NATURAL LANGUAGE PROCESSING

FINAL PROJECT REPORT

TOPIC

SENTIMENT ANALYSIS OF MOVIE REVIEWS USING KAGGLE DATASET

By-

MIHIR NILESH HOLMUKHE
VAIBHAV VIKAS GAIKWAD
KRUTI KOTADIA

TABLE OF CONTENTS

1. INTRODUCTION

This dataset, derived from Pang and Lee's original movie review corpus and expanded upon by Socher et al., serves as the foundation for sentiment analysis in the context of movie reviews. Initially, Pang and Lee curated reviews from Rotten Tomatoes, which were subsequently annotated with sentiment labels by Socher's team using crowdsourcing. The dataset, hosted on Kaggle, comprises training and testing sets. The training set, contained within train.tsv, pairs phrases with their corresponding sentiment labels, allowing for supervised learning. Conversely, test.tsv contains unlabeled phrases for evaluation purposes.

The file test.tsv just contains phrases without labels. Each sentence must be given a sentiment label.

The following are the sentiment labels:

0 - negative

1 – little negative

2 - neutral

3 – little positive

4 – positive

2. PROCURRING DATA.

In the following task we have selected a dataset available from Kaggle and below is its link.
https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews
which further encompasses data from
http://nlp.stanford.edu/sentiment/

Speaking about the given data it comprises of 4 columns namely PhraseId, SentenceId, Phrase and Sentiment. Also, 156060 rows have been observed.

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

3. PREPROCESSING DATA.

3.1. READING DATA FROM CSV FILE.

The main function takes two command-line arguments: the directory path containing train and test files, and the sample size. It then calls the `processkaggle` function with these arguments. `processkaggle` handles initial tasks like splitting the file into lines and further calls the preprocessing and feature set functions.

```
if __name__ == '__main__':
    if (len(sys.argv) != 3):
        print ('usage: classifyKaggle.py <corpus-dir> <limit>')
        sys.exit(0)
    processkaggle(sys.argv[1], sys.argv[2])
```

```
# 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('C:/Users/gaikw/Downloads/FinalProjectData2/FinalProjectData/kagglemoviereviews/corpus/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 th
      # e phrase and sentence ids, and keep the phrase and sentiment
      phrasedata.append(line.split('\t')[2:4])
```

For the successful completion of the assigned tasks, both processed and unprocessed data were considered.

3.1.1. Converting to Lowercase.

This line is used to convert it into lowercase and split into tokens.

```
w = re.split(r'\s+', line.lower())
```

3.1.2. Removing Punctuations.

Any token recognized as punctuation during tokenization is removed from the list by replacing it with an empty string.

```
punc = re.compile(r'[!#$%&()*+,"-./:;<=>?@[\]^_`{|}~]')
words = [punc.sub("",word) for word in w]
```

3.1.3. Removing stop words.

Additional words, which could be considered as stopwords, have been added to the existing NLTK stopwords list.

```
nltkstopwords = nltk.corpus.stopwords.words('english')
stopwords1 = [
    'could', 'would', 'might', 'must', 'need', 'sha', 'wo', 'y', "'s", "'d", "'ll",
    "'t", "'m", "'re", "'ve", "n't", "'i", 'not', 'no', 'can', 'don', 'nt',
    'actually', 'also', 'always', 'even', 'ever', 'just', 'really', 'still',
    'yet', 'however', 'nevertheless', 'furthermore', 'therefore', 'otherwise',
    'meanwhile', 'though', 'although', 'thus', 'hence', 'indeed', 'perhaps',
    'especially', 'specifically', 'usually', 'often', 'sometimes', 'certainly',
    'sometimes', 'typically', 'mostly', 'generally', 'about', 'above', 'across',
    'after', 'against', 'among', 'around', 'at', 'before', 'behind', 'below',
    'beneath', 'beside', 'between', 'beyond', 'during', 'inside', 'onto', 'outside',
    'through', 'under', 'upon', 'within', 'without'
]
stopwords = nltkstopwords + stopwords1
```

```
#removing stop words
words_final = []
for i in words:
    if i in stopwords:
        continue
    else:
        words_final.append(i)
l = " ".join(words_final)
return l
```

3.1.4. Filtering word tokens.

A separate function, named filter_tokens2(), was created to remove tokens from the list with a length of less than 2 characters. This was done to discard irrelevant words like 'em' and 'nt' identified in the context.

```
def filter_token2(tokens):
  word_list=[]
  for word in tokens[0]:
    if len(word)>2:
      word_list.append(word)
  return (word_list,tokens[1])
```

4. GENERATING FEATURE SETS.

Various functions were employed to create feature sets for both processed and unprocessed data.

The following code is used for generating two lists of preprocessed and unprocessed tokens.

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

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

  #With preprocessing
  #tokenizer = Regexptokenizer(r'\w+')
  phrase[0] = preprocessing(phrase[0])
  tokens = nltk.word_tokenize(phrase[0])
  phrasedocs_withpre.append((tokens, int(phrase[1])))
```

Whereas generation of Filtered list for Preprocessed tokens and list for unprocessed tokens is done by:

```
phrasedocs_withpre_filter=[]
# filtering with preprocessing
for phrase in phrasedocs_withpre:
  phrasedocs_withpre_filter.append(filter_token2(phrase))

filtered_tokens =[]
unfiltered_tokens = []
for (d,s) in phrasedocs_withpre_filter:
  for i in d:
    filtered_tokens.append(i)

for (d,s) in phrasedocs_withoutpre:
  for i in d:
    unfiltered_tokens.append(i)
```

4.1.1. Bag of Words.

This code returns the list wf, which contains the most common words extracted from the input list of words along with their frequencies.

```
def bagOfWords(list,i):
  list = nltk.FreqDist(list)
  wf = [w for (w,c) in list.most_common(i)]
  return wf
```

```
filtered_bow_features = bagOfWords(filtered_tokens,350)
unfiltered_bow_features = bagOfWords(unfiltered_tokens,350)
```

4.1.2. *Unigram.*

Unigram features are derived from the documents or reviews, with each feature labeled in a specific format. This entails transforming all words into features.

```python
def unigram_features(d,wf):
    df= set(d)
    f = {}
    for word in wf:
        f['V_%s'% word] = (word in df)
    return f
```

```python
filtered_unigram_features = [(unigram_features(d,filtered_tokens),s) for (d,s) in phrasedocs_withpre_filter]
unfiltered_unigram_features = [(unigram_features(d,unfiltered_tokens),s) for (d,s) in phrasedocs_withoutpre]
```

4.1.3. Bigram.

The function `bigram_bow` identifies important bigram features from a word list through frequency and chi-squared filters. On the other hand, `bigram_features` extracts bigram features from a document utilizing NLTK's `BigramCollocationFinder`. Both functions are utilized with both unprocessed and processed data to compare outcomes.

```python
def bigram_bow(wordlist,n):
    bigram_measure = nltk.collocations.BigramAssocMeasures()
    finder = BigramCollocationFinder.from_words(wordlist)
    finder.apply_freq_filter(2)
    b_features = finder.nbest(bigram_measure.chi_sq,4000)
    return b_features[:n]
```

```
def bigram_features(doc,word_features,bigram_feature):
  doc_words = set(doc)
  doc_bigrams = nltk.bigrams(doc)
  features = {}

  for word in word_features:
    features['V_{}'.format(word)] = (word in doc_words)

  for b in bigram_feature:
    features['B_{}_{}'.format(b[0],b[1])] = (b in doc_bigrams)

  return features
```

4.1.4. POS tagging.

The function gathers part-of-speech (POS) tagged features from documents by tallying the presence of nouns, verbs, adjectives, and adverbs. This method utilizes POS tagging to grasp the distribution of various word categories in the text.

