IST 664
Natural Language Processing
Final Project

Sri Venkata Subhramanya Abhishek Namana

1. INTRODUCTION

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

The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site. Socher's group used crowd-sourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: "negative", "somewhat negative", "neutral", "somewhat positive", "positive".

The train/test split has been preserved for the purposes of benchmarking, but the sentences have been shuffled from their original order. Each Sentence has been parsed into many phrases by the Stanford parser. Each phrase has a Phrase Id. Each sentence has a Sentence Id. Phrases that are repeated (such as short/common words) are only included once in the data.

The train.tsv contains the phrases and their associated sentiment labels test.tsv contains just phrases. You must assign a sentiment label to each phrase.

The sentiment labels are:
0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 – positive

The various steps involved are -
1. Extraction of the train data from the dataset.
2. Tokenizing and filtering the unnecessary data from the dataset.
3. Generating and executing the feature sets from the dataset.
4. Naive-Bayes algorithm classification.
5. Logistic regression classification.
6. Decision tree classification

STEPS INVOLVED IN THE PROJECT

1. Extraction of the train data from the dataset.

The function reads a CSV file bit by bit and saves the data into a list named "data." There might be a limit on how many rows we can add to this list. Sometimes, analyzing a small amount of data isn't very helpful, so we want to use as much data as possible. However, because of the limits of our computer's processing power, we can't use all the data available. Our goal is still to analyze as much of the dataset as we can.

```
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])

  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
```

2. Tokenizing and filtering the unnecessary data from the dataset.

The process described in the code involves using the NLTK tokenize function to split sentences into individual words. Before this tokenization step, a preprocessing step is applied to the tokens using the regular expression 'w+'. The reason for using this specific regular expression is that it matches any sequence of alphanumeric characters, including underscores. This ensures that the tokenization process effectively breaks down phrases into their basic word components while adhering to this pattern.

After the data is tokenized, we create two lists -

1. "normaltokens" -> contains the unfiltered word phrases

2. "preprocessedTokens" -> contains the filtered word 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(phrase[0])
  tokens = tokenizer.tokenize(phrase[0])
  phrasedocs.append((tokens, int(phrase[1])))

# possibly filter tokens
normaltokens = get_words_from_docs_usual(phrasedocs_without)
preprocessedTokens = get_words_from_docs(phrasedocs)
```

Preprocessing documents -

This function converts all sentences to lowercase and splits them into individual lines. It then uses regular expressions to remove punctuation from the words. After stripping away punctuation, the words are compiled into a "word list" file. Next, this list is further refined into a "final word list" by removing any stop words. Essentially, this means the function filters out common words (like 'the', 'and', 'in') that don't contribute much meaning to the text.

```
def pre_processing(document):
  # "Pre_processing_documents"
  # "create list of lower case words"
  word_list = re.split('\s+', document.lower())

  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
```

Retrieving words/tokens -

There are three functions defined in this script. Using documents longer than three pages, the first function returns a list of tokens/words. The second feature compiles a dictionary of terms that convey an emotion. The third method returns every line that could be found among the tokens.

```
def get_words_from_docs(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_docs_usual(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_dataset(lines):
  all_words = []
  for id,words in lines:
    all_words.extend(words)
  return all_words
```

3. Generating and executing the feature sets from the dataset.

Creating Feature Sets -

1) Unigram features or bag of words (Preprocessed) -

Two separate functions were developed to produce unigrams. The first procedure extracts the 200 most frequent words from the 'wordlist' of processed tokens and returns them as a list. The second procedure provides a tool that creates a unique vocabulary list from the texts.

2) Unigram features or bag of words (Without Preprocessed):

I worked on creating bigram features from documents to acquire high frequency bigrams.I have sorted the results by frequency after removing special characters. To find the best possible bigrams given the values provided in both dimensions, I used the nbest function

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


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

3)Bigram features –

I worked on creating bigram features from documents to acquire high frequency bigrams.I have sorted the results by frequency after removing special characters. To find the best possible bigrams given the values provided in both dimensions, I used the nbest function.

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

4) Sentiment Lexicons -

We will begin by importing the subjectivity lexicon file developed by Janyce Wiebe and her team as part of the MPQA project at the University of Pittsburgh. Our focus will be on augmenting each document with two specific metrics: the count of positively and negatively subjective phrases. This is achieved using the `readSubjectivity` function from the Professor’s Subjectivity.py module. Instead of a simple dictionary, this function produces a Subjectivity Lexicon. This lexicon is distinct in that it assigns each word a numerical value representing its intensity and polarity. Our feature extraction tool includes these word features, along with 'positive count' and 'negative count.' These attributes account for the frequency of each subjective word, counting it once under normal circumstances, but twice if the word is deemed to have exceptional strength.

```
def readSubjectivity(path):
  flexicon = open(path, 'r')
  # initialize an empty dictionary
  sldict = { }
  for line in flexicon:
    fields = line.split()
    strength = fields[0].split("=")[1]
    word = fields[2].split("=")[1]
    posTag = fields[3].split("=")[1]
    stemmed = fields[4].split("=")[1]
    polarity = fields[5].split("=")[1]
    if (stemmed == 'y'):
      isStemmed = True
    else:
      isStemmed = False
    # put a dictionary entry with the word as the keyword
    #     and a list of the other values
    sldict[word] = [strength, posTag, isStemmed, polarity]
  return sldict
```

```
SLpath = "./SentimentLexicons/subjclueslen1-HLTEMNLP05.tff"
SL = readSubjectivity(SLpath)
def SL_features(document, word_features, SL):
  document_words = set(document)
  features = {}
  for word in word_features:
    features['contains({})'.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)

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

5) Negation word features -

I examined the use of negation in contractions like "doesn't," “ , ” , “t” as well as in complete words like "not," "never," and "nor." For example, in the first document I analyzed, phrases like 'if', 'you don't like', 'this', 'film', indicate the use of negation. In language, negation words can function differently: some negate only the immediately following word, while others negate every word until the next punctuation, or the scope of negation might be determined by the sentence structure. In my approach, I adopted the former strategy, where I added word features for each word in the document sequentially, but modified the feature to a negated version when it followed a negation 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
```

6) POS features –

By leveraging the features of part-of-speech (POS) tags, I managed to effectively tackle this categorization task. Nowadays, there's an increasing trend towards employing shorter units for classification, such as phrase-level classification, especially in brief forms of social media like tweets. This dataset includes a substantial training set, presenting a challenge for demonstration with NLTK because the standard NLTK POS tagger takes an inordinate amount of time to process on a regular system. Due to this limitation, I was restricted to analysing only 2000 training sentences. The most common approach in utilizing data from POS systems involves tallying various types of word tags.

```python
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
```

7)Trigram features –

I used part-of-speech (POS) tags to categorize text effectively. With the rise of social media, like Twitter, there's more focus on classifying short text units, such as phrases. The dataset I worked with was large, which made it difficult to use the NLTK POS tagger, as it's slow on standard systems. Because of this, I could only analyze 2000 sentences from the training set. A common method in POS tagging is to count different types of word tags in the text.

```python
def trigram_features(document, word_features,trigram_features):
  document_words = set(document)
  document_trigrams = nltk.trigrams(document)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  for trigram in trigram_features:
    #print(trigram)
    features['trigram({} {} {})'.format(trigram[0], trigram[1], trigram[2])] = (trigram in document_trigrams)
  return features

def get_trigram_features(tokens):
  trigram_measures = nltk.collocations.TrigramAssocMeasures()
  finder = TrigramCollocationFinder.from_words(tokens,window_size=3)
  #finder.apply_freq_filter(6)
  trigram_features = finder.nbest(trigram_measures.chi_sq, 3000)
  return trigram_features[:500]
```

8. Combination of features sets:

```
def combined_document_features(document, word_features, SL, bigram_features):
  document_words = set(document)
  document_bigrams = nltk.bigrams(document)
  features = {}
  #print(bigram_features[0])

  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
```

4. Naive-Bayes algorithm classification -

Feature Set	**Achieved Accuracy**
Normal features without preprocessing	60%
Preprocessed features	70%
Bigram features	60%
Negation word features	60%
Sentiment lexicon features	70%
Trigram Features	70%
POS features	90%
Combined feature set (Word features after preprocessing, SL, Bigram)	60%

Command used to execute the program -

For Naïve Bayes Classifier -

python classifykaggle.py <foldername> limit

For Logistic Regression -

python sklearn_model_performance.py <csv file path>

For Decision Tree Classifier-

python sklearn_model_performance_DT.py <csv file path>

Naive Bayes Classifier:

1. Normal features without preprocessing:

```
Read 156060 phrases, using 100 random phrases
---------------------------------------------------
Accuracy with normal features, without pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.6
---------------------------------------------------
Showing most informative features
Most Informative Features
           contains(its) = True              4 : 2      =     10.8 : 1.0
          contains(with) = True              4 : 2      =     10.8 : 1.0
             contains(.) = True              0 : 2      =      8.6 : 1.0
           contains(and) = True              4 : 2      =      8.4 : 1.0
          contains(that) = True              0 : 2      =      6.1 : 1.0
          contains(film) = True              0 : 2      =      6.1 : 1.0
           contains(has) = True              0 : 2      =      6.1 : 1.0
            contains(of) = True              4 : 2      =      6.0 : 1.0
            contains(an) = True              4 : 3      =      5.8 : 1.0
        contains(honest) = True              4 : 3      =      5.8 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<1>. . . |
2 | .<4>. . |
3 | 2 2<.>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

