Final Project Report

Processing and Classification of Sentiment or other Data

Dataset-Kaggle Movie Review

The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site.

We are going to use training data- “train.tsv”, and test data- “test.tsv”.

train.tsv contains the phrases and their associated sentiment labels. The sentiment labels are:

0 - negative
1 - slightly negative
2 - neutral
3 - slightly positive
4 – positive

test.tsv contains just phrases.

Goal of this project- To predict the sentiments of reviews using basic classification algorithms and compare the results by varying different parameters.

Steps for Classification and Sentiments Analysis-

1.) Fetch data from train.tsv

```
# function to read kaggle training file, train and test a classifier
def processkaggle(dirPath,limitStr):
  # convert the limit argument from a string to an int
  limit = int(limitStr)
  os.chdir(dirPath)
  f = open('./train.tsv', 'r')
  # loop over lines in the file and use the first limit of them
  phrasedata = []
  for line in f:
    # ignore the first line starting with Phrase and read all lines
    if (not line.startswith('Phrase')):
      # remove final end of line character
      line = line.strip()
      # each line has 4 items separated by tabs
      # ignore the phrase and sentence ids, and keep the phrase and sentiment
      phrasedata.append(line.split('\t')[2:4])
```

2.) Randomize data and select a certain no. of phrases from phrase data.

```
  # 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)
```

Note- Commandline interface takes a directory name with kaggle subdirectory for train.tsv and a limit to the number of kaggle phrases to use

```
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])
```

3.) Tokenization-

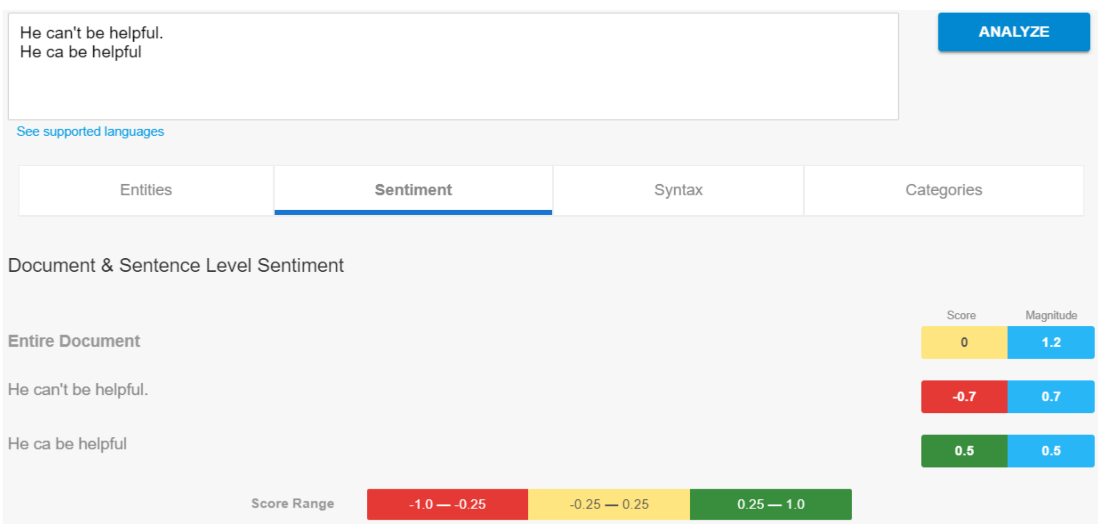
I tried three different types of tokenizers-

• Countvectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
def countvector_token(phraselist):
  phrasedocs4 = []
  for phrase in phraselist:
    cvtokens = CountVectorizer().build_tokenizer()(phrase[0])
    phrasedocs4.append((cvtokens, int(phrase[1])))
  return phrasedocs4
```

This divided sentences into words similar to wordpunct tokenizer and removed all single character words like ‘ca n’t’ resulted into ‘ca’ and ‘U.S.A.’ was completed removed.

Observation-Change of negative words like can’t to ca won’t be useful in sentiment analysis. For example (Based on Google NL API)-



• word_punct tokenizer

```
def wordpunct_token(phraselist):
  phrasedocs3 = []
  for phrase in phraselist:
    wptokens = nltk.wordpunct_tokenize(phrase[0])
```

```
    phrasedocs3.append((wptokens, int(phrase[1])))
  return phrasedocs3
```

This divided word like ‘U.S.A.’ into six words- ‘U’,’.’,’S’,’.’,’A’,’.’ and ‘ca n’t’ into four words- ‘ca’,’n’, ’’’ ,’t’.

• word tokenizer

```
def word_token(phraselist):
  phrasedocs = []
  for phrase in phraselist:
    tokens = nltk.word_tokenize(phrase[0])
    phrasedocs.append((tokens, int(phrase[1])))
  return phrasedocs
```

This approach preserved unique words like U.S.A but negative words like ‘woud n’t’ are divided into two parts ‘would’ and ‘n’t’. It would be interesting to see how various classification algorithms react to this

Clearly if we want to do classification without pre-processing then word tokenizer would be the most useful compared to other two.

4.) **Pre-processing**

To remove unnecessary words like non-alphanumeric words and stop list, I did some pre-processing on words generated by above tokenizers. I have divided tokenization into two categories-

a. Word tokenizer without pre-processing (as discussed earlier)
b. Word tokenizer with pre-processing

In first one, I have not applied pre-processing on sentences and then created pairs of (tokensOf(sentence), label) list for classification. In second one I have used some pre-processing steps before classification.

Lower case:

I converted all tokens into lower case as many functions of nltk are case-sensitive

```
def lower_case(doc):
  return [w.lower( ) for w in doc]
```

Clean text:

Removing punctuation, stop words and single character would again result in change of negative words like can’t to can. As we have seen in count vector case, this won’t be useful in our sentiment analysis. In order to avoid such scenario, we will need to expand some stop words with apostrophe.

```
def clean_text(doc):
  cleantext = []
  for review_text in doc:
    review_text = re.sub(r"it 's", "it is", review_text)
    review_text = re.sub(r"that 's", "that is", review_text)
```

```
    review_text = re.sub(r"\'s", "\'s", review_text)
    review_text = re.sub(r"\'ve", "have", review_text)
    review_text = re.sub(r"wo n't", "will not", review_text)
    review_text = re.sub(r"do n't", "do not", review_text)
    review_text = re.sub(r"ca n't", "can not", review_text)
    review_text = re.sub(r"sha n't", "shall not", review_text)
    review_text = re.sub(r"n\'t", "not", review_text)
    review_text = re.sub(r"\'re", "are", review_text)
    review_text = re.sub(r"\'d", "would", review_text)
    review_text = re.sub(r"\'ll", "will", review_text)
    cleantext.append(review_text)
  return cleantext
```

Removing punctuation and numbers-

As punctuation and numbers will be unnecessary for sentiment analysis

```
def rem_no_punct(doc):
  remtext = []
  for text in doc:
    punctuation = re.compile(r'[-_.?!/\%@,":;\'{}<>~`\()|0-9]')
    word = punctuation.sub("", text)
    remtext.append(word)
  return remtext
```

Removing some stop words-

I am not going to remove negative words like not, cannot, would not as they will be useful in our sentiment analysis.

```
from nltk.corpus import stopwords
def rem_stopword(doc):
  stopwords = nltk.corpus.stopwords.words('english')
  updatestopwords = [word for word in stopwords if word not in ['not', 'no', 
'can','has','have','had','must','shan','do', 'should','was','were','won','are','cannot','does','ain', 'could', 
'did', 'is', 'might', 'need', 'would']]
  return [w for w in doc if not w in updatestopwords]
```

Stemming and Lemmatization-

In assignment 1, I had tried three stemmers-Lancaster, Porter and Snowball stemmer. I had also examined document using WordNet lemmatizer. I found that Lancaster stemmer was severe on some words like event and ever resulted into ev whereas Snowball stemmer hardly changed any word compared to other two stemmers. The WordNet lemmatizer only removes affixes if the resulting word is in its dictionary like lying remains same instead of changing to lie. So, I decided to use combination of WordNet lemmatization and Porter stemming.

```
def lemmatizer(doc):
  wnl = nltk.WordNetLemmatizer()
  lemma = [wnl.lemmatize(t) for t in doc]
  return lemma
```

```
def stemmer(doc):
  porter = nltk.PorterStemmer()
  stem = [porter.stem(t) for t in doc]
  return stem
```

So, our final word tokenizer with preprocessing will look like this-

```
def process_token(phraselist):
  phrasedocs2 = []
  for phrase in phraselist:
    tokens = nltk.word_tokenize(phrase[0])
    tokens = lower_case(tokens)
    tokens = clean_text(tokens)
    tokens = rem_no_punct(tokens)
    tokens = rem_stopword(tokens)
    tokens = stemmer(tokens)
    tokens = lemmatizer(tokens)
    phrasedocs2.append((tokens, int(phrase[1])))
  return phrasedocs2
```

5.) Filtering-

Removing 1 and 2 characters-

Single characters and double character words that might be generated through above mentioned pre-processing won’t be useful in our classification and sentiment analysis.

```
def rem_character(doc):
  word_list=[]
  for word in doc:
      if (len(word) > 1):
        word_list.append(word)
  return word_list
```

Similarly, for unprocessed tokens we can extract words in following ways:

```
def get_words(doc):
  word_list = []
  for (word, sentiment) in doc:
    word_list.extend(word)
  return word_list
```

```
(C:\ProgramData\Miniconda3) C:\Users\bobby\Downloads\kagglemoviereviews\kagglemoviereviews>python classifyKaggle.py C:\Users\bobby\Downloads\kagglemoviereviews\kagglemoviereviews\corpus 156060
Read 156060 phrases, using 156060 random phrases
All Phrases-
['the Magazine', '2']
["Those who are n't put off by the film 's austerity will find it more than capable of rewarding them .", '4']
['A piece of mildly entertaining , inoffensive fluff that drifts aimlessly for 90 minutes before lodging in the cracks of that ever-growing category : unembarrassing but unmemorable', '3']
['single facet', '2']
['Too much power , not enough puff .', '1']
['lists and ways', '3']
['its dying , in this shower of black-and-white psychedelia ,', '1']
["will probably sink the film for anyone who does n't think about percentages all day long", '1']
['in mystery and a ravishing , baroque beauty', '4']
['creepiest conventions', '1']

Word Tokenized but without pre processing-
(['the', 'Magazine'], 2)
(['Those', 'who', 'are', "n't", 'put', 'off', 'by', 'the', 'film', "'s", 'austerity', 'will', 'find', 'it', 'more', 'than', 'capable', 'of', 'rewarding', 'them', '.'], 4)
(['A', 'piece', 'of', 'mildly', 'entertaining', ',', 'inoffensive', 'fluff', 'that', 'drifts', 'aimlessly', 'for', '90', 'minutes', 'before', 'lodging', 'in', 'the', 'cracks', 'of', 'that', 'ever-growing', 'cate
gory', ':', 'unembarrassing', 'but', 'unmemorable'], 3)
(['single', 'facet'], 2)
(['Too', 'much', 'power', ',', 'not', 'enough', 'puff', '.'], 1)
(['lists', 'and', 'ways'], 3)
(['its', 'dying', ',', 'in', 'this', 'shower', 'of', 'black-and-white', 'psychedelia', ','], 1)
(['will', 'probably', 'sink', 'the', 'film', 'for', 'anyone', 'who', 'does', "n't", 'think', 'about', 'percentages', 'all', 'day', 'long'], 1)
(['in', 'mystery', 'and', 'a', 'ravishing', ',', 'baroque', 'beauty'], 4)
(['creepiest', 'conventions'], 1)

Word Tokenized and pre processed-
(['magazin'], 2)
(['are', 'not', 'put', 'film', 'auster', 'find', 'capabl', 'reward', ''], 4)
(['piec', 'mildli', 'entertain', '', 'inoffens', 'fluff', 'drift', 'aimlessli', '', 'minut', 'lodg', 'crack', 'evergrow', 'categori', '', 'unembarrass', 'unmemor'], 3)
(['singl', 'facet'], 2)
(['much', 'power', '', 'not', 'enough', 'puff', ''], 1)
(['list', 'way'], 3)
(['die', '', 'shower', 'blackandwhit', 'psychedelia', ''], 1)
(['probabl', 'sink', 'film', 'anyon', 'doe', 'not', 'think', 'percentag', 'day', 'long'], 1)
(['mysteri', 'ravish', '', 'baroqu', 'beauti'], 4)
(['creepiest', 'convent'], 1)
```

6.) Writing featuresets to a csv file-

We need to generate csv files of feature set so that they can be later use it with our Weka Classifer. We will use function- writeFeatureSets(featuresets, outpath) defined in save_features.py file.We will import this file using- import save_features. I will update writeFeatureSets function in order to convert integer value from 0-4 to corresponding sentiment labels.

```
def writeFeatureSets(featuresets, outpath):
    # open outpath for writing
    f = open(outpath, 'w')
    # get the feature names from the feature dictionary in the first featureset
    featurenames = featuresets[0][0].keys()
    # create the first line of the file as comma separated feature names
    #    with the word class as the last feature name
    featurenameline = ''
    for featurename in featurenames:
        # replace forbidden characters with text abbreviations
        featurename = featurename.replace(',','CM')
        featurename = featurename.replace("'","DQ")
        featurename = featurename.replace('"','QU')
        featurenameline += featurename + ','
    featurenameline += 'class'
    # write this as the first line in the csv file
    f.write(featurenameline)
    f.write('\n')
    for featureset in featuresets:
```

```
        featureline = ''
        for key in featurenames:
              try:
                featureline += str(featureset[0].get(key,[])) + ','
             except KeyError:
               continue
        if featureset[1] == 0:
          featureline += str("strongly negative")
        elif featureset[1] == 1:
          featureline += str("slightly negative")
        elif featureset[1] == 2:
          featureline += str("neutral")
        elif featureset[1] == 3:
          featureline += str("slightly positive")
        elif featureset[1] == 4:
          featureline += str("strongly positive")
        # write each feature set values to the file
        f.write(featureline)
        f.write('\n')
    f.close()
```

7.) Feature Selection-

Bag of words feature:

```
from nltk import FreqDist
def bag_of_words(corpus ,wordcount):
  wordlist = nltk.FreqDist(corpus)
  word_features = [w for (w, c) in wordlist.most_common(wordcount)]
  return word_features
```

This function collects all the words in the corpus and select some number (depending on wordcount passed as argument) of most frequent words to be the word features. This function will be useful in other features that we are going to define now.

Bag of words for bigram:

```
from nltk.collocations import *
def bag_of_words_biagram(wordlist,bigramcount):
  bigram_measures = nltk.collocations.BigramAssocMeasures()
  finder = BigramCollocationFinder.from_words(wordlist,window_size=3)
  finder.apply_ngram_filter(lambda w1, w2: len(w1) < 2)
  finder.apply_freq_filter(3)
  bigram_features = finder.nbest(bigram_measures.chi_sq, 3000)
  return bigram_features[:bigramcount]
```

This function collects all the words in the corpus and select some number (depending on bigramcount passed as argument) of most frequent bigrams. This function will be useful in other features that we are going to define now. We use the chi-squared measure to get bigrams that are informative features. Freq_filter would remove words that only occurred with a frequency less than 3. Ngram_filter will filter out bigrams in which the first word's length is less than 2

Note-It is better to apply this feature to only un-processed tokens as bigram finder must have the words in order. So, this will not produce enough bigrams (with pre-processed tokens) for more accurate results