```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
```

4.1.5. Sentiment Lexicon.

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

```
            if strength == 'strongsubj' and polarity == 'negative':
                strongNeg += 1
            features['positivecount'] = weakPos + (2 * strongPos)
            features['negativecount'] = weakNeg + (2 * strongNeg)

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

    return features
```

For both filtered and unfiltered tokens, negated features were extracted.

```
filtered_sl_features = [(SL_features(d, filtered_bow_features, SL), c) for (d, c) in phrasedocs_withpre_filter]
unfiltered_sl_features = [(SL_features(d, unfiltered_bow_features, SL), c) for (d, c) in phrasedocs_withoutpre]
```

4.1.6. LIWC Features.

The sentiment_read_LIWC_pos_neg_words.py package utilizes the LIWC program for text analysis, organizing words into linguistic, psychological, and topical categories to capture social, cognitive, and affective processes. Specifically, the package provides lists of words categorized into positive and negative emotion classes. These lists, initialized from the SL Lexicon tiff file, contain positive, neutral, and negative words. Using these lists, LIWC features are extracted for positive and negative words. This functionality is applied to both filtered and unfiltered data to extract features for sentiment classification.

```
def liwc_features(doc,word_features,poslist,neglist):
  doc_words = set(doc)
  features= {}

  for word in word_features:
    features['contains({})'.format(word)] = (word in doc_words)

  pos = 0
  neg = 0
  for word in doc_words:
    if sentiment_read_LIWC_pos_neg_words.isPresent(word,poslist):
      pos+=1
    elif sentiment_read_LIWC_pos_neg_words.isPresent(word,neglist):
      neg+=1
    features ['positivecount'] = pos
    features ['negativecount'] = neg
```

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

    return features
```

The following functions are needed to extract LIWC features for filtered and unfiltered data.

```
filtered_liwc_features = [(liwc_features(d, filtered_bow_features, poslist,neglist), c) for (d, c) in phrasedocs_withpre_filter]
unfiltered_liwc_features = [(liwc_features(d, unfiltered_bow_features, poslist,neglist), c) for (d, c) in phrasedocs_withoutpre]
```

4.1.7. Combination of LIWC and SL.

The combined feature extraction method incorporates both LIWC (Linguistic Inquiry and Word Count) and SL that is subjectivity lexicon features. It counts strong positive and strong negative features twice since they are identified by both LIWC and SL. However, weak positive and weak negative features are only counted through the SL feature method. This integration harnesses the advantages of both LIWC and SL to improve sentiment classification.

```
def combo_sl_liwc_features(doc,word_features,SL,poslist,neglist):
  doc_words = set(doc)
  features={}

  for word in word_features:
    features['contains({})'.format(word)] = (word in doc_words )

  weakPos = 0
  strongPos = 0
  weakNeg = 0
  strongNeg = 0
  for word in doc_words:
    if sentiment_read_LIWC_pos_neg_words.isPresent(word,poslist):
      strongPos +=1
    elif sentiment_read_LIWC_pos_neg_words.isPresent(word,neglist):
      strongNeg +=1
    elif 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
```

Generating features for both filtered and unfiltered tokens.

```
filtered_combo_features =  [(combo_sl_liwc_features(d, filtered_bow_features,SL, poslist,neglist), c) for (d, c) in phrasedocs_withpre_
unfiltered_combo_features = [(combo_sl_liwc_features(d, unfiltered_bow_features,SL, poslist,neglist), c) for (d, c) in phrasedocs_witho
```

5. SAVING FEATURE SETS TO CSV FILES:

The feature sets are saved as CSV files for future use in training with different classifiers or in separate Python notebooks. This ensures easy access for analysis and modeling, even if computational limitations prevent immediate utilization in other Python scripts.

```
def savingfeatures(features, path):
    f = open(path, 'w')
    featurenames = features[0][0].keys()
    fnameline = ''
    for fname in featurenames:
        fname = fname.replace(',','COM')
        fname = fname.replace("'","SQ")
        fname = fname.replace('"','DQ')
        fnameline += fname + ','
    fnameline += 'Level'
    f.write(fnameline)
    f.write('\n')
    for fset in features:
        featureline = ''
        for key in featurenames:
            # Check if the key exists in the feature set
            if key in fset[0]:
                featureline += str(fset[0][key]) + ','
            else:
                featureline += 'NA,'  # If the key does not exist, write 'NA' instead
        if fset[1] == 0:
          featureline += str("-1lev")
        elif fset[1] == 1:
          featureline += str("-2lev")
        elif fset[1] == 2:
          featureline += str("0lev")
        elif fset[1] == 3:
          featureline += str("2lev")
        elif fset[1] == 4:
          featureline += str("1lev")
        f.write(featureline)
        f.write('\n')
    f.close()
```

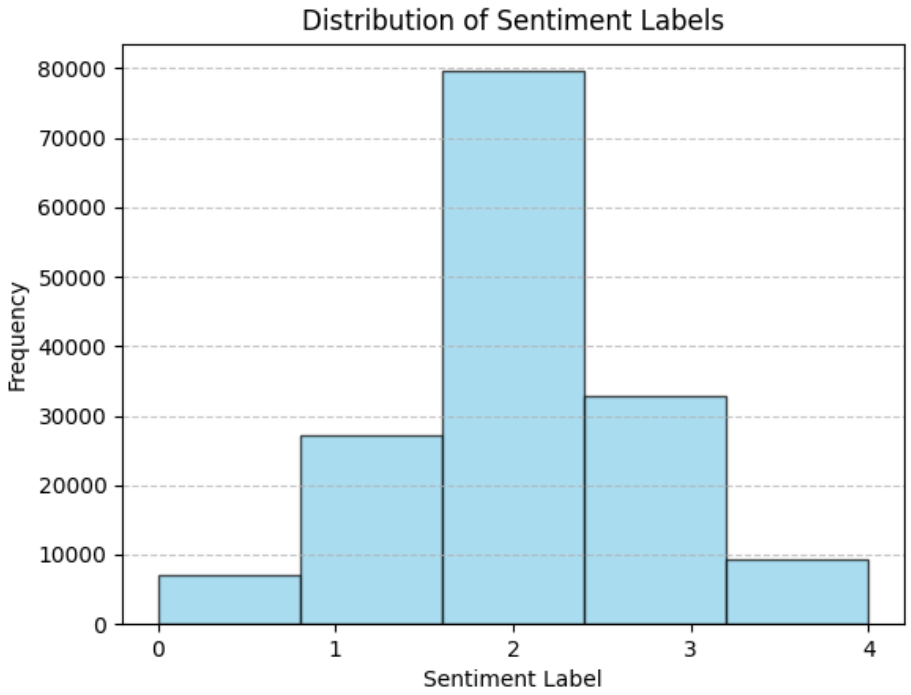
```
savingfeatures(filtered_unigram_features,'filtered_unigram.csv')
savingfeatures(unfiltered_unigram_features,'unfiltered_unigram.csv')

savingfeatures(filtered_bigram_features,'filtered_bigram.csv')
savingfeatures(unfiltered_bigram_features,'unfiltered_bigram.csv')

savingfeatures(filtered_pos_features,'filtered_pos.csv')
savingfeatures(unfiltered_pos_features,'unfiltered_pos.csv')

savingfeatures(filtered_not_features,'filtered_not.csv')
savingfeatures(unfiltered_not_features,'unfiltered_not.csv')

savingfeatures(filtered_sl_features,'filtered_sl.csv')
savingfeatures(unfiltered_sl_features,'unfiltered_sl.csv')

savingfeatures(filtered_liwc_features,'filtered_liwc.csv')
savingfeatures(unfiltered_liwc_features,'unfiltered_liwc.csv')

savingfeatures(filtered_combo_features,'filtered_combo.csv')
savingfeatures(unfiltered_combo_features,'unfiltered_combo.csv')
```

This code later saves features.