2. Preprocessed features:

```
---------------------------------------------------
Accuracy with pre-processed features :
Training and testing a classifier
Accuracy of classifier :
0.7
---------------------------------------------------
Showing most informative features
Most Informative Features
          contains(film) = True              0 : 2      =      6.1 : 1.0
        contains(honest) = True              4 : 3      =      5.8 : 1.0
    contains(historical) = True              4 : 1      =      4.5 : 1.0
         contains(movie) = True              1 : 2      =      4.0 : 1.0
         contains(could) = True              1 : 2      =      2.4 : 1.0
           contains(not) = True              1 : 2      =      2.4 : 1.0
        contains(rather) = True              1 : 2      =      2.4 : 1.0
          contains(life) = True              3 : 2      =      1.9 : 1.0
          contains(real) = True              3 : 2      =      1.9 : 1.0
       contains(serious) = True              3 : 2      =      1.9 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<.>. 1 . |
2 | .<4>. . |
3 | . 2<2>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

3. Bigram features:

```
---------------------------------------------------
Accuracy with bigram featuresets :
Training and testing a classifier
Accuracy of classifier :
0.6
---------------------------------------------------
Showing most informative features
Most Informative Features
          contains(film) = True              0 : 2      =      6.1 : 1.0
        contains(honest) = True              4 : 3      =      5.8 : 1.0
    contains(historical) = True              4 : 1      =      4.5 : 1.0
         contains(movie) = True              1 : 2      =      4.0 : 1.0
         contains(could) = True              1 : 2      =      2.4 : 1.0
           contains(not) = True              1 : 2      =      2.4 : 1.0
        contains(rather) = True              1 : 2      =      2.4 : 1.0
          contains(life) = True              3 : 2      =      1.9 : 1.0
          contains(real) = True              3 : 2      =      1.9 : 1.0
       contains(serious) = True              3 : 2      =      1.9 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<.>. 1 . |
2 | .<4>. . |
3 | . 3<1>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

4. Negation word features:

```
---------------------------------------------------
Accuracy with NOT_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.6
---------------------------------------------------
Showing most informative features
Most Informative Features
          contains(film) = True                0 : 2      =      6.1 : 1.0
        contains(honest) = True                4 : 3      =      5.8 : 1.0
    contains(historical) = True                4 : 1      =      4.5 : 1.0
         contains(movie) = True                1 : 2      =      4.0 : 1.0
             contains(s) = False               1 : 2      =      4.0 : 1.0
         contains(could) = True                1 : 2      =      2.4 : 1.0
          contains(life) = True                3 : 2      =      1.9 : 1.0
          contains(real) = True                3 : 2      =      1.9 : 1.0
       contains(serious) = True                3 : 2      =      1.9 : 1.0
   contains(beautifully) = False               2 : 4      =      1.6 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<.>. 1 . |
2 | .<4>. . |
3 | . 3<1>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

5. Sentiment lexicon:

```
---------------------------------------------------
Accuracy with SL_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.7
---------------------------------------------------
Showing most informative features
Most Informative Features
           positivecount = 4                   4 : 2      =      6.6 : 1.0
           positivecount = 3                   4 : 2      =      6.6 : 1.0
          contains(film) = True                0 : 2      =      6.1 : 1.0
        contains(honest) = True                4 : 3      =      5.8 : 1.0
           negativecount = 3                   1 : 2      =      5.2 : 1.0
           negativecount = 4                   0 : 2      =      5.0 : 1.0
    contains(historical) = True                4 : 1      =      4.5 : 1.0
         contains(movie) = True                1 : 2      =      4.0 : 1.0
           negativecount = 2                   3 : 2      =      3.4 : 1.0
           positivecount = 1                   3 : 1      =      3.0 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<.>. 1 . |
2 | .<4>. . |
3 | . 2<2>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

6. Trigram features:

```
--------------------------------------------------
Accuracy with Trigram featuresets :
Training and testing a classifier
Accuracy of classifier :
0.7
--------------------------------------------------
Showing most informative features
Most Informative Features
          contains(film) = True                0 : 2      =      6.1 : 1.0
        contains(honest) = True                4 : 3      =      5.8 : 1.0
    contains(historical) = True                4 : 1      =      4.5 : 1.0
         contains(movie) = True                1 : 2      =      4.0 : 1.0
         contains(could) = True                1 : 2      =      2.4 : 1.0
           contains(not) = True                1 : 2      =      2.4 : 1.0
        contains(rather) = True                1 : 2      =      2.4 : 1.0
          contains(life) = True                3 : 2      =      1.9 : 1.0
          contains(real) = True                3 : 2      =      1.9 : 1.0
       contains(serious) = True                3 : 2      =      1.9 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<.>. 1 . |
2 | .<4>. . |
3 | . 2<2>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

7. POS features:

```
--------------------------------------------------
Accuracy with POS_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.9
--------------------------------------------------
Showing most informative features
Most Informative Features
                 adverbs = 2                   4 : 2      =      8.8 : 1.0
              adjectives = 2                   4 : 2      =      7.5 : 1.0
                   verbs = 2                   4 : 2      =      7.5 : 1.0
                   nouns = 3                   4 : 2      =      6.2 : 1.0
          contains(film) = True                0 : 2      =      6.1 : 1.0
        contains(honest) = True                4 : 3      =      5.8 : 1.0
    contains(historical) = True                4 : 1      =      4.5 : 1.0
         contains(movie) = True                1 : 2      =      4.0 : 1.0
                   nouns = 1                   2 : 0      =      3.4 : 1.0
                   verbs = 0                   2 : 4      =      3.1 : 1.0

The confusion matrix
  | 1 2 3 4 |
--+---------+
1 |<1>. . . |
2 | .<4>. . |
3 | . 1<3>. |
4 | . . .<1>|
--+---------+
(row = reference; col = test)
```

8. Combined feature set (Word features after preprocessing, SL, Bigram) :

```
Accuracy with combined featuresets :
Training and testing a classifier
Accuracy of classifier :
0.6
----------------------------------------------------
```