```
-------------------------------------------------
Top 10 Unprocessed tokens word features
['the', ',', 'a', 'of', 'and', 'to', '.', "'s", 'in', 'is']
-------------------------------------------------
Top 10 Pre-processed tokens word features
['film', 'not', 'movi', 'are', 'one', 'have', 'like', 'make', 'charact', 'stori']
-------------------------------------------------
Top 10 Unprocessed tokens word features(Bigrams)
[('ANTWONE', 'FISHER'), ('LITTLE', 'EYE'), ('ART', 'FILM'), ('CELL', 'PHONE'), ('Nouvelle', 'Vague'), ('Greatest', 'Musicians'), ('THE', 'GUYS'), ('Unbearable', 'Ligh
)]
-------------------------------------------------
Top 10 Pre-processed tokens word features(Bigrams)
[('ellen', 'pompeo'), ('disloy', 'satyr'), ('pooti', 'tang'), ('dover', 'kosashvili'), ('cherri', 'orchard'), ('toilethumor', 'codswallop'), ('swan', 'dive'), ('hong'
 'callar')]
-------------------------------------------------
```

Unigram feature (Baseline feature for comparison):

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

This function returns a dictionary who's each element is a word (obtained from bag of words function defined earlier) with a Boolean value indicating whether that word occurred in document or not. The feature label will be 'contains(keyword)' for each keyword (aka word) in the bag of words set

For example

Unigramsets_without_preprocessing -

({'contains(the)': False, 'contains(yet)': False, 'contains(after)': False, 'contains(him)': False, 'contains(take)': False, 'contains(tale)': False, 'contains(years)': False, 'contains(music)': False, 'contains(romantic)': False, 'contains(same)': False, 'contains(documentary)': False, 'contains(subject)': False, 'contains(comes)': False, 'contains(year)': False, 'contains(watching)': False, 'contains(making)': False, 'contains(me)': False, 'contains(worth)': False, 'contains(seem)': False, 'contains(give)': False, 'contains(anything)': False, 'contains(special)': False………

Bigram feature:

```
def bigram_features(doc,word_features,bigram_features):
  document_words = set(doc)
  document_bigrams = nltk.bigrams(doc)
  features = {}
  for word in word_features:
    features['contains(%s)' % word] = (word in document_words)
```

```
  for bigram in bigram_features:
    features['bigram(%s %s)' % bigram] = (bigram in document_bigrams)
  return features
```

This function takes the list of words in a document as an argument and returns a feature dictionary. It depends on the variables word_features and bigram_features

For example
Bigramsets_without_preprocessing -
………..'bigram(smallest sensitivities)': False, 'bigram(Bermuda Triangle)': False, 'bigram(Digital stereo)': False, 'bigram(Les Vampires)': False, 'bigram(Movies Ago)': False, 'bigram(Plutonium Circus)': False, 'bigram(craven concealment)': False……….

Negative features:
For this feature, I first created my own negative words dictionary and also added processed version negative words (clean text+ stem+lemma) in this dictionary.

Note- I took care of whitespaces in some negative words just like in original corpus so I added ca n't instead of can't

```
negative_words =
['abysmal','adverse','alarming','angry','annoy','anxious','apathy','appalling','atrocious','awful',
'bad','banal','barbed','belligerent','bemoan','beneath','boring','broken',
'callous','ca n\'t','clumsy','coarse','cold','cold-
hearted','collapse','confused','contradictory','contrary','corrosive','corrupt','crazy','creepy','criminal'
,'cruel','cry','cutting','dead','decaying','damage','damaging','dastardly','deplorable','depressed','depr
ived','deformed''deny','despicable','detrimental','dirty','disease','disgusting','disheveled','dishonest',
'dishonorable','dismal','distress','do n\'t','dreadful','dreary',
'enraged','eroding','evil','fail','faulty','fear','feeble','fight','filthy','foul','frighten','frightful',
'gawky','ghastly','grave','greed','grim','grimace','gross','grotesque','gruesome','guilty',
'haggard','hard','hard-
hearted','harmful','hate','hideous','horrendous','horrible','hostile','hurt','hurtful',
'icky','ignore','ignorant','ill','immature','imperfect','impossible','inane','inelegant','infernal','injure','i
njurious','insane','insidious','insipid',
'jealous','junky','lose','lousy','lumpy','malicious','mean','menacing','messy','misshapen','missing','m
isunderstood','moan','moldy','monstrous',
'naive','nasty','naughty','negate','negative','never','no','nobody','nondescript','nonsense','noxious',
'objectionable','odious','offensive','old','oppressive',
'pain','perturb','pessimistic','petty','plain','poisonous','poor','prejudice','questionable','quirky','quit',
'reject','renege','repellant','reptilian','repulsive','repugnant','revenge','revolting','rocky','rotten','rude
','ruthless',
'sad','savage','scare','scary','scream','severe','shoddy','shocking','sick',
'sickening','sinister','slimy','smelly','sobbing','sorry','spiteful','sticky','stinky','stormy','stressful','stu
ck','stupid','substandard','suspect','suspicious',
'tense','terrible','terrifying','threatening',
'ugly','undermine','unfair','unfavorable','unhappy','unhealthy','unjust','unlucky','unpleasant','upset','
unsatisfactory',
```

'unsightly','untoward','unwanted','unwelcome','unwholesome','unwieldy','unwise','upset','vice','vicious','vile','villainous','vindictive',
'wary','weary','wicked','woeful','worthless','wound','yell','yucky',
'are n\'t','cannot','ca n\'t','could n\'t','did n\'t','does n\'t','do n\'t','had n\'t','has n\'t','have n\'t','is n\'t','must n\'t','sha n\'t','should n\'t','was n\'t','were n\'t','would n\'t',
'no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']

This function will pre-process above mentioned negative words dictionary:

```
def negativewordproc(negativewords):
  nwords = []
  nwords = clean_text(negativewords)
  nwords = lemmatizer(nwords)
  nwords = stemmer(nwords)
  return nwords

  processnwords = negativewordproc(negative_words)
  negative_words = negative_words + processnwords
```

I look for negation words and negate the word following the negation word. I will go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.

```
def negative_features(doc, word_features, negationwords):
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = False
    features['contains(NOT{})'.format(word)] = False
  # go through document words in order
  for i in range(0, len(doc)):
    word = doc[i]
    if ((i + 1) < len(doc)) and (word in negationwords):
      i += 1
      features['contains(NOT{})'.format(doc[i])] = (doc[i] in word_features)
    else:
      if ((i + 3) < len(doc)) and (word.endswith('n') and doc[i+1] == "'" and doc[i+2] == 't'):
        i += 3
        features['contains(NOT{})'.format(doc[i])] = (doc[i] in word_features)
      else:
        features['contains({})'.format(word)] = (word in word_features)
  return features
```

For Example

Negativesets_without_preprocessing -

({'contains(the)': False, 'contains(NOTthe)': False, 'contains(and)': False, 'contains(NOTand)': False, 'contains(,)': False, 'contains(NOT,)': False, 'contains(of)': False, 'contains(NOTof)': False, 'contains(a)': True, 'contains(NOTa)': False, 'contains(to)': False, 'contains(NOTto)': False………………

POS feature:
It runs the default POS tagger (Stanford tagger) on the document and counts 4 types of pos tags to use as features

```
def POS_features(doc, word_features):
    document_words = set(doc)
    tagged_words = nltk.pos_tag(doc)
    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
```

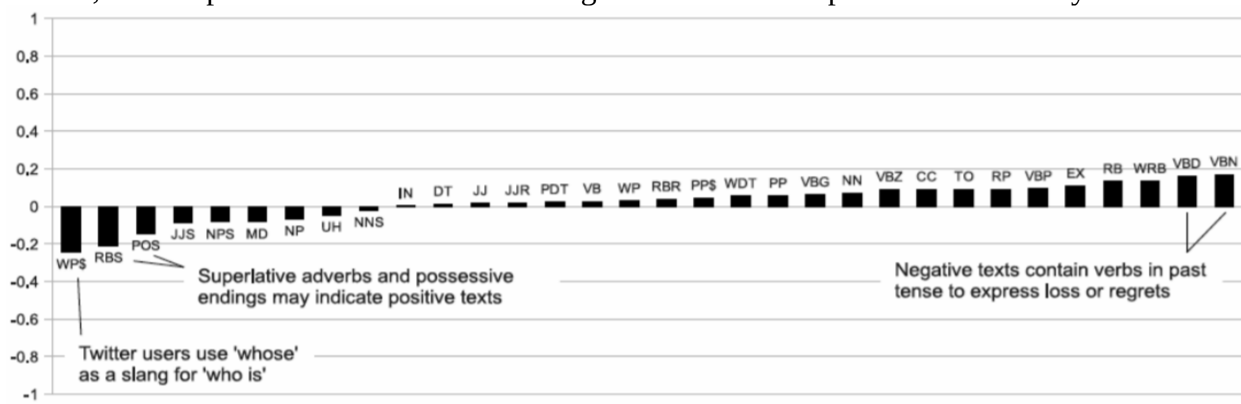
For example
POSsets_without_preprocessing -
…………..'contains(funny)': False, 'contains(comes)': False, 'contains(along)': False, 'contains(occasionally)': False, 'contains(unconventional)': False, 'contains(gutsy)': False, 'contains(perfectly)': False, 'nouns': 1, 'verbs': 0, 'adjectives': 2, 'adverbs': 0}, 1)

Since POS tagger cannot detect normalized form of token we will create a new function for pre-processed form of a sentence that takes un processed form of tokens that will be tagged and then pre-processed.

```
def POS2_features(doc,word_features):
    tagged_words = nltk.pos_tag(doc)
    document_words = set(doc)
    nwords = clean_text(document_words)
    nwords = rem_no_punct(nwords)
    nwords = rem_stopword(nwords)
    nwords = lemmatizer(nwords)
    nwords = stemmer(nwords)
    features = {}
```

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

Note- Based on a study, more past tense verbs mean negative sentiment and more superlative adverb, means positive sentiment so counting POS will also help in sentiment analysis



Sentiment Lexicon(Subjectivity) feature:
In order to use this function, we will define one additional function that reads subjectivity words from the subjectivity lexicon file and returns dictionary, where each word is mapped to a list containing the strength and polarity.

```
def readSubjectivity(path):
    flexicon = open(path, 'r')
    sldict = { }
    for line in flexicon:
        fields = line.split()   #  split on whitespace
        # split each field on the '=' and keep the second part as the value
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
```

```
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        # put a dictionary entry with the word as the keyword
        #     and a list of the other values
        procword = wordproc(word)
        sldict[procword] = [strength, posTag, isStemmed, polarity]
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict

SL = readSubjectivity(SLpath)
```

Note- I have not imported this function from sentiment_read_Subjectivity.py as this function is not same as the one in sentiment_read_Subjectivity.py. I have modified it to include pre-processed version of all words in SL for our pre-processed tokens. In order to pre-process individual words in SL dictionary, I have defined another function. This function takes word and returns stemmed and lemmatized version of it.

```
def wordproc(word):
  wnl = nltk.WordNetLemmatizer()
  porter = nltk.PorterStemmer()
  nwords = wnl.lemmatize(word)
  nwords = porter.stem(nwords)
  return nwords
```

This feature function will calculate word counts of subjectivity words. Negative feature will have number of weakly negative words + 2 * number of strongly negative words. Same way it will count for positive features. It will not count neutral words

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

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

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

For example-
Subjectivitysets_without_preprocessing -
({'contains(the)': False, 'contains(and)': False, 'contains(,)': False, 'contains(of)': False, 'contains(a)': True, 'contains(to)': False, 'contains(that)': False, "contains('s)": False, 'contains(.)': False, 'contains(in)': False, 'contains(is)': False,……………… 'positivecount': 2, 'negativecount': 0}, 1)

Sentiment Lexicon(LIWC) feature:
I have defined another function that will calculate word counts of positive and negative words just like we did subjectivity count earlier.

```
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
    if 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
```

For example
liwcsets_without_preprocessing -
………..'contains(young)': False, 'contains(set)': False, 'contains(conquer)': False, 'contains(online)': False, 'contains(world)': False, 'contains(laptops)': False, 'contains(cell)': False, 'contains(phones)': False, 'contains(sketchy)': False, 'positivecount': 0, 'negativecount': 1}, 2)

I have added pre-processed version of positive words and negative words to their respective dictionary that I got by reading LIWC sentiment lexicon file. For this I have reused function define for pre-processing of negative words dictionary.

```
import sentiment_read_LIWC_pos_neg_words
poslist,neglist = sentiment_read_LIWC_pos_neg_words.read_words()
poslist = poslist+negativewordproc(poslist)
neglist = neglist+negativewordproc(neglist)
```

Sentiment lexicon Combination approach:
If a word is found in positive dictionary of LIWC sentiment lexicon then it will be considered as strongly positive. Similarly, if a word is found in negative dictionary of LIWC sentiment lexicon then it will be considered as strongly negative. Rest of the approach is like subjectivity feature.

```
def SL_liwc_features(doc, word_features, SL,poslist,neglist):
  document_words = set(doc)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  # count variables for the 4 classes of subjectivity
  weakPos = 0
  strongPos = 0
  weakNeg = 0
  strongNeg = 0
  for word in document_words:
    if 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
```

Bing Liu's Opinion Lexicon

Bing Liu maintains and freely distributes a sentiment lexicon consisting of lists of strings.
Positive words: 2006
Negative words: 4783
I have defined another function to read this lexicon features and get two dictionaries of positive and negative list so that we can reuse feature function defined for LIWC.

```
def read_opinionlexicon():
  POLARITY_DATA_DIR = os.path.join('polarity-data', 'rt-polaritydata')
  POSITIVE_REVIEWS = os.path.join(POLARITY_DATA_DIR, 'rt-polarity-pos.txt')
  NEGATIVE_REVIEWS = os.path.join(POLARITY_DATA_DIR, 'rt-polarity-neg.txt')
  pos_features = []
  neg_features = []

  for line in open(POSITIVE_REVIEWS, 'r').readlines()[35:]:
    pos_words = re.findall(r"[\w']+|[.,!?;]", line.rstrip())
    pos_features.append(pos_words[0])
  for line in open(NEGATIVE_REVIEWS, 'r').readlines()[35:]:
    neg_words = re.findall(r"[\w']+|[.,!?;]", line.rstrip())
    neg_features.append(neg_words[0])

  return pos_features,neg_features
```

Note-Files related to this lexicon can be found in corpus/polarity-data folder

8.) Building Feature set and saving it in csv file-

```
  unigramsets_without_preprocessing = [(unigram_features(d, uword_features), s) for (d, s) in wordtoken]
  print(" ")
  print("Unigramsets_without_preprocessing -")
  print(unigramsets_without_preprocessing[0])
save_features.writeFeatureSets(unigramsets_without_preprocessing,"outputcsv/unigramsets_without_preprocessing.csv")
  print(" ")
```

NLTK Classifiers-
Naive Bayes Classifier:
I am using Naïve Bayes classifier to train and test data with 90 % of data as training set and 10% as test set initially.

