Sentiment Classification Of Movie Reviews

by

Dixitha Kasturi
Venkata Naga Abhiram Kuppa

Table of Contents

DATASET :

This dataset was created for the Kaggle competition, which can be found here: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews , and it incorporates data from Socher et al's sentiment analysis, which can be found at http://nlp.stanford.edu/sentiment/ .

The data came from Pang and Lee's original movie review corpus, which was based on Rotten Tomatoes reviews. Socher's team used crowd-sourcing to manually mark all of the sub-sentences with a sentiment label that ranged from "negative," "little negative," "neutral," "little positive," and "positive."

The data is split into training and testing The phrases and their related sentiment labels can be found in train.tsv. 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

APPROACH

Step 1: Reading Data from CSV file

Step 2: Preprocessing and filtering Data

Step 3: Generating Feature sets(included new features and combined features)

Step 4 : Saving Featuresets to CSV files

Step 5: Running cross validation on all Feature sets

Step 6: Running Naïve Bayes Classifier on Feature sets

Step 7: Running Random Forest Classifier on Feature sets(advanced experiment)

1 Reading data from csv

The main function takes 2 system arguments when executed. The first one is the path to the directory where the train and test files are at, the second one is a number which represents the sample size. Within this function, the function processkaggle takes these arguments, does initial processing like splitting the file into lines to further call the preprocessing function 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])


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 th
      # e phrase and sentence ids, and keep the phrase and sentiment
      phrasedata.append(line.split('\t')[2:4])
```

2 Preprocessing and filtering Data

Both processed and unprocessed data was considered for all the experiments that were done.

2.1 Converting to lowercase:

Each line was converted inti lowercase and split into tokens

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

2.2 Removing punctuation marks:

For each of the split tokens, the ones that classify as punctuations are removed from the list by substituting them with nothing.

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

2.3 Removing stop words:

The predefined nltkstopwords list, some additional list of words were added that would classify as stopwords

```
words_final = []
  for i in words:
```

```
        if i in stopwords:
          continue
        else:
          words_final.append(i)
```

2.4 Filtering word tokens:

A separate function (*filter_tokens2()*)was written to filter out tokens that were less than 2 in length, because we found a few words like 'em, 'nt etc which did not make sense.

```
def filter_token2(line):
  li=[]
  #print("line 0 ",line[0])
  for i in line[0]:
    if len(i)>2:
      li.append(i)
  return (li,line[1])
```

3 Generating Feature sets

To generate feature sets for both preprocessed and unprocessed data, different methods listed below were used. For each of the features generated, different experiments were conducted

Generating two lists of preprocessed and unprocessed tokens :

```
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
    phrase[0] = preprocessing(phrase[0])
    tokens = nltk.word_tokenize(phrase[0])
    phrasedocs_withpre.append((tokens, int(phrase[1])))
```

Generating Filtered list for Preprocessed tokens and list for unprocessed tokens:

```
phrasedocs_withpre_filter=[]
# filtering with preprocessing and length > 2
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)
```

3.1 Bag of Words :

```
def bagOfWords(wordlist):
  wordlist = nltk.FreqDist(wordlist)
  word_feature = [w for (w,c) in wordlist.most_common(100)]
  return word_feature


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

3.2 Unigram :

For the document/reviews, unigram features are extracted, the features have a label "V_labelname". We take all words and turn them into Features

```
def unigram_features(doc,word_features):
  doc_words = set(doc)
  features = {}
  for word in word_features:
    features['V_%s'% word] = (word in doc_words)
  return features
```

Unigram features are extracted for both filtered and unfiltered tokens.

```
filtered_unigram_features = [(unigram_features(d,filtered_bow_features),s) for
(d,s) in phrasedocs_withpre]
unfiltered_unigram_features = [(unigram_features(d,unfiltered_bow_features),s)
for (d,s) in phrasedocs_withoutpre]
```

3.3 Bigram

The below two functions are for getting out the bigrams that are present in the data given. The function bigram_bow is like the bigram function we have learnt in class where we find the bigram collections from words in the wordlist with window size 3 and then apply filter to get bigram of only those words which have a frequency count of 6 and higher with the apply_freq_filter function. The we get the bigrams with the chi squared to find the nbest bigrams features which is returned. The bigram feature extraction's function i.e bigram_features is the snippet we have used in one of our labs to extract bigram features. This functions will be applied for both unfiltered and filtered data to compare the results.

```
def bigram_bow(wordlist):
  bigram_measure = nltk.collocations.BigramAssocMeasures()
  finder = BigramCollocationFinder.from_words(wordlist,window_size=3)
  finder.apply_freq_filter(6)
  b_features = finder.nbest(bigram_measure.chi_sq,3000)
  return b_features[:300]


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
```

Bigram features Extraction using the above function for filtered and unfiltered data

```
filtered_bow_bigram_features=bigram_bow(filtered_tokens)
unfiltered_bow_bigram_features=bigram_bow(unfiltered_tokens)

filtered_bow_bigram_features=bigram_bow(filtered_tokens)
unfiltered_bigram_features =
[(bigram_features(d,unfiltered_bow_features,unfiltered_bow_bigram_features),s) for
(d,s) in phrasedocs_withoutpre]
```

3.4 POS tagging

Part-of-speech tagged features were extracted, Counts of various sorts of word tags are the most popular way to use POS tagging information. the function below is a feature function that counts nouns, verbs, adjectives, and adverbs.

```
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

  filtered_pos_features = [(POS_features(d,filtered_bow_features),s) for (d,s) in
phrasedocs_withpre]

  unfiltered_pos_features = [(POS_features(d,unfiltered_bow_features),s) for (d,s)
in phrasedocs_withoutpre]
```