```
The confusion matrix
  | 1 2 3 |
--+-------+
1 |<.>3 . |
2 | .<6>. |
3 | . 1<.>|
--+-------+
(row = reference; col = test)
```

Showing 1 most informative features:

B_{'contains(like)': False, 'contains(rrb)': False, 'contains(love)': False, 'contains(old)': False, 'contains(good)': False, 'contains(two)': False, 'contains(year)': False, 'contains(less)': False, 'contains(comes)': False, 'contains(whole)': False, 'contains(animation)': False, 'contains(clever)': False, 'contains(visual)': False, 'contains(shows)': False, 'contains(can)': False, 'contains(really)': False, 'contains(amy)': False, 'contains(lrb)': False, 'contains(movie)': False, 'contains(character)': False, 'contains(material)': False, 'contains(not)': False, 'contains(searching)': False, 'contains(quarter)': False, 'contains(dark)': False, 'contains(disturbing)': False, 'contains(match)': False, 'contains(sensibilities)': False, 'contains(directors)': False, 'contains(stylish)': False, 'contains(closely)': False, 'contains(resembles)': False, 'contains(version)': False, 'contains(tomcats)': False, 'contains(cheap)': False, 'contains(junk)': False, 'contains(insult)': False, 'contains(deathdefying)': False, 'contains(efforts)': False, 'contains(trying)': False, 'contains(eat)': False, 'contains(brussels)': False, 'contains(sprouts)': False, 'contains(entertaining)': False, 'contains(dispense)': False, 'contains(advice)': False, 'contains(never)': False, 'contains(together)': False, 'contains(coherent)': False, 'contains(cop)': False, 'contains(flick)': False, 'contains(cliches)': False, 'contains(oily)': False, 'contains(arms)': False, 'contains(dealer)': False, 'contains(squad)': False, 'contains(car)': False, 'contains(pileups)': False, 'contains(requisite)': False, 'contains(screaming)': False, 'contains(captain)': False, 'contains(sorority)': False, 'contains(boys)': False, 'contains(takes)': False, 'contains(title)': False, 'contains(literally)': False, 'contains(adorns)': False, 'contains(childhood)': False, 'contains(imagination)': False, 'contains(big)': False, 'contains(chill)': False, 'contains(history)': False, 'contains(academy)': False, 'contains(stretched)': False, 'contains(barely)': False, 'contains(feature)': False, 'contains(length)': False, 'contains(little)': False, 'contains(attention)': False, 'contains(paid)': False, 'contains(litmus)': False, 'contains(test)': False, 'contains(carries)': False, 'contains(day)': False, 'contains(lightweight)': False, 'contains(filmmaking)': False, 'contains(sure)': False, 'contains(look)': False, 'contains(angle)': False, 'contains(women)': False, 'contains(augustine)': False, 'contains(brand)': False, 'contains(trickery)': False, 'contains(stops)': False, 'contains(devolves)': False, 'contains(flashy)': False, 'contains(vaguely)': False, 'contains(silly)': False, 'contains(overkill)': False, 'contains(predictable)': False, 'contains(outcome)': False, 'contains(disbelief)': False, 'contains(several)': False, 'contains(right)': False, 'contains(actors)': False, 'contains(kind)': False, 'contains(flair)': False, 'contains(great)': False, 'contains(cinema)': False, 'contains(plenty)': False, 'contains(nudity)': False, 'contains(prevalence)': False, 'contains(fastforward)': False, 'contains(technology)': False, 'contains(beautiful)': False, 'contains(timeless)': False, 'contains(universal)': False, 'contains(tale)': False, 'contains(heated)': False, 'contains(passions)': False, 'contains(jealousy)': False, 'contains(betrayal)': False, 'contains(forgiveness)': False, 'contains(murder)': False, 'contains(thin)': False, 'contains(period)': False, 'contains(onehour)': False, 'contains(documentary)': False, 'contains(treat)': False, 'contains(delightful)': False, 'contains(witty)': False, 'contains(improbable)': False, 'contains(romantic)': False, 'contains(comedy)': False, 'contains(zippy)': False, 'contains(jazzy)': False, 'contains(score)': False, 'contains(looks)': False, 'contains(drag)': False, 'contains(queen)': False, 'contains(cuteness)': False, 'contains(career)': False, 'contains(success)': False, 'contains(bestselling)': False, 'contains(writer)': False, 'contains(selfhelp)': False, 'contains(books)': False, 'contains(help)': False, 'contains(neuroses)': False, 'contains(men)': False, 'contains(expresses)': False, 'contains(basic)': False, 'contains(emotions)': False, 'contains(terrorism)': False, 'contains(creeping)': False, 'contains(predictability)': False, 'contains(makes)': False, 'contains(work)': False, 'contains(admittedly)': False, 'contains(limited)': False, 'contains(extent)': False, 'contains(commitment)': False, 'contains(genuinely)': False, 'contains(engaging)': False, 'contains(performers)': False, 'contains(pay)': False, 'contains(handbagclutching)': False, 'contains(sarandon)': False, 'contains(tumult)': False, 'contains(something)': False, 'contains(full)': False, 'contains(frontal)': False, 'contains(guess)': False, 'contains(artifice)': False, 'contains(acting)': False, 'contains(distorts)': False, 'contains(reality)': False, 'contains(people)': False, 'contains(make)': False, 'contains(movies)': False, 'contains(watch)': False, 'contains(creating)': False, 'contains(screenplay)': False, 'contains(weirder)': False, 'contains(middleclass)': False, 'contains(angst)': False, 'contains(actually)': False, 'contains(watchable)': False, 'contains(biggest)': False, 'contains(disappointments)': False, 'contains(even)': False, 'contains(worse)': False, 'contains(usual)': False, 'contains(fluttering)': False, 'contains(stammering)': False, 'contains(friendships)': False, 'contains(bazadona)': False, 'contains(liar)': False, 'contains(making)': False, 'contains(understandable)': False, 'bigram(abhors cute)': False, 'bigram(abhors sort)': False, 'bigram(academy barely)': False, 'bigram(academy stretched)': False, 'bigram(acidic allmale)': False, 'bigram(acidic eve)': False, 'bigram(acting distorts)': False, 'bigram(acting reality)': False, 'bigram(actors kind)': False, 'bigram(actually biggest)': False, 'bigram(actually watchable)': False, 'bigram(admittedly extent)': False, 'bigram(admittedly limited)': False, 'bigram(adorns childhood)': False, 'bigram(adorns imagination)': False, 'bigram(adult get)': False, 'bigram(advice never)': False, 'bigram(alienation monstrous)': False, 'bigram(alienation murk)': False, 'bigram(alive kiss)': False, 'bigram(alive last)': False, 'bigram(allmale characterization)': False, 'bigram(allmale eve)': False, 'bigram(angle augustine)': False, 'bigram(angle women)': False, 'bigram(angst actually)': False, 'bigram(angst watchable)': False, 'bigram(appears screen)': False, 'bigram(arms dealer)': False, 'bigram(arms squad)': False, 'bigram(artifice acting)': False, 'bigram(artifice distorts)': False, 'bigram(attention paid)': False, 'bigram(audiences endorsement)': False, 'bigram(audiences mistake)': False, 'bigram(augustine brand)': False, 'bigram(barely feature)': False, 'bigram(barely length)': False, 'bigram(basic emotions)': False, 'bigram(bazadona liar)': False, 'bigram(bazadona making)': False, 'bigram(bean abhors)': False, 'bigram(bean sort)': False, 'bigram(beautiful timeless)': False, 'bigram(beautiful universal)': False, 'bigram(bestselling selfhelp)': False, 'bigram(bestselling writer)': False, 'bigram(betrayal forgiveness)': False, 'bigram(betrayal murder)': False, 'bigram(better conversation)': False, 'bigram(better seen)': False, 'bigram(big chill)':

False, 'bigram(biggest disappointments)': False, 'bigram(blair videocam)': False, 'bigram(blair witch)': False, 'bigram(books help)': False, 'bigram(box office)': False, 'bigram(box pie)': False, 'bigram(boys takes)': False, 'bigram(boys title)': False, 'bigram(brand trickery)': False, 'bigram(bros costumer)': False, 'bigram(bros jived)': False, 'bigram(brussels sprouts)': False, 'bigram(captain boys)': False, 'bigram(captain sorority)': False, 'bigram(captured chaos)': False, 'bigram(captured urban)': False, 'bigram(capturing innercity)': False, 'bigram(capturing life)': False, 'bigram(car pileups)': False, 'bigram(car requisite)': False, 'bigram(career success)': False, 'bigram(carol appears)': False, 'bigram(carol kane)': False, 'bigram(carries day)': False, 'bigram(carries lightweight)': False, 'bigram(caught intricate)': False, 'bigram(caught plot)': False, 'bigram(chaos conflagration)': False, 'bigram(chaos urban)': False, 'bigram(characterization contrived)': False, 'bigram(characterization hitler)': False, 'bigram(cheap insult)': False, 'bigram(cheap junk)': False, 'bigram(chief films)': False, 'bigram(chief scarpia)': False, 'bigram(childhood big)': False, 'bigram(childhood imagination)': False, 'bigram(chill history)': False, 'bigram(cliches oily)': False, 'bigram(closely resembles)': False, 'bigram(cloying far)': False, 'bigram(coherent cop)': False, 'bigram(college kids)': False, 'bigram(college subject)': False, 'bigram(comedy jazzy)': False, 'bigram(comedy zippy)': False, 'bigram(commitment genuinely)': False, 'bigram(conclusion may)': False, 'bigram(conclusion stars)': False, 'bigram(conflagration forget)': False, 'bigram(conflagration fury)': False, 'bigram(confront joy)': False, 'bigram(confront