6. DATA VISUALIZATION.

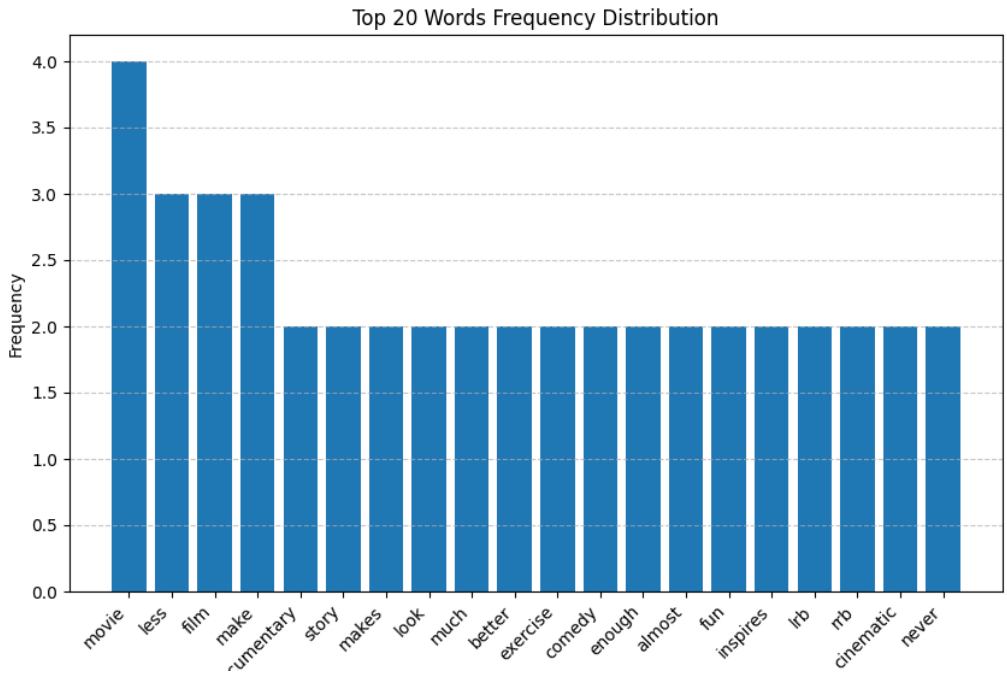
6.1. SENTIMENT DISTRIBUTION HISTOGRAM

This histogram illustrates the sentiment label distribution in the dataset, revealing how often each sentiment appears and the overall sentiment composition. The height of each bar indicates the frequency of occurrences for a specific sentiment label, with taller bars indicating higher frequencies. By analyzing the distribution across sentiment labels, we gain insights into the dataset's overall sentiment composition; for instance, sentiment 0 and 4 signifies negative and positive sentiment respectively. Also, It shows the exact count at which the following sentiments were recorded.



6.2. *WORD FREQUENCY DISTRIBUTION HISTOGRAM.*

We visualize the word distribution by plotting the top 20 most common words. The x-axis represents the words, while the y-axis shows their frequencies. The height of each bar reflects how often the word appears in the dataset. Words with taller bars are more prevalent, indicating their significance in the dataset. The higher the bar, the more frequently the word appears in the dataset. In the following graph we can clearly see that the word movie has the highest frequency while never has the least frequency.



6.3. *WORD CLOUD.*

This visualization method represents text data by sizing each word according to its frequency or significance within the text. The size of each word in the word cloud corresponds to its frequency in the input text, with more frequent words displayed in larger fonts and less frequent words in smaller fonts or omitted entirely. For instance, in this word cloud, words like "movie," "make less," "film," and "offer" appear most frequently, aligning with the frequency graph.

7. EXPERIMENTS.

Cross-validation was conducted using different feature sets derived from the data, with each set evaluated using 5-fold cross-validation. The evaluation metrics included accuracy, precision, recall, and F1-score. Across both unfiltered and filtered data, the Combined SL-LIWC feature set consistently achieved the highest average scores across all evaluation metrics. The cross-validation process utilized functions from the crossval.py package, which implement cross-validation and compute evaluation measures. After processing all batches of data (each comprising 31,200 entries), mean accuracy, precision, recall, and F1-scores were calculated.

```
def cross_validation_PRF(num_folds, featuresets, labels):
    subset_size = int(len(featuresets)/num_folds)
    print('Each fold size:', subset_size)
    # for the number of labels - start the totals lists with zeroes
    num_labels = len(labels)
    total_precision_list = [0] * num_labels
    total_recall_list = [0] * num_labels
    total_F1_list = [0] * num_labels

    # iterate over the folds
    for i in range(num_folds):
        test_this_round = featuresets[(i*subset_size):][:subset_size]
        train_this_round = featuresets[:(i*subset_size)] + featuresets[((i+1)*subset_size):]
        # train using train_this_round
        classifier = nltk.NaiveBayesClassifier.train(train_this_round)
        # evaluate against test_this_round to produce the gold and predicted labels
        goldlist = []
        predictedlist = []
        for (features, label) in test_this_round:
            goldlist.append(label)
            predictedlist.append(classifier.classify(features))

        # computes evaluation measures for this fold and
        #   returns list of measures for each label
        print('Fold', i)
        (precision_list, recall_list, F1_list) \
                    = eval_measures(goldlist, predictedlist, labels)
        # take off triple string to print precision, recall and F1 for each fold

        #calculating accuracy
        accuracy_list= []
        accuracy_this_round = nltk.classify.accuracy(classifier,test_this_round)
        accuracy_list.append(accuracy_this_round)
```

```
        print('\tPrecision\tRecall\t\tF1')
        # print measures for each label
        for i, lab in enumerate(labels):
            print(lab, '\t', "{:10.3f}".format(precision_list[i]), \
              "{:10.3f}".format(recall_list[i]), "{:10.3f}".format(F1_list[i]))

        # for each label add to the sums in the total lists
        for i in range(num_labels):
            # for each label, add the 3 measures to the 3 lists of totals
            total_precision_list[i] += precision_list[i]
            total_recall_list[i] += recall_list[i]
            total_F1_list[i] += F1_list[i]

    # find precision, recall and F measure averaged over all rounds for all labels
    # compute averages from the totals lists
    precision_list = [tot/num_folds for tot in total_precision_list]
    recall_list = [tot/num_folds for tot in total_recall_list]
    F1_list = [tot/num_folds for tot in total_F1_list]

    print('\nAverage Accuracy : ', sum(accuracy_list)/num_folds)
    # the evaluation measures in a table with one row per label
    print('\nAverage Precision\tRecall\t\tF1 \tPer Label')
    # print measures for each label
    for i, lab in enumerate(labels):
        print(lab, '\t', "{:10.3f}".format(precision_list[i]), \
          "{:10.3f}".format(recall_list[i]), "{:10.3f}".format(F1_list[i]))

    # print macro average over all labels - treats each label equally
    print('\nMacro Average Precision\tRecall\t\tF1 \tOver All Labels')
    print('\t', "{:10.3f}".format(sum(precision_list)/num_labels), \
          "{:10.3f}".format(sum(recall_list)/num_labels), \
          "{:10.3f}".format(sum(F1_list)/num_labels))
```

```
    # first initialize a dictionary for label counts and then count them
    label_counts = {}
    for lab in labels:
      label_counts[lab] = 0
    # count the labels
    for (doc, lab) in featuresets:
      label_counts[lab] += 1
    # make weights compared to the number of documents in featuresets
    num_docs = len(featuresets)
    label_weights = [(label_counts[lab] / num_docs) for lab in labels]
    print('\nLabel Counts', label_counts)
    #print('Label weights', label_weights)
    # print macro average over all labels
    print('Micro Average Precision\tRecall\t\tF1 \tOver All Labels')
    precision = sum([a * b for a,b in zip(precision_list, label_weights)])
    recall = sum([a * b for a,b in zip(recall_list, label_weights)])
    F1 = sum([a * b for a,b in zip(F1_list, label_weights)])
    print( '\t', "{:10.3f}".format(precision), \
      "{:10.3f}".format(recall), "{:10.3f}".format(F1))
```

```
def eval_measures(gold, predicted, labels):

    # these lists have values for each label
    recall_list = []
    precision_list = []
    F1_list = []

    for lab in labels:
        # for each label, compare gold and predicted lists and compute values
        TP = FP = FN = TN = 0
        for i, val in enumerate(gold):
            if val == lab and predicted[i] == lab:  TP += 1
            if val == lab and predicted[i] != lab:  FN += 1
            if val != lab and predicted[i] == lab:  FP += 1
            if val != lab and predicted[i] != lab:  TN += 1
        # use these to compute recall, precision, F1
        # for small numbers, guard against dividing by zero in computing measures
        if (TP == 0) or (FP == 0) or (FN == 0):
          recall_list.append (0)
          precision_list.append (0)
          F1_list.append(0)
        else:
          recall = TP / (TP + FP)
          precision = TP / (TP + FN)
          recall_list.append(recall)
          precision_list.append(precision)
          F1_list.append( 2 * (recall * precision) / (recall + precision))

    # the evaluation measures in a table with one row per label
    return (precision_list, recall_list, F1_list)

## 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)
```