3.5 Not features

Negation or contradictions in opinions should be considered when performing sentiment classification. The list of negation words considered are listed below. We can expand the list.One of the methods to analyze negation words is to find the negation word and then negate the word that follows it. This was used in the function defined below. Extracting all features and if a word is preceeded by a negation word, then we negate the current word and take it as a feature.

```
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone',
'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither',
'nor','oppressive','revenge','revolting','rude','sticky','sick',
'sorry','tense','ugly','unwanted','unwelcome','pain','vile','vicious','quit','nons
ense','guilty','impossible','hate','damage','dead','alarming','angry','annoy','cor
rupt','creepy','cruel','cry','dishonest','dirty','evil',
'enraged','reject','sad','terrifying','stupid','yell']

def NOT_features(document, new_word_features, negationwords):
    features = {}
    for word in new_word_features:
        features['V_{}'.format(word)] = False
        features['V_NOT{}'.format(word)] = False
    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
```

```
        if ((i + 1) < len(document)) and ((word in negationwords) or
(word.endswith("n't"))):
            i += 1
            features['V_NOT{}'.format(document[i])] = (document[i] in
new_word_features)
        else:
            features['V_{}'.format(word)] = (word in new_word_features)
    return features
```

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

```
  filtered_not_features = [(NOT_features(d, filtered_bow_features, negationwords),
c) for (d, c) in phrasedocs_withpre]

  unfiltered_not_features = [(NOT_features(d, unfiltered_bow_features,
negationwords), c) for (d, c) in phrasedocs_withoutpre]
```

3.6 SL features(Sentiment lexicon)

First, read subjective words from a subjective lexicon file prepared as part of the MPQA project by Janice Wiebe and her team at the University of Pittsburgh. These words are frequently employed as a function or in combination with other information, but they serve two purposes, one of which is to count the number of subjective positive and negative words included in each sentence document. Each word is matched to its counterpart in a list that includes both intensity and polarity. Each weak subjective word contains both positive and negative subjective words. Each strong subjective word is counted twice, and each strong subjective word is counted once. Positive count and negative count are kept track of.

```
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 these features were generated.

```
  filtered_sl_features = [(SL_features(d, filtered_bow_features, SL), c) for (d,
c) in phrasedocs_withpre]

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

3.7 LIWC features

Linguistic Inquiry and Word Count(LIWC) is a text analysis program that calculates the percentage of words in each text that fall into one or more of over 80 linguistic, psychological and topical categories indicating various social, cognitive, and affective processes. Here we try to label the text to either a positive tweet or a negative one based on the sentiment_read_LIWC_pos_neg_words.py package given which returns words in positive emotion class and words in negative emotion class. The parameters poslist and neglist are initialized from the SL Lexicon tiff file given and brings out the positive, neutral and negative lists using which LIWC features on pos and neg words are extracted below. This function is then used to extract features for both the filtered and unfiltered data.

```
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
```

LIWC features Extraction using the above function for filtered and unfiltered data

```
  filtered_liwc_features = [(liwc_features(d, filtered_bow_features,
poslist,neglist), c) for (d, c) in phrasedocs_withpre]

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

3.8 Combination of LIWC and SL:

Both LIWC AND SL features extraction methods were combined, here the strong positive and strong negative features were counted twice, by LIWC and by SL. Whereas weak positive and weak negative feature are counted only through SL feature method.

```
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
```

For both filtered and unfiltered tokens, features were generated.

```
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_withoutpre]
```

4 Saving Feature sets to CSV files :

All featuresets generated so far have been saved into csv files so they can be used as training sets for any other classifier or in a different python notebook. Because of computational issues, we were not able to use these featuresets in a different python script.

A function to save the csv files was written:

```
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:
            featureline += str(fset[0][key]) + ','
        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()
```

To save features, the lines below were used for all features

```
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')
```

filtered_combo.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,036 KB
unfiltered_combo.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,034 KB
filtered_liwc.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,036 KB
filtered_not.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	2,061 KB
filtered_sl.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,033 KB
unfiltered_liwc.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,034 KB
unfiltered_not.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	2,059 KB
unfiltered_sl.csv	5/2/2022 11:06 AM	Microsoft Excel Com...	1,032 KB
filtered_bigram.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	1,078 KB
filtered_pos.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	1,038 KB
unfiltered_bigram.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	1,547 KB
unfiltered_pos.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	1,036 KB
unfiltered_unigram.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	4,629 KB
filtered_unigram.csv	5/2/2022 11:05 AM	Microsoft Excel Com...	4,079 KB

Sample CSV file

contains(film)	contains(movie)	contains(one)	contains(rrb)	contains(lrb)	contains(g	contains(k	contains(l	contains(t	contains(s	contains(r	positivecc	negativec	Level
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	1	-2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	1	-1lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	1	-2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	0	0	-2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	1	-2lev
TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	-2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	1	0	2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	2	0	2lev
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	2	0	-2lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0lev

Experiments :

5 Running Cross Validation on all Feature sets

For all feature sets generated, cross validation was done using 5 folds. The average accuracy, precision, recall and f-score measures for all features were very close with *Combined SL-LIWC features* giving the highest average accuracy, precision, recall and f-score: 0.1064, 0.531, 0.494, 0.498 scores respectively for **unfiltered** data while similarly for **filtered** data we have scores in the form 0.1085, 0.543 ,0.495 ,0.499 respectively for each evaluation measure.

All the above cross validation functions are a result of running the data on the package given in crossval.py which contains the functions for cross validation and evaluation measure which we have learnt in our labs, which provide the accuracy scores for each batch of data that we run which is 31200 entries for each batch and a total of 5 batches. We then find the mean accuracy, precision, recall and F1 scores for each batch and collectively.

```
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))
    # for micro averaging, weight the scores for each label by the number of items
    #    this is better for labels with imbalance
    # first intialize 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))
# Function to compute precision, recall and F1 for each label
#  and for any number of labels
# Input: list of gold labels, list of predicted labels (in same order)
# Output: returns lists of precision, recall and F1 for each label
#       (for computing averages across folds and labels)
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)
```

5.1 Unigram:

For unfiltered tokens:

```
print("\n Unigram Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_unigram_features,labels)
```

```
 Unigram Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.207      0.170      0.187
1             0.182      0.338      0.237
2             0.840      0.594      0.696
3             0.160      0.326      0.214
4             0.126      0.277      0.173
Fold 1
        Precision       Recall          F1
0             0.211      0.161      0.182
1             0.171      0.336      0.226
2             0.847      0.598      0.701
3             0.173      0.352      0.232
4             0.119      0.262      0.163
Fold 2
        Precision       Recall          F1
0             0.187      0.150      0.167
1             0.166      0.329      0.221
2             0.846      0.602      0.703
3             0.163      0.336      0.220
4             0.121      0.227      0.158
Fold 3
        Precision       Recall          F1
0             0.201      0.173      0.186
1             0.174      0.336      0.229
2             0.850      0.593      0.698
3             0.159      0.329      0.214
4             0.133      0.275      0.179
Fold 4
        Precision       Recall          F1
0             0.210      0.178      0.193
1             0.187      0.346      0.243
2             0.844      0.600      0.701
3             0.168      0.344      0.225
4             0.122      0.234      0.161
```

```
Average Accuracy :  0.10319871794871796

Average Precision       Recall          F1      Per Label
0             0.203      0.167      0.183
1             0.176      0.337      0.231
2             0.845      0.597      0.700
3             0.165      0.337      0.221
4             0.124      0.255      0.167

Macro Average Precision Recall          F1      Over All Labels
              0.303      0.339      0.300

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
              0.513      0.457      0.462
```

For Unigram filtered tokens:

```
print("\n Unigram filtered : ")
crossval.cross_validation_PRF(5,filtered_unigram_features,labels)
```

```
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0           0.051        0.346       0.089
1           0.089        0.328       0.140
2           0.927        0.559       0.697
3           0.158        0.380       0.224
4           0.065        0.359       0.111
Fold 1
        Precision       Recall          F1
0           0.050        0.320       0.086
1           0.090        0.345       0.142
2           0.931        0.558       0.698
3           0.168        0.417       0.240
4           0.067        0.320       0.111
Fold 2
        Precision       Recall          F1
0           0.048        0.322       0.083
1           0.090        0.353       0.143
2           0.928        0.557       0.696
3           0.162        0.397       0.231
4           0.066        0.340       0.110
Fold 3
        Precision       Recall          F1
0           0.055        0.330       0.095
1           0.082        0.349       0.132
2           0.929        0.555       0.695
3           0.154        0.388       0.220
4           0.062        0.302       0.104
Fold 4
        Precision       Recall          F1
0           0.053        0.294       0.090
1           0.083        0.340       0.133
2           0.928        0.554       0.693
3           0.157        0.380       0.223
4           0.071        0.405       0.121
```

```
Average Accuracy :  0.10510897435897434

Average Precision       Recall          F1      Per Label
0               0.051       0.322       0.089
1               0.087       0.343       0.138
2               0.928       0.556       0.696
3               0.160       0.393       0.227
4               0.066       0.345       0.111

Macro Average Precision Recall          F1      Over All Labels
                0.259       0.392       0.252

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
                0.529       0.461       0.437
```

5.2 Bigram:

For unfiltered :

```
print("\n Bigram Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_bigram_features,labels)
```

```
 Bigram Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.207      0.170      0.187
1             0.182      0.338      0.237
2             0.840      0.594      0.696
3             0.160      0.326      0.214
4             0.126      0.277      0.173
Fold 1
        Precision       Recall          F1
0             0.211      0.161      0.182
1             0.171      0.336      0.226
2             0.847      0.598      0.701
3             0.173      0.352      0.232
4             0.119      0.262      0.163
Fold 2
        Precision       Recall          F1
0             0.187      0.150      0.167
1             0.166      0.329      0.221
2             0.846      0.602      0.703
3             0.163      0.336      0.220
4             0.121      0.227      0.157
Fold 3
        Precision       Recall          F1
0             0.200      0.173      0.185
1             0.174      0.336      0.229
2             0.850      0.593      0.698
3             0.159      0.329      0.214
4             0.133      0.275      0.179
Fold 4
        Precision       Recall          F1
0             0.210      0.178      0.193
1             0.187      0.346      0.243
2             0.844      0.600      0.701
3             0.168      0.344      0.226
4             0.122      0.234      0.161
```

```
Average Accuracy :  0.10320512820512821

Average Precision       Recall          F1      Per Label
0             0.203      0.167      0.183
1             0.176      0.337      0.231
2             0.845      0.597      0.700
3             0.165      0.337      0.221
4             0.124      0.255      0.167

Macro Average Precision Recall          F1      Over All Labels
              0.303      0.339      0.300

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
              0.513      0.457      0.462
```

For Bigram filtered tokens

```
print("\n Bigram filtered : ")
crossval.cross_validation_PRF(5,filtered_bigram_features,labels)
```

```
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.051      0.346       0.089
1             0.089      0.328       0.140
2             0.927      0.559       0.697
3             0.158      0.380       0.224
4             0.065      0.359       0.111
Fold 1
        Precision       Recall          F1
0             0.050      0.320       0.086
1             0.090      0.345       0.142
2             0.931      0.558       0.698
3             0.168      0.417       0.240
4             0.067      0.320       0.111
Fold 2
        Precision       Recall          F1
0             0.048      0.322       0.083
1             0.090      0.353       0.143
2             0.928      0.557       0.696
3             0.163      0.397       0.231
4             0.066      0.340       0.110
Fold 3
        Precision       Recall          F1
0             0.055      0.330       0.095
1             0.082      0.349       0.132
2             0.929      0.555       0.695
3             0.154      0.388       0.220
4             0.062      0.302       0.104
Fold 4
        Precision       Recall          F1
0             0.053      0.294       0.090
1             0.083      0.340       0.133
2             0.928      0.554       0.693
3             0.157      0.380       0.223
4             0.071      0.405       0.121
```