rising)': False, 'bigram(constantly defies)': False, 'bigram(constantly expectation)': False, 'bigram(contrived nature)': False, 'bigram(contrived provocative)': False, 'bigram(conversation starter)': False, 'bigram(conversation worlds)': False, 'bigram(cop cliches)': False, 'bigram(cop flick)': False, 'bigram(costumer jived)': False, 'bigram(costumer sex)': False, 'bigram(creating screenplay)': False, 'bigram(creating weirder)': False, 'bigram(credits roll)': False, 'bigram(credits stomach)': False, 'bigram(creeping makes)': False, 'bigram(creeping predictability)': False, 'bigram(cute cloying)': False, 'bigram(cuteness career)': False, 'bigram(danger sophisticated)': False, 'bigram(dark disturbing)': False, 'bigram(day filmmaking)': False, 'bigram(day lightweight)': False, 'bigram(deadly dull)': False, 'bigram(deadly watching)': False, 'bigram(dealer car)': False, 'bigram(dealer squad)': False, 'bigram(deathdefying efforts)': False, 'bigram(defies expectation)': False, 'bigram(defies interfering)': False, 'bigram(degraded blair)': False, 'bigram(degraded handheld)': False, 'bigram(delightful improbable)': False, 'bigram(delightful witty)': False, 'bigram(devolves flashy)': False, 'bigram(devolves vaguely)': False, 'bigram(directors closely)': False, 'bigram(directors stylish)': False, 'bigram(disappointments even)': False, 'bigram(disbelief right)': False, 'bigram(disbelief several)': False, 'bigram(discovery homosexuality)': False, 'bigram(discovery rare)': False, 'bigram(dispense advice)': False, 'bigram(dispense never)': False, 'bigram(distorts people)': False, 'bigram(distorts reality)': False, 'bigram(disturbing match)': False, 'bigram(documentary delightful)': False, 'bigram(documentary treat)': False, 'bigram(drag cuteness)': False, 'bigram(drag queen)': False, 'bigram(dry welcome)': False, 'bigram(dry would)': False, 'bigram(dual patronising)': False, 'bigram(dual performance)': False, 'bigram(dubious caught)': False, 'bigram(dubious product)': False, 'bigram(dull proverbial)': False, 'bigram(dull watching)': False, 'bigram(dumb deadly)': False, 'bigram(dumb humor)': False, 'bigram(eat brussels)': False, 'bigram(eat sprouts)': False, 'bigram(efforts trying)': False, 'bigram(emotions terrorism)': False, 'bigram(endorsement bean)': False, 'bigram(endorsement things)': False, 'bigram(engaging pay)': False, 'bigram(engaging performers)': False, 'bigram(entertaining advice)': False, 'bigram(entertaining dispense)': False, 'bigram(entirely wholesome)': False, 'bigram(eve characterization)': False, 'bigram(eve hitler)': False, 'bigram(even usual)': False, 'bigram(even worse)': False, 'bigram(expectation interfering)': False, 'bigram(expectation son)': False, 'bigram(expresses basic)': False, 'bigram(expresses emotions)': False, 'bigram(extent commitment)': False, 'bigram(family genuine)': False, 'bigram(far forte)': False, 'bigram(far zhang)': False, 'bigram(fastforward beautiful)': False, 'bigram(fastforward technology)': False, 'bigram(feature length)': False, 'bigram(feature little)': False, 'bigram(feel credits)': False, 'bigram(feel roll)': False, 'bigram(filmmaking look)': False, 'bigram(filmmaking sure)': False, 'bigram(films captured)': False, 'bigram(films chaos)': False, 'bigram(flair great)': False, 'bigram(flashy silly)': False, 'bigram(flashy vaguely)': False, 'bigram(flick cliches)': False, 'bigram(fluttering friendships)': False, 'bigram(fluttering stammering)': False, 'bigram(footage box)': False, 'bigram(footage holiday)': False, 'bigram(forget feel)': False, 'bigram(forget pain)': False, 'bigram(forgiveness murder)': False, 'bigram(forgiveness thin)': False, 'bigram(forte constantly)': False, 'bigram(friendships bazadona)': False, 'bigram(friendships liar)': False, 'bigram(frontal artifice)': False, 'bigram(frontal guess)': False, 'bigram(full frontal)': False, 'bigram(full guess)': False, 'bigram(fury forget)': False, 'bigram(fury pain)': False, 'bigram(genre goofy)': False, 'bigram(genre offering)': False, 'bigram(genuine sweet)': False, 'bigram(genuine without)': False, 'bigram(genuinely engaging)': False, 'bigram(genuinely performers)': False, 'bigram(get capturing)': False, 'bigram(get substituting)': False, 'bigram(great cinema)': False, 'bigram(grub unconditional)': False, 'bigram(grumbling grub)': False, 'bigram(grumbling tasty)': False, 'bigram(guess acting)': False, 'bigram(guess artifice)': False, 'bigram(guts confront)': False, 'bigram(guts joy)': False, 'bigram(handbagclutching sarandon)': False, 'bigram(handbagclutching tumult)': False, 'bigram(handheld blair)': False, 'bigram(handheld witch)': False, 'bigram(heated jealousy)': False, 'bigram(heated passions)': False, 'bigram(history academy)': False, 'bigram(history stretched)': False, 'bigram(hitler contrived)': False, 'bigram(hitler nature)': False, 'bigram(holiday box)': False, 'bigram(holiday office)': False, 'bigram(homosexuality family)': False, 'bigram(homosexuality rare)': False, 'bigram(humor deadly)': False, 'bigram(humor dull)': False, 'bigram(identity alienation)': False, 'bigram(identity monstrous)': False, 'bigram(imagination big)': False, 'bigram(imagination chill)': False, 'bigram(improbable comedy)': False, 'bigram(improbable romantic)': False, 'bigram(innercity dual)': False, 'bigram(innercity life)': False, 'bigram(insult deathdefying)': False, 'bigram(insult efforts)': False, 'bigram(intelligently intentional)': False, 'bigram(intelligently made)': False, 'bigram(intentional police)': False, 'bigram(interfering discovery)': False, 'bigram(interfering son)': False, 'bigram(intricate plot)': False, 'bigram(jazzy looks)': False, 'bigram(jazzy score)': False, 'bigram(jealousy betrayal)': False, 'bigram(jealousy forgiveness)': False, 'bigram(jived sex)': False, 'bigram(joy rising)': False, 'bigram(joy stale)': False, 'bigram(junk deathdefying)': False, 'bigram(junk insult)': False, 'bigram(kane appears)': False, 'bigram(kane screen)': False, 'bigram(kids matter)': False, 'bigram(kids subject)': False, 'bigram(kind flair)': False, 'bigram(kiss performances)': False, 'bigram(label milder)': False, 'bigram(label stands)': False, 'bigram(last kiss)': False, 'bigram(length attention)': False, 'bigram(length little)': False, 'bigram(liar making)': False, 'bigram(life dual)': False, 'bigram(life performance)': False, 'bigram(lightweight filmmaking)': False, 'bigram(lightweight sure)': False, 'bigram(limited commitment)': False, 'bigram(limited extent)': False, 'bigram(literally adorns)': False, 'bigram(literally childhood)': False, 'bigram(litmus carries)': False, 'bigram(litmus test)': False, 'bigram(little attention)': False, 'bigram(little paid)': False, 'bigram(look angle)': False, 'bigram(looks drag)': False, 'bigram(lot scarier)': False, 'bigram(lot tepid)': False, 'bigram(made intentional)': False, 'bigram(make movies)': False, 'bigram(make watch)': False, 'bigram(makes work)': False, 'bigram(making understandable)': False, 'bigram(match sensibilities)': False, 'bigram(matter adult)': False, 'bigram(may college)': False, 'bigram(may kids)': False, 'bigram(meditation alienation)': False, 'bigram(meditation identity)': False, 'bigram(men basic)': False, 'bigram(men expresses)': False, 'bigram(mib label)': False, 'bigram(mib stands)': False, 'bigram(middleclass actually)': False, 'bigram(middleclass angst)': False, 'bigram(milder better)': False, 'bigram(milder seen)': False, 'bigram(mistake endorsement)': False, 'bigram(mistake things)': False, 'bigram(monstrous murk)': False, 'bigram(monstrous need)': False, 'bigram(movies creating)': False, 'bigram(movies watch)': False, 'bigram(murder period)': False, 'bigram(murder thin)': False, 'bigram(murk need)': False, 'bigram(murk sell)': False, 'bigram(nature conclusion)': False, 'bigram(nature provocative)': False, 'bigram(need sell)': False, 'bigram(need twisted)': False, 'bigram(neuroses men)': False, 'bigram(never together)': False, 'bigram(nudity fastforward)': False, 'bigram(nudity prevalence)': False, 'bigram(observant poetic)': False, 'bigram(observant unfussily)': False, 'bigram(offering goofy)': False, 'bigram(office dubious)': False, 'bigram(office pie)': False, 'bigram(oily arms)': False, 'bigram(oily dealer)': False, 'bigram(onehour documentary)': False, 'bigram(onehour treat)': False, 'bigram(outcome disbelief)': False, 'bigram(outcome several)': False, 'bigram(overkill outcome)': False, 'bigram(overkill predictable)': False, 'bigram(paid litmus)': False, 'bigram(pain credits)': False, 'bigram(pain feel)': False, 'bigram(paint dry)': False, 'bigram(paint would)': False, 'bigram(passions betrayal)': False, 'bigram(passions jealousy)': False, 'bigram(patronising carol)': False, 'bigram(patronising reverence)': False, 'bigram(pay handbagclutching)': False, 'bigram(people make)': False, 'bigram(people movies)': False, 'bigram(performance patronising)': False, 'bigram(performance reverence)': False, 'bigram(performances intelligently)': False, 'bigram(performances made)': False, 'bigram(performers pay)': False, 'bigram(period documentary)': False, 'bigram(period onehour)': False, 'bigram(pie dubious)': False, 'bigram(pie product)': False, 'bigram(pileups requisite)': False, 