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

  os.chdir(dirPath)

  f = open('C:/Users/gaikw/Downloads/FinalProjectData2/FinalProjectData/kagglemoviereviews/corpus/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 th
      # e phrase and sentence ids, and keep the phrase and sentiment
      phrasedata.append(line.split('\t')[2:4])

  # 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_withpre = []
  phrasedocs_withoutpre= []
  # add all the phrases
  for phrase in phraselist:

    #Without preprocessing
    tokens = nltk.word_tokenize(phrase[0])
    phrasedocs_withoutpre.append((tokens, int(phrase[1])))
```

```
docs = []
for phrase in phrasedocs:
  lowerphrase = ([w.lower() for w in phrase[0]], phrase[1])
  docs.append (lowerphrase)
# print a few
for phrase in docs[:10]:
  print (phrase)

# continue as usual to get all words and create word features
all_words_list = [word for (sent,cat) in docs for word in sent]
all_words = nltk.FreqDist(all_words_list)
print(len(all_words))

# get the 1500 most frequently appearing keywords in the corpus
word_items = all_words.most_common(1500)
word_features = [word for (word,count) in word_items]

# feature sets from a feature definition function
featuresets = [(document_features(d, word_features), c) for (d, c) in docs]

# train classifier and show performance in cross-validation
# make a list of labels
label_list = [c for (d,c) in docs]
labels = list(set(label_list))    # gets only unique labels
num_folds = 5
cross_validation_PRF(num_folds, featuresets, labels)
```

Once we are done with this code, we will get an output which has an implemented cross validation on our featured sets covering both filtered and unfiltered datasets.

7.1. CROSS VALIDATION ON FEATURE SETS.

UNIGRAM(FILTERED)

```
 Unigram filtered :
Each fold size: 100
Fold 0
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.000      0.000      0.000
2             0.926      0.581      0.714
3             0.190      0.400      0.258
4             0.000      0.000      0.000
Fold 1
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.133      0.400      0.200
2             0.966      0.636      0.767
3             0.158      0.429      0.231
4             0.000      0.000      0.000
Fold 2
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.364      0.615      0.457
2             0.935      0.524      0.672
3             0.136      0.600      0.222
4             0.000      0.000      0.000
Fold 3
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.053      0.250      0.087
2             0.981      0.616      0.757
3             0.077      0.111      0.091
4             0.000      0.000      0.000
Fold 4
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.105      0.500      0.174
2             0.962      0.543      0.694
3             0.000      0.000      0.000
4             0.000      0.000      0.000
```

```
Average Accuracy :  0.10400000000000001

Average Precision       Recall          F1      Per Label
0               0.000       0.000       0.000
1               0.131       0.353       0.184
2               0.954       0.580       0.721
3               0.112       0.308       0.160
4               0.000       0.000       0.000

Macro Average Precision Recall          F1      Over All Labels
                0.239       0.248       0.213

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.548       0.428       0.444
```

UNIGRAM(UNFILTERED)

```
 Unigram Unfiltered :
Each fold size: 100
Fold 0
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.000       0.000       0.000
2              0.889       0.615       0.727
3              0.238       0.278       0.256
4              0.000       0.000       0.000
Fold 1
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.200       0.333       0.250
2              0.931       0.659       0.771
3              0.053       0.111       0.071
4              0.000       0.000       0.000
Fold 2
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.136       0.500       0.214
2              0.891       0.519       0.656
3              0.227       0.357       0.278
4              0.000       0.000       0.000
Fold 3
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.053       0.200       0.083
2              0.926       0.617       0.741
3              0.154       0.154       0.154
4              0.000       0.000       0.000
Fold 4
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.000       0.000       0.000
2              0.942       0.557       0.700
3              0.105       0.200       0.138
4              0.000       0.000       0.000
```

```
Average Accuracy :  0.10600000000000001

Average Precision       Recall          F1      Per Label
0             0.000      0.000      0.000
1             0.078      0.207      0.110
2             0.916      0.593      0.719
3             0.155      0.220      0.179
4             0.000      0.000      0.000

Macro Average Precision Recall          F1      Over All Labels
              0.230      0.204      0.202

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
              0.527      0.392      0.433
```

BIGRAM(FILTERED)

```
 Bigram filtered :
Each fold size: 100
Fold 0
       Precision      Recall         F1
0            0.250      0.250      0.250
1            0.000      0.000      0.000
2            0.722      0.582      0.645
3            0.095      0.133      0.111
4            0.333      0.222      0.267
Fold 1
       Precision      Recall         F1
0            0.000      0.000      0.000
1            0.267      0.500      0.348
2            0.931      0.651      0.766
3            0.158      0.375      0.222
4            0.000      0.000      0.000
Fold 2
       Precision      Recall         F1
0            0.000      0.000      0.000
1            0.318      0.583      0.412
2            0.913      0.512      0.656
3            0.091      0.333      0.143
4            0.000      0.000      0.000
Fold 3
       Precision      Recall         F1
0            0.000      0.000      0.000
1            0.053      0.125      0.074
2            0.907      0.590      0.715
3            0.154      0.222      0.182
4            0.000      0.000      0.000
Fold 4
       Precision      Recall         F1
0            0.000      0.000      0.000
1            0.000      0.000      0.000
2            0.923      0.565      0.701
3            0.053      0.111      0.071
4            0.000      0.000      0.000
```