```
def nltk_naive_bayes(featuresets,percent):
  training_size = int(percent*len(featuresets))
  train_set, test_set = featuresets[training_size:], featuresets[:training_size]
  classifier = nltk.NaiveBayesClassifier.train(train_set)
  print("Naive Bayes Classifier ")
  print("Accuracy : ",nltk.classify.accuracy(classifier, test_set))
  print("Showing most informative features:")
  print(classifier.show_most_informative_features(10))
  confusionmatrix(classifier,test_set)
  print(" ")
```

I am also printing confusion matrix to know how many of the actual class labels (the gold standard labels) match with the predicted labels

```
from nltk.metrics import ConfusionMatrix
def confusionmatrix(classifier_type, test_set):
  reflist = []
  testlist = []
  for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier_type.classify(features))
  print("Confusion matrix:")
  cm = ConfusionMatrix(reflist, testlist)
  print(cm)
```

Output of Naïve Bayes Classifier will look like this-

```
Naive Bayes Classifier
--------------------------------------------------
Accuracy with Unigramsets_without_preprocessing -:
Naive Bayes Classifier
Accuracy :  0.7
Showing most informative features:
Most Informative Features
           contains(off) = True              0 : 2      =     17.2 : 1.0
           contains(...) = True              0 : 2      =     17.2 : 1.0
             contains(.) = True              4 : 2      =     14.7 : 1.0
            contains(as) = True              0 : 2      =     12.3 : 1.0
         contains(could) = True              4 : 2      =     11.4 : 1.0
          contains(next) = True              4 : 2      =     11.4 : 1.0
      contains(director) = True              4 : 2      =     11.4 : 1.0
          contains(does) = True              4 : 2      =     11.4 : 1.0
        contains(called) = True              4 : 2      =     11.4 : 1.0
     contains(landscape) = True              4 : 2      =     11.4 : 1.0
             contains(`) = True              0 : 2      =     10.3 : 1.0
          contains(with) = True              3 : 2      =      8.6 : 1.0
            contains(is) = True              4 : 2      =      8.2 : 1.0
             contains(') = True              0 : 2      =      7.4 : 1.0
            contains(on) = True              0 : 2      =      7.4 : 1.0
          contains(from) = True              4 : 2      =      6.9 : 1.0
           contains(n't) = True              4 : 2      =      6.9 : 1.0
            contains(be) = True              4 : 2      =      6.9 : 1.0
           contains(all) = True              4 : 2      =      6.9 : 1.0
          contains(than) = True              1 : 2      =      5.5 : 1.0
None
Confusion matrix:
  | 0 1 2 3 4 |
--+-----------+
0 |<1>. . 1 . |
1 | .<3>1 . 1 |
2 | . 1<9>. . |
3 | . 1 1<.>. |
4 | . . . .<1>|
--+-----------+
(row = reference; col = test)


--------------------------------------------------
Accuracy with Bigramsets_without_preprocessing -:
Naive Bayes Classifier
Accuracy :  0.7
Showing most informative features:
Most Informative Features
           contains(off) = True              0 : 2      =     17.2 : 1.0
           contains(...) = True              0 : 2      =     17.2 : 1.0
             contains(.) = True              4 : 2      =     14.7 : 1.0
```

Maximum Entropy Classifier-

Max Entropy classifier is a probabilistic classifier which belongs to the class of exponential models. Unlike the Naive Bayes classifier, the Max Entropy does not assume that the features are conditionally independent of each other. Max Entropy classifier can be used to solve a large variety of text classification problems such as language detection, topic classification, sentiment analysis and more.

We are going to use three different algorithms of max entropy to train and test our data:

a.) Generalized Iterative Scaling (GIS) algorithm

b.) Improved Iterative Scaling (IIS)

```
from nltk.classify import MaxentClassifier
def maximum_entropy(featuresets,percent):
  training_size = int(percent*len(featuresets))
  train_set, test_set = featuresets[training_size:], featuresets[:training_size]
  classifier1 = MaxentClassifier.train(train_set, 'GIS', max_iter = 1)
  print("Maximum Entropy Classifier- Generalized Iterative Scaling (GIS) algorithm")
```

```
print("Accuracy : ",nltk.classify.accuracy(classifier1, test_set))
print("Showing most informative features:")
print(classifier1.show_most_informative_features(10))
confusionmatrix(classifier1,test_set)
print(" ")
classifier2 = MaxentClassifier.train(train_set, 'IIS', max_iter = 1)
print("Maximum Entropy Classifier- Iterative Scaling (IIS) algorithm")
print("Accuracy : ",nltk.classify.accuracy(classifier2, test_set))
print("Showing most informative features:")
print(classifier2.show_most_informative_features(10))
confusionmatrix(classifier2,test_set)
print(" ")
```

Output of Maximum Entropy Classifier will look like this-

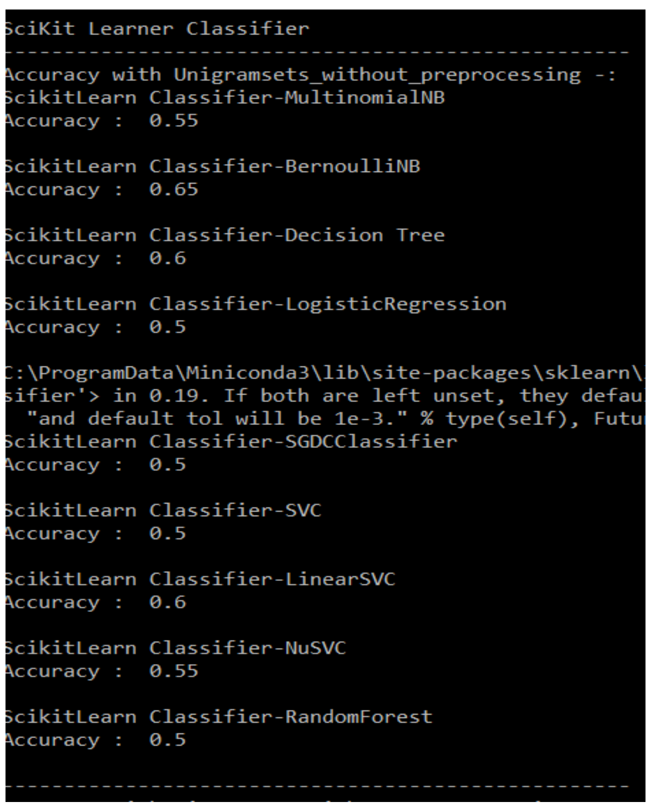
```
Maximum Entropy
---------------------------------------------------
Accuracy with Unigramsets_without_preprocessing -:
  ==> Training (1 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -1.60944        0.044
         Final          -0.93925        0.567
Maximum Entropy Classifier- Generalized Iterative Scaling (GIS) algorithm
Accuracy :  0.5
Showing most informative features:
  -0.012 contains(as)==False and label is 0
  -0.011 contains(of)==False and label is 0
  -0.011 contains(fails)==False and label is 0
  -0.011 contains(scummy)==False and label is 0
  -0.011 contains(ripoff)==False and label is 0
  -0.011 contains(David)==False and label is 0
  -0.011 contains(Cronenberg)==False and label is 0
  -0.011 contains(brilliant)==False and label is 0
  -0.011 contains(Videodrome)==False and label is 0
  -0.010 contains(off)==False and label is 0
None
Confusion matrix:
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  2  .  . |
1 |  . <.> 5  .  . |
2 |  .  .<10> .  . |
3 |  .  .  2 <.> . |
4 |  .  .  1  . <.>|
--+----------------+
(row = reference; col = test)


  ==> Training (1 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -1.60944        0.044
         Final          -1.09172        0.567
Maximum Entropy Classifier- Iterative Scaling (IIS) algorithm
Accuracy :  0.5
Showing most informative features:
   0.008 contains(Rock)==True and label is 2
   0.008 contains(modern)==True and label is 2
   0.008 contains(cinema)==True and label is 2
```

Sci-Kit Learner Classifiers-
We will also train and test our dataset using 8 algorithms from Sci-kit learner classifiers:
a.) Random Forest
b.) MultinomialNB
c.) BernoulliNB’
d.) Logistic Regressions
e.) SGDClassifer
f.) SVC
g.) Linear SVC
h.) NuSVC
i.) Decision Tree Classifier

```
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.naive_bayes import  MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
def sklearn(featuresets,percent):
  training_size = int(percent*len(featuresets))
  train_set, test_set = featuresets[training_size:], featuresets[:training_size]
  classifier1 = SklearnClassifier(MultinomialNB())
  classifier1.train(train_set)
  print("ScikitLearn Classifier-MultinomialNB")
  print("Accuracy : ",nltk.classify.accuracy(classifier1, test_set))
  print(" ")
  classifier2 = SklearnClassifier(BernoulliNB())
  classifier2.train(train_set)
  print("ScikitLearn Classifier-BernoulliNB")
  print("Accuracy : ",nltk.classify.accuracy(classifier2, test_set))
  print(" ")
  classifier3 = SklearnClassifier(DecisionTreeClassifier())
  classifier3.train(train_set)
  print("ScikitLearn Classifier-Decision Tree")
  print("Accuracy : ",nltk.classify.accuracy(classifier3, test_set))
  print(" ")
  classifier4 = SklearnClassifier(LogisticRegression())
  classifier4.train(train_set)
  print("ScikitLearn Classifier-LogisticRegression")
  print("Accuracy : ",nltk.classify.accuracy(classifier4, test_set))
  print(" ")
  classifier5 = SklearnClassifier(SGDClassifier())
  classifier5.train(train_set)
  print("ScikitLearn Classifier-SGDCClassifier")
  print("Accuracy : ",nltk.classify.accuracy(classifier5, test_set))
  print(" ")
```

```
classifier6 = SklearnClassifier(SVC())
classifier6.train(train_set)
print("ScikitLearn Classifier-SVC")
print("Accuracy : ",nltk.classify.accuracy(classifier6, test_set))
print(" ")
classifier7 = SklearnClassifier(LinearSVC())
classifier7.train(train_set)
print("ScikitLearn Classifier-LinearSVC")
print("Accuracy : ",nltk.classify.accuracy(classifier7, test_set))
print(" ")
classifier8 = SklearnClassifier(NuSVC(nu=0.09))
classifier8.train(train_set)
print("ScikitLearn Classifier-NuSVC")
print("Accuracy : ",nltk.classify.accuracy(classifier8, test_set))
print(" ")
classifier9 = SklearnClassifier(RandomForestClassifier())
classifier9.train(train_set)
print("ScikitLearn Classifier-RandomForest")
print("Accuracy : ",nltk.classify.accuracy(classifier9, test_set))
print(" ")
```

Output of SciKit Learn Classifier will look like this-



```
SciKit Learner Classifier
---------------------------------------------------
Accuracy with Unigramsets_without_preprocessing -:
ScikitLearn Classifier-MultinomialNB
Accuracy :  0.55

ScikitLearn Classifier-BernoulliNB
Accuracy :  0.65

ScikitLearn Classifier-Decision Tree
Accuracy :  0.6

ScikitLearn Classifier-LogisticRegression
Accuracy :  0.5

C:\ProgramData\Miniconda3\lib\site-packages\sklearn\
sifier'> in 0.19. If both are left unset, they defau
  "and default tol will be 1e-3." % type(self), Futu
ScikitLearn Classifier-SGDCClassifier
Accuracy :  0.5

ScikitLearn Classifier-SVC
Accuracy :  0.5

ScikitLearn Classifier-LinearSVC
Accuracy :  0.6

ScikitLearn Classifier-NuSVC
Accuracy :  0.55

ScikitLearn Classifier-RandomForest
Accuracy :  0.5

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

Single Fold Performances of all classifiers against all feature sets -
Dataset- limited to 30000 phrases (to avoid memory error)
Bag of words size-500(most frequent words)
90/10 split

a.) Without preprocessing –

	Naïve Bayes	GIS	IIS	Multi Nomial NB	Bernoulli NB	Decision Tree	Logistic Regression	SGDC	SVC	Linear SVC	Nu SVC	Random Forest
Unigram	0.5400	0.503	0.503	0.5516	0.5383	0.5060	0.5563	0.5483	0.5030	0.5536	0.1513	0.5296
Bigram	0.5400	0.503	0.503	0.5480	0.5430	0.5100	0.5563	0.5493	0.5030	0.5536	0.0663	0.5230
POS	0.5270	0.503	0.179	0.5536	0.5283	0.4793	0.5570	0.5190	0.5096	0.5570	0.2206	0.5170
Negation	0.5220	0.503	0.043	0.5303	0.5430	0.5086	0.5630	0.5496	0.5030	0.5606	0.4143	0.5273
SL	0.5456	0.503	0.503	0.5663	0.5423	0.5350	0.5680	0.5580	0.5410	0.5670	0.1896	0.5576
LIWC	0.5443	0.503	0.179	0.5526	0.5450	0.5260	0.5663	0.5420	0.5400	0.5606	0.1323	0.5386
SL+ LIWC	0.5500	0.503	0.179	0.5703	0.5443	0.5416	0.5690	0.5310	0.5573	0.5663	0.1773	0.5510
Opinion	0.5443	0.503	0.179	0.5526	0.545	0.5250	0.5663	0.5463	0.5400	0.5603	0.1323	0.5530

b.) With preprocessing –

	Naïve Bayes	GIS	IIS	Multi Nomial NB	Bernoulli NB	Decision Tree	Logistic Regression	SGDC	SVC	Linear SVC	Nu SVC	Random Forest
Unigram	0.5610	0.503	0.503	0.5546	0.5603	0.5570	0.5600	0.5556	0.5030	0.5563	0.1696	0.5596
Bigram	0.5606	0.503	0.503	0.5533	0.5580	0.5600	0.5600	0.5520	0.5030	0.5556	0.1220	0.5633
POS	0.551	0.503	0.503	0.5526	0.5486	0.5170	0.5633	0.5516	0.5050	0.5626	0.1166	0.5320
Negation	0.5306	0.503	0.043	0.5326	0.5333	0.5556	0.5530	0.5520	0.5030	0.5546	0.2516	0.5590
SL	0.5716	0.503	0.503	0.5613	0.5613	0.5556	0.5773	0.5626	0.5350	0.5650	0.129	0.5746
LIWC	0.5576	0.503	0.503	0.5520	0.5546	0.5590	0.5670	0.5543	0.5396	0.5633	0.132	0.5663
SL+ LIWC	0.5706	0.503	0.503	0.5593	0.5616	0.5560	0.5746	0.5320	0.5333	0.5650	0.216	0.5676
Opinion	0.5443	0.503	0.503	0.5520	0.5546	0.5580	0.5670	0.5626	0.5396	0.5633	0.1320	0.5700

Observations-
I will consider unigram feature (naïve bayes) without preprocessing accuracy i.e. 0.54 as baseline for comparison
We will use single fold results to drop some classifiers from cross validation testing especially ones whose accuracy remained low and did not show any improvement for any feature set.

a. Our maximum entropy classifiers (GIS and IIS) were just able to achieve 50% accuracy. Moreover, they did not show any improvement with any additional feature set. In fact, accuracy remained same for both processed and pre-processed version of document. So, we will drop this classifier for cross validation test.
b. SVC classifier also showed similar results, so we will drop that also.
c. NuSVC classidfier never achieved more than 25% accuracy for any featureset so we will drop this also.
d. Bigram featureset achieved similar accuracy as unigram featureset. Whereas POS and Negation featureset frequency was below unigramset
e. Sentiment lexicons showed slight improvement in performance compared to unigrams
f. Overall, Preprocessed version achieved higher accuracy compared to un processed version.

Note- Green color depicts better accuracy compared to baseline
Yellow color depicts similar level of accuracy
Red color depicts accuracy below standards
I have included screenshot of every result mentioned above as evidence in corpus/Single fold output folder

Cross Validation-

I defined cross-validation functions for every classifier so that they can be trained in multifold and also calculated accuracy, fscore, recall and precision

```
def naive_bayes(num_folds, featuresets, label_list):
    subset_size = int(len(featuresets)/num_folds)
    # overall gold labels for each instance (reference) and predicted labels (test)
    reflist = []
    testlist = []
    accuracy_list = []
    print("Naive Bayes Classifier")
    # iterate over the folds
    for i in range(num_folds):
        print('Start Fold', i)
        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 and save accuracy
        accuracy_this_round = nltk.classify.accuracy(classifier, test_this_round)
        print(i, accuracy_this_round)
        accuracy_list.append(accuracy_this_round)

        # add the gold labels and predicted labels for this round to the overall lists
        for (features, label) in test_this_round:
            reflist.append(label)
            testlist.append(classifier.classify(features))
    print('Done with cross-validation')
    # call the evaluation measures function
    print('mean accuracy-', sum(accuracy_list) / num_folds)
    (precision_list, recall_list) = eval_measures(reflist, testlist, label_list)
    print_evaluation (precision_list, recall_list, label_list)
    print(" ")
```

Similarly, I have defined functions for every classifier.I have also modified original f-score and print_evaluation functions to take care of DivisionByZero error and when precision type is None.

```
def eval_measures(reflist, testlist, label_list):
    #initialize sets
    # for each label in the label list, make a set of the indexes of the ref and test items
    #   store them in sets for each label, stored in dictionaries
    # first create dictionaries
    ref_sets = {}
    test_sets = {}
    # create empty sets for each label
    for lab in label_list:
```

```
        ref_sets[lab] = set()
        test_sets[lab] = set()

    # get gold labels
    for j, label in enumerate(reflist):
        ref_sets[label].add(j)
    # get predicted labels
    for k, label in enumerate(testlist):
        test_sets[label].add(k)

    # lists to return precision and recall for all labels
    precision_list = []
    recall_list = []
    #compute precision and recall for all labels using the NLTK functions
    for lab in label_list:
        precision_list.append ( precision(ref_sets[lab], test_sets[lab]))
        recall_list.append ( recall(ref_sets[lab], test_sets[lab]))
    return (precision_list, recall_list)

# This function computes F-measure (beta = 1) from precision and recall
def Fscore (precision, recall):
    print(precision)
    print(recall)
    if (precision == 0.0) and  (recall == 0.0 ):
      return 0.0
    else:
      return (2.0 * precision * recall) / (precision + recall)

# this function prints precision, recall and F-measure for each label
def print_evaluation(precision_list, recall_list, label_list):
    fscore=[]
    num_folds=0
    num=0
    for index, lab in enumerate(label_list):
        num +=1
        if precision_list[index] is None:
          precision_list[index]=0.0
        if recall_list[index] is None:
          recall_list[index]=0.0
        fscore.append(Fscore(precision_list[index],recall_list[index]))
        if fscore[num_folds]==0:
          num-=1
        num_folds += 1
    print('average precision', sum(precision_list)/num_folds)
    print('average recall   ', sum(recall_list)/num_folds)
    print('F-score  ',sum(fscore)/num)
```