'bigram(pileups screaming)': False, 'bigram(plenty nudity)': False, 'bigram(plenty prevalence)': False, 'bigram(plot scored)': False, 'bigram(poetic identity)': False, 'bigram(poetic meditation)': False, 'bigram(police chief)': False, 'bigram(police scarpia)': False, 'bigram(powered danger)': False, 'bigram(powered real)': False, 'bigram(predictability makes)': False, 'bigram(predictable disbelief)': False, 'bigram(predictable outcome)': False, 'bigram(prevalence fastforward)': False, 'bigram(prevalence technology)': False, 'bigram(product caught)': False, 'bigram(product intricate)': False, 'bigram(proverbial dry)': False, 'bigram(proverbial paint)': False, 'bigram(provocative conclusion)': False, 'bigram(provocative stars)': False, 'bigram(quarter dark)': False, 'bigram(quarter disturbing)': False, 'bigram(queen cuteness)': False, 'bigram(rare family)': False, 'bigram(real danger)': False, 'bigram(reality make)': False, 'bigram(reality people)': False, 'bigram(relying dumb)': False, 'bigram(requisite captain)': False, 'bigram(requisite screaming)': False, 'bigram(resembles version)': False, 'bigram(reverence carol)': False, 'bigram(reverence kane)': False, 'bigram(right actors)': False, 'bigram(right kind)': False, 'bigram(rising stale)': False, 'bigram(roll grumbling)': False, 'bigram(roll stomach)': False, 'bigram(romantic comedy)': False, 'bigram(romantic zippy)': False, 'bigram(sarandon something)': False, 'bigram(sarandon tumult)': False, 'bigram(scarier genre)': False, 'bigram(scarier tepid)': False, 'bigram(scarpia captured)': False, 'bigram(scarpia films)': False, 'bigram(score looks)': False, 'bigram(scored powered)': False, 'bigram(scored real)': False, 'bigram(screaming captain)': False, 'bigram(screaming sorority)': False, 'bigram(screenplay middleclass)': False, 'bigram(screenplay weirder)': False, 'bigram(searching dark)': False, 'bigram(searching quarter)': False, 'bigram(seeks acidic)': False, 'bigram(seen conversation)': False, 'bigram(seen starter)': False, 'bigram(selfhelp books)': False, 'bigram(selfhelp help)': False, 'bigram(sell twisted)': False, 'bigram(sensibilities directors)': False, 'bigram(several actors)': False, 'bigram(several right)': False, 'bigram(sex lot)': False, 'bigram(silly overkill)': False, 'bigram(silly predictable)': False, 'bigram(something frontal)': False, 'bigram(something full)': False, 'bigram(son discovery)': False, 'bigram(son homosexuality)': False, 'bigram(sophisticated audiences)': False, 'bigram(sophisticated mistake)': False, 'bigram(sorority boys)': False, 'bigram(sorority takes)': False, 'bigram(sort cloying)': False, 'bigram(sort cute)': False, 'bigram(sprouts entertaining)': False, 'bigram(squad car)': False, 'bigram(squad pileups)': False, 'bigram(stale time)': False, 'bigram(stammering bazadona)': False, 'bigram(stammering friendships)': False, 'bigram(stands better)': False, 'bigram(stands milder)': False, 'bigram(stars college)': False, 'bigram(stars may)': False, 'bigram(starter observant)': False, 'bigram(starter worlds)': False, 'bigram(stomach grumbling)': False, 'bigram(stomach tasty)': False, 'bigram(stops devolves)': False, 'bigram(story degraded)': False, 'bigram(story handheld)': False, 'bigram(stretched barely)': False, 'bigram(stretched feature)': False, 'bigram(stylish closely)': False, 'bigram(stylish resembles)': False,

'bigram(subject adult)': False, 'bigram(subject matter)': False, 'bigram(substituting capturing)': False, 'bigram(substituting innercity)': False, 'bigram(success bestselling)': False, 'bigram(sure look)': False, 'bigram(sweet relying)': False, 'bigram(sweet without)': False, 'bigram(takes literally)': False, 'bigram(takes title)': False, 'bigram(tale heated)': False, 'bigram(tale passions)': False, 'bigram(tasty grub)': False, 'bigram(tasty unconditional)': False, 'bigram(technology beautiful)': False, 'bigram(technology timeless)': False, 'bigram(tepid genre)': False, 'bigram(tepid offering)': False, 'bigram(terrorism creeping)': False, 'bigram(terrorism predictability)': False, 'bigram(test carries)': False, 'bigram(test day)': False, 'bigram(thin onehour)': False, 'bigram(thin period)': False}_1 = False
0 : 4 = 1.0 : 1.0

Combined feature set (Word features after preprocessing, SL, Trigram)
Showing 1 most informative feature:

T_{'contains(like)': False, 'contains(rrb)': False, 'contains(love)': False, 'contains(old)': False, 'contains(good)': False, 'contains(two)': False, 'contains(year)': False, 'contains(less)': False, 'contains(comes)': False, 'contains(whole)': False, 'contains(animation)': False, 'contains(clever)': False, 'contains(visual)': False, 'contains(shows)': False, 'contains(can)': False, 'contains(really)': False, 'contains(amy)': False, 'contains(lrb)': False, 'contains(movie)': False, 'contains(character)': False, 'contains(material)': False, 'contains(not)': False, 'contains(searching)': False, 'contains(quarter)': False, 'contains(dark)': False, 'contains(disturbing)': False, 'contains(match)': False, 'contains(sensibilities)': False, 'contains(directors)': False, 'contains(stylish)': False, 'contains(closely)': False, 'contains(resembles)': False, 'contains(version)': False, 'contains(tomcats)': False, 'contains(cheap)': False, 'contains(junk)': False, 'contains(insult)': False, 'contains(deathdefying)': False, 'contains(efforts)': False, 'contains(trying)': False, 'contains(eat)': False, 'contains(brussels)': False, 'contains(sprouts)': False, 'contains(entertaining)': False, 'contains(dispense)': False, 'contains(advice)': False, 'contains(never)': False, 'contains(together)': False, 'contains(coherent)': False, 'contains(cop)': False, 'contains(flick)': False, 'contains(cliches)': False, 'contains(oily)': False, 'contains(arms)': False, 'contains(dealer)': False, 'contains(squad)': False, 'contains(car)': False, 'contains(pileups)': False, 'contains(requisite)': False, 'contains(screaming)': False, 'contains(captain)': False, 'contains(sorority)': False, 'contains(boys)': False, 'contains(takes)': False, 'contains(title)': False, 'contains(literally)': False, 'contains(adorns)': False, 'contains(childhood)': False, 'contains(imagination)': False, 'contains(big)': False, 'contains(chill)': False, 'contains(history)': False, 'contains(academy)': False, 'contains(stretched)': False, 'contains(barely)': False, 'contains(feature)': False, 'contains(length)': False, 'contains(little)': False, 'contains(attention)': False, 'contains(paid)': False, 'contains(litmus)': False, 'contains(test)': False, 'contains(carries)': False, 'contains(day)': False, 'contains(lightweight)': False, 'contains(filmmaking)': False, 'contains(sure)': False, 'contains(look)': False, 'contains(angle)': False, 'contains(women)': False, 'contains(augustine)': False, 'contains(brand)': False, 'contains(trickery)': False, 'contains(stops)': False, 'contains(devolves)': False, 'contains(flashy)': False, 'contains(vaguely)': False, 'contains(silly)': False, 'contains(overkill)': False, 'contains(predictable)': False, 'contains(outcome)': False, 'contains(disbelief)': False, 'contains(several)': False, 'contains(right)': False, 'contains(actors)': False, 'contains(kind)': False, 'contains(flair)': False, 'contains(great)': False, 'contains(cinema)': False, 'contains(plenty)': False, 'contains(nudity)': False, 'contains(prevalence)': False, 'contains(fastforward)': False, 'contains(technology)': False, 'contains(beautiful)': False, 'contains(timeless)': False, 'contains(universal)': False, 'contains(tale)': False, 'contains(heated)': False, 'contains(passions)': False, 'contains(jealousy)': False, 'contains(betrayal)': False, 'contains(forgiveness)': False, 'contains(murder)': False, 'contains(thin)': False, 'contains(period)': False, 'contains(onehour)': False, 'contains(documentary)': False, 'contains(treat)': False, 'contains(delightful)': False, 'contains(witty)':