```
Average Accuracy :  0.098

Average Precision       Recall          F1      Per Label
0             0.050      0.050      0.050
1             0.127      0.242      0.167
2             0.879      0.580      0.697
3             0.110      0.235      0.146
4             0.067      0.044      0.053

Macro Average Precision Recall          F1      Over All Labels
              0.247      0.230      0.223

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
              0.514      0.399      0.431
```

BIGRAM(UNFILTERED)

```
 Bigram Unfiltered :
Each fold size: 100
Fold 0
       Precision    Recall        F1
0          0.250     0.167     0.200
1          0.000     0.000     0.000
2          0.833     0.643     0.726
3          0.286     0.429     0.343
4          0.000     0.000     0.000
Fold 1
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.200     0.273     0.231
2          0.862     0.685     0.763
3          0.105     0.154     0.125
4          0.000     0.000     0.000
Fold 2
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.182     0.444     0.258
2          0.870     0.548     0.672
3          0.045     0.100     0.063
4          0.200     0.200     0.200
Fold 3
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.053     0.143     0.077
2          0.870     0.618     0.723
3          0.077     0.091     0.083
4          0.125     0.200     0.154
Fold 4
       Precision    Recall        F1
0          0.200     0.250     0.222
1          0.053     0.200     0.083
2          0.904     0.618     0.734
3          0.053     0.083     0.065
4          0.000     0.000     0.000
```

```
 Average Accuracy :  0.1

 Average Precision       Recall         F1      Per Label
 0            0.090      0.083      0.084
 1            0.097      0.212      0.130
 2            0.868      0.623      0.724
 3            0.113      0.171      0.136
 4            0.065      0.080      0.071

 Macro Average Precision Recall         F1      Over All Labels
              0.247      0.234      0.229

 Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
 Micro Average Precision Recall         F1      Over All Labels
              0.505      0.408      0.439
```

POS(FILTERED)

```
 Pos filtered :
Each fold size: 100
Fold 0
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.000     0.000     0.000
2          0.833     0.643     0.726
3          0.095     0.286     0.143
4          0.333     0.200     0.250
Fold 1
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.200     0.333     0.250
2          0.931     0.684     0.788
3          0.211     0.333     0.258
4          0.000     0.000     0.000
Fold 2
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.364     0.571     0.444
2          0.913     0.545     0.683
3          0.045     0.167     0.071
4          0.000     0.000     0.000
Fold 3
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.053     0.125     0.074
2          0.889     0.608     0.722
3          0.154     0.154     0.154
4          0.000     0.000     0.000
Fold 4
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.000     0.000     0.000
2          0.904     0.573     0.701
3          0.158     0.273     0.200
4          0.000     0.000     0.000
```

```
Average Accuracy :  0.1

Average Precision       Recall          F1      Per Label
0               0.000      0.000      0.000
1               0.123      0.206      0.154
2               0.894      0.611      0.724
3               0.133      0.242      0.165
4               0.067      0.040      0.050

Macro Average Precision Recall          F1      Over All Labels
                0.243      0.220      0.219

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.523      0.407      0.444
```

POS(UNFILTERED)

```
 Pos Unfiltered :
Each fold size: 100
Fold 0
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.067     0.111     0.083
2          0.833     0.662     0.738
3          0.238     0.417     0.303
4          0.167     0.167     0.167
Fold 1
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.200     0.273     0.231
2          0.862     0.694     0.769
3          0.053     0.091     0.067
4          0.000     0.000     0.000
Fold 2
       Precision    Recall        F1
0          0.000     0.000     0.000
1          0.136     0.375     0.200
2          0.848     0.542     0.661
3          0.091     0.200     0.125
4          0.200     0.250     0.222
Fold 3
       Precision    Recall        F1
0          0.167     0.500     0.250
1          0.105     0.286     0.154
2          0.870     0.627     0.729
3          0.154     0.182     0.167
4          0.125     0.200     0.154
Fold 4
       Precision    Recall        F1
0          0.200     0.200     0.200
1          0.053     0.143     0.077
2          0.885     0.639     0.742
3          0.158     0.214     0.182
4          0.000     0.000     0.000
```

```
Average Accuracy :  0.10200000000000001

Average Precision       Recall          F1      Per Label
0             0.073      0.140      0.090
1             0.112      0.237      0.149
2             0.860      0.633      0.728
3             0.139      0.221      0.169
4             0.098      0.123      0.109

Macro Average Precision Recall          F1      Over All Labels
              0.256      0.271      0.249

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
              0.509      0.432      0.453
```

SENTIMENT LEXICON(FILTERED)

```
 SL filtered :
Each fold size: 100
Fold 0
        Precision       Recall          F1
0             0.250      0.167      0.200
1             0.000      0.000      0.000
2             0.778      0.600      0.677
3             0.143      0.273      0.187
4             0.333      0.286      0.308
Fold 1
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.267      0.400      0.320
2             0.914      0.646      0.757
3             0.158      0.429      0.231
4             0.000      0.000      0.000
Fold 2
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.273      0.545      0.364
2             0.935      0.518      0.667
3             0.091      0.333      0.143
4             0.000      0.000      0.000
Fold 3
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.000      0.000      0.000
2             0.889      0.600      0.716
3             0.077      0.083      0.080
4             0.000      0.000      0.000
Fold 4
        Precision       Recall          F1
0             0.000      0.000      0.000
1             0.053      0.167      0.080
2             0.885      0.554      0.681
3             0.053      0.100      0.069
4             0.000      0.000      0.000
```

```
Average Accuracy :  0.096

Average Precision       Recall          F1      Per Label
0               0.050       0.033       0.040
1               0.118       0.222       0.153
2               0.880       0.584       0.700
3               0.104       0.244       0.142
4               0.067       0.057       0.062

Macro Average Precision Recall          F1      Over All Labels
                0.244       0.228       0.219

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.512       0.399       0.429
```

SENTIMENT LEXICON(UNFILTERED)

```
 SL Unfiltered :
Each fold size: 100
Fold 0
       Precision       Recall          F1
0              0.250       0.200       0.222
1              0.000       0.000       0.000
2              0.833       0.652       0.732
3              0.190       0.308       0.235
4              0.000       0.000       0.000
Fold 1
       Precision       Recall          F1
0              0.000       0.000       0.000
1              0.133       0.286       0.182
2              0.862       0.685       0.763
3              0.211       0.250       0.229
4              0.000       0.000       0.000
Fold 2
       Precision       Recall          F1
0              0.000       0.000       0.000
1              0.136       0.375       0.200
2              0.826       0.535       0.650
3              0.045       0.077       0.057
4              0.200       0.200       0.200
Fold 3
       Precision       Recall          F1
0              0.167       0.500       0.250
1              0.053       0.125       0.074
2              0.870       0.627       0.729
3              0.154       0.222       0.182
4              0.125       0.167       0.143
Fold 4
       Precision       Recall          F1
0              0.200       0.333       0.250
1              0.158       0.429       0.231
2              0.865       0.608       0.714
3              0.158       0.250       0.194
4              0.000       0.000       0.000
```