```
Average Accuracy :  0.10510897435897434

Average Precision       Recall          F1      Per Label
0             0.051      0.322       0.089
1             0.087      0.343       0.138
2             0.928      0.556       0.696
3             0.160      0.393       0.227
4             0.066      0.345       0.111

Macro Average Precision Recall          F1      Over All Labels
              0.259      0.392       0.252

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
              0.529      0.461       0.438
```

5.3 POS :

For unfiltered :

```
print("\n Pos Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_pos_features,labels)
```

```
 Pos Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.274      0.141      0.186
1             0.189      0.325      0.239
2             0.809      0.613      0.698
3             0.153      0.321      0.207
4             0.151      0.245      0.187
Fold 1
        Precision       Recall          F1
0             0.284      0.139      0.187
1             0.184      0.328      0.236
2             0.815      0.616      0.702
3             0.164      0.339      0.221
4             0.136      0.231      0.172
Fold 2
        Precision       Recall          F1
0             0.252      0.134      0.175
1             0.186      0.326      0.237
2             0.816      0.623      0.706
3             0.155      0.327      0.210
4             0.150      0.205      0.173
Fold 3
        Precision       Recall          F1
0             0.272      0.145      0.189
1             0.190      0.334      0.242
2             0.818      0.612      0.700
3             0.153      0.329      0.209
4             0.161      0.246      0.195
Fold 4
        Precision       Recall          F1
0             0.288      0.154      0.200
1             0.198      0.334      0.249
2             0.809      0.619      0.702
3             0.164      0.334      0.220
4             0.152      0.215      0.178
```

```
Average Accuracy :  0.10088461538461539

Average Precision       Recall          F1      Per Label
0             0.274      0.143      0.188
1             0.190      0.329      0.241
2             0.813      0.617      0.701
3             0.158      0.330      0.213
4             0.150      0.228      0.181

Macro Average Precision Recall          F1      Over All Labels
              0.317      0.329      0.305

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
              0.502      0.462      0.464
```

For filtered tokens :

```
print("\n Pos Unfiltered : ")
crossval.cross_validation_PRF(5,filtered_pos_features,labels)
```

```
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.135      0.204      0.163
1             0.167      0.307      0.216
2             0.851      0.605      0.707
3             0.199      0.340      0.251
4             0.138      0.248      0.178
Fold 1
        Precision       Recall          F1
0             0.141      0.195      0.164
1             0.164      0.326      0.218
2             0.856      0.604      0.708
3             0.214      0.366      0.270
4             0.133      0.224      0.167
Fold 2
        Precision       Recall          F1
0             0.144      0.215      0.172
1             0.176      0.322      0.227
2             0.846      0.600      0.702
3             0.198      0.353      0.254
4             0.145      0.236      0.180
Fold 3
        Precision       Recall          F1
0             0.139      0.210      0.167
1             0.160      0.315      0.212
2             0.847      0.597      0.700
3             0.197      0.341      0.250
4             0.144      0.248      0.182
Fold 4
        Precision       Recall          F1
0             0.131      0.190      0.155
1             0.159      0.309      0.210
2             0.849      0.596      0.701
3             0.198      0.345      0.252
4             0.143      0.252      0.183
```

```
Average Accuracy :  0.10308333333333333

Average Precision       Recall          F1      Per Label
0             0.138      0.203      0.164
1             0.165      0.316      0.217
2             0.850      0.601      0.704
3             0.201      0.349      0.255
4             0.141      0.242      0.178

Macro Average Precision Recall          F1      Over All Labels
              0.299      0.342      0.304

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
              0.519      0.459      0.469
```

5.4 NOT

NOT Unfiltered Tokens:

```
print("\n NOT unfiltered : ")
crossval.cross_validation_PRF(5, unfiltered_not_features,labels)
```

```
 NOT filtered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.595       0.228       0.329
1             0.426       0.445       0.436
2             0.642       0.767       0.699
3             0.441       0.489       0.464
4             0.562       0.333       0.418
Fold 1
        Precision       Recall          F1
0             0.597       0.244       0.346
1             0.424       0.447       0.436
2             0.649       0.758       0.699
3             0.427       0.496       0.459
4             0.572       0.330       0.419
Fold 2
        Precision       Recall          F1
0             0.601       0.247       0.350
1             0.428       0.445       0.436
2             0.646       0.771       0.703
3             0.444       0.492       0.467
4             0.555       0.316       0.402
Fold 3
        Precision       Recall          F1
0             0.611       0.232       0.336
1             0.423       0.453       0.438
2             0.629       0.752       0.685
3             0.432       0.483       0.456
4             0.582       0.324       0.417
Fold 4
        Precision       Recall          F1
0             0.594       0.237       0.339
1             0.418       0.442       0.430
2             0.636       0.752       0.689
3             0.445       0.488       0.465
4             0.561       0.348       0.429

Average Accuracy :  0.11024358974358975
```

```
Average Accuracy :  0.11024358974358975

Average Precision       Recall          F1      Per Label
0             0.600      0.237      0.340
1             0.424      0.447      0.435
2             0.641      0.760      0.695
3             0.438      0.490      0.462
4             0.566      0.330      0.417

Macro Average Precision Recall          F1      Over All Labels
              0.534      0.453      0.470

Label Counts {0: 7069, 1: 27263, 2: 79554, 3: 32914, 4: 9200}
Micro Average Precision Recall          F1      Over All Labels
              0.554      0.599      0.568
```

NOT Filtered tokens:

```
print("\n NOT filtered : ")
crossval.cross_validation_PRF(5, filtered_not_features,labels)
```

```
 NOT filtered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.595      0.228       0.329
1             0.426      0.445       0.436
2             0.642      0.767       0.699
3             0.441      0.489       0.464
4             0.562      0.333       0.418
Fold 1
        Precision       Recall          F1
0             0.597      0.244       0.346
1             0.424      0.447       0.436
2             0.649      0.758       0.699
3             0.427      0.496       0.459
4             0.572      0.330       0.419
Fold 2
        Precision       Recall          F1
0             0.601      0.247       0.350
1             0.428      0.445       0.436
2             0.646      0.771       0.703
3             0.444      0.492       0.467
4             0.555      0.316       0.402
Fold 3
        Precision       Recall          F1
0             0.611      0.232       0.336
1             0.423      0.453       0.438
2             0.629      0.752       0.685
3             0.432      0.483       0.456
4             0.582      0.324       0.417
Fold 4
        Precision       Recall          F1
0             0.594      0.237       0.339
1             0.418      0.442       0.430
2             0.636      0.752       0.689
3             0.445      0.488       0.465
4             0.561      0.348       0.429

Average Accuracy :  0.11024358974358975
```

```
Average Accuracy :  0.11024358974358975

Average Precision       Recall          F1      Per Label
0             0.600      0.237       0.340
1             0.424      0.447       0.435
2             0.641      0.760       0.695
3             0.438      0.490       0.462
4             0.566      0.330       0.417

Macro Average Precision Recall          F1      Over All Labels
              0.534      0.453       0.470

Label Counts {0: 7069, 1: 27263, 2: 79554, 3: 32914, 4: 9200}
Micro Average Precision Recall          F1      Over All Labels
              0.554      0.599       0.568
```

5.5 SL:

For unfiltered:

```
print("\n SL Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_sl_features,labels)
```

```
 SL Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0           0.249        0.192       0.217
1           0.210        0.368       0.267
2           0.822        0.627       0.712
3           0.247        0.398       0.305
4           0.217        0.307       0.254
Fold 1
        Precision       Recall          F1
0           0.260        0.190       0.220
1           0.196        0.364       0.255
2           0.820        0.625       0.710
3           0.254        0.393       0.309
4           0.213        0.319       0.255
Fold 2
        Precision       Recall          F1
0           0.233        0.177       0.202
1           0.193        0.355       0.250
2           0.824        0.632       0.715
3           0.252        0.399       0.309
4           0.213        0.285       0.244
Fold 3
        Precision       Recall          F1
0           0.257        0.209       0.230
1           0.197        0.371       0.258
2           0.829        0.622       0.711
3           0.246        0.397       0.304
4           0.226        0.300       0.258
Fold 4
        Precision       Recall          F1
0           0.249        0.197       0.220
1           0.212        0.373       0.271
2           0.821        0.629       0.712
3           0.258        0.402       0.314
4           0.215        0.297       0.250
```

```
Average Accuracy :  0.10691666666666666

Average Precision       Recall          F1      Per Label
0              0.250      0.193      0.218
1              0.202      0.366      0.260
2              0.823      0.627      0.712
3              0.251      0.398      0.308
4              0.217      0.301      0.252

Macro Average Precision Recall          F1      Over All Labels
               0.349      0.377      0.350

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
               0.532      0.494      0.498
```

For filtered tokens :

```
print("\n SL Unfiltered : ")
crossval.cross_validation_PRF(5,filtered_sl_features,labels)
```

```
 SL Unfiltered :
Each fold size: 31200
Fold 0
         Precision        Recall          F1
0              0.106       0.321       0.159
1              0.148       0.344       0.207
2              0.849       0.612       0.711
3              0.322       0.386       0.351
4              0.149       0.349       0.209
Fold 1
         Precision        Recall          F1
0              0.103       0.305       0.154
1              0.144       0.354       0.205
2              0.858       0.616       0.717
3              0.339       0.403       0.368
4              0.140       0.317       0.194
Fold 2
         Precision        Recall          F1
0              0.100       0.305       0.151
1              0.157       0.357       0.218
2              0.844       0.610       0.708
3              0.325       0.394       0.356
4              0.147       0.326       0.202
Fold 3
         Precision        Recall          F1
0              0.108       0.312       0.161
1              0.148       0.353       0.209
2              0.851       0.608       0.709
3              0.320       0.395       0.354
4              0.143       0.326       0.199
Fold 4
         Precision        Recall          F1
0              0.123       0.329       0.179
1              0.151       0.356       0.212
2              0.852       0.609       0.710
3              0.329       0.402       0.362
4              0.164       0.383       0.230
```

```
Average Accuracy :  0.10878205128205128

Average Precision        Recall          F1      Per Label
0              0.108       0.314       0.161
1              0.150       0.353       0.210
2              0.851       0.611       0.711
3              0.327       0.396       0.358
4              0.149       0.340       0.207

Macro Average Precision Recall          F1      Over All Labels
               0.317       0.403       0.329

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
               0.543       0.491       0.494
```

5.6 LIWC:

For unfiltered :

```
print("\n LIWC Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_liwc_features,labels)
```

```
 LIWC Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.239      0.193      0.213
1             0.199      0.360      0.256
2             0.830      0.620      0.710
3             0.239      0.398      0.298
4             0.193      0.300      0.235
Fold 1
        Precision       Recall          F1
0             0.249      0.190      0.215
1             0.188      0.349      0.244
2             0.830      0.619      0.709
3             0.238      0.396      0.297
4             0.185      0.299      0.229
Fold 2
        Precision       Recall          F1
0             0.219      0.175      0.194
1             0.189      0.350      0.245
2             0.832      0.625      0.714
3             0.230      0.388      0.289
4             0.189      0.267      0.221
Fold 3
        Precision       Recall          F1
0             0.233      0.195      0.212
1             0.187      0.355      0.245
2             0.836      0.613      0.707
3             0.228      0.392      0.288
4             0.195      0.289      0.233
Fold 4
        Precision       Recall          F1
0             0.241      0.199      0.218
1             0.199      0.360      0.257
2             0.832      0.623      0.712
3             0.237      0.395      0.296
4             0.191      0.279      0.227
```

```
Average Accuracy :  0.10632692307692308

Average Precision       Recall          F1      Per Label
0             0.236      0.190      0.211
1             0.192      0.355      0.250
2             0.832      0.620      0.710
3             0.234      0.394      0.294
4             0.191      0.287      0.229

Macro Average Precision Recall          F1      Over All Labels
              0.337      0.369      0.339

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
              0.529      0.487      0.491
```

For filtered tokens :

```
print("\n LIWC Unfiltered : ")
crossval.cross_validation_PRF(5,filtered_liwc_features,labels)
```

```
 LIWC Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0             0.097      0.347       0.151
1             0.117      0.361       0.177
2             0.878      0.595       0.709
3             0.283      0.384       0.326
4             0.124      0.336       0.181
Fold 1
        Precision       Recall          F1
0             0.098      0.328       0.151
1             0.105      0.373       0.164
2             0.888      0.594       0.712
3             0.288      0.399       0.334
4             0.127      0.311       0.180
Fold 2
        Precision       Recall          F1
0             0.093      0.311       0.143
1             0.110      0.357       0.169
2             0.872      0.590       0.704
3             0.284      0.389       0.328
4             0.133      0.344       0.192
Fold 3
        Precision       Recall          F1
0             0.105      0.336       0.161
1             0.104      0.368       0.162
2             0.882      0.588       0.706
3             0.267      0.383       0.314
4             0.129      0.315       0.183
Fold 4
        Precision       Recall          F1
0             0.116      0.319       0.171
1             0.109      0.371       0.168
2             0.879      0.590       0.706
3             0.281      0.391       0.327
4             0.142      0.371       0.206
```

```
Average Accuracy :  0.10776282051282052

Average Precision       Recall          F1      Per Label
0              0.102      0.328      0.155
1              0.109      0.366      0.168
2              0.880      0.591      0.707
3              0.281      0.389      0.326
4              0.131      0.336      0.188

Macro Average Precision Recall          F1      Over All Labels
               0.300      0.402      0.309

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
               0.539      0.482      0.477
```

5.7 SL+LIWC:

For unfiltered:

```
print("\n Combined SL LIWC Unfiltered : ")
crossval.cross_validation_PRF(5,unfiltered_combo_features,labels)
```

```
 Combined SL LIWC Unfiltered :
Each fold size: 31200
Fold 0
        Precision       Recall          F1
0           0.253       0.191       0.218
1           0.210       0.367       0.267
2           0.818       0.629       0.711
3           0.254       0.394       0.309
4           0.214       0.309       0.253
Fold 1
        Precision       Recall          F1
0           0.263       0.186       0.218
1           0.197       0.364       0.256
2           0.818       0.626       0.710
3           0.254       0.388       0.307
4           0.200       0.308       0.242
Fold 2
        Precision       Recall          F1
0           0.241       0.183       0.208
1           0.201       0.364       0.259
2           0.821       0.634       0.715
3           0.256       0.397       0.312
4           0.210       0.289       0.243
Fold 3
        Precision       Recall          F1
0           0.258       0.205       0.228
1           0.195       0.366       0.254
2           0.826       0.623       0.710
3           0.247       0.390       0.303
4           0.214       0.291       0.247
Fold 4
        Precision       Recall          F1
0           0.253       0.197       0.222
1           0.214       0.371       0.271
2           0.815       0.630       0.711
3           0.262       0.394       0.314
4           0.201       0.287       0.236
```

```
Average Accuracy :  0.10642307692307693

Average Precision       Recall          F1      Per Label
0             0.254       0.192       0.219
1             0.203       0.367       0.261
2             0.820       0.628       0.711
3             0.255       0.393       0.309
4             0.208       0.297       0.244

Macro Average Precision Recall          F1      Over All Labels
              0.348       0.375       0.349

Label Counts {0: 7067, 1: 27264, 2: 79551, 3: 32914, 4: 9204}
Micro Average Precision Recall          F1      Over All Labels
              0.531       0.494       0.498
```

For filtered tokens:

```
print("\n Unigram Unfiltered : ")
crossval.cross_validation_PRF(5,filtered_combo_features,labels)
```

```
 Unigram Unfiltered :
Each fold size: 31200
Fold 0
          Precision       Recall          F1
0               0.102      0.317       0.154
1               0.162      0.354       0.222
2               0.837      0.623       0.714
3               0.359      0.387       0.373
4               0.144      0.351       0.204
Fold 1
          Precision       Recall          F1
0               0.101      0.304       0.151
1               0.158      0.350       0.218
2               0.845      0.625       0.718
3               0.369      0.405       0.386
4               0.138      0.317       0.193
Fold 2
          Precision       Recall          F1
0               0.100      0.303       0.151
1               0.168      0.348       0.226
2               0.830      0.616       0.707
3               0.345      0.391       0.366
4               0.143      0.321       0.198
Fold 3
          Precision       Recall          F1
0               0.113      0.326       0.167
1               0.164      0.361       0.226
2               0.838      0.614       0.709
3               0.335      0.388       0.360
4               0.147      0.333       0.204
Fold 4
          Precision       Recall          F1
0               0.124      0.316       0.178
1               0.161      0.354       0.221
2               0.839      0.615       0.710
3               0.347      0.396       0.370
4               0.157      0.382       0.222
```

```
Average Accuracy :  0.10851923076923078

Average Precision       Recall          F1      Per Label
0              0.108      0.313      0.160
1              0.162      0.353      0.223
2              0.838      0.618      0.712
3              0.351      0.393      0.371
4              0.146      0.341      0.204

Macro Average Precision Recall          F1      Over All Labels
               0.321      0.404      0.334

Label Counts {0: 7069, 1: 27263, 2: 79557, 3: 32912, 4: 9199}
Micro Average Precision Recall          F1      Over All Labels
               0.543      0.495      0.499
```

6 Running Naïve Bayes Classifier on Feature sets:

For each of the feature sets generated. Naïve Bayes classifier was run without cross validation.

The highest accuracy for unfiltered tokens was obtained by both SL and SL +LIWC feature sets

The highest accuracy for filtered tokens was obtained by Not feature set

Feature set	Unigram	Bigram	POS	NOT	SL	LIWC	SL +LIWC
Filtered	0.531	0.526	0.520	0.538	0.546	0.540	0.546
Unfiltered	0.513	0.515	0.501	0.560	0.531	0.530	0.531

```
def naivebayesaccuracy(features):
  train_set,test_set = features[int(0.1*len(features)):],
features[:int(0.1*len(features))]
  classifier = nltk.NaiveBayesClassifier.train(train_set)
  print("\nAccuracy : ")
  print(nltk.classify.accuracy(classifier,test_set),"\n")
  l1 = []
  tl=[]
  for (features,label) in test_set:
    l1.append(label)
    tl.append(classifier.classify(features))
  print(ConfusionMatrix(l1,tl))
```

6.1 Unigram

```
print("\n Unigram filtered : ")
naivebayesaccuracy(filtered_unigram_features)
print("\n Unigram Unfiltered : ")
naivebayesaccuracy(unfiltered_unigram_features)
```

```
Unigram filtered :

Accuracy :
0.5312179487179487

  |    0    1    2    3    4 |
--+--------------------------+
0 |  <27> 107  496   55    7 |
1 |   31 <228>2245  185   24 |
2 |   26  184<7434> 327   24 |
3 |    8  134 2550 <532>  64 |
4 |    4   41  591  210  <66>|
--+--------------------------+
(row = reference; col = test)
```

```
 Unigram Unfiltered :

Accuracy :
0.5135897435897436

  |    0    1    2    3    4 |
--+--------------------------+
0 | <142> 126  278   94   29 |
1 |  228 <506>1634  309   65 |
2 |  165  518<6725> 464   68 |
3 |  206  275 2119 <518> 165 |
4 |   82   56  502  205 <121>|
--+--------------------------+
(row = reference; col = test)
```

6.2 Bigram

```
print("\n Bigram filtered : ")
naivebayesaccuracy(filtered_bigram_features)
print("\n Bigram unfiltered : ")
naivebayesaccuracy(unfiltered_bigram_features)
```

```
 Bigram filtered :

Accuracy :
0.5268589743589743

  |    0    1    2    3    4 |
--+--------------------------+
0 |  <35>   94  528   76   13 |
1 |   43 <224>2225  197   28 |
2 |   18  217<7384> 342   21 |
3 |   18  136 2541 <518>  53 |
4 |    3   39  575  214 <58>|
--+--------------------------+
(row = reference; col = test)
```

```
 Bigram Unfiltered :

Accuracy :
0.5154487179487179

  |    0    1    2    3    4 |
--+--------------------------+
0 | <134> 143  338   98   33 |
1 |  206 <525>1618  296   72 |
2 |  188  511<6726> 484   73 |
3 |  214  252 2107 <543> 150 |
4 |   80   53  466  177 <113>|
--+--------------------------+
(row = reference; col = test)
```

6.3 POS:

```
print("\n Pos filtered : ")
naivebayesaccuracy(filtered_pos_features)
print("\n Pos Unfiltered : ")
naivebayesaccuracy(unfiltered_pos_features)
```

```
 Pos filtered :

Accuracy :
0.5201282051282051

  |    0    1    2    3    4 |
--+--------------------------+
0 |  <85> 159  326   85   37 |
1 |  162 <456>1729  282   84 |
2 |  115  451<6777> 547  105 |
3 |  102  300 2026 <665> 195 |
4 |   42   79  412  248 <131>|
--+--------------------------+
(row = reference; col = test)
```

```
 Pos Unfiltered :

Accuracy :
0.5017948717948718

  |    0    1    2    3    4 |
--+--------------------------+
0 | <192> 113  238   92   34 |
1 |  394 <521>1430  312   85 |
2 |  309  549<6464> 493  125 |
3 |  348  295 1921 <500> 219 |
4 |  123   70  427  195 <151>|
--+--------------------------+
(row = reference; col = test)
```

1.3 NOT

```
print("\n NOT filtered : ")
naivebayesaccuracy(filtered_not_features)
print("\n NOT unfiltered : ")
naivebayesaccuracy(unfiltered_not_features)
```

```
 NOT filtered :

Accuracy :
0.5382692307692307

  |    0    1    2    3    4 |
--+-------------------------+
0 | <387> 227   18   12    9 |
1 |  621<1383> 487  205   68 |
2 |  416 1435<4396>1458  309 |
3 |  104  248  530<1701> 674 |
4 |   13   18   32  319 <530>|
--+-------------------------+
(row = reference; col = test)
```

```
 NOT Unfiltered :

Accuracy :
0.5601282051282052

  |    0    1    2    3    4 |
--+-------------------------+
0 | <364> 223   50    8    8 |
1 |  679<1183> 702  137   63 |
2 |  444 1008<5223>1072  267 |
3 |  181  178  775<1446> 677 |
4 |   20   13   61  296 <522>|
--+-------------------------+
(row = reference; col = test)
```

6.4 SL:

```
print("\n SL filtered : ")
naivebayesaccuracy(filtered_sl_features)
print("\n SL Unfiltered : ")
naivebayesaccuracy(unfiltered_sl_features)
```

```
 SL filtered :

Accuracy :
0.5467307692307692

  |    0    1    2    3    4 |
--+-------------------------+
0 |  <77> 146  363   90   16 |
1 |   85 <401>1801  376   50 |
2 |   52  336<6798> 744   65 |
3 |   32  174 1819<1111> 152 |
4 |   11   34  302  423 <142>|
--+-------------------------+
(row = reference; col = test)
```

```
 SL Unfiltered :

Accuracy :
0.5316025641025641

  |    0    1    2    3    4 |
--+-------------------------+
0 | <165> 158  252   67   27 |
1 |  284 <537>1499  339   83 |
2 |  195  500<6537> 604  104 |
3 |  189  231 1747 <829> 287 |
4 |   56   42  338  305 <225>|
--+-------------------------+
(row = reference; col = test)
```

1.4 LIWC:

```
print("\n LIWC filtered : ")
naivebayesaccuracy(filtered_liwc_features)
print("\n LIWC Unfiltered : ")
naivebayesaccuracy(unfiltered_liwc_features)
```

```
 LIWC filtered :

Accuracy :
0.5407692307692308

  |    0    1    2    3    4 |
--+-------------------------+
0 |  <65> 126  403   86   12 |
1 |   82 <294>1917  375   45 |
2 |   39  250<7018> 636   52 |
3 |   34  128 2067 <938> 121 |
4 |    7   31  394  359 <121>|
--+-------------------------+
(row = reference; col = test)
```

```
 LIWC Unfiltered :

Accuracy :
0.53

  |    0    1    2    3    4 |
--+-------------------------+
0 | <161> 160  248   74   26 |
1 |  264 <518>1547  327   86 |
2 |  182  495<6604> 562   97 |
3 |  180  253 1810 <789> 251 |
4 |   56   38  395  281 <196>|
--+-------------------------+
(row = reference; col = test)
```

6.5 SL+LIWC:

```
print("\n Combined SL LIWC filtered : ")
naivebayesaccuracy(filtered_combo_features)
print("\n Combined SL LIWC Unfiltered : ")
naivebayesaccuracy(unfiltered_combo_features)
```

```
 Combined SL LIWC filtered :

Accuracy :
0.546025641025641

  |    0    1    2    3    4 |
--+-------------------------+
0 |  <76> 162  347   97   10 |
1 |   76 <454>1729  408   46 |
2 |   49  382<6700> 808   56 |
3 |   31  204 1748<1155> 150 |
4 |    6   39  285  449 <133>|
--+-------------------------+
(row = reference; col = test)
```

```
 Combined SL LIWC Unfiltered :

Accuracy :
0.5312179487179487

  |    0    1    2    3    4 |
--+-------------------------+
0 | <172> 156  249   67   25 |
1 |  284 <558>1480  334   86 |
2 |  204  492<6500> 645   99 |
3 |  196  241 1738 <843> 265 |
4 |   53   39  345  315 <214>|
--+-------------------------+
(row = reference; col = test)
```

7 Running Random Forest Classifier on all feature sets :

One of the advanced experiments that was chosen was to run the feature sets by saving them into csv files and using these features on a random forest classifier to compare it with the naïve bayes classifier. Taking the internal 90:10 train-test set split, we trained out random forest classifier to calculate the accuracy. For not or negation features, the classifier did not work and gave us memory error. This classifier surprisingly worked very well for unfiltered feature sets as compared to Naïve Bayes.The accuracies were high for filtered data compared to naïve bayes results

Feature set	Unigram	Bigram	POS	SL	LIWC	SL +LIWC
Filtered	0.547	0.546	0.565	0.582	0.567	0.579
Unfiltered	0.567	0.571	0.580	0.605	0.591	0.595

```
def rf(featuresets):
  n = 0.1
  cutoff = int(n*len(featuresets))
  train_set, test_set = featuresets[cutoff:], featuresets[:cutoff]
  classifier_rf = SklearnClassifier(RandomForestClassifier())
  classifier_rf.train(train_set)
  print("Classifier-RandomForest \n")
  print("Accuracy : ",nltk.classify.accuracy(classifier_rf, test_set))
```

For filtered feature sets :

```
Unigram filtered :
Classifier-RandomForest

Accuracy :  0.5472435897435898

 Bigram filtered :
Classifier-RandomForest

Accuracy :  0.5467307692307692

 Pos filtered :
Classifier-RandomForest

Accuracy :  0.5655128205128205

 SL filtered :
Classifier-RandomForest

Accuracy :  0.5820512820512821

 LIWC filtered :
Classifier-RandomForest

Accuracy :  0.5673717948717949

 Combined SL LIWC filtered :
Classifier-RandomForest

Accuracy :  0.5796794871794871
```

For unfiltered feature sets :

```
Unigram Unfiltered :
Classifier-RandomForest

Accuracy :  0.5679487179487179

 Bigram Unfiltered :
Classifier-RandomForest

Accuracy :  0.5712820512820512

 Pos Unfiltered :
Classifier-RandomForest

Accuracy :  0.5805769230769231

SL Unfiltered :
Classifier-RandomForest

Accuracy :  0.6053846153846154

 LIWC Unfiltered :
Classifier-RandomForest

Accuracy :  0.5918589743589744


 Combined SL LIWC unfiltered :
Classifier-RandomForest

Accuracy :  0.5955151541545454
```

Summary/Comparisons:

LIWC feature sets which were not implemented in the course , were implemented and tested out on the reviews dataset.

Combined LIWC and SL features were generated

Random Forest Classifier and Naïve Bayes Classifiers were run, and accuracies were calculated. Random Forest classifier performed better in all cases, but for unfiltered data the accuracy scores were much higher than the filtered featuresets.

Challenges

Although a lot was learnt through trial and error, we faced issues while running the programs as these classifiers are computationally intensive. It was time consuming to run.