False, 'contains(improbable)': False, 'contains(romantic)': False, 'contains(comedy)': False, 'contains(zippy)': False, 'contains(jazzy)': False, 'contains(score)': False, 'contains(looks)': False, 'contains(drag)': False, 'contains(queen)': False, 'contains(cuteness)': False, 'contains(career)': False, 'contains(success)': False, 'contains(bestselling)': False, 'contains(writer)': False, 'contains(selfhelp)': False, 'contains(books)': False, 'contains(help)': False, 'contains(neuroses)': False, 'contains(men)': False, 'contains(expresses)': False, 'contains(basic)': False, 'contains(emotions)': False, 'contains(terrorism)': False, 'contains(creeping)': False, 'contains(predictability)': False, 'contains(makes)': False, 'contains(work)': False, 'contains(admittedly)': False, 'contains(limited)': False, 'contains(extent)': False, 'contains(commitment)': False, 'contains(genuinely)': False, 'contains(engaging)': False, 'contains(performers)': False, 'contains(pay)': False, 'contains(handbagclutching)': False, 'contains(sarandon)': False, 'contains(tumult)': False, 'contains(something)': False, 'contains(full)': False, 'contains(frontal)': False, 'contains(guess)': False, 'contains(artifice)': False, 'contains(acting)': False, 'contains(distorts)': False, 'contains(reality)': False, 'contains(people)': False, 'contains(make)': False, 'contains(movies)': False, 'contains(watch)': False, 'contains(creating)': False, 'contains(screenplay)': False, 'contains(weirder)': False, 'contains(middleclass)': False, 'contains(angst)': False, 'contains(actually)': False, 'contains(watchable)': False, 'contains(biggest)': False, 'contains(disappointments)': False, 'contains(even)': False, 'contains(worse)': False, 'contains(usual)': False, 'contains(fluttering)': False, 'contains(stammering)': False, 'contains(friendships)': False, 'contains(bazadona)': False, 'contains(liar)': False, 'contains(making)': False, 'contains(understandable)': False, 'trigram(abhors sort cute)': False, 'trigram(academy stretched barely)': False, 'trigram(acidic allmale eve)': False, 'trigram(acting distorts reality)': False, 'trigram(actually watchable biggest)': False, 'trigram(admittedly limited extent)': False, 'trigram(adorns childhood imagination)': False, 'trigram(alienation monstrous murk)': False, 'trigram(alive last kiss)': False, 'trigram(allmale eve characterization)': False, 'trigram(angle women augustine)': False, 'trigram(angst actually watchable)': False, 'trigram(arms dealer squad)': False, 'trigram(artifice acting distorts)': False, 'trigram(audiences mistake endorsement)': False, 'trigram(barely feature length)': False, 'trigram(bazadona liar making)': False, 'trigram(bean abhors sort)': False, 'trigram(beautiful timeless universal)': False, 'trigram(bestselling writer selfhelp)': False, 'trigram(betrayal forgiveness murder)': False, 'trigram(better seen conversation)': False, 'trigram(blair witch videocam)': False, 'trigram(box office pie)': False, 'trigram(boys takes title)': False, 'trigram(bros costumer jived)': False, 'trigram(captain sorority boys)': False, 'trigram(captured chaos urban)': False, 'trigram(capturing innercity life)': False, 'trigram(car pileups requisite)': False, 'trigram(carol kane appears)': False, 'trigram(carries day lightweight)': False, 'trigram(caught intricate plot)': False, 'trigram(chaos urban conflagration)': False, 'trigram(characterization hitler contrived)': False, 'trigram(cheap junk insult)': False, 'trigram(chief scarpia films)': False, 'trigram(childhood imagination big)': False, 'trigram(college kids subject)': False, 'trigram(comedy zippy jazzy)': False, 'trigram(conclusion stars may)': False, 'trigram(conflagration fury forget)': False, 'trigram(confront joy rising)': False, 'trigram(constantly defies expectation)': False, 'trigram(contrived nature provocative)': False, 'trigram(conversation starter worlds)': False, 'trigram(cop flick cliches)': False, 'trigram(costumer jived sex)': False, 'trigram(creating screenplay weirder)': False, 'trigram(credits roll stomach)': False, 'trigram(creeping predictability makes)': False, 'trigram(day lightweight filmmaking)': False, 'trigram(deadly dull watching)': False, 'trigram(dealer squad car)': False, 'trigram(defies expectation interfering)': False, 'trigram(degraded handheld blair)': False, 'trigram(delightful witty improbable)': False, 'trigram(devolves flashy vaguely)': False, 'trigram(directors stylish closely)': False, 'trigram(disbelief several right)': False, 'trigram(discovery homosexuality rare)': False, 'trigram(dispense advice never)': False, 'trigram(distorts reality people)': False, 'trigram(documentary treat delightful)': False, 'trigram(drag queen cuteness)': False, 'trigram(dry would welcome)': False, 'trigram(dual performance patronising)': False, 'trigram(dubious product caught)': False, 'trigram(dull watching proverbial)': False, 'trigram(dumb humor deadly)': False, 'trigram(eat brussels sprouts)': False, 'trigram(endorsement things bean)': False, 'trigram(engaging performers pay)': False, 'trigram(entertaining dispense advice)': False, 'trigram(eve characterization hitler)': False, 'trigram(even worse usual)': False, 'trigram(expectation interfering son)': False, 'trigram(expresses basic emotions)': False, 'trigram(far zhang forte)': False, 'trigram(fastforward technology beautiful)': False, 'trigram(feature length little)': False, 'trigram(feel credits roll)': False, 'trigram(filmmaking sure look)': False, 'trigram(films captured chaos)': False, 'trigram(flashy vaguely silly)': False, 'trigram(fluttering stammering friendships)': False, 'trigram(footage holiday box)': False, 'trigram(forget pain feel)': False, 'trigram(forgiveness murder thin)': False, 'trigram(friendships bazadona liar)': False, 'trigram(frontal guess artifice)': False, 'trigram(full frontal guess)': False, 'trigram(fury forget pain)': False, 'trigram(genre offering goofy)': False, 'trigram(genuine sweet without)': False, 'trigram(genuinely engaging performers)': False, 'trigram(get substituting capturing)': False, 'trigram(grumbling tasty grub)': False, 'trigram(guess artifice acting)': False, 'trigram(guts confront joy)': False, 'trigram(handbagclutching sarandon tumult)': False, 'trigram(handheld blair witch)': False, 'trigram(heated passions jealousy)': False, 'trigram(history academy stretched)': False, 'trigram(hitler contrived nature)': False, 'trigram(holiday box office)': False, 'trigram(homosexuality rare family)': False, 'trigram(humor deadly dull)': False, 'trigram(identity alienation monstrous)': False, 'trigram(imagination big chill)': False, 'trigram(improbable romantic comedy)': False, 'trigram(innercity life dual)': False, 'trigram(insult deathdefying efforts)': False, 'trigram(intelligently made intentional)': False, 'trigram(interfering son discovery)': False, 'trigram(jazzy score looks)': False, 'trigram(jealousy betrayal forgiveness)': False, 'trigram(joy rising stale)': False, 'trigram(junk insult deathdefying)': False, 'trigram(kane appears screen)': False, 'trigram(kids subject matter)': False, 'trigram(label stands milder)': False, 'trigram(length little attention)': False, 'trigram(life dual performance)': False, 'trigram(lightweight filmmaking sure)': False, 'trigram(limited extent commitment)': False, 'trigram(literally adorns childhood)': False, 'trigram(litmus test carries)': False, 'trigram(little attention paid)': False, 'trigram(lot scarier tepid)': False, 'trigram(make movies watch)': False, 'trigram(may college kids)': False, 'trigram(meditation identity alienation)': False, 'trigram(men expresses basic)': False, 'trigram(mib label stands)': False, 'trigram(middleclass angst actually)': False, 'trigram(milder better seen)': False, 'trigram(mistake endorsement things)': False, 'trigram(monstrous murk need)': False, 'trigram(movies watch creating)': False, 'trigram(murder thin period)': False, 'trigram(murk need sell)': False, 'trigram(nature provocative conclusion)': False, 'trigram(need sell twisted)': False, 'trigram(nudity prevalence fastforward)': False, 'trigram(observant unfussily poetic)': False, 'trigram(office pie dubious)': False, 'trigram(oily arms dealer)': False, 'trigram(onehour documentary treat)': False, 'trigram(outcome disbelief several)': False, 'trigram(overkill predictable outcome)': False, 'trigram(pain feel credits)': False, 'trigram(paint dry would)': False, 'trigram(passions jealousy betrayal)': False, 'trigram(patronising reverence carol)': False, 'trigram(people make movies)': False, 'trigram(performance patronising reverence)': False, 'trigram(performances intelligently made)': False, 'trigram(period onehour documentary)': False, 'trigram(pie dubious product)': False, 'trigram(pileups requisite screaming)': False, 'trigram(plenty nudity prevalence)': False, 'trigram(poetic meditation identity)': False, 'trigram(police chief scarpia)': False, 'trigram(powered real danger)': False, 'trigram(predictable outcome disbelief)': False, 'trigram(prevalence fastforward technology)': False, 'trigram(product caught intricate)': False, 'trigram(proverbial paint dry)': False, 'trigram(provocative conclusion stars)': False, 'trigram(quarter dark disturbing)': False, 'trigram(reality people make)': False, 'trigram(requisite screaming captain)': False, 'trigram(reverence carol kane)': False, 'trigram(right actors kind)': False, 'trigram(roll stomach grumbling)': False, 'trigram(romantic comedy zippy)': False, 'trigram(sarandon tumult something)':