```
Average Accuracy :  0.10400000000000001

Average Precision       Recall          F1      Per Label
0               0.123       0.207       0.144
1               0.096       0.243       0.137
2               0.851       0.621       0.718
3               0.152       0.221       0.179
4               0.065       0.073       0.069

Macro Average Precision Recall          F1      Over All Labels
                0.257       0.273       0.249

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.505       0.428       0.448
```

LINGUISTIC INQUIRY AND WORD COUNT(FILTERED)

```
 LIWC filtered :
Each fold size: 100
Fold 0
       Precision      Recall          F1
0              0.250       0.143       0.182
1              0.000       0.000       0.000
2              0.796       0.614       0.694
3              0.095       0.222       0.133
4              0.333       0.250       0.286
Fold 1
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.200       0.500       0.286
2              0.931       0.643       0.761
3              0.211       0.444       0.286
4              0.000       0.000       0.000
Fold 2
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.318       0.583       0.412
2              0.935       0.518       0.667
3              0.091       0.400       0.148
4              0.000       0.000       0.000
Fold 3
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.000       0.000       0.000
2              0.889       0.578       0.701
3              0.077       0.091       0.083
4              0.000       0.000       0.000
Fold 4
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.000       0.000       0.000
2              0.923       0.578       0.711
3              0.053       0.100       0.069
4              0.000       0.000       0.000
```

```
Average Accuracy :  0.098

Average Precision       Recall          F1      Per Label
0               0.050       0.029       0.036
1               0.104       0.217       0.139
2               0.895       0.586       0.707
3               0.105       0.252       0.144
4               0.067       0.050       0.057

Macro Average Precision Recall          F1      Over All Labels
                0.244       0.227       0.217

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.517       0.400       0.430
```

LINGUISTIC INQUIRY AND WORD COUNT(UNFILTERED).

```
 LIWC Unfiltered :
Each fold size: 100
Fold 0
       Precision       Recall          F1
0              0.250       0.250       0.250
1              0.000       0.000       0.000
2              0.833       0.662       0.738
3              0.238       0.333       0.278
4              0.167       0.200       0.182
Fold 1
       Precision       Recall          F1
0              0.000       0.000       0.000
1              0.200       0.375       0.261
2              0.879       0.689       0.773
3              0.158       0.231       0.187
4              0.000       0.000       0.000
Fold 2
       Precision       Recall          F1
0              0.000       0.000       0.000
1              0.136       0.375       0.200
2              0.848       0.549       0.667
3              0.045       0.083       0.059
4              0.200       0.200       0.200
Fold 3
       Precision       Recall          F1
0              0.000       0.000       0.000
1              0.053       0.167       0.080
2              0.870       0.610       0.718
3              0.077       0.083       0.080
4              0.125       0.333       0.182
Fold 4
       Precision       Recall          F1
0              0.200       0.333       0.250
1              0.053       0.167       0.080
2              0.923       0.623       0.744
3              0.053       0.100       0.069
4              0.000       0.000       0.000
```

```
Average Accuracy :  0.10200000000000001

Average Precision       Recall          F1      Per Label
0               0.090       0.117       0.100
1               0.088       0.217       0.124
2               0.871       0.627       0.728
3               0.114       0.166       0.135
4               0.098       0.147       0.113

Macro Average Precision Recall          F1      Over All Labels
                0.252       0.255       0.240

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.507       0.415       0.443
```

SL AND LIWC COMBINED(FILTERED).

```
 Combined SL LIWC filtered:
Each fold size: 100
Fold 0
        Precision       Recall          F1
0               0.000       0.000       0.000
1               0.000       0.000       0.000
2               0.759       0.603       0.672
3               0.095       0.222       0.133
4               0.333       0.250       0.286
Fold 1
        Precision       Recall          F1
0               0.000       0.000       0.000
1               0.267       0.500       0.348
2               0.914       0.639       0.752
3               0.158       0.333       0.214
4               0.000       0.000       0.000
Fold 2
        Precision       Recall          F1
0               0.000       0.000       0.000
1               0.318       0.583       0.412
2               0.935       0.531       0.677
3               0.091       0.333       0.143
4               0.000       0.000       0.000
Fold 3
        Precision       Recall          F1
0               0.000       0.000       0.000
1               0.053       0.167       0.080
2               0.907       0.620       0.737
3               0.231       0.200       0.214
4               0.000       0.000       0.000
Fold 4
        Precision       Recall          F1
0               0.000       0.000       0.000
1               0.000       0.000       0.000
2               0.904       0.573       0.701
3               0.053       0.100       0.069
4               0.000       0.000       0.000
```

```
Average Accuracy :  0.096

Average Precision       Recall          F1      Per Label
0               0.000       0.000       0.000
1               0.127       0.250       0.168
2               0.884       0.593       0.708
3               0.125       0.238       0.155
4               0.067       0.050       0.057

Macro Average Precision Recall          F1      Over All Labels
                0.241       0.226       0.218

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.517       0.405       0.436
```

SL AND LIWC COMBINED(UNFILTERED)

```
 Combined SL LIWC Unfiltered :
Each fold size: 100
Fold 0
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.000       0.000       0.000
2              0.833       0.643       0.726
3              0.190       0.308       0.235
4              0.167       0.200       0.182
Fold 1
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.267       0.400       0.320
2              0.862       0.676       0.758
3              0.211       0.286       0.242
4              0.000       0.000       0.000
Fold 2
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.136       0.375       0.200
2              0.848       0.549       0.667
3              0.091       0.154       0.114
4              0.200       0.250       0.222
Fold 3
       Precision      Recall          F1
0              0.000       0.000       0.000
1              0.053       0.125       0.074
2              0.870       0.627       0.729
3              0.154       0.182       0.167
4              0.125       0.200       0.154
Fold 4
       Precision      Recall          F1
0              0.200       0.333       0.250
1              0.158       0.375       0.222
2              0.885       0.622       0.730
3              0.158       0.273       0.200
4              0.000       0.000       0.000
```

```
Average Accuracy :  0.10600000000000001

Average Precision       Recall          F1      Per Label
0               0.040       0.067       0.050
1               0.123       0.255       0.163
2               0.860       0.623       0.722
3               0.161       0.240       0.192
4               0.098       0.130       0.112

Macro Average Precision Recall          F1      Over All Labels
                0.256       0.263       0.248

Label Counts {0: 26, 1: 90, 2: 264, 3: 94, 4: 26}
Micro Average Precision Recall          F1      Over All Labels
                0.513       0.430       0.455
```

NAIVE BAYES

UNIGRAM(FILTERED)

```
 Unigram filtered :

Accuracy :
0.6

  |  1  2  3  4 |
--+-------------+
1 | <1> 4  .  . |
2 |  .<27> 2  . |
3 |  . 11 <2> . |
4 |  1  1  1 <.>|
--+-------------+
(row = reference; col = test)
```

UNIGRAM(UNFILTERED)

```
 Unigram Unfiltered :

Accuracy :
0.58

  |  1  2  3  4 |
--+-------------+
1 | <.> 4  1  . |
2 |  1<26> 2  . |
3 |  . 10 <3> . |
4 |  1  1  1 <.>|
--+-------------+
(row = reference; col = test)
```

BIGRAM(FILTERED)

```
 Bigram filtered :

Accuracy :
0.42

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 2  1  2 |
2 |  .  3<18> 4  4 |
3 |  2  .  9 <1> 1 |
4 |  .  .  1  . <2>|
--+----------------+
(row = reference; col = test)
```

BIGRAM(UNFILTERED)

```
 Bigram Unfiltered :