Cross Validation results-

Dataset- limited to 50000 phrases (to avoid memory error)
Bag of words size-500(most frequent words)
90/10 split
No. of folds- 5

Note- We will use complete dataset in Weka classifier and some of the current classifiers as it takes long time to train and test entire dataset. We will shortlist classifiers here based on certain experiment like we did in single fold run so that we are left with few classifiers for training on Weka classifier and some of current classifiers

Unigram Feature sets:

Without Pre-processing

```
Unigramsets_without_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5361
Start Fold 1
1 0.5316
Start Fold 2
2 0.5239
Start Fold 3
3 0.5293
Start Fold 4
4 0.5353
Done with cross-validation
mean accuracy- 0.5312399999999999
0.18710743801652893
0.2538116591928251
0.36483967188665173
0.22130498699536355
0.6206227397018612
0.8279919978033186
0.413594677216328
0.21927733816379064
0.3094654242275625
0.21426146010186758
average precision 0.37912599020978643
average recall    0.3473294884514331
F-score   0.3480382859288907
```

```
MultinomialNB Classifier
Start Fold 0
0 0.554
Start Fold 1
1 0.5515
Start Fold 2
2 0.5491
Start Fold 3
3 0.5476
Start Fold 4
4 0.5627
Done with cross-validation
mean accuracy- 0.55298
0.4536741214057508
0.06367713004484304
0.4211111111111111
0.12857627501978966
0.5723866734216304
0.9306868552151571
0.47778675282714056
0.22556964438936028
0.476027397260274
0.09439728353140917
average precision 0.48019721120518144
average recall    0.2885814376401118
F-score   0.29630428870197734
```

With Pre-processing

```
Unigramsets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.554
Start Fold 1
1 0.5607
Start Fold 2
2 0.5495
Start Fold 3
3 0.549
Start Fold 4
4 0.5597
Done with cross-validation
mean accuracy- 0.55458
0.2604166666666667
0.13452914798206278
0.3841345117482216
0.20151532285423499
0.6083708667472262
0.8689444161142275
0.4613855329561095
0.2876346648870245
0.3799682034976153
0.16230899830220713
average precision 0.41885515632316783
average recall    0.3309865100279514
F-score   0.3478507282884228
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5536
Start Fold 1
1 0.5559
Start Fold 2
2 0.5466
Start Fold 3
3 0.5489
Start Fold 4
4 0.5587
Done with cross-validation
mean accuracy- 0.55274
0.5
0.03811659192825112
0.4353808353808354
0.10019224245165667
0.564638783269962
0.9436708115953399
0.5032217834130118
0.23081323291066833
0.4973544973544973
0.06383701188455009
average precision 0.5001191798836613
average recall    0.2753259781540932
F-score   0.2739764958195707
```

```
BernouliNB Classifier
Start Fold 0
0 0.5356
Start Fold 1
1 0.5319
Start Fold 2
2 0.523
Start Fold 3
3 0.5276
Start Fold 4
4 0.5346
Done with cross-validation
mean accuracy- 0.53054
0.18309859154929578
0.25650224215246636
0.3671222475322703
0.2187040597082438
0.6197624981674241
0.8291295649786216
0.4113138686131387
0.21489179140051481
0.31141868512110726
0.21392190152801357
average precision 0.3785431781966472
average recall    0.346629911953572
F-score   0.3466047317839497
```

```
Decision Tree Classifier
Start Fold 0
0 0.5099
Start Fold 1
1 0.5214
Start Fold 2
2 0.5106
Start Fold 3
3 0.5105
Start Fold 4
4 0.5205
Done with cross-validation
mean accuracy- 0.51458
0.19989615784008308
0.1726457399103139
0.3292328956461645
0.26936559990953296
0.6223134969911166
0.7666810496999176
0.38448137228625034
0.27352464486605016
0.2781725888324873
0.18607809847198642
average precision 0.3628193023192204
average recall    0.33365902657156027
F-score   0.3422426630546028
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5615
Start Fold 1
1 0.5608
Start Fold 2
2 0.5595
Start Fold 3
3 0.5518
Start Fold 4
4 0.5667
Done with cross-validation
mean accuracy- 0.56006
0.4380165289256198
0.09506726457399103
0.4242671009771987
0.17674997172905121
0.590993135938806
0.9152708586670851
0.4579033134166214
0.24110973400705502
0.4420289855072464
0.12427843803056027
average precision 0.47064181295309854
average recall    0.3104952534015485
F-score   0.32677837401118376
```

```
BernouliNB Classifier
Start Fold 0
0 0.5523
Start Fold 1
1 0.56
Start Fold 2
2 0.549
Start Fold 3
3 0.549
Start Fold 4
4 0.56
Done with cross-validation
mean accuracy- 0.55406
0.2506203473945409
0.1358744394618834
0.3859457092819615
0.19936673074748387
0.6075211429509813
0.8707096065586631
0.4630987673584023
0.2829631042044046
0.36961628817541115
0.16027164685908318
average precision 0.41536045103225944
average recall    0.3298371055663037
F-score   0.3459386729582749
```

```
Decision Tree Classifier
Start Fold 0
0 0.5707
Start Fold 1
1 0.5716
Start Fold 2
2 0.5628
Start Fold 3
3 0.5608
Start Fold 4
4 0.5738
Done with cross-validation
mean accuracy- 0.56794
0.3306505700871898
0.2210762331838565
0.4299489506522972
0.2571525500395793
0.6219840695148443
0.8423488800847292
0.49529824561403507
0.3364477071217466
0.3993630573248408
0.2129032258064516
average precision 0.4554489786386414
average recall    0.37398571924727264
F-score   0.3961671037413327
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5616
Start Fold 1
1 0.565
Start Fold 2
2 0.5569
Start Fold 3
3 0.5543
Start Fold 4
4 0.567
Done with cross-validation
mean accuracy- 0.56096
0.4032258064516129
0.07847533632286996
0.4217322834645669
0.15141920162840664
0.5892505677517033
0.9160161612991802
0.4742524916943522
0.27218991324244446
0.44129554655870445
0.11103565365025467
average precision 0.46595133918418796
average recall    0.30582725322863114
F-score   0.3189359665258462
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pa
sifier'> in 0.19. If both are left un
  "and default tol will be 1e-3." % t
0 0.5539
Start Fold 1
1 0.5436
Start Fold 2
2 0.5452
Start Fold 3
3 0.5465
Start Fold 4
4 0.5579
Done with cross-validation
mean accuracy- 0.54942
0.2725615314494075
0.13408071748878925
0.4042697390715012
0.1349089675449508
0.591060911246526
0.9093476640646452
0.43492007728789744
0.23605682143197634
0.3092485549132948
0.10899830220713073
average precision 0.4024121627937253
average recall    0.30467849454749846
F-score   0.31313930007989044
```

```
Linear SVC Classifier
Start Fold 0
0 0.5599
Start Fold 1
1 0.5582
Start Fold 2
2 0.5586
Start Fold 3
3 0.5498
Start Fold 4
4 0.5645
Done with cross-validation
mean accuracy- 0.5582
0.46321525885558584
0.07623318385650224
0.41968325791855204
0.16781635191677033
0.5873071441074241
0.9213117326324873
0.45665267384559227
0.23853560873295834
0.4258373205741627
0.09066213921901528
average precision 0.4705391310602634
average recall    0.2989118032715467
F-score   0.3101779088987275
```

```
Random Forests Classifier
Start Fold 0
0 0.5382
Start Fold 1
1 0.5495
Start Fold 2
2 0.539
Start Fold 3
3 0.5323
Start Fold 4
4 0.5522
Done with cross-validation
mean accuracy- 0.54224
0.29411764705882354
0.16591928251121077
0.37595748006878227
0.2719665271966527
0.6202283751619415
0.8075157886478641
0.4207751120485104
0.30431881018209556
0.3565051020408163
0.1898132427843803
average precision 0.4135167432757748
average recall    0.347906730264407
F-score   0.3660569465193266
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5553
Start Fold 1
1 0.5537
Start Fold 2
2 0.5486
Start Fold 3
3 0.5412
Start Fold 4
4 0.5592
Done with cross-validation
mean accuracy- 0.5516
0.23859191655801826
0.08206278026905829
0.40119485294117646
0.09872215311545855
0.5830426284668102
0.9244106225238301
0.46655172413793106
0.2579845552483554
0.30071599045346065
0.08556876061120543
average precision 0.39801942251147937
average recall    0.28974977435358157
F-score   0.2922253402515768
```

```
Linear SVC Classifier
Start Fold 0
0 0.5619
Start Fold 1
1 0.5638
Start Fold 2
2 0.5557
Start Fold 3
3 0.5517
Start Fold 4
4 0.5646
Done with cross-validation
mean accuracy- 0.55954
0.41867469879518807
0.06233183856502242
0.42880311385014597
0.14949677711183987
0.586567760627404
0.9212332797238457
0.4671030547163478
0.26532557917818667
0.42105263157894735
0.08421052631578947
average precision 0.4644402519136051
average recall    0.2965196001789368
F-score   0.3051481456435414
```

```
Random Forests Classifier
Start Fold 0
0 0.5762
Start Fold 1
1 0.5698
Start Fold 2
2 0.5649
Start Fold 3
3 0.565
Start Fold 4
4 0.5779
Done with cross-validation
mean accuracy- 0.57076
0.3818011257035647
0.18251121076233184
0.444017007491395
0.2479927626371141
0.6190244739249772
0.8492919624995097
0.4910762574364521
0.3462675183525598
0.403573629081947
0.22241086587436332
average precision 0.46789849872766726
average recall    0.36969486402517576
F-score   0.3948475315703952
```

Bigram Feature sets:

Without Pre-processing

```
Bigramsets_without_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5361
Start Fold 1
1 0.5316
Start Fold 2
2 0.5239
Start Fold 3
3 0.5293
Start Fold 4
4 0.5353
Done with cross-validation
mean accuracy- 0.5312399999999999
0.18710743801652893
0.2538116591928251
0.36483967188665173
0.22130498699536355
0.6206227397018612
0.8279919978033186
0.4136690647482014
0.21927733816379064
0.3093137254901961
0.21426146010186758
average precision 0.3791105279686878
average recall    0.3473294884514331
F-score   0.34803169865775485
```

```
MultinomialNB Classifier
Start Fold 0
0 0.552
Start Fold 1
1 0.552
Start Fold 2
2 0.5481
Start Fold 3
3 0.5473
Start Fold 4
4 0.5614
Done with cross-validation
mean accuracy- 0.55216
0.5402843601895735
0.051121076233183856
0.4237031427489587
0.12654076670813072
0.5701075629446853
0.9335111599262542
0.4757771497779572
0.22471160263132806
0.4878048780487805
0.07470288624787776
average precision 0.49953541874199103
average recall    0.28211749834935496
F-score   0.2861989038074016
```

```
BernouliNB Classifier
Start Fold 0
0 0.5377
Start Fold 1
1 0.5333
Start Fold 2
2 0.527
Start Fold 3
3 0.5302
Start Fold 4
4 0.5382
Done with cross-validation
mean accuracy- 0.53328
0.1902891824348447
0.2390134529147982
0.366469937292512
0.22469750084812845
0.6182416686087159
0.8325030400502098
0.4138913624220837
0.22156544951854323
0.3255179934569248
0.20271646859083192
average precision 0.38288202884301625
average recall    0.3440991823845024
F-score   0.3476974834735248
```

With Pre-processing

```
Bigramsets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.554
Start Fold 1
1 0.5607
Start Fold 2
2 0.5495
Start Fold 3
3 0.549
Start Fold 4
4 0.5597
Done with cross-validation
mean accuracy- 0.55458
0.2604166666666667
0.13452914798206278
0.38405172413793104
0.20151532285423499
0.6083875751833238
0.8689444161142275
0.4613855329561095
0.28763466488870245
0.3799682034976153
0.16230899830220713
average precision 0.4188419404883293
average recall    0.3309865100279514
F-score   0.3478491192341261
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5538
Start Fold 1
1 0.5548
Start Fold 2
2 0.5449
Start Fold 3
3 0.5466
Start Fold 4
4 0.5559
Done with cross-validation
mean accuracy- 0.5512
0.5102040816326531
0.02242152466367713
0.4348500517063082
0.09510347167250933
0.5624023687953184
0.9462205311261915
0.5013510704635211
0.2299551911526361
0.5075187969924813
0.04584040747028863
average precision 0.5032652739180564
average recall    0.26790822521706054
F-score   0.26077903617290693
```

```
BernouliNB Classifier
Start Fold 0
0 0.5559
Start Fold 1
1 0.5597
Start Fold 2
2 0.5509
Start Fold 3
3 0.5481
Start Fold 4
4 0.5592
Done with cross-validation
mean accuracy- 0.55476
0.271259418729817
0.11300448430493273
0.3847812028454408
0.20185457423951148
0.6051731625230703
0.8746322519907426
0.4595008421374981
0.28610925731718945
0.3812677388836329
0.1368421052631579
average precision 0.4203964730238918
average recall    0.32248853462310684
F-score   0.33875092002289925
```

```
Decision Tree Classifier
Start Fold 0
0 0.5125
Start Fold 1
1 0.5213
Start Fold 2
2 0.5107
Start Fold 3
3 0.509
Start Fold 4
4 0.5197
Done with cross-validation
mean accuracy- 0.51464
0.19862651875330165
0.16860986547085202
0.3330111832113765
0.27275811376229786
0.6225838295704232
0.7669164084258424
0.3806209850107066
0.27114119553818283
0.27601809954751133
0.1864176570458404
average precision 0.3621721232186639
average recall    0.33316864804860313
F-score   0.3417514298510356
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5615
Start Fold 1
1 0.5608
Start Fold 2
2 0.5594
Start Fold 3
3 0.5519
Start Fold 4
4 0.5668
Done with cross-validation
mean accuracy- 0.56008
0.4380165289256198
0.09506726457399103
0.4242671009771987
0.17674997172905121
0.5910334346504559
0.9153100851214059
0.45782041998551776
0.24110973400705502
0.4420289855072464
0.12427843803056027
average precision 0.4706332940092078
average recall    0.31050309869241266
F-score   0.32678279586293824
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pa
sifier'> in 0.19. If both are left un
  "and default tol will be 1e-3." % t
0 0.5465
Start Fold 1
1 0.5359
Start Fold 2
2 0.543
Start Fold 3
3 0.54
Start Fold 4
4 0.5579
Done with cross-validation
mean accuracy- 0.54466
0.28669275929549903
0.131390134529148
0.41339931480776554
0.12280900147008933
0.5894943119639631
0.9086023614325501
0.44046897109359207
0.20774144341691295
0.24253908100426338
0.1738539898132428
average precision 0.3945188876330166
average recall    0.30887938613238863
F-score   0.31389618569183686
```

```
Decision Tree Classifier
Start Fold 0
0 0.5731
Start Fold 1
1 0.5711
Start Fold 2
2 0.5604
Start Fold 3
3 0.5618
Start Fold 4
4 0.5734
Done with cross-validation
mean accuracy- 0.56796
0.33628922237380626
0.2210762331838565
0.4283295711060948
0.25749180142485584
0.6215809430084807
0.8423881065390499
0.4960696238068501
0.3369243969873201
0.4006472491909385
0.2101867572156197
average precision 0.45658332189723405
average recall    0.37361345907014043
F-score   0.3961517410165871
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5618
Start Fold 1
1 0.5654
Start Fold 2
2 0.5569
Start Fold 3
3 0.5541
Start Fold 4
4 0.5671
Done with cross-validation
mean accuracy- 0.56106
0.4032258064516129
0.07847533632286996
0.42196349905601005
0.15164536921859098
0.5893443036696785
0.9159769348448594
0.47460159362549803
0.2725712651349032
0.44070080862533695
0.11103565365025467
average precision 0.46596720228562727
average recall    0.3059409118342956
F-score   0.31907338621882503
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pack
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % typ
0 0.5522
Start Fold 1
1 0.5559
Start Fold 2
2 0.5457
Start Fold 3
3 0.5446
Start Fold 4
4 0.5584
Done with cross-validation
mean accuracy- 0.55136
0.28169014084507044
0.08071748878923767
0.3941958887545345
0.1105959516001357
0.5829334652485778
0.9244890754324716
0.4690462911321807
0.24053770616836687
0.29785247432306255
0.10831918505942276
average precision 0.40514365206068514
average recall    0.29293188140992693
F-score   0.2980177555736996
```

```
Linear SVC Classifier
Start Fold 0
0 0.5599
Start Fold 1
1 0.5582
Start Fold 2
2 0.5586
Start Fold 3
3 0.5497
Start Fold 4
4 0.5645
Done with cross-validation
mean accuracy- 0.55818
0.46321525885558584
0.07623318385650224
0.41968325791855204
0.16781635191677033
0.5872674534906981
0.9212725061781666
0.45673603504928806
0.23853560873295834
0.4258373205741627
0.09066213921901528
average precision 0.47054786517765734
average recall    0.2989039579806826
F-score   0.31017353509579715
```

```
Random Forests Classifier
Start Fold 0
0 0.54
Start Fold 1
1 0.5508
Start Fold 2
2 0.5431
Start Fold 3
3 0.5353
Start Fold 4
4 0.5531
Done with cross-validation
mean accuracy- 0.54446
0.31047765793528503
0.18071748878923766
0.38644227744573767
0.27784688454144524
0.6198163595991119
0.8102616404503197
0.4226158763985083
0.3025073886929164
0.3536423841059603
0.18132427843803056
average precision 0.41859891109692066
average recall    0.3505315361823899
F-score   0.36928586036765515
```

```
Linear SVC Classifier
Start Fold 0
0 0.5619
Start Fold 1
1 0.5639
Start Fold 2
2 0.5558
Start Fold 3
3 0.5518
Start Fold 4
4 0.5647
Done with cross-validation
mean accuracy- 0.55962
0.4186746987951807
0.06233183856502242
0.4293125810635538
0.1497229447020242
0.5866073881659465
0.9212725061781666
0.46727089627391744
0.26542091715130134
0.42105263157894735
0.08421052631578947
average precision 0.46458363917550916
average recall    0.2965917465824608
F-score   0.30524411492182296
```

```
Random Forests Classifier
Start Fold 0
0 0.574
Start Fold 1
1 0.5716
Start Fold 2
2 0.5653
Start Fold 3
3 0.5631
Start Fold 4
4 0.5764
Done with cross-validation
mean accuracy- 0.57008
0.3713768115942029
0.18385650224215247
0.4468795658526404
0.24211240529232161
0.6181496590876673
0.8499588122229632
0.4883438889637515
0.3455048145676423
0.4052728387492336
0.22444821731748726
average precision 0.4660045528494991
average recall    0.36917615032851336
F-score   0.39387212235367924
```