False, 'trigram(scarier tepid genre)': False, 'trigram(scarpia films captured)': False, 'trigram(scored powered real)': False, 'trigram(screaming captain sorority)': False, 'trigram(screenplay weirder middleclass)': False, 'trigram(searching quarter dark)': False, 'trigram(seen conversation starter)': False, 'trigram(selfhelp books help)': False, 'trigram(several right actors)': False, 'trigram(silly overkill predictable)': False, 'trigram(something full frontal)': False, 'trigram(son discovery homosexuality)': False, 'trigram(sophisticated audiences mistake)': False, 'trigram(sorority boys takes)': False, 'trigram(sort cute cloying)': False, 'trigram(squad car pileups)': False, 'trigram(stammering friendships bazadona)': False, 'trigram(stands milder better)': False, 'trigram(stars may college)': False, 'trigram(starter worlds observant)': False, 'trigram(stomach grumbling tasty)': False, 'trigram(story degraded handheld)': False, 'trigram(stretched barely feature)': False, 'trigram(stylish closely resembles)': False, 'trigram(subject matter adult)': False, 'trigram(substituting capturing innercity)': False, 'trigram(sweet without relying)': False, 'trigram(takes title literally)': False, 'trigram(tale heated passions)': False, 'trigram(tasty grub unconditional)': False, 'trigram(technology beautiful timeless)': False, 'trigram(tepid genre offering)': False, 'trigram(terrorism creeping predictability)': False, 'trigram(test carries day)': False, 'trigram(thin period onehour)': False, 'trigram(things bean abhors)': False, 'trigram(timeless universal tale)': False, 'trigram(title literally adorns)': False, 'trigram(tomcats cheap junk)': False, 'trigram(treat delightful witty)': False, 'trigram(trying eat brussels)': False, 'trigram(tumult something full)': False, 'trigram(understandable guts confront)': False, 'trigram(unfussily poetic meditation)': False, 'trigram(universal tale heated)': False, 'trigram(urban conflagration fury)': False, 'trigram(usual fluttering stammering)': False, 'trigram(vaguely silly overkill)': False, 'trigram(version tomcats cheap)': False, 'trigram(videocam footage holiday)': False, 'trigram(warner bros costumer)': False, 'trigram(watch creating screenplay)': False, 'trigram(watchable biggest disappointments)': False, 'trigram(watching proverbial paint)': False, 'trigram(weirder middleclass angst)': False, 'trigram(witch videocam footage)': False, 'trigram(witty improbable romantic)': False, 'trigram(women augustine brand)': False, 'trigram(work admittedly limited)': False, 'trigram(worlds observant unfussily)': False, 'trigram(worse usual fluttering)': False, 'trigram(would welcome improvement)': False, 'trigram(writer selfhelp books)': False, 'trigram(zippy jazzy score)': False, 'trigram(actors kind visual)': False, 'trigram(adult can get)': False, 'trigram(advice never comes)': False, 'trigram(amy career success)': False, 'trigram(animation dumb humor)': False, 'trigram(animation litmus test)': False, 'trigram(attention paid animation)': False, 'trigram(augustine brand visual)': False, 'trigram(biggest disappointments year)': False, 'trigram(brand visual trickery)': False, 'trigram(brussels sprouts less)': False, 'trigram(can get substituting)': False, 'trigram(career success lrb)': False, 'trigram(character acidic allmale)': False, 'trigram(character understandable guts)': False, 'trigram(clever angle women)': False, 'trigram(clever devolves flashy)': False, 'trigram(closely resembles year)': False, 'trigram(cloying material far)': False, 'trigram(coherent whole cop)': False, 'trigram(comes men expresses)': False, 'trigram(comes together coherent)': False, 'trigram(commitment two genuinely)': False, 'trigram(cute cloying material)': False, 'trigram(cuteness amy career)': False, 'trigram(danger less sophisticated)': False, 'trigram(dark disturbing good)': False, 'trigram(disappointments year even)': False, 'trigram(disturbing good match)': False, 'trigram(extent commitment two)': False, 'trigram(family movie genuine)': False, 'trigram(flair shows great)': False, 'trigram(forte shows constantly)': False, 'trigram(good handbagclutching sarandon)': False, 'trigram(good match sensibilities)': False, 'trigram(great cinema can)': False, 'trigram(intentional not police)': False, 'trigram(jived sex whole)': False, 'trigram(kind visual flair)': False, 'trigram(kiss really performances)': False, 'trigram(last kiss really)': False, 'trigram(less entertaining dispense)': False, 'trigram(less sophisticated audiences)': False, 'trigram(liar making character)': False, 'trigram(look clever angle)': False, 'trigram(lrb bestselling writer)': False, 'trigram(made intentional not)': False, 'trigram(makes movie work)': False, 'trigram(making character understandable)': False, 'trigram(match sensibilities two)': False, 'trigram(material far zhang)': False, 'trigram(matter adult can)': False, 'trigram(movie genuine sweet)': False, 'trigram(movie work admittedly)': False, 'trigram(neuroses comes men)': False, 'trigram(never comes together)': False, 'trigram(not entirely wholesome)': False, 'trigram(not police chief)': False, 'trigram(offering goofy lrb)': False, 'trigram(paid animation litmus)': False, 'trigram(pay good handbagclutching)': False, 'trigram(performers pay good)': False, 'trigram(predictability makes movie)': False, 'trigram(queen cuteness amy)': False, 'trigram(rare family movie)': False, 'trigram(real danger less)': False, 'trigram(really performances intelligently)': False, 'trigram(really plenty nudity)': False, 'trigram(relying animation dumb)': False, 'trigram(resembles year version)': False, 'trigram(rising stale material)': False, 'trigram(seeks character acidic)': False, 'trigram(sensibilities two directors)': False, 'trigram(sex whole lot)': False, 'trigram(shows constantly defies)': False, 'trigram(shows great cinema)': False, 'trigram(sprouts less entertaining)': False, 'trigram(stale material time)': False, 'trigram(stops clever devolves)': False, 'trigram(success lrb bestselling)': False, 'trigram(sure look clever)': False, 'trigram(together coherent whole)': False, 'trigram(trickery stops clever)': False, 'trigram(two directors stylish)': False, 'trigram(two genuinely engaging)': False, 'trigram(visual trickery stops)': False, 'trigram(whole cop flick)': False, 'trigram(whole lot scarier)': False, 'trigram(without relying animation)': False, 'trigram(year even worse)': False, 'trigram(year version tomcats)': False, 'trigram(zhang forte shows)': False, 'trigram(basic emotions love)': False, 'trigram(big chill rrb)': False, 'trigram(books help rrb)': False, 'trigram(chill rrb history)': False, 'trigram(emotions love terrorism)': False, 'trigram(entirely wholesome rrb)': False, 'trigram(grub unconditional love)': False, 'trigram(intricate plot old)': False, 'trigram(love story degraded)': False, 'trigram(love terrorism creeping)': False, 'trigram(old mib label)': False, 'trigram(old scored powered)': False, 'trigram(old warner bros)': False, 'trigram(plot old scored)': False, 'trigram(rrb alive last)': False, 'trigram(rrb history academy)': False, 'trigram(sell twisted love)': False, 'trigram(time old mib)': False, 'trigram(twisted love story)': False, 'trigram(unconditional love seeks)': False, 'trigram(wholesome rrb alive)': False, 'trigram(cliches like oily)': False, 'trigram(deathdefying efforts like)': False, 'trigram(efforts like trying)': False, 'trigram(flick cliches like)': False, 'trigram(like drag queen)': False, 'trigram(like oily arms)': False, 'trigram(like trying eat)': False, 'trigram(looks like drag)': False, 'trigram(score looks like)': False, 'trigram(welcome improvement like)': False, 'trigram(amy neuroses comes)': False, 'trigram(can really plenty)': False, 'trigram(cinema can really)': False, 'trigram(goofy lrb not)': False, 'trigram(lrb not entirely)': False, 'trigram(visual flair shows)': False, 'trigram(help rrb amy)': False, 'trigram(love seeks character)': False, 'trigram(material time old)': False, 'trigram(rrb amy neuroses)': False, 'trigram(improvement like old)': False, 'trigram(like old warner)': False}_1 = False 0 : 4 = 1.0 : 1.0

5. Logistic regression classification -

In logistic regression classification, the following key parameters are used:

1. Class Weight: This parameter assigns weights to each class. By default, all classes have a weight of one. In the "balanced" mode, weights are automatically adjusted inversely to class frequencies in the input data using the formula: .: n samples / (n classes * np.bincount(y)).
2. Solver: This refers to the algorithm used for solving the optimization problem.
3. Max Iter: This is the maximum number of iterations the solver performs to achieve convergence.

In this specific context, the class weight was set to "balanced," the solver used was L-BFGS, and the maximum number of iterations was set to 1000. These settings were applied in trials using various feature sets with the logistic regression algorithm.

1.Normal features without preprocessing:

	precision	recall	f1-score	support
neg	0.15	0.33	0.21	6
neu	0.67	0.70	0.68	46
pos	0.00	0.00	0.00	4
sneg	0.17	0.17	0.17	18
spos	0.43	0.35	0.38	26
accuracy			0.46	100
macro avg	0.28	0.31	0.29	100
weighted avg	0.46	0.46	0.46	100

Predicted	neg	neu	sneg	spos	All
Actual					
neg	2	2	1	1	6
neu	3	32	6	5	46
pos	2	0	1	1	4
sneg	4	6	3	5	18
spos	2	8	7	9	26
All	13	48	18	21	100

2.Preprocessed features:

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.55	0.91	0.68	46
pos	0.00	0.00	0.00	4
sneg	0.22	0.11	0.15	18
spos	0.46	0.23	0.31	26
accuracy			0.50	100
macro avg	0.25	0.25	0.23	100
weighted avg	0.41	0.50	0.42	100

Predicted Actual	neu	pos	sneg	spos	All
neg	6	0	0	0	6
neu	42	0	1	3	46
pos	2	0	1	1	4
sneg	13	0	2	3	18
spos	14	1	5	6	26
All	77	1	9	13	100

3.Bigram features:

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.55	0.91	0.68	46
pos	0.00	0.00	0.00	4
sneg	0.22	0.11	0.15	18
spos	0.46	0.23	0.31	26
accuracy			0.50	100
macro avg	0.25	0.25	0.23	100
weighted avg	0.41	0.50	0.42	100

Predicted Actual	neu	pos	sneg	spos	All
neg	6	0	0	0	6
neu	42	0	1	3	46
pos	2	0	1	1	4
sneg	13	0	2	3	18
spos	14	1	5	6	26
All	77	1	9	13	100

4. Negation word features:

```
  warnings.warn(
              precision    recall  f1-score   support

         neg       0.00      0.00      0.00         4
         neu       0.74      0.76      0.75        59
         pos       0.14      0.20      0.17         5
        sneg       0.20      0.20      0.20        15
        spos       0.33      0.24      0.28        17

    accuracy                           0.53       100
   macro avg       0.28      0.28      0.28       100
weighted avg       0.53      0.53      0.53       100



Predicted  neg  neu  pos  sneg  spos  All
Actual
neg          0    0    2     2     0    4
neu          2   45    0     9     3   59
pos          0    1    1     0     3    5
sneg         2    7    1     3     2   15
spos         1    8    3     1     4   17
All          5   61    7    15    12  100
```

5. Sentiment lexicon:

```
  warnings.warn(
              precision    recall  f1-score   support

         neg       0.00      0.00      0.00         4
         neu       0.74      0.76      0.75        59
         pos       0.14      0.20      0.17         5
        sneg       0.20      0.20      0.20        15
        spos       0.33      0.24      0.28        17

    accuracy                           0.53       100
   macro avg       0.28      0.28      0.28       100
weighted avg       0.53      0.53      0.53       100



Predicted  neg  neu  pos  sneg  spos  All
Actual
neg          0    0    2     2     0    4
neu          2   45    0     9     3   59
pos          0    1    1     0     3    5
sneg         2    7    1     3     2   15
spos         1    8    3     1     4   17
All          5   61    7    15    12  100
```

6. Trigram features:

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.55	0.91	0.68	46
pos	0.00	0.00	0.00	4
sneg	0.22	0.11	0.15	18
spos	0.46	0.23	0.31	26
accuracy			0.50	100
macro avg	0.25	0.25	0.23	100
weighted avg	0.41	0.50	0.42	100

Predicted Actual	neu	pos	sneg	spos	All
neg	6	0	0	0	6
neu	42	0	1	3	46
pos	2	0	1	1	4
sneg	13	0	2	3	18
spos	14	1	5	6	26
All	77	1	9	13	100

7.POS features:

	precision	recall	f1-score	support
neg	0.10	0.17	0.12	6
neu	0.57	0.65	0.61	46
pos	0.00	0.00	0.00	4
sneg	0.20	0.17	0.18	18
spos	0.41	0.35	0.38	26
accuracy			0.43	100
macro avg	0.26	0.27	0.26	100
weighted avg	0.41	0.43	0.42	100

Predicted Actual	neg	neu	sneg	spos	All
neg	1	5	0	0	6
neu	5	30	6	5	46
pos	1	0	2	1	4
sneg	1	7	3	7	18
spos	2	11	4	9	26
All	10	53	15	22	100

8. Combined feature set (Word features after preprocessing, SL, Bigram) :

	precision	recall	f1-score	support
neg	0.04	1.00	0.08	4
neu	0.00	0.00	0.00	59
pos	0.00	0.00	0.00	5
sneg	0.00	0.00	0.00	15
spos	0.00	0.00	0.00	17
accuracy			0.04	100
macro avg	0.01	0.20	0.02	100
weighted avg	0.00	0.04	0.00	100

Predicted	neg	All
Actual		
neg	4	4
neu	59	59
pos	5	5
sneg	15	15
spos	17	17
All	100	100

6. Decision Tree Classifier

The Decision Tree Classifier uses these parameters:

1. Criterion: This is a method to measure the quality of a split in the tree. It supports "gini" for Gini impurity and "entropy" for information gain.

2. Max Depth: This is the maximum depth the tree can reach. If not set (None), the tree grows until all leaves are pure or until there are only a few samples left.

3. Min Samples Split: This is the minimum number of samples required to split an internal node.

In our case, we've set the criterion to "gini", the maximum depth to 7, and the minimum samples required to split a node to 5. We used these settings in the decision tree classifier to test various sets of features.

1. Normal features without preprocessing:

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.59	0.83	0.69	46
pos	0.00	0.00	0.00	4
sneg	0.33	0.17	0.22	18
spos	0.32	0.23	0.27	26
accuracy			0.47	100
macro avg	0.25	0.24	0.24	100
weighted avg	0.42	0.47	0.43	100

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	0	4	0	1	1	6
neu	2	38	0	1	5	46
pos	0	0	0	1	3	4
sneg	2	9	0	3	4	18
spos	1	13	3	3	6	26
All	5	64	3	9	19	100

2. Preprocessed features:

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.48	0.96	0.64	46
pos	0.00	0.00	0.00	4
sneg	0.33	0.06	0.10	18
spos	0.75	0.12	0.20	26
accuracy			0.48	100
macro avg	0.31	0.23	0.19	100
weighted avg	0.47	0.48	0.36	100

Predicted Actual	neu	pos	sneg	spos	All
neg	6	0	0	0	6
neu	44	0	2	0	46
pos	3	0	0	1	4
sneg	17	0	1	0	18
spos	22	1	0	3	26
All	92	1	3	4	100

3. Bigram features

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.47	0.96	0.63	46
pos	0.00	0.00	0.00	4
sneg	0.33	0.06	0.10	18
spos	0.75	0.12	0.20	26
accuracy			0.48	100
macro avg	0.31	0.23	0.19	100
weighted avg	0.47	0.48	0.36	100

Predicted	neu	sneg	spos	All
Actual				
neg	6	0	0	6
neu	44	2	0	46
pos	3	0	1	4
sneg	17	1	0	18
spos	23	0	3	26
All	93	3	4	100

4. Negation word features

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	4
neu	0.68	0.90	0.77	59
pos	0.20	0.20	0.20	5
sneg	0.00	0.00	0.00	15
spos	0.23	0.18	0.20	17
accuracy			0.57	100
macro avg	0.22	0.25	0.23	100
weighted avg	0.45	0.57	0.50	100

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	0	2	1	0	1	4
neu	0	53	1	1	4	59
pos	0	1	1	0	3	5
sneg	0	12	1	0	2	15
spos	2	10	1	1	3	17
All	2	78	5	2	13	100

5. Sentiment lexicon

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	4
neu	0.66	0.92	0.77	59
pos	0.25	0.20	0.22	5
sneg	0.00	0.00	0.00	15
spos	0.22	0.12	0.15	17
accuracy			0.57	100
macro avg	0.23	0.25	0.23	100
weighted avg	0.44	0.57	0.49	100

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	0	2	1	0	1	4
neu	2	54	0	1	2	59
pos	0	1	1	0	3	5
sneg	0	13	1	0	1	15
spos	0	12	1	2	2	17
All	2	82	4	3	9	100

6. Trigram features

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.47	0.96	0.63	46
pos	0.00	0.00	0.00	4
sneg	0.33	0.06	0.10	18
spos	0.75	0.12	0.20	26
accuracy			0.48	100
macro avg	0.31	0.23	0.19	100
weighted avg	0.47	0.48	0.36	100

Predicted Actual	neu	sneg	spos	All
neg	6	0	0	6
neu	44	2	0	46
pos	3	0	1	4
sneg	17	1	0	18
spos	23	0	3	26
All	93	3	4	100

7. POS features

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	6
neu	0.53	0.89	0.66	46
pos	0.00	0.00	0.00	4
sneg	0.30	0.17	0.21	18
spos	0.50	0.23	0.32	26
accuracy			0.50	100
macro avg	0.27	0.26	0.24	100
weighted avg	0.43	0.50	0.42	100

Predicted	neu	sneg	spos	All
Actual				
neg	6	0	0	6
neu	41	2	3	46
pos	2	1	1	4
sneg	13	3	2	18
spos	16	4	6	26
All	78	10	12	100

8. Combined feature set (Word features after preprocessing, SL, Bigram)

	precision	recall	f1-score	support
neg	0.00	0.00	0.00	4
neu	0.59	1.00	0.74	59
pos	0.00	0.00	0.00	5
sneg	0.00	0.00	0.00	15
spos	0.00	0.00	0.00	17
accuracy			0.59	100
macro avg	0.12	0.20	0.15	100
weighted avg	0.35	0.59	0.44	100

Predicted	neu	All
Actual		
neg	4	4
neu	59	59
pos	5	5
sneg	15	15
spos	17	17
All	100	100

Comparative Analysis of Logistic Regression and Decision Tree classifier:

I will use Sci-kit learn's tools to classify opinions in this section. Since our target labels are similar to classification labels, I'll use Logistic Regression and Decision Tree Classifier methods. Additionally,I plan to test the effectiveness of both methods using five different sets of features.

Feature Set Type	Logistic Regression			Decision Tree Classifier		
	Precision	**Recall**	**F-1 score**	**Precision**	**Recall**	**F-1 score**
Normal features without preprocessing	**0.46**	**0.46**	**0.46**	**0.42**	**0.47**	**0.43**
Preprocessed features	**0.41**	**0.50**	**0.42**	**0.47**	**0.48**	**0.36**
Bigram features	**0.41**	**0.50**	**0.42**	**0.47**	**0.48**	**0.36**
Negation word features	**0.53**	**0.53**	**0.53**	**0.45**	**0.57**	**0.50**
Sentiment Lexicon Features	**0.53**	**0.53**	**0.53**	**0.44**	**0.57**	**0.59**
Trigram features	**0.41**	**0.50**	**0.42**	**0.47**	**0.48**	**0.36**
POS features	**0.41**	**0.43**	**0.42**	**0.43**	**0.50**	**0.42**
Combined feature set	**0.00**	**0.04**	**0.00**	**0.35**	**0.59**	**0.44**

In sentiment analysis using logistic regression, the use of Sentiment Lexicon features leads to better performance compared to other feature functions. This is largely because the train data has fewer words that are not seen before, thanks to the inclusion of these words and symbols in the Lexicon. The Lexicon includes entries for each word and token. We observed that recall rates are higher than the F-measure, but precision rates are somewhat lower.

Advanced Task:

Train the classifier on the entire training set and test it on a separately available test set.

```
def processkaggle(dirPath,limitStr):
  # convert the limit argument from a string to an int
  limit = int(limitStr)

  os.chdir(dirPath)

  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 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
      templist=[]
      if len(line.split('\t'))==3:
          templist.append(line.split('\t')[0])
          templist.append(line.split('\t')[2])
          testphrasedata.append(templist)
      else:
          templist.append(line.split('\t')[0])
          templist.append("")
          testphrasedata.append(templist)
  phraselist=testphrasedata

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

  # create list of phrase documents as (list of words, label)
  phrasedocs = []
```

```
  # add all the phrases
  for id,phrase in phraselist:

    # with pre processing
    tokenizer = RegexpTokenizer(r'\w+')
    phrase = pre_processing_documents(phrase)
    tokens = tokenizer.tokenize(phrase)
    phrasedocs.append((id, tokens))
```

```python
def create_test_submission(featuresets,test_featuresets,fileName):
  print ("---------------------------------------------------")
  print ("Training and testing a classifier ")
  test_set = test_featuresets
  training_set = featuresets
  classifier = nltk.NaiveBayesClassifier.train(training_set)
  fw = open(fileName,"w")
  fw.write("PhraseId"+','+"Sentiment"+'\n')
  for test,id in test_featuresets:
    fw.write(str(id)+','+str(classifier.classify(test))+'\n')
  fw.close()
```

Result:

```
---------------------------------------------------
Accuracy with pre-processed features :
Training and testing a classifier
Accuracy of classifier :
0.559400230681
---------------------------------------------------

The confusion matrix
  |    0    1    2    3    4 |
--+----------------------------+
0 |  <95> 193  375   66   10 |
1 |   92 <560>1830  205   20 |
2 |   54  401<7024> 466   32 |
3 |   26  183 2053 <879> 120 |
4 |   10   42  375  323 <172>|
--+----------------------------+
(row = reference; col = test)


('Read', 66292, 'phrases, using', 66292, 'test phrases')
---------------------------------------------------
```

Conclusion:

To handle memory issues when processing 1,556,060 records in the classifyKaggle.py file, I limited the dataset to 100 records of each type. This was done for two reasons: Firstly, while I could gather some feature sets from 1,56,060 records, the system struggled with combined feature sets or sometimes with Trigram feature sets. Secondly, to effectively test nine different feature sets, we found it best to work with 100 records of each. The highest accuracy I achieved was 90% using the Naive Bayes classifier on the POS feature set. In sentiment analysis using logistic regression, the use of Sentiment Lexicon features leads to better performance compared to other feature functions with precision 0.55, recall value as 0.55 and f1-value as 0.55 while using the logistic regression. While looking at the results from Decision tree analysis it is visible that preprocessed ,bigram, trigram have highest performance with precision of 0.47 ,recall 0.48 , f1-value as 0.36 .Considering all the three experiments the highest accuracy is achieved using Naïve Bayes classifier .The analysis led to conclude that classifying target variables into three categories - "Negative," "Neutral," and "Positive" - could improve accuracy. However, changing the composition of target variables should be considered for more complex cases. Later testing the features on test data and found the accuracy to be 0.56.Overall, the report showcases both the capabilities and challenges in sentiment analysis, offering insights for future explorations in the field.

Lessons Learned:

- During the project duration was able to work with various features necessary to get the required results
- Gained the knowledge necessary to categorize and forecast the feeling conveyed by a set of data.
- In this lesson, we analyzed all of the ideas that were presented by the lecturer in order to arrive at the required outcomes.
- Utilizing all of the concepts taught in the course helped to complete the project.
- Learnt to write programs in Python as per requirements.
- Testing features on different sizes of dataset gives us knowledge on the performance and the accuracy of various features.