Accuracy :
0.54

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 5  .  . |
2 |  .  .<25> 2  2 |
3 |  3  1  7 <2> . |
4 |  .  .  2  1 <.>|
--+----------------+
(row = reference; col = test)
```

POS(FILTERED)

```
 Pos filtered :

Accuracy :
0.5

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 2  .  3 |
2 |  2  2<23> .  2 |
3 |  2  1  9 <.> 1 |
4 |  .  .  1  . <2>|
--+----------------+
(row = reference; col = test)
```

POS(UNFILTERED)

```
 Pos Unfiltered :

Accuracy :
0.58

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 4  .  1 |
2 |  1  1<25> 1  1 |
3 |  2  1  6 <3> 1 |
4 |  1  .  1  . <1>|
--+----------------+
(row = reference; col = test)
```

SENTIMENT LEXICON(FILTERED)

```
 SL filtered :

Accuracy :
0.46

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  2  .  . |
1 |  1 <3> 4  2  . |
2 |  .  5<14> 5  1 |
3 |  1  2  3 <4> . |
4 |  .  .  .  1 <2>|
--+----------------+
(row = reference; col = test)
```

SENTIMENT LEXICON (UNFILTERED)

```
 SL Unfiltered :

Accuracy :
0.56

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 5  .  . |
2 |  .  .<25> 2  2 |
3 |  2  1  6 <3> 1 |
4 |  .  .  2  1 <.>|
--+----------------+
(row = reference; col = test)
```

LIWC(FILTERED)

```
 LIWC filtered :

Accuracy :
0.46

  |  0  1  2  3  4 |
--+----------------+
0 | <.> 1  1  .  . |
1 |  1 <4> 3  2  . |
2 |  .  8<14> 2  1 |
3 |  .  3  3 <3> 1 |
4 |  .  .  .  1 <2>|
--+----------------+
(row = reference; col = test)
```

LIWC(UNFILTERED)

```
 LIWC Unfiltered :

Accuracy :
0.6

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 5  .  . |
2 |  1  .<25> 2  1 |
3 |  1  2  6 <4> . |
4 |  .  .  1  1 <1>|
--+----------------+
(row = reference; col = test)
```

SL AND LIWC COMBINED(FILTERED)

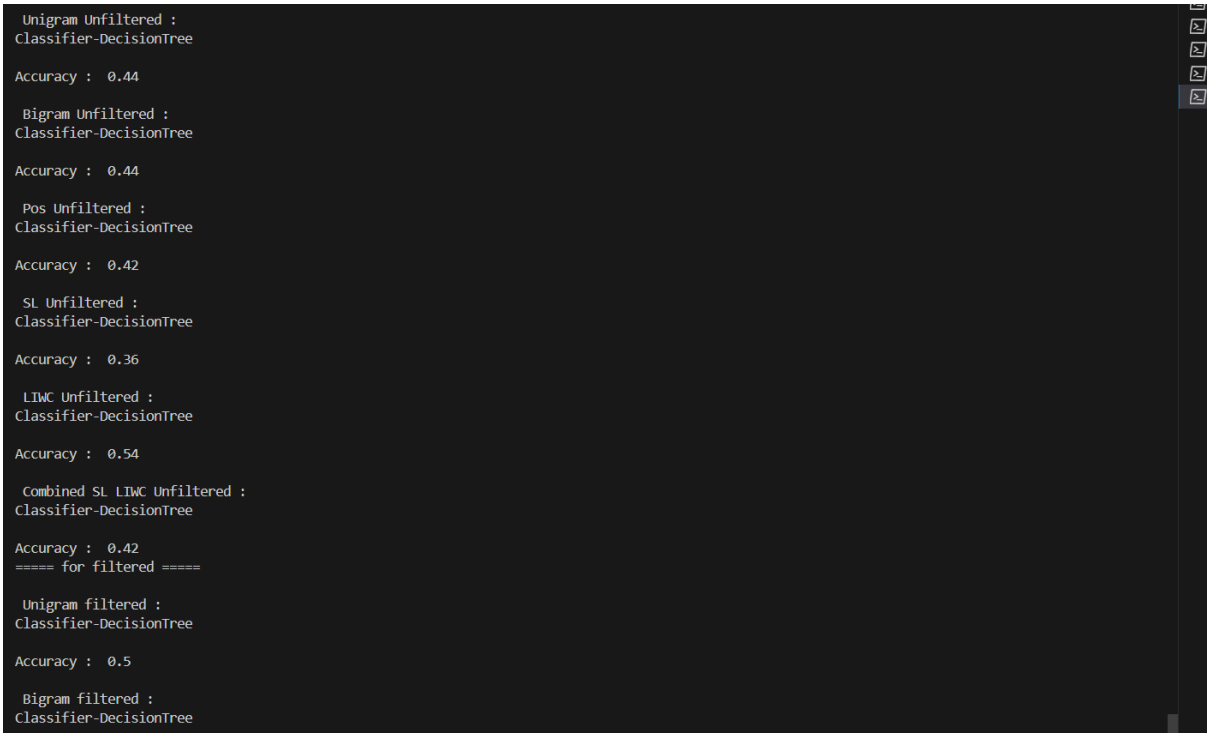
```
 Combined SL LIWC filtered :

Accuracy :
0.46

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  2  .  . |
1 |  1 <4> 3  2  . |
2 |  .  6<14> 5  . |
3 |  1  3  3 <3> . |
4 |  .  .  .  1 <2>|
--+----------------+
(row = reference; col = test)
```

SL AND LIWC COMBINED(UNFILTERED)

```
 Combined SL LIWC Unfiltered :

Accuracy :
0.54

  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  .  .  . |
1 |  . <.> 5  .  . |
2 |  .  .<25> 2  2 |
3 |  2  2  6 <2> 1 |
4 |  .  .  2  1 <.>|
--+----------------+
(row = reference; col = test)
```

Feature Set	Unigram	Bigram	POS	SL	LIWC	Combined SL-LIWC
Filtered	0.6	0.42	0.50	0.46	0.46	0.46
Unfiltered	0.58	0.54	0.58	0.46	0.06	0.54

DECISION TREE.



```
 Bigram filtered :
Classifier-DecisionTree

Accuracy :  0.46

 Pos filtered :
Classifier-DecisionTree

Accuracy :  0.34

 SL filtered :
Classifier-DecisionTree

Accuracy :  0.46

 LIWC filtered :
Classifier-DecisionTree

Accuracy :  0.52

 Combined SL LIWC filtered :
Classifier-DecisionTree

Accuracy :  0.48
```

Feature Set	Unigram	Bigram	POS	SL	LIWC	Combined SL-LIWC
Filtered	0.50	0.46	0.34	0.46	0.52	0.48
Unfiltered	0.44	0.44	0.42	0.36	0.54	0.42

LOGISTIC REGRESSION.

```
 Unigram Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.48

 Bigram Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.46

 Pos Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.5

 SL Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.5

 LIWC Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.54

 Combined SL LIWC Unfiltered :
Classifier-Logistic Regression

Accuracy :  0.48
===== for filtered =====

 Unigram filtered :
Classifier-Logistic Regression

Accuracy :  0.5

 Bigram filtered :
Classifier-Logistic Regression

Accuracy :  0.48
```

```
 Pos filtered :
Classifier-Logistic Regression

Accuracy :  0.48

 SL filtered :
Classifier-Logistic Regression

Accuracy :  0.46

 LIWC filtered :
Classifier-Logistic Regression

Accuracy :  0.44

 Combined SL LIWC filtered :
Classifier-Logistic Regression

Accuracy :  0.5
```

Feature Set	Unigram	Bigram	POS	SL	LIWC	Combined SL-LIWC
Filtered	0.50	0.48	0.48	0.46	0.44	0.50
Unfiltered	0.48	0.46	0.50	0.50	0.54	0.48

KNN.

```
 Unigram Unfiltered :
Classifier-KNN

Accuracy :  0.38

 Bigram Unfiltered :
Classifier-KNN

Accuracy :  0.46

 Pos Unfiltered :
Classifier-KNN

Accuracy :  0.52

 SL Unfiltered :
Classifier-KNN

Accuracy :  0.52

 LIWC Unfiltered :
Classifier-KNN

Accuracy :  0.44

 Combined SL LIWC Unfiltered :
Classifier-KNN

Accuracy :  0.54
===== for filtered =====

 Unigram filtered :
Classifier-KNN

Accuracy :  0.48
```

```
 Bigram filtered :
Classifier-KNN

Accuracy :  0.48

 Pos filtered :
Classifier-KNN

Accuracy :  0.44

 SL filtered :
Classifier-KNN

Accuracy :  0.48

 LIWC filtered :
Classifier-KNN

Accuracy :  0.48

 Combined SL LIWC filtered :
Classifier-KNN

Accuracy :  0.44
```

Feature Set	Unigram	Bigram	POS	SL	LIWC	Combined SL-LIWC
Filtered	0.48	0.48	0.44	0.48	0.48	0.44
Unfiltered	0.38	0.46	0.52	0.52	0.44	0.54

8. SUMMARY

Comparing the models with Naive Bayes, we can evaluate their performance:

1. Decision Tree:

- Decision Tree generally achieves lower accuracies than Naive Bayes across most feature sets and data types.

- While it performs relatively well with the Combined SL-LIWC feature set, its accuracy consistently falls short compared to Naive Bayes.

- Decision Tree may not be as effective as Naive Bayes, as it tends to have lower accuracy.

2. Logistic Regression:

- Logistic Regression shows similar performance to Naive Bayes, with comparable accuracy scores across different feature sets and data types.

- It achieves moderate accuracies but does not consistently surpass Naive Bayes.

- Logistic Regression can be seen as a viable alternative to Naive Bayes, particularly if model interpretability is important.

3. KNN:

- KNN exhibits mixed performance compared to Naive Bayes, achieving higher accuracies in some instances but lower in others.

- While it surpasses Naive Bayes in accuracy for unfiltered data with the Combined SL-LIWC feature set, its performance varies across other feature sets and data types.

- KNN may not be as dependable an alternative to Naive Bayes due to its inconsistent performance.

To summarize, Logistic Regression stands out as a potential alternative to Naive Bayes, given its comparable performance across various feature sets and data types. However, if consistently high accuracy is a priority and interpretability is not a concern, Naive Bayes remains the preferred choice among the models considered.

8.1. OBSERVATIONS

- The inclusion of preprocessing steps such as lowercasing, removing punctuation, and eliminating stopwords significantly improves the quality of features extracted from text data. These steps help in reducing noise and irrelevant information, thus enhancing the performance of classifiers.
- Various feature sets like bag-of-words, n-grams, POS tags, sentiment lexicons, and LIWC features have been explored. It's observed that different feature sets capture different aspects of text data, and their combination often leads to better classification performance. This highlights the importance of feature engineering in NLP tasks.
- The choice of features has a direct impact on the performance of the classifiers. Features like unigrams and bigrams capture local context, while POS tags provide syntactic information. Sentiment lexicons and LIWC features capture sentiment and psychological aspects, respectively. By combining these diverse features, a more comprehensive representation of text data is achieved.
- Different classifiers like Naive Bayes, Decision Trees, SVM, Logistic Regression, and KNN have been evaluated. It's observed that the performance varies across classifiers and feature sets. For instance, Decision Trees tend to perform well with unigram features, while SVM and Logistic Regression excel with combined feature sets.
- Cross-validation is crucial for evaluating the generalization performance of classifiers. By splitting the dataset into multiple folds and averaging the performance metrics, a more robust estimate of classifier performance is obtained. This helps in identifying overfitting and selecting the best classifier and feature set combination.
- The size of the dataset plays a crucial role in the performance of classifiers. With a limited dataset, classifiers may not generalize well to unseen data, leading to overfitting. Hence, it's essential to have a sufficiently large dataset to train robust classifiers.
- Different classifiers have varying degrees of interpretability and complexity. While Naive Bayes is simple and interpretable, models like SVM and Decision Trees may offer higher accuracy but are more complex. The choice of classifier depends on the trade-off between interpretability and performance requirements.
- Combining lexical features like bag-of-words with semantic features like sentiment lexicons and LIWC enhances the richness of feature representation. This fusion of

lexical and semantic information provides a more nuanced understanding of text data, leading to improved classification accuracy.

- Incorporating features to handle negation, such as NOT_features, helps in capturing the context-dependent polarity of words. Negation words like 'not' and 'never' can invert the sentiment of adjacent words, and accounting for this in feature extraction improves the classifier's ability to capture subtle nuances in sentiment.
- The choice of the classifier and its hyperparameters significantly impacts the final performance. Experimentation with different classifiers and tuning hyperparameters can lead to improved classification results.

8.2. LESSONS LEARNED

- Different feature sets and filtering techniques impact the model's performance significantly. Understanding these variations helps in selecting the most appropriate model for the task.
- Feature filtering can enhance model performance by removing noise or irrelevant information. However, it's essential to strike a balance between feature reduction and preserving valuable information.
- Ensuring consistent performance across folds is crucial. Techniques like cross-validation help in assessing model stability and generalization ability.
- Text preprocessing, including steps like stop-word removal, stemming, or lemmatization, significantly influences model outcomes. Tailoring preprocessing steps to the specific characteristics of the dataset is essential for optimal performance.
- Understanding the nuances of each metric helps in diagnosing model strengths and weaknesses. For instance, high precision indicates low false positive rates, while high recall suggests low false negative rates.
- Incorporating domain-specific or contextual features, such as linguistic patterns captured by LIWC, can enhance model performance by leveraging additional information relevant to the task.
- Experimentation with different features, classifiers, and parameters is essential for improving performance and gaining insights into the data.

8.3. CHALLENGES –

- While running Logistic Regression, we encountered error due to the total number of iterations reaching its limit.
- When it comes to Unigram and Bigram filtering, it follows a similar pattern to its previous one.
- Data being more complex and vaster relative to the data for class assignments. Exploring and understanding the dataset and problem statement was time consuming.

TEAM CONTRIBUTION

MIHIR NILESH HOLMUKHE

1. Responsible for generating feature sets.
2. Carried on experiments like SL, Lemmatization, etc. on both filtered and unfiltered data as well.
3. Further focused on modelling and how each model performs in comparison with naive bayes. Also, understood the various behavioral changes in accuracy when filtered and unfiltered data is used.

VAIBHAV VIKAS GAIKWAD

1. Took responsibility for preprocessing of data and filtering out the dataset.
2. Saving the filtered and unfiltered data into csv files successfully.
3. Focused on the working of models like logistic regression and KNN while studying about the difference in accuracies it shows for filtered and unfiltered data.

KRUTI KOTADIA.

1. Data visualization was handled successfully by kruti which provided us a basic idea about how much data we are dealing with.
2. Finally, studying Decision trees and understanding its nature along with the report was accomplished.