POS Feature sets:

Without Pre-processing

```
POSsets_without_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.523
Start Fold 1
1 0.5198
Start Fold 2
2 0.5146
Start Fold 3
3 0.517
Start Fold 4
4 0.5258
Done with cross-validation
mean accuracy- 0.5200400000000001
0.16270611057225995
0.3008968609865471
0.3493223965763195
0.2215311545855479
0.6314950492014346
0.8080649590083553
0.3992716120375695
0.19858899799790256
0.28353909465020577
0.23395585738539898
average precision 0.36526685260755787
average recall    0.3526075659927504
F-score   0.3425802118793061
```

With Pre-processing

```
Bigramsets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5453
Start Fold 1
1 0.5446
Start Fold 2
2 0.5374
Start Fold 3
3 0.5377
Start Fold 4
4 0.5529
Done with cross-validation
mean accuracy- 0.5435800000000001
0.21324448146605582
0.2295964125560538
0.37222500417292603
0.25217686305552417
0.6324657740333985
0.8245400698230887
0.43662635464111826
0.2650395652588426
0.31754735792622135
0.2162988115449915
average precision 0.39442179444794395
average recall    0.35753034444770015
F-score   0.364959269240626
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5519
Start Fold 1
1 0.5492
Start Fold 2
2 0.5466
Start Fold 3
3 0.5414
Start Fold 4
4 0.5605
Done with cross-validation
mean accuracy- 0.54992
0.3824175824175824
0.07802690582959641
0.373036093418259
0.1986882279769309
0.5860925449871466
0.8943239320597811
0.4657560235249478
0.23405472399656782
0.4683734939759036
0.10560271646859083
average precision 0.4551351476647679
average recall    0.3021393012662934
F-score   0.3161805817645387
```

```
BernouliNB Classifier
Start Fold 0
0 0.5247
Start Fold 1
1 0.5234
Start Fold 2
2 0.5192
Start Fold 3
3 0.5231
Start Fold 4
4 0.529
Done with cross-validation
mean accuracy- 0.52388
0.16945169712793734
0.29103139013452917
0.350684017350684
0.23770213728372724
0.6337692854713539
0.8040638606676342
0.40754785120982306
0.2151778053198589
0.29978213507625273
0.233616298811545
average precision 0.3722469972472102
average recall    0.3563182984434589
F-score   0.3501228228693371
```

```
Decision Tree Classifier
Start Fold 0
0 0.5021
Start Fold 1
1 0.5016
Start Fold 2
2 0.4995
Start Fold 3
3 0.4993
Start Fold 4
4 0.5052
Done with cross-validation
mean accuracy- 0.50154
0.19178743961352657
0.1780269058295964
0.3170913610938471
0.2884767612801086
0.6337669395102401
0.731965637626015
0.3492428831011508
0.2748593764896558
0.267946959304984
0.19898132427843804
average precision 0.3519671165247497
average recall    0.33446200110076274
F-score   0.3404169132793221
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5576
Start Fold 1
1 0.5584
Start Fold 2
2 0.5474
Start Fold 3
3 0.5499
Start Fold 4
4 0.5613
Done with cross-validation
mean accuracy- 0.55492
0.4260089686098655
0.042600896860986545
0.39694267515923565
0.1761845527535904
0.5800409059163923
0.9122111952300631
0.4927289896128423
0.24873677185623033
0.4924731182795699
0.07775891341256366
average precision 0.4776389315155812
average recall    0.2914984660226868
F-score   0.2991117279550333
```

```
BernouliNB Classifier
Start Fold 0
0 0.5484
Start Fold 1
1 0.549
Start Fold 2
2 0.5457
Start Fold 3
3 0.5434
Start Fold 4
4 0.5563
Done with cross-validation
mean accuracy- 0.54856
0.2300981461286805
0.18923766816143497
0.372999844648128
0.2715141920162841
0.6333434099153568
0.8218334444749539
0.4445249130938586
0.29259223948898844
0.335243553008596
0.19864176570458403
average precision 0.403241973358924
average recall    0.3547638619692491
F-score   0.3679385062138576
```

```
Decision Tree Classifier
Start Fold 0
0 0.5307
Start Fold 1
1 0.5387
Start Fold 2
2 0.5238
Start Fold 3
3 0.5209
Start Fold 4
4 0.5328
Done with cross-validation
mean accuracy- 0.52938
0.23601570166830227
0.215695067264574
0.34585650345856506
0.2883636774850164
0.6333023315480582
0.774604793472718
0.41504557203101616
0.290876155972924
0.31113271754982985
0.21731748726655348
average precision 0.38827056525115433
average recall    0.3573714362923572
F-score   0.36694065300830403
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5627
Start Fold 1
1 0.5617
Start Fold 2
2 0.5615
Start Fold 3
3 0.5516
Start Fold 4
4 0.5673
Done with cross-validation
mean accuracy- 0.56096
0.42708333333333333
0.09192825112107623
0.41997934950955085
0.18398733461494968
0.595352646125499
0.9125642333189503
0.4506796793307773
0.24654399847459244
0.43990384615384615
0.12427843803056027
average precision 0.4665997708906013
average recall     0.31186045111202587
F-score   0.32805911957636924
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5508
Start Fold 1
1 0.5316
Start Fold 2
2 0.5175
Start Fold 3
3 0.5459
Start Fold 4
4 0.5521
Done with cross-validation
mean accuracy- 0.53958
0.25755879059350506
0.1031390134529148
0.3319636408915453
0.30148139771570737
0.6243023255813953
0.8424273329933707
0.4661137440758294
0.18752979311659834
0.26058631921824105
0.21731748726655348
average precision 0.38810496407210326
average recall     0.330379004909029
F-score   0.3369754778305652
```

```
Linear SVC Classifier
Start Fold 0
0 0.5613
Start Fold 1
1 0.5586
Start Fold 2
2 0.5601
Start Fold 3
3 0.5486
Start Fold 4
4 0.5654
Done with cross-validation
mean accuracy- 0.5588
0.4563953488372093
0.07040358744394619
0.4139584443260522
0.17573221757322174
0.5905220195505392
0.9194288628250893
0.4525560538116592
0.24053770616836687
0.4204724409448819
0.09066213921901528
average precision 0.4667808614940684
average recall     0.2993529026459279
F-score   0.3102296498566521
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5658
Start Fold 1
1 0.5676
Start Fold 2
2 0.5558
Start Fold 3
3 0.5567
Start Fold 4
4 0.567
Done with cross-validation
mean accuracy- 0.56258
0.4245939675174014
0.08206278026905829
0.40790842872008326
0.17731539070451205
0.5972493573264781
0.9113482132350057
0.4635990139687757
0.26894842215654496
0.43783783783783786
0.11001697792869927
average precision 0.46623772107411526
average recall     0.3099383568587627
F-score   0.3245170666758786
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5356
Start Fold 1
1 0.5457
Start Fold 2
2 0.5181
Start Fold 3
3 0.5366
Start Fold 4
4 0.558
Done with cross-validation
mean accuracy- 0.5388
0.3401709401709402
0.08923766816143498
0.3396571808953237
0.23080402578310527
0.6378163259106236
0.8235594084650688
0.40125517425557483
0.2864906092096482
0.23333333333333334
0.23769100169779286
average precision 0.39044659091315914
average recall     0.33355654266341006
F-score   0.34098076228729185
```

```
Linear SVC Classifier
Start Fold 0
0 0.5658
Start Fold 1
1 0.5649
Start Fold 2
2 0.5543
Start Fold 3
3 0.5529
Start Fold 4
4 0.5668
Done with cross-validation
mean accuracy- 0.56094
0.4269340974212034
0.06681614349775784
0.4107929515418502
0.16872102227750763
0.5932697672059644
0.9177028988349744
0.46053729350909395
0.26313280579654874
0.4239864864864865
0.08522920203735145
average precision 0.46310411923291966
average recall     0.300320414488828
F-score   0.31044831280079155
```

```
Random Forests Classifier
Start Fold 0
0 0.5353
Start Fold 1
1 0.5333
Start Fold 2
2 0.5364
Start Fold 3
3 0.5332
Start Fold 4
4 0.5405
Done with cross-validation
mean accuracy- 0.53574
0.29014740108611325
0.16771300448430493
0.35339240841158576
0.30215990048626035
0.6291955247735749
0.78754952339858
0.402818331852228
0.3025073886929164
0.35997067448680353
0.166723259762309
average precision 0.4071048681220611
average recall    0.34533061536487414
F-score   0.3622564278341349
```

```
Random Forests Classifier
Start Fold 0
0 0.5512
Start Fold 1
1 0.5537
Start Fold 2
2 0.5451
Start Fold 3
3 0.543
Start Fold 4
4 0.5547
Done with cross-validation
mean accuracy- 0.54954
0.30527817403708984
0.19192825112107623
0.38239842541824826
0.30758792265068413
0.6346910726557131
0.7970423253442122
0.4331250796888946
0.3238630946706073
0.37653562653562656
0.20814940577249574
average precision 0.42640567566711446
average recall    0.36571419991181514
F-score   0.3843972618382042
```

Negation Feature sets:

Without Pre-processing

```
Negativesets_without_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5358
Start Fold 1
1 0.5343
Start Fold 2
2 0.5281
Start Fold 3
3 0.5265
Start Fold 4
4 0.5308
Done with cross-validation
mean accuracy- 0.5311
0.189232239566741
0.5327354260089686
0.4281949934123847
0.3307701006445776
0.7164714615638403
0.6814811909151531
0.4768627450980392
0.3477929259223949
0.2845984378129381
0.4825127334465195
average precision 0.4190719754907887
average recall    0.4750584753875227
F-score   0.42225729806899615
```

