Comparative Analysis of Logistic Regression and Decision Tree classifier:

In this section we will try to use Sci-kit learn algorithms to classify the sentiments. Since our outcome labels are like classification type of labels, we have decided to implement Logistic Regression and Decision tree classifier algorithms. Further we will run both the algorithms on each of the five feature sets.

Before delving into any details, we will do a comparative analysis of the metrics,

Feature set type	Logistic Regression			Decision Tree Classifier		
	Precision	Recall	f1-score	Precision	Recall	f1-score
Normal Features without preprocessing	0.49	0.47	0.46	0.47	0.53	0.42
Preprocessed Features	0.46	0.44	0.43	0.49	0.52	0.38
Bigram Features	0.46	0.44	0.43	0.49	0.52	0.38
Negation word Features	0.53	0.51	0.51	0.51	0.55	0.50
Sentiment Lexicon features	0.53	0.50	0.51	0.51	0.55	0.50

For Logistic regression classifier, Sentiment Lexicon features performance better in classification compared with other features functions because fewer words are unseen in train data as features. These words or tokens covers on Lexicon dictionary. Plus, we observed recall score lesser than F-measure which is greater than Precision.

For Decision Tree classifier, Sentiment Lexicon feature and Negation word features perform better in classification compared with other features functions. These words or tokens covers on Lexicon dictionary. Plus, we observed recall score higher than F-measure which is lesser than Precision.

Let us now discuss outputs for each of feature set for an algorithm,

Logistic Regression:

The parameters used in logistic regression are,

- Class_weight: Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).
- solver: Algorithm to use in the optimization problem.

- max_iter: Maximum number of iterations taken for the solvers to converge.

For our analysis, we have used the following parameters,

Parameter_Description	Parameter_value
Class_weight	Balanced
Solver	Lbfgs
max_iter	1000

We have done experiments on different feature sets created using Logistic regression algorithm,

a. Normal Features without preprocessing

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklear
n_model_performance.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews\
corpus\features_normal.csv"
Shape of feature data - num instances with num features + class label
(156000, 201)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.15      0.39      0.22      7069
         neu       0.66      0.68      0.67     79548
         pos       0.20      0.48      0.28      9202
        sneg       0.33      0.20      0.25     27263
        spos       0.34      0.17      0.23     32918

    accuracy                           0.47    156000
   macro avg       0.34      0.39      0.33    156000
weighted avg       0.49      0.47      0.46    156000



Predicted    neg    neu    pos   sneg   spos     All
Actual
neg         2761   1681   1165    991    471    7069
neu         5601  54437   5607   6942   6961   79548
pos         1087   2097   4398    492   1128    9202
sneg        5437  10812   2979   5531   2504   27263
spos        3370  13470   7702   2667   5709   32918
All        18256  82497  21851  16623  16773  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Terminal

Event Log

41:39 LF UTF-8 4 spaces Python 3.8 (base)

b. Bigram Features

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklear
n_model_performance.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews\
corpus\features_biagram.csv"
Shape of feature data - num instances with num features + class label
(156000, 701)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.13      0.37      0.20      7069
         neu       0.62      0.67      0.64     79548
         pos       0.19      0.45      0.27      9202
        sneg       0.28      0.16      0.20     27263
        spos       0.35      0.15      0.21     32918

    accuracy                           0.44    156000
   macro avg       0.31      0.36      0.30    156000
weighted avg       0.46      0.44      0.43    156000



Predicted    neg    neu    pos   sneg   spos     All
Actual
neg         2628   2418    845    847    331    7069
neu         7356  52937   6702   6888   5665   79548
pos          934   2574   4162    572    960    9202
sneg        5376  12723   2965   4325   1874   27263
spos        3354  14559   7461   2769   4775   32918
All        19648  85211  22135  15401  13605  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Terminal

Event Log

41:39 LF UTF-8 4 spaces Python 3.8 (base)

c. Negation Word Features

```
Terminal:  Local  +

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklear
n_model_performance.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews\
corpus\features_NOT.csv"
Shape of feature data - num instances with num features + class label
(156000, 203)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.18      0.47      0.26      7069
         neu       0.71      0.67      0.69     79548
         pos       0.25      0.55      0.35      9202
        sneg       0.34      0.26      0.30     27263
        spos       0.41      0.27      0.33     32918

    accuracy                           0.50    156000
   macro avg       0.38      0.45      0.39    156000
weighted avg       0.53      0.50      0.51    156000



Predicted    neg    neu    pos   sneg   spos     All
Actual
neg         3343   1247    505   1480    494    7069
neu         5453  53376   4038   9015   7666   79548
pos          575   1006   5058    410   2153    9202
sneg        6225   9372   2014   7121   2531   27263
spos        2615  10299   8258   2722   9024   32918
All        18211  75300  19873  20748  21868  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>

Event Log
41:39  LF  UTF-8  4 spaces  Python 3.8 (base)
```

d. Preprocessed Features

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklear
n_model_performance.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews\
corpus\features_preprocessed.csv"
Shape of feature data - num instances with num features + class label
(156000, 201)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.13      0.37      0.20      7069
         neu       0.62      0.67      0.64     79548
         pos       0.19      0.45      0.27      9202
        sneg       0.28      0.16      0.20     27263
        spos       0.35      0.15      0.21     32918

    accuracy                           0.44    156000
   macro avg       0.31      0.36      0.30    156000
weighted avg       0.46      0.44      0.43    156000



Predicted    neg    neu    pos   sneg   spos     All
Actual
neg         2626   2418    846    847    332    7069
neu         7367  52924   6693   6887   5677   79548
pos          934   2574   4162    572    960    9202
sneg        5379  12723   2959   4327   1875   27263
spos        3359  14549   7459   2769   4782   32918
All        19665  85188  22119  15402  13626  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

e. Sentiment Lexicon Features

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklear
n_model_performance.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews\
corpus\features_SL.csv"
Shape of feature data - num instances with num features + class label
(156000, 203)
** Results from Logistic Regression with liblinear
              precision    recall  f1-score   support

         neg       0.18      0.47      0.26      7069
         neu       0.71      0.67      0.69     79548
         pos       0.25      0.55      0.35      9202
        sneg       0.34      0.26      0.30     27263
        spos       0.41      0.27      0.33     32918

    accuracy                           0.50    156000
   macro avg       0.38      0.45      0.39    156000
weighted avg       0.53      0.50      0.51    156000



Predicted    neg    neu    pos   sneg   spos     All
Actual
neg         3343   1247    505   1480    494    7069
neu         5453  53376   4038   9015   7666   79548
pos          575   1006   5058    410   2153    9202
sneg        6225   9372   2014   7121   2531   27263
spos        2615  10299   8258   2722   9024   32918
All        18211  75300  19873  20748  21868  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Decision Tree Classifier

The parameters used in Decision Tree Classifier are,

For our analysis, we have used the following parameters,

- criterion: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- min_samples_split: The minimum number of samples required to split an internal node

Parameter_Description	Parameter_value
criterion	gini
max_depth	7
min_samples_split	5

We have done experiments on different feature sets created using Logistic regression algorithm,

a. Normal Features without preprocessing

Terminal: Local

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklearn_
model_performance_Decision_Tree.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemov
iereviews\corpus\features_normal.csv"
Shape of feature data - num instances with num features + class label
(156000, 201)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.39      0.02      0.04      7069
         neu       0.56      0.94      0.70     79548
         pos       0.41      0.01      0.03      9202
        sneg       0.43      0.05      0.10     27263
        spos       0.32      0.17      0.22     32918

    accuracy                           0.53    156000
   macro avg       0.42      0.24      0.22    156000
weighted avg       0.47      0.53      0.42    156000



Predicted  neg     neu  pos  sneg   spos     All
Actual
neg        155    4745    7   457   1705    7069
neu         60   74620   34   903   3931   79548
pos          5    6115  120    95   2867    9202
sneg       157   21860   30  1465   3751   27263
spos        25   26657  104   453   5679   32918
All        402  133997  295  3373  17933  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Event Log

7:1 LF UTF-8 4 spaces Python 3.8 (base)

b. Bigram Features

```
Terminal:   Local   +
FileNotFoundError: [Errno 2] No such file or directory: 'C:\\Users\\pkous\\Desktop\
\WorkingDir NLP\\kagglemoviereviews\\corpus\\features_bigram.csv'

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklearn_
model_performance_Decision_Tree.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemov
iereviews\corpus\features_biagram.csv"
Shape of feature data - num instances with num features + class label
(156000, 701)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.55      0.02      0.03      7069
         neu       0.52      0.99      0.68     79548
         pos       0.51      0.03      0.05      9202
        sneg       0.43      0.02      0.03     27263
        spos       0.45      0.06      0.10     32918

    accuracy                           0.52    156000
   macro avg       0.49      0.22      0.18    156000
weighted avg       0.49      0.52      0.38    156000



Predicted  neg     neu  pos  sneg  spos     All
Actual
neg        108    6498    7   343   113    7069
neu         24   78375   61   237   851   79548
pos          1    8088  246    10   857    9202
sneg        58   26196   35   492   482   27263
spos         5   30821  134    73  1885   32918
All        196  149978  483  1155  4188  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
                                                            Event Log
                      7:1  LF  UTF-8  4 spaces  Python 3.8 (base)
```

c. Negation Word Features

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklearn_
model_performance_Decision_Tree.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemov
iereviews\corpus\features_NOT.csv"
Shape of feature data - num instances with num features + class label
(156000, 203)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.44      0.03      0.06      7069
         neu       0.63      0.83      0.71     79548
         pos       0.42      0.04      0.07      9202
        sneg       0.36      0.16      0.23     27263
        spos       0.41      0.46      0.43     32918

    accuracy                           0.55    156000
   macro avg       0.45      0.30      0.30    156000
weighted avg       0.51      0.55      0.50    156000



Predicted  neg     neu  pos   sneg   spos     All
Actual
neg        242    3415   21   2224   1167    7069
neu         67   66099   90   3551   9741   79548
pos         16    2201  334    325   6326    9202
sneg       176   18117   51   4452   4467   27263
spos        45   15799  297   1719  15058   32918
All        546  105631  793  12271  36759  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

d. Preprocessed Features

```
Terminal: Local

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklearn_
model_performance_Decision_Tree.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemov
iereviews\corpus\features_preprocessed.csv"
Shape of feature data - num instances with num features + class label
(156000, 201)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.55      0.02      0.03      7069
         neu       0.52      0.99      0.68     79548
         pos       0.51      0.03      0.05      9202
        sneg       0.43      0.02      0.03     27263
        spos       0.45      0.06      0.10     32918

    accuracy                           0.52    156000
   macro avg       0.49      0.22      0.18    156000
weighted avg       0.49      0.52      0.38    156000



Predicted  neg     neu  pos  sneg  spos     All
Actual
neg        108    6498    7   343   113    7069
neu         24   78374   62   237   851   79548
pos          1    8088  246    10   857    9202
sneg        58   26196   33   494   482   27263
spos         5   30819  134    73  1887   32918
All        196  149975  482  1157  4190  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Event Log

7:1 LF UTF-8 4 spaces Python 3.8 (base)

e. Sentiment Lexicon Features

```
(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>python run_sklearn_
model_performance_Decision_Tree.py "C:\Users\pkous\Desktop\WorkingDir NLP\kagglemov
iereviews\corpus\features_SL.csv"
Shape of feature data - num instances with num features + class label
(156000, 203)
** Results from Decision Tree Classifier
              precision    recall  f1-score   support

         neg       0.44      0.03      0.06      7069
         neu       0.63      0.83      0.71     79548
         pos       0.42      0.04      0.07      9202
        sneg       0.36      0.16      0.23     27263
        spos       0.41      0.46      0.43     32918

    accuracy                           0.55    156000
   macro avg       0.45      0.30      0.30    156000
weighted avg       0.51      0.55      0.50    156000



Predicted  neg     neu  pos   sneg   spos     All
Actual
neg        242    3415   21   2224   1167    7069
neu         68   66099   90   3550   9741   79548
pos         16    2201  333    326   6326    9202
sneg       178   18118   51   4449   4467   27263
spos        45   15799  298   1719  15057   32918
All        549  105632  793  12268  36758  156000

(base) C:\Users\pkous\Desktop\WorkingDir NLP\kagglemoviereviews>
```

Highlights of Sci-kit learn algorithms:

After running through different experiments, it can be said that Sentiment Lexicon feature sets work best when classifying the movie review datasets. The important aspect is that both logistic regression and decision tree classifier work similarly for Sentiment Lexicon features with highest precision, recall and f1-measures.

Comparison of evaluation metrics:

Average subjectivity accuracy was 1.35% higher than its corresponding unigram feature set for un-processed tokens. Average subjectivity accuracy was 0.45% higher than its corresponding unigram feature set for pre-processed tokens. In fact, Logistic regression produced best accuracy using subjectivity feature set on pre-processed version.

Based on NLTK classifiers and Sci-kit learn algorithms outputs, highest precision, recall and f1-measures was obtained for Sci-kit learn algorithms. The reason for this might be because the logic for Naïve Bayes is based on naïve algorithm wherein each of the feature is take multiple times i.e., the train data has some reoccurrences of data because of which redundant learning is made by the Naïve Bayes algorithm however, this is not the case with Logistic regression and Decision tree classifier algorithms. Therefore, in our case Sci-kit learn algorithms work best.

Conclusion:

The maximum accuracy attained is 70%, we obtained this accuracy for combined feature sets where we ran the Naïve Bayes classifier. After running through the entire analysis, we concluded that grouping the target variables into three categories such as Negative, Neutral and Positive would give more accuracy, however, changing the target variable composition is something which we need to consider in extreme situations.

Lessons Learned:

- Learned how to combine and implement most of the concepts learned during the semester.
- Leveraged python file system to write reusable code. Gained experience in dealing with problems that arise from working with large datasets.
- Observed how combining various feature sets affects the accuracy of the model.

Learned how different machine learning models are implemented and how their outputs differ, and what causes these differences in output.