With Pre-processing

```
Negativesets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5335
Start Fold 1
1 0.5348
Start Fold 2
2 0.526
Start Fold 3
3 0.5241
Start Fold 4
4 0.5363
Done with cross-validation
mean accuracy- 0.53094
0.20297115218517878
0.5269058295964125
0.4103730664240218
0.4080063326925252
0.7503177517299816
0.6252304554191347
0.4743066007310256
0.42063113738201924
0.28990562166598277
0.4797962648556876
average precision 0.42557483854723815
average recall    0.4921140039891559
F-score   0.4383226819311009
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5392
Start Fold 1
1 0.5335
Start Fold 2
2 0.5319
Start Fold 3
3 0.5305
Start Fold 4
4 0.5474
Done with cross-validation
mean accuracy- 0.5365
0.0
0.0
0.43977272727272726
0.04376342870066719
0.5401213755900203
0.9740320872396344
0.5106450587861455
0.15320812279530938
0.0
0.0
average precision 0.2981078323297786
average recall    0.23420072774712217
F-score   0.33673618306501263
```

```
BernouliNB Classifier
Start Fold 0
0 0.5442
Start Fold 1
1 0.5473
Start Fold 2
2 0.5466
Start Fold 3
3 0.54
Start Fold 4
4 0.5513
Done with cross-validation
mean accuracy- 0.54588
0.6923076923076923
0.008071748878923767
0.38640429338103754
0.21983489765916545
0.5844119587838181
0.9032675636449221
0.4142130134496546 7
0.2172752407283821
0.6666666666666666
0.008828522920203734
average precision 0.5488007249177739
average recall    0.2714555947663194
F-score   0.26166498642743563
```

```
Decision Tree Classifier
Start Fold 0
0 0.5069
Start Fold 1
1 0.5243
Start Fold 2
2 0.5076
Start Fold 3
3 0.5096
Start Fold 4
4 0.5174
Done with cross-validation
mean accuracy- 0.51316
0.19743178170144463
0.1654708520179372
0.32496473906911144
0.2605450638923442
0.6207312046167993
0.7678970697838622
0.37997587454764775
0.2702831537801506
0.2809593734703867
0.19490662139219014
average precision 0.36081259468107796
average recall    0.3318205521732969
F-score   0.3403601312710733
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5417
Start Fold 1
1 0.5436
Start Fold 2
2 0.5337
Start Fold 3
3 0.5355
Start Fold 4
4 0.5494
Done with cross-validation
mean accuracy- 0.54078
0.8
0.0017937219730941704
0.46068548387096775
0.05167929435711863
0.5462203502549324
0.9665398344643628
0.4979434447300771
0.1846696539231576
0.3333333333333333
0.00033955857385398983
average precision 0.5276365224378622
average recall    0.24100441265831743
F-score   0.212919853220695
```

```
BernouliNB Classifier
Start Fold 0
0 0.5456
Start Fold 1
1 0.5538
Start Fold 2
2 0.5397
Start Fold 3
3 0.5387
Start Fold 4
4 0.5508
Done with cross-validation
mean accuracy- 0.54572
0.9
0.008071748878923767
0.41921236658078764
0.12880244260997398
0.5667841847478883
0.9396304868002981
0.4333199033037873
0.2050719801697016
0.6666666666666666
0.008149405772495755
average precision 0.597196624259826
average recall    0.25794521284627864
F-score   0.24292397614421354
```

```
Decision Tree Classifier
Start Fold 0
0 0.5726
Start Fold 1
1 0.5713
Start Fold 2
2 0.5617
Start Fold 3
3 0.5604
Start Fold 4
4 0.5716
Done with cross-validation
mean accuracy- 0.56752
0.336173001310616
0.23004484304932735
0.43226311667971806
0.24968901956349654
0.6201208981001727
0.8450555054328639
0.49363147466742147
0.3325388502240442
0.4
0.21188455008488966
average precision 0.45643769815158564
average recall    0.37384255367092434
F-score   0.3958853489985822
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5654
Start Fold 1
1 0.5616
Start Fold 2
2 0.5591
Start Fold 3
3 0.5543
Start Fold 4
4 0.5673
Done with cross-validation
mean accuracy- 0.56154
0.390057361376673
0.09147982062780269
0.4311951754385965
0.17788080967997286
0.5917948458049312
0.9160946142078218
0.46393797331410025
0.24530460482410144
0.4548780487804878
0.1266553480475382
average precision 0.4663726809429577
average recall    0.3114830394774474
F-score   0.3276396848565528
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5523
Start Fold 1
1 0.5486
Start Fold 2
2 0.5466
Start Fold 3
3 0.54
Start Fold 4
4 0.5586
Done with cross-validation
mean accuracy- 0.54922
0.26290516206482595
0.09820627802690583
0.3952760387023335
0.15707339138301482
0.5873946408353252
0.9157808025732554
0.44766100272708204
0.20345123462675183
0.32690622261174407
0.1266553480475382
average precision 0.4040286133882621
average recall    0.30023341093149325
F-score   0.30917203419278455
```

```
Linear SVC Classifier
Start Fold 0
0 0.5632
Start Fold 1
1 0.5593
Start Fold 2
2 0.5588
Start Fold 3
3 0.5515
Start Fold 4
4 0.5666
Done with cross-validation
mean accuracy- 0.55988
0.3794642857142857
0.07623318385650224
0.4337950138504155
0.1770892231143277
0.5895094339622642
0.9191935040991644
0.45916154680159016
0.24225378968443131
0.4316109422492401
0.09643463497453311
average precision 0.45870824451555914
average recall    0.30224086714579174
F-score   0.3143222981046313
```

```
Logistic Regression Classifier
Start Fold 0
0 0.564
Start Fold 1
1 0.565
Start Fold 2
2 0.558
Start Fold 3
3 0.5562
Start Fold 4
4 0.5664
Done with cross-validation
mean accuracy- 0.56192
0.3837471783295711
0.07623318385650224
0.43947797716150083
0.15232387198914396
0.5887968163614841
0.9169968226572001
0.47565419012918186
0.27381065878539423
0.4394141145139814
0.11205432937181664
average precision 0.46541805529914376
average recall    0.3062837733320114
F-score   0.3193372345813893
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5553
Start Fold 1
1 0.5527
Start Fold 2
2 0.5506
Start Fold 3
3 0.5464
Start Fold 4
4 0.5596
Done with cross-validation
mean accuracy- 0.55292
0.2745664739884393
0.08520179372197309
0.4133385032958511
0.12054732556824607
0.5860837868877016
0.9208410151806378
0.45858343337334934
0.2549337401086853
0.28554502369668244
0.08183361629881154
average precision 0.4036234442484048
average recall    0.29267149817567073
F-score   0.29757831715984995
```

```
Linear SVC Classifier
Start Fold 0
0 0.5644
Start Fold 1
1 0.5646
Start Fold 2
2 0.557
Start Fold 3
3 0.5546
Start Fold 4
4 0.5681
Done with cross-validation
mean accuracy- 0.56174
0.3713733075435203
0.08609865470852018
0.4421900161030596
0.1552640506615402
0.5902086596271409
0.9164868787510297
0.47143089695588475
0.27609877014014683
0.4036979969183359
0.08896434634974533
average precision 0.4557801754295883
average recall    0.30458254012219643
F-score   0.31633613195240123
```

```
Random Forests Classifier
Start Fold 0
0 0.5491
Start Fold 1
1 0.5537
Start Fold 2
2 0.5432
Start Fold 3
3 0.5397
Start Fold 4
4 0.5535
Done with cross-validation
mean accuracy- 0.54784
0.3445887445887446
0.17847533632286997
0.39539712753071465
0.2583964717855931
0.6115099583140343
0.8286196210724512
0.4320250284414107
0.289636762322433
0.36711409395973155
0.18573853989813244
average precision 0.43012699056692716
average recall    0.34817334628029595
F-score   0.3689710534416786
```

```
Random Forests Classifier
Start Fold 0
0 0.5729
Start Fold 1
1 0.5697
Start Fold 2
2 0.5694
Start Fold 3
3 0.5622
Start Fold 4
4 0.5756
Done with cross-validation
mean accuracy- 0.56996
0.38046511627906976
0.18340807174887894
0.45407706093189965
0.229220852651815
0.61273646855003322
0.8601576903463696
0.4892005610098177
0.3325388502240442
0.4183937823834197
0.21935483870967742
average precision 0.4709745978308478
average recall    0.36493606073615703
F-score   0.39031442577218006
```

Subjectivity Feature sets:

Without Pre-processing

```
Subjectivitysets_without_preprocessing
Naive Bayes Classifier
Start Fold 0
0 0.5464
Start Fold 1
1 0.5409
Start Fold 2
2 0.5375
Start Fold 3
3 0.5368
Start Fold 4
4 0.551
Done with cross-validation
mean accuracy- 0.54252
0.2008816120906801
0.2860986547085202
0.37960180315552217
0.228542349881262
0.6430719622980808
0.8135958890675872
0.4345273047563124
0.2822004004194871
0.31457905544147846
0.2601018675721562
average precision 0.39453234754841476
average recall    0.37410783232980255
F-score   0.37332657471153896
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5642
Start Fold 1
1 0.568
Start Fold 2
2 0.566
Start Fold 3
3 0.567
Start Fold 4
4 0.5789
Done with cross-validation
mean accuracy- 0.56882
0.44419134396355353
0.08744394618834081
0.4216902220940294
0.16532850842474273
0.6050411106618142
0.8803985407758993
0.4798239178283199
0.3741062065020498
0.5066991473812423
0.14125636672325975
average precision 0.4914891483857919
average recall    0.3297067137228585
F-score   0.34843918247667716
```

With Pre-processing

```
Subjectivitysets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5624
Start Fold 1
1 0.5615
Start Fold 2
2 0.5584
Start Fold 3
3 0.5544
Start Fold 4
4 0.5655
Done with cross-validation
mean accuracy- 0.56044
0.28295605858885486
0.1905829596412556
0.3919470920054464
0.22786384711070903
0.6435345753660109
0.8224610677440866
0.4438353099127322
0.3782057393459815
0.352557127312296
0.22003395585573854
average precision 0.4229660326370068
average recall    0.3678295139398836
F-score   0.38347709369497845
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5637
Start Fold 1
1 0.5665
Start Fold 2
2 0.5583
Start Fold 3
3 0.5608
Start Fold 4
4 0.5733
Done with cross-validation
mean accuracy- 0.56452
0.4676470588235294
0.07130044843049327
0.43566666666666665
0.14780052018545742
0.5913354981701974
0.9000509943906171
0.4834566522406813
0.33015540089617695
0.5064748201438849
0.11952461799660441
average precision 0.4969161392089919
average recall    0.31376639637986986
F-score   0.3287935846144904
```

```
BernouliNB Classifier
Start Fold 0
0 0.5474
Start Fold 1
1 0.5378
Start Fold 2
2 0.5348
Start Fold 3
3 0.5328
Start Fold 4
4 0.545
Done with cross-validation
mean accuracy- 0.53956
0.18947046219773492
0.27757847533632285
0.3826598368431038
0.22809001470089335
0.6412820672091434
0.8099478288157533
0.42902572997976296
0.2829631042044046
0.309462915601023
0.24651952461799662
average precision 0.39038020236615356
average recall    0.3690197895350741
F-score   0.36845666492395845
```

```
Decision Tree Classifier
Start Fold 0
0 0.5414
Start Fold 1
1 0.5425
Start Fold 2
2 0.5385
Start Fold 3
3 0.5387
Start Fold 4
4 0.5463
Done with cross-validation
mean accuracy- 0.54148
0.26954314720812184
0.23811659192825113
0.37565147668047577
0.317878548004071
0.653653344320835
0.7467540109049543
0.4323876807594657
0.3734388406902469
0.3291032148900169
0.2641765704584041
average precision 0.4120677727717831
average recall    0.38807291239718544
F-score   0.39763382287260585
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5703
Start Fold 1
1 0.5725
Start Fold 2
2 0.5677
Start Fold 3
3 0.5623
Start Fold 4
4 0.5803
Done with cross-validation
mean accuracy- 0.57062
0.42429906542056073
0.10179372197309416
0.4154402895054282
0.19473029514870518
0.6073496893755923
0.9050327540893579
0.4773573200992556
0.29345028124702066
0.48868778280542985
0.1466893039049236
average precision 0.48262682944125335
average recall    0.3283392712726203
F-score   0.3490737587999231
```

```
BernouliNB Classifier
Start Fold 0
0 0.5602
Start Fold 1
1 0.5586
Start Fold 2
2 0.5555
Start Fold 3
3 0.5534
Start Fold 4
4 0.5624
Done with cross-validation
mean accuracy- 0.55802
0.2597864768683274
0.16367713004484305
0.39358877086494687
0.23464887481623883
0.6432931552691104
0.8143804181540031
0.43760557432432434
0.3951758985603966
0.3517110266159696
0.18845500848896435
average precision 0.4171970007885357
average recall    0.35926746601288917
F-score   0.3748714970830652
```

```
Decision Tree Classifier
Start Fold 0
0 0.5689
Start Fold 1
1 0.5668
Start Fold 2
2 0.5614
Start Fold 3
3 0.5642
Start Fold 4
4 0.5691
Done with cross-validation
mean accuracy- 0.56608
0.29047875201721357
0.242152466367713
0.41123863466984645
0.31199819065927853
0.6491078746611184
0.8077119209194681
0.4807394002068252
0.354561922013538
0.35207700101317124
0.23599320882852293
average precision 0.436728332513635
average recall    0.39048354175770406
F-score   0.405881536751728
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5679
Start Fold 1
1 0.5717
Start Fold 2
2 0.5628
Start Fold 3
3 0.5646
Start Fold 4
4 0.5758
Done with cross-validation
mean accuracy- 0.56856
0.43956043956043955
0.08968609865470852
0.41285569105691056
0.18376116702476536
0.6049343914512454
0.9060134154473777
0.4735817991318665
0.30164934693488415
0.4578313253012048
0.11612903225806452
average precision 0.47775272930033336
average recall    0.31944781206396006
F-score   0.33651823450737334
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5451
Start Fold 1
1 0.5598
Start Fold 2
2 0.5636
Start Fold 3
3 0.54
Start Fold 4
4 0.5485
Done with cross-validation
mean accuracy- 0.5514
0.30668257756563244
0.11524663677130045
0.36122971818958155
0.23917222661992538
0.613082161687l704
0.8757698191660456
0.48434489402697495
0.19172466393364476
0.3143483023001095
0.2923599320882852
average precision 0.41593753075389384
average recall    0.3428546557158403
F-score   0.35084918978003
```

```
Linear SVC Classifier
Start Fold 0
0 0.5685
Start Fold 1
1 0.5696
Start Fold 2
2 0.5647
Start Fold 3
3 0.5599
Start Fold 4
4 0.5761
Done with cross-validation
mean accuracy- 0.56776
0.4716981132075472
0.07847533632286996
0.4161691542288557
0.1891891891891892
0.6019286951575791
0.9132703094967246
0.47176972955875374
0.284393173801125
0.45304777594728174
0.0933786078098472
average precision 0.4829226936200035
average recall    0.3117413233239512
F-score   0.3260019806364859
```

```
Random Forests Classifier
Start Fold 0
0 0.5626
Start Fold 1
1 0.5673
Start Fold 2
2 0.5654
Start Fold 3
3 0.5626
Start Fold 4
4 0.5752
Done with cross-validation
mean accuracy- 0.56662
0.3642483171278983
0.21838565022421524
0.4121133275513154
0.3224018998077576
0.6518028105020608
0.7878241085788256
0.46200043677658875
0.4033749642482601
0.38218714768883877
0.2302207130730051
average precision 0.45447040792934035
average recall    0.3924414671864127
F-score   0.41325522627315314
```

```
SGDC Classifier
tart Fold 0
:\ProgramData\Miniconda3\lib\site-pac
ifier'> in 0.19. If both are left uns
 "and default tol will be 1e-3." % ty
 0.5325
tart Fold 1
 0.544
tart Fold 2
 0.54
tart Fold 3
 0.549
tart Fold 4
 0.5746
one with cross-validation
ean accuracy- 0.54802
.1867145421903052
.23318385650224216
.36764705882352944
.1498360284971l636
.6112345545198352
.8848311301141489
.48084945013272656
.24177709981885784
.32309839497557574
.1572156196943973
verage precision 0.3939088001283944
verage recall    0.33336874692535245
-score   0.3353148741442934
```

```
inear SVC Classifier
tart Fold 0
 0.5655
tart Fold 1
 0.5681
tart Fold 2
 0.5597
tart Fold 3
 0.5589
tart Fold 4
 0.571
one with cross-validation
ean accuracy- 0.56464
.44376899696048633
.06547085201793722
.40690018435607056
.17471446341739227
.5991399289282587
.9126819126819127
.4669344870210136
.28811135475259797
.44366197183098594
.08556876061120543
verage precision 0.472081113819363
verage recall    0.3053094686962091
-score   0.3163561548398753
```

```
Random Forests Classifier
Start Fold 0
0 0.5787
Start Fold 1
1 0.5811
Start Fold 2
2 0.5724
Start Fold 3
3 0.5732
Start Fold 4
4 0.5852
Done with cross-validation
mean accuracy- 0.57812
0.3416974169741697
0.20762331838565024
0.43898598645882536
0.31527762071695126
0.6501434988769653
0.8175185344996666
0.48754962107542404
0.386404805033845
0.3953246753246753
0.25840407470288623
average precision 0.4627402397420119
average recall    0.39704567066779983
F-score   0.41864451588475216
```

LIWC Feature sets:

Without Pre-processing

```
LIWCsets_without_preprocessing
Naive Bayes Classifier
Start Fold 0
0 0.5447
Start Fold 1
1 0.54
Start Fold 2
2 0.5389
Start Fold 3
3 0.5366
Start Fold 4
4 0.5487
Done with cross-validation
mean accuracy- 0.5417799999999999
0.20492610837438424
0.27982062780269057
0.3760151085930123
0.2251498360284971
0.6374279703649501
0.820107480484839
0.43310131477184843
0.26694632472113644
0.3201168614357262
0.2604414261460102
average precision 0.39431747270798423
average recall    0.37049313903663467
F-score   0.37061546566836323
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5584
Start Fold 1
1 0.5592
Start Fold 2
2 0.5562
Start Fold 3
3 0.5544
Start Fold 4
4 0.5688
Done with cross-validation
mean accuracy- 0.5594
0.4572127139364303
0.08385650224215246
0.42534787514103045
0.12789777224923668
0.5840327737809752
0.9171145020201624
0.4718775181305399
0.2791495852798169
0.4949640287769784
0.1168081494057725
average precision 0.4866869819531908
average recall    0.3049653022394282
F-score   0.3183596512635774
```

```
BernouliNB Classifier
Start Fold 0
0 0.5429
Start Fold 1
1 0.54
Start Fold 2
2 0.5384
Start Fold 3
3 0.5341
Start Fold 4
4 0.5484
Done with cross-validation
mean accuracy- 0.54076
0.19993642720915447
0.2820627802690583
0.37425437752549545
0.21994798145425762
0.637522879804759
0.8197544423959519
0.4330418775885872
0.2691390981027743
0.31509754028838
0.25229202037351445
average precision 0.3919706204832752
average recall    0.36863926451911133
F-score   0.36809867407220115
```

With Pre-processing

```
LIWCsets_with_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5554
Start Fold 1
1 0.5555
Start Fold 2
2 0.5528
Start Fold 3
3 0.5526
Start Fold 4
4 0.5642
Done with cross-validation
mean accuracy- 0.5561
0.28111273792093705
0.17219730941704037
0.3903205531112508
0.21067511025670022
0.6321815154038302
0.8339151924057584
0.4367544584858864
0.35255982457812945
0.34031710079275196
0.20407470288624788
average precision 0.41613727314293125
average recall    0.35468442790877525
F-score   0.3703405346273316
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5618
Start Fold 1
1 0.5602
Start Fold 2
2 0.5536
Start Fold 3
3 0.5558
Start Fold 4
4 0.5639
Done with cross-validation
mean accuracy- 0.55906
0.49387755102040815
0.054260089686098655
0.4416745061147695
0.10618568359154133
0.5758077072790969
0.928411720864551
0.4925767073573078
0.2846791877204691
0.5161987041036717
0.08115449915110357
average precision 0.5040270351750509
average recall    0.2909382362027527
F-score   0.2961705308235909
```

```
BernouliNB Classifier
Start Fold 0
0 0.5545
Start Fold 1
1 0.5559
Start Fold 2
2 0.5515
Start Fold 3
3 0.5508
Start Fold 4
4 0.5648
Done with cross-validation
mean accuracy- 0.5555
0.27489481065918653
0.1757847533632287
0.391488460724116
0.20909193712540994
0.6319506526014661
0.8318754167810771
0.4350282485875706
0.3597101725617313
0.3419753086419753
0.18811544991511037
average precision 0.4150674962428629
average recall    0.3529155459493115
F-score   0.3683624594196733
```

```
Decision Tree Classifier
Start Fold 0
0 0.5257
Start Fold 1
1 0.5355
Start Fold 2
2 0.5235
Start Fold 3
3 0.527
Start Fold 4
4 0.5448
Done with cross-validation
mean accuracy- 0.5313
0.23663725998962118
0.20448430493273542
0.36109976861303933
0.30001130837950923
0.6435082033623658
0.7477346722629742
0.4172050323019381
0.35093907903517974
0.3125822007891276
0.24210526315789474
average precision 0.3942064930112184
average recall    0.3690549255536587
F-score   0.37858393190973516
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5674
Start Fold 1
1 0.572
Start Fold 2
2 0.5654
Start Fold 3
3 0.5575
Start Fold 4
4 0.5763
Done with cross-validation
mean accuracy- 0.56772
0.4293785310734463
0.10224215246636771
0.41700404858299595
0.1863620943118851
0.6020116436219981
0.9086023614325501
0.4759436254657379
0.2801029650109639
0.47119815668202764
0.13887945670628182
average precision 0.4791072010852412
average recall    0.32323780598560975
F-score   0.34282813901946707
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5563
Start Fold 1
1 0.5504
Start Fold 2
2 0.5557
Start Fold 3
3 0.5453
Start Fold 4
4 0.5587
Done with cross-validation
mean accuracy- 0.55328
0.29205607476635514
0.1681614349775785
0.40121096239643084
0.14237249802103358
0.5975861890732413
0.9070333032597183
0.46294804671768025
0.21918200019067594
0.3169968717413973
0.2064516129032258
average precision 0.41415962893902103
average recall    0.3286401698704465
F-score   0.3383293921009122
```

```
Decision Tree Classifier
Start Fold 0
0 0.5719
Start Fold 1
1 0.5675
Start Fold 2
2 0.5601
Start Fold 3
3 0.5627
Start Fold 4
4 0.5747
Done with cross-validation
mean accuracy- 0.56738
0.30269607843137253
0.22152466367713006
0.4263157894736842
0.283953409476422
0.6318385650224215
0.82905111206998
0.48976750661283586
0.33539898941748497
0.3853658536585366
0.24142614601018675
average precision 0.44719675863977015
average recall    0.3822708641302407
F-score   0.40176787434468564
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5638
Start Fold 1
1 0.5721
Start Fold 2
2 0.5589
Start Fold 3
3 0.5549
Start Fold 4
4 0.5741
Done with cross-validation
mean accuracy- 0.56476
0.41956521739130437
0.08654708520179372
0.41446111869031377
0.17177428474499604
0.5984601870511032
0.9086415878868709
0.4723610243597751
0.28839736867194204
0.44052287581699345
0.11443123938879457
average precision 0.4690740846618979
average recall    0.31395831317887946
F-score   0.32956356420158156
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5615
Start Fold 1
1 0.5633
Start Fold 2
2 0.5534
Start Fold 3
3 0.5539
Start Fold 4
4 0.5684
Done with cross-validation
mean accuracy- 0.5601
0.3241042345276873
0.08923766816143498
0.39753710135775183
0.14237249802103358
0.6061001598295152
0.8925195151610246
0.4450392132453629
0.3408332538850224
0.33900928792569657
0.07436332767402377
average precision 0.42235799937720275
average recall    0.30786525258050784
F-score   0.3159081366326518
```

```
Linear SVC Classifier
Start Fold 0
0 0.5653
Start Fold 1
1 0.5638
Start Fold 2
2 0.5618
Start Fold 3
3 0.5541
Start Fold 4
4 0.5724
Done with cross-validation
mean accuracy- 0.56348
0.46944444444444444
0.0757847533632287
0.4144050104384134
0.1795770666063553
0.5960398569238631
0.9151139528498019
0.46496604273645853
0.26761369053293926
0.44532488114104596
0.09541595925297114
average precision 0.4780360471368451
average recall    0.30670108452105926
F-score   0.31996623627683374
```

```
Random Forests Classifier
Start Fold 0
0 0.5503
Start Fold 1
1 0.5625
Start Fold 2
2 0.5533
Start Fold 3
3 0.5546
Start Fold 4
4 0.5727
Done with cross-validation
mean accuracy- 0.55868
0.32633158289572395
0.19506726457399104
0.403409933283914
0.3077010064457763
0.6440374023312412
0.7889224492998078
0.4484180035650624
0.3837353417866336
0.3731082654249127
0.21765704584040746
average precision 0.43906103750017084
average recall    0.37861662158932324
F-score   0.3981878812579376
```

```
Linear SVC Classifier
Start Fold 0
0 0.5609
Start Fold 1
1 0.5696
Start Fold 2
2 0.5561
Start Fold 3
3 0.5532
Start Fold 4
4 0.5687
Done with cross-validation
mean accuracy- 0.5617
0.43465045592705165
0.06412556053811659
0.4127210496292071
0.16363225149836028
0.593376405351783
0.9150747263954812
0.46632620491541654
0.2785775574411288
0.4188034188034188
0.0831918505942275
average precision 0.46517550692537546
average recall    0.3009203892934628
F-score   0.31072744025125576
```

```
Random Forests Classifier
Start Fold 0
0 0.5777
Start Fold 1
1 0.5749
Start Fold 2
2 0.5711
Start Fold 3
3 0.5712
Start Fold 4
4 0.5852
Done with cross-validation
mean accuracy- 0.57602
0.36102236421725240
0.20269058295964126
0.453318335208099
0.27343661653285084
0.6330909735768754
0.8402306515514063
0.48844969199178645
0.36285632567451614
0.39429530201342283
0.23938879456706283
average precision 0.46603533340148723
average recall    0.3837205942570955
F-score   0.4074266506003303
```

SL + LIWC Feature sets:

Without Pre-processing

```
SL+LIWC sets_without_preprocessing
Naive Bayes Classifier
Start Fold 0
0 0.5484
Start Fold 1
1 0.5428
Start Fold 2
2 0.5406
Start Fold 3
3 0.5394
Start Fold 4
4 0.5495
Done with cross-validation
mean accuracy- 0.54414
0.20321361058601134
0.289237668161435
0.38688338688338686
0.23148252855365825
0.6452115396597918
0.8123798689836426
0.43430290872617855
0.2889693965106302
0.3148901545972335
0.2628183361629881
average precision 0.3969003200905204
average recall    0.3769775596744708
F-score   0.3762239771005327
```

With Pre-processing

```
SL + LIWCsets_with_preprocessing
Naive Bayes Classifier
Start Fold 0
0 0.5618
Start Fold 1
1 0.5627
Start Fold 2
2 0.5592
Start Fold 3
3 0.5551
Start Fold 4
4 0.5675
Done with cross-validation
mean accuracy- 0.56126
0.27994616419919244
0.18654708520179372
0.3936027580923195
0.23238719891439558
0.6470078861914335
0.8206566508453301
0.4416110320674182
0.38468872151778055
0.34925864909390447
0.21595925297113752
average precision 0.4222852979288536
average recall    0.3680477818900875
F-score   0.38355427845107340
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5683
Start Fold 1
1 0.5719
Start Fold 2
2 0.565
Start Fold 3
3 0.5661
Start Fold 4
4 0.5789
Done with cross-validation
mean accuracy- 0.57004
0.44110854503464203
0.08565022421524664
0.4206652126499455
0.17448829582720796
0.6102957404546457
0.8677676224846036
0.47907136322049404
0.39937076937744304
0.503858875413451
0.15517826825127334
average precision 0.4909999473546357
average recall     0.33649103603115493
F-score    0.35592029014108767
```

```
BernouliNB Classifier
Start Fold 0
0 0.547
Start Fold 1
1 0.5383
Start Fold 2
2 0.5367
Start Fold 3
3 0.5351
Start Fold 4
4 0.5457
Done with cross-validation
mean accuracy- 0.54056
0.19003971891231286
0.2789237668161435
0.39088502269288955
0.23374420445550154
0.6434086717309075
0.807358882830581
0.42608939330250806
0.2899227762417771
0.3095546908776481
0.2431239388794567
average precision 0.3919954995032532
average recall     0.370614713844692
F-score    0.37042639535697175
```

```
Decision Tree Classifier
Start Fold 0
0 0.5381
Start Fold 1
1 0.5483
Start Fold 2
2 0.5356
Start Fold 3
3 0.539
Start Fold 4
4 0.5474
Done with cross-validation
mean accuracy- 0.54168
0.2449975597852611
0.22511210762331837
0.3763144058885384
0.32375890534886353
0.6590829418775853
0.737496567685247
0.4357695979632969
0.391648393555153
0.3389121338912134
0.27504244482173174
average precision 0.4110153278811791
average recall     0.39061168380686273
F-score    0.39899487320243665
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5659
Start Fold 1
1 0.5658
Start Fold 2
2 0.5604
Start Fold 3
3 0.5592
Start Fold 4
4 0.571
Done with cross-validation
mean accuracy- 0.56446
0.4778156996587031
0.06278026905829596
0.44062400509391914
0.15650797240755399
0.5938594199111575
0.8914996273486839
0.4771042879830598
0.34369339307846314
0.49594594594594593
0.12461799660441426
average precision 0.4970698717185571
average recall     0.31581985169948223
F-score    0.3307109466147486
```

```
BernouliNB Classifier
Start Fold 0
0 0.5587
Start Fold 1
1 0.5581
Start Fold 2
2 0.5543
Start Fold 3
3 0.5531
Start Fold 4
4 0.5616
Done with cross-validation
mean accuracy- 0.55716
0.25621007806955287
0.1618834080717489
0.3921975122502827
0.2353273775867918
0.6448866294482737
0.8110853959910563
0.4342963653308481
0.3998474592430165
0.34824281150159747
0.18505942275042445
average precision 0.415166679320111
average recall     0.3586406127286076
F-score    0.37382143494390946
```

```
Decision Tree Classifier
Start Fold 0
0 0.5609
Start Fold 1
1 0.5686
Start Fold 2
2 0.5595
Start Fold 3
3 0.5647
Start Fold 4
4 0.5717
Done with cross-validation
mean accuracy- 0.56508
0.29661472326706073
0.24753363228699551
0.4121592068814696
0.31968788872554565
0.6506400050945679
0.8015533675911034
0.4727272727272727
0.3569453713414053
0.35670419651995905
0.2366723259762309
average precision 0.437769080898066
average recall     0.39247851718425614
F-score    0.4079005546112914
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5714
Start Fold 1
1 0.5719
Start Fold 2
2 0.569
Start Fold 3
3 0.5634
Start Fold 4
4 0.5779
Done with cross-validation
mean accuracy- 0.57072
0.4197292069632495
0.09730941704035874
0.413197729422895
0.19755739002600928
0.6081387979296503
0.9033460165535637
0.47689011325374964
0.29707312422537896
0.5005861664712778
0.14499151103565366
average precision 0.48370840280816446
average recall    0.32805549177619286
F-score   0.34863239962783815
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
 "and default tol will be 1e-3." % ty
0 0.57
Start Fold 1
1 0.5305
Start Fold 2
2 0.5587
Start Fold 3
3 0.5431
Start Fold 4
4 0.5471
Done with cross-validation
mean accuracy- 0.54988
0.29454841334418225
0.16233183856502242
0.4209650582362729
0.11444080063326925
0.6020568566328298
0.8864001882869808
0.42272529386155855
0.27771951568309655
0.3139475038600103
0.2071307300509338
average precision 0.41084862518697074
average recall    0.32960461464386054
F-score   0.3382282036407832
```

```
Linear SVC Classifier
Start Fold 0
0 0.5706
Start Fold 1
1 0.5684
Start Fold 2
2 0.5658
Start Fold 3
3 0.5599
Start Fold 4
4 0.5754
Done with cross-validation
mean accuracy- 0.56802
0.4684931506849315
0.07668161434977579
0.4173170731707317
0.1934863734026914
0.6025474732800664
0.911112854509081
0.47225705329153606
0.28725331299456575
0.4596375617792422
0.09473684210526316
average precision 0.48405046244130157
average recall    0.31265419947227546
F-score   0.3271757765902334
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5665
Start Fold 1
1 0.5721
Start Fold 2
2 0.5644
Start Fold 3
3 0.5645
Start Fold 4
4 0.5773
Done with cross-validation
mean accuracy- 0.56896
0.4462242562929062
0.08744394618834081
0.41376739562624254
0.18828451882845187
0.6056034482758621
0.903855960459734
0.4751594245884621
0.30546286585947185
0.4572192513368984
0.11612903225806452
average precision 0.4795947552240743
average recall    0.3202352647188126
F-score   0.3374757753313976
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pa
sifier'> in 0.19. If both are left un
 "and default tol will be 1e-3." % t
0 0.5676
Start Fold 1
1 0.5325
Start Fold 2
2 0.5401
Start Fold 3
3 0.5514
Start Fold 4
4 0.5278
Done with cross-validation
mean accuracy- 0.54388
0.19004524886877827
0.22600896860986547
0.3931933381607531
0.12280900147008933
0.6297263573314762
0.8512532852155493
0.43589396808222886
0.307274287348651
0.24899304284145002
0.23089983022071306
average precision 0.3795703910569373
average recall    0.3476490745729736
F-score   0.34352291106870253
```

```
Linear SVC Classifier
Start Fold 0
0 0.5645
Start Fold 1
1 0.569
Start Fold 2
2 0.5585
Start Fold 3
3 0.5591
Start Fold 4
4 0.5707
Done with cross-validation
mean accuracy- 0.56436
0.44089456869009586
0.06188340807174888
0.4091381100726895
0.17822006106524935
0.5981546211576025
0.9129172714078374
0.4684797987737777
0.2841071598817809
0.44346289752650175
0.08522920203735145
average precision 0.47202599924413347
average recall    0.30447142049279363
F-score   0.315252222966102
```

```
Random Forests Classifier
Start Fold 0
0 0.5682
Start Fold 1
1 0.5732
Start Fold 2
2 0.5636
Start Fold 3
3 0.5605
Start Fold 4
4 0.5781
Done with cross-validation
mean accuracy- 0.56872
0.3499627699180938З
0.21076233183856502
0.41454701343240924
0.3280560895623657
0.6574388755073085
0.7815871023418193
0.4622779519331243
0.42177519305939554
0.40179573512906847
0.2431239388794567
average precision 0.4572044691840008
average recall    0.3970609311363205
F-score   0.4175092035927741
```

```
Random Forests Classifier
Start Fold 0
0 0.5737
Start Fold 1
1 0.5815
Start Fold 2
2 0.573
Start Fold 3
3 0.5743
Start Fold 4
4 0.5819
Done with cross-validation
mean accuracy- 0.57688
0.36281859070464767
0.21704035874439462
0.4339506172839506
0.3179916317991632
0.6513715239504929
0.8113207547169812
0.4823487879501059
0.3907903517971208
0.3958656330749354
0.2601018675721562
average precision 0.4652710305928265
average recall    0.3994489929259632
F-score   0.42138782607483155
```

Opinion Lexicon Feature sets:

Without Pre-processing

```
Opinionlexicon_without_preprocessing-
Naive Bayes Classifier
Start Fold 0
0 0.5447
Start Fold 1
1 0.54
Start Fold 2
2 0.5389
Start Fold 3
3 0.5366
Start Fold 4
4 0.5487
Done with cross-validation
mean accuracy- 0.5417799999999999
0.20492610837438424
0.27982062780269057
0.3760151085930123
0.2251498360284971
0.6374279703649501
0.820107480484839
0.43310131477184843
0.26694632472113644
0.3201168614357262
0.2604414261460102
average precision 0.39431747270798423
average recall    0.37049313903663467
F-score   0.37061546566836323
```

With Pre-processing

```
Opinionlexicon_with_preprocessing
Naive Bayes Classifier
Start Fold 0
0 0.5554
Start Fold 1
1 0.5555
Start Fold 2
2 0.5528
Start Fold 3
3 0.5526
Start Fold 4
4 0.5642
Done with cross-validation
mean accuracy- 0.5561
0.28111273792093705
0.17219730941704037
0.3903205531112508
0.21067511025670022
0.6321815154038302
0.8339151924057584
0.43675445848588864
0.35255982457812945
0.34031710079275196
0.20407470288624788
average precision 0.41613727314293125
average recall    0.35468442790877525
F-score   0.3703405346273316
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5584
Start Fold 1
1 0.5592
Start Fold 2
2 0.5562
Start Fold 3
3 0.5544
Start Fold 4
4 0.5688
Done with cross-validation
mean accuracy- 0.5594
0.4572127139364303
0.08385650224215246
0.42534787514103045
0.12789777224923668
0.5840327737809752
0.9171145020201624
0.4718775181305399
0.2791495852798169
0.4949640287769784
0.1168081494057725
average precision 0.4866869819531908
average recall    0.3049653022394282
F-score   0.3183596512635774
```

```
BernoulliNB Classifier
Start Fold 0
0 0.5429
Start Fold 1
1 0.54
Start Fold 2
2 0.5384
Start Fold 3
3 0.5341
Start Fold 4
4 0.5484
Done with cross-validation
mean accuracy- 0.54076
0.19993642720915447
0.2820627802690583
0.37425437752549545
0.21994798145425762
0.637522879804759
0.8197544423959519
0.4330418775885872
0.2691390981027743
0.31509754028838
0.25229202037351445
average precision 0.3919706204832752
average recall    0.36863926451911133
F-score   0.36809867407220115
```

```
Decision Tree Classifier
Start Fold 0
0 0.5265
Start Fold 1
1 0.5385
Start Fold 2
2 0.525
Start Fold 3
3 0.5265
Start Fold 4
4 0.5486
Done with cross-validation
mean accuracy- 0.53302
0.24314536989136057
0.21076233183856502
0.36620680234940584
0.3031776546420898
0.6444324360836232
0.7484799748950692
0.41785310734463277
0.35255982457812945
0.31526016615653696
0.24482173174872665
average precision 0.39737957636511184
average recall    0.371960303540516
F-score   0.3816289833301652
```

```
MultinomialNB Classifier
Start Fold 0
0 0.5618
Start Fold 1
1 0.5602
Start Fold 2
2 0.5536
Start Fold 3
3 0.5558
Start Fold 4
4 0.5639
Done with cross-validation
mean accuracy- 0.55906
0.49387755102040815
0.054260089686098655
0.4416745061147695
0.10618568359154133
0.5758077072790969
0.928411720864551
0.4925767073573078
0.2846791877204691
0.5161987041036717
0.08115449915110357
average precision 0.5040270351750509
average recall    0.2909382362027527
F-score   0.2961705308235909
```

```
BernoulliNB Classifier
Start Fold 0
0 0.5545
Start Fold 1
1 0.5559
Start Fold 2
2 0.5515
Start Fold 3
3 0.5508
Start Fold 4
4 0.5648
Done with cross-validation
mean accuracy- 0.5555
0.27489481065918653
0.1757847533632287
0.391488460724116
0.20909193712540994
0.6319506526014661
0.8318754167810771
0.4350282485875706
0.3597101725617313
0.34197530864197553
0.18811544991511037
average precision 0.4150674962428629
average recall    0.3529155459493115
F-score   0.3683624594196733
```

```
Decision Tree Classifier
Start Fold 0
0 0.5753
Start Fold 1
1 0.566
Start Fold 2
2 0.5597
Start Fold 3
3 0.5617
Start Fold 4
4 0.5763
Done with cross-validation
mean accuracy- 0.5678
0.3046683046683047
0.22242152466367712
0.42481012658227846
0.284631912246975
0.6321567454382291
0.8289726591613384
0.49255807483655584
0.3375917627991229
0.38457330415754926
0.23870967741935484
average precision 0.4477533111365835
average recall    0.3824655072580937
F-score   0.40209876086219865
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5674
Start Fold 1
1 0.572
Start Fold 2
2 0.5654
Start Fold 3
3 0.5575
Start Fold 4
4 0.5763
Done with cross-validation
mean accuracy- 0.56772
0.4293785310734463
0.10224215246636771
0.41700404858299595
0.1863620943118851
0.6020116436219981
0.9086023614325501
0.4759436254657379
0.2801029650109639
0.47119815668202764
0.13887945670628182
average precision 0.4791072010852412
average recall    0.32323780598560975
F-score   0.34282813901946707
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pa
sifier'> in 0.19. If both are left ur
  "and default tol will be 1e-3." % t
0 0.553
Start Fold 1
1 0.559
Start Fold 2
2 0.5377
Start Fold 3
3 0.5421
Start Fold 4
4 0.5722
Done with cross-validation
mean accuracy- 0.5528
0.28956521739130436
0.1493273542600897
0.3819022838970801
0.14938369331674772
0.6102484892827141
0.8833797513042796
0.4276765375854214
0.2863952712365335
0.3155737704918033
0.15687606112054328
average precision 0.4049932597296647
average recall    0.3250724262476387
F-score   0.33725503281038055
```

```
Linear SVC Classifier
Start Fold 0
0 0.5653
Start Fold 1
1 0.5637
Start Fold 2
2 0.5617
Start Fold 3
3 0.5541
Start Fold 4
4 0.5723
Done with cross-validation
mean accuracy- 0.56342
0.46944444444444444
0.0757847533632287
0.4142521534847298
0.17946398281126313
0.5960295357571731
0.9150747263954812
0.46473509933774837
0.26761369053293926
0.4444444444444444
0.09507640067911714
average precision 0.47778113549370804
average recall    0.3066027107564059
F-score   0.3198192473559429
```

```
Logistic Regression Classifier
Start Fold 0
0 0.5638
Start Fold 1
1 0.5721
Start Fold 2
2 0.5589
Start Fold 3
3 0.5549
Start Fold 4
4 0.5741
Done with cross-validation
mean accuracy- 0.56476
0.41956521739130437
0.08654708520179372
0.41446111869031377
0.17177428474499604
0.5984601870511032
0.9086415878868709
0.4723610243597751
0.28839736867194204
0.44052287581699345
0.11443123938879457
average precision 0.4690740846618979
average recall    0.31395831317887946
F-score   0.32956356420158156
```

```
SGDC Classifier
Start Fold 0
C:\ProgramData\Miniconda3\lib\site-pac
sifier'> in 0.19. If both are left uns
  "and default tol will be 1e-3." % ty
0 0.5513
Start Fold 1
1 0.5551
Start Fold 2
2 0.5525
Start Fold 3
3 0.5496
Start Fold 4
4 0.543
Done with cross-validation
mean accuracy- 0.5503
0.2797074954296161
0.06860986547085202
0.36361063950198075
0.14531267669342984
0.6053279665280831
0.8966775193190287
0.45505713798396724
0.25436171226999715
0.2398604448320977
0.1867572156196944
average precision 0.38871273685514895
average recall    0.3103437978746004
F-score   0.31538089771964983
```

```
Linear SVC Classifier
Start Fold 0
0 0.561
Start Fold 1
1 0.5696
Start Fold 2
2 0.5561
Start Fold 3
3 0.5532
Start Fold 4
4 0.5687
Done with cross-validation
mean accuracy- 0.56172
0.43465045592705165
0.06412556053811659
0.4127210496292071
0.16363225149836028
0.5933914989952433
0.9150747263954812
0.46632620491541654
0.2785775574411288
0.4197952218430034
0.0835314091680815
average precision 0.46537688626198437
average recall    0.3009883010082336
F-score   0.31083508201083293
```

```
Random Forests Classifier
Start Fold 0
0 0.5546
Start Fold 1
1 0.5638
Start Fold 2
2 0.5501
Start Fold 3
3 0.5575
Start Fold 4
4 0.5702
Done with cross-validation
mean accuracy- 0.55924
0.3300229182582124
0.1937219730941704
0.41004373397677574
0.307474838855592
0.641946255366513
0.7918252069195465
0.44289382605783184
0.3782057393459815
0.3968636911942099
0.2234295415959253
average precision 0.44435408497070855
average recall    0.37893145996224314
F-score   0.39970381208892625
```

```
Random Forests Classifier
Start Fold 0
0 0.5794
Start Fold 1
1 0.5721
Start Fold 2
2 0.566
Start Fold 3
3 0.5654
Start Fold 4
4 0.5831
Done with cross-validation
mean accuracy- 0.5732
0.34991974317817015
0.19551569506726457
0.444896449704142
0.2720796109917449
0.6320964914886664
0.8346212685835327
0.4858382240367441
0.36304700162074555
0.39685977260422306
0.24889643463497454
average precision 0.46192213620238914
average recall    0.3828320021796524
F-score   0.4058776804750793
```

Summary(Accuracy)-

Dataset- limited to 50000 phrases (to avoid memory error)
Bag of words size-500(most frequent words)
90/10 split
No. of folds- 5

a.) Without pre-processing

	Naïve Bayes	Multi Nomial NB	Bernoulli NB	Decision Tree	Logistic Regression	SGDC	Linear SVC	Random Forest	**Average (Feature-wise)**
Unigram	0.5312	0.5529	0.5305	0.5145	0.5600	0.5494	0.5582	0.5422	**0.5423**
Bigram	0.5312	0.5521	0.5332	0.5146	0.5600	0.5446	0.5581	0.5446	**0.5423**
POS	0.5200	0.5499	0.5238	0.5015	0.5609	0.5395	0.5588	0.5357	**0.5362**
Negation	0.5311	0.5365	0.5458	0.5131	0.5615	0.5492	0.5598	0.5478	**0.5431**
SL	0.5425	0.5688	0.5395	0.5414	0.5706	0.5514	0.5677	0.5666	**0.5560**
LIWC	0.5417	0.5594	0.5407	0.5313	0.5677	0.5532	0.5634	0.5586	**0.5520**
SL+ LIWC	0.5441	0.5700	0.5405	0.5416	0.5707	0.5498	0.5680	0.5687	**0.5566**
Opinion	0.5417	0.5594	0.5407	0.5330	0.5677	0.5528	0.5634	0.5592	**0.5522**
Average (Classifier-wise)	**0.5354**	**0.5561**	**0.5368**	**0.5238**	**0.5648**	**0.5487**	**0.5621**	**0.5529**	

a.) With pre-processing

	Naïve Bayes	Multi Nomial NB	Bernoulli NB	Decision Tree	Logistic Regression	SGDC	Linear SVC	Random Forest	**Average (Feature-wise)**
Unigram	0.5545	0.5527	0.5540	0.5679	0.5609	0.5516	0.5595	0.5707	**0.5589**
Bigram	0.5545	0.5512	0.5547	0.5679	0.5610	0.5513	0.5596	0.5700	**0.5587**
POS	0.5435	0.5549	0.5485	0.5293	0.5625	0.5388	0.5609	0.5495	**0.5484**
Negation	0.5309	0.5407	0.5457	0.5675	0.5619	0.5529	0.5617	0.5699	**0.5539**
SL	0.5604	0.5645	0.5580	0.5660	0.5685	0.5480	0.5646	**0.5781**	**0.5635**
LIWC	0.5561	0.5590	0.5555	0.5673	0.5647	0.5601	0.5617	0.5760	**0.5625**
SL+ LIWC	0.5612	0.5644	0.5571	0.5650	0.5689	0.5438	0.5643	0.5768	**0.5626**
Opinion	0.5561	0.5590	0.5555	0.5678	0.5647	0.5503	0.5617	0.5732	**0.5610**
Average (Classifier-wise)	**0.5521**	**0.5558**	**0.5536**	**0.5623**	**0.5641**	**0.5496**	**0.5617**	**0.5705**	

Feature set comparison-

Unigram Vs Bigram Feature set: No improvement in accuracy

Bigram accuracy was almost equal to its corresponding unigram feature set for both pre-processed as well un processed tokens and for all classifiers.

Unigram Vs POS Feature set: Accuracy declined

Average POS accuracy was 1% lower than its corresponding unigram feature set for both pre-processed as well un processed tokens. We need to classify parts of speech in more specific categories rather than general noun, verb, adverb and adjectives especially for sentiment analysis in order to obtain higher accuracies.

Unigram Vs Negation Feature set: No improvement in accuracy

Negation accuracy was almost equal to its corresponding unigram feature set for both pre-processed as well un processed tokens and for all classifiers.

Unigram Vs Lexicon Feature sets: Improvement in accuracy

- Average subjectivity accuracy was 1.35% higher than its corresponding unigram feature set for un-processed tokens. Average subjectivity accuracy was 0.45% higher than its corresponding unigram feature set for pre-processed tokens. In fact, Random Forest classifier produced best accuracy (0.5781) using subjectivity feature set on pre-processed version.
- Average LIWC accuracy was 1.00% higher than its corresponding unigram feature set for un-processed tokens. Average LIWC accuracy was 0.35% higher than its corresponding unigram feature set for pre-processed tokens.
- Average SL + LIWC accuracy was 1.40% higher than its corresponding unigram feature set for un-processed tokens. Average SL + LIWC accuracy was 0.36% higher than its corresponding unigram feature set for pre-processed tokens.
- Average opinion lexicon accuracy was 1.01% higher than its corresponding unigram feature set for un-processed tokens. Average opinion lexicon accuracy was 0.25% higher than its corresponding unigram feature set for pre-processed tokens.

Classifier comparison-

- BernoulliNB classifier average accuracy was almost equal to its corresponding average accuracy of Naïve Bayes classifier for both pre-processed as well unprocessed tokens.
- Logistic Regression, LinearSVC, Multi nomialNB and Random Forest classifiers produced better accuracy than its corresponding Naïve Bayes classifier for both pre-processed as well as un-processed tokens.
- DecisionTree and SGDC classifier performance varied compared to its Naïve Bayes counterpart.
  DecisionTree classifier produced better performance than Naive Bayes for pre-processed version whereas its performance was lower than Naive Bayes for unprocessed version.
  SGDC classifier produced better performance than Naive Bayes for un-processed version whereas its performance was lower than Naive Bayes for pre-processed version.

Pre-processed Vs Un-processed-

	Accuracy	F-measure	Precision	Recall
Un-processed	0.5475	0.3467	0.4281	0.3365
Pre-processed	**0.5587**	0.3455	**0.4497**	0.3362

Accuracy - It is ratio of correctly predicted observation to the total observations. If we have high accuracy, then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, we have to look at other parameters to evaluate the performance of your model. Our pre-processed version achieved higher accuracy compared to un processed version.

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate. We achieved higher precision with pre-processed version
Precision = TP/TP+FP

Recall - Recall is the ratio of correctly predicted positive observations to the all observations in actual class. We achieved similar levels of recall (0.336)
Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. We achieved similar levels of F1-measure (0.34)
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Observation- With higher accuracy and precision, pre-processed version of dataset is more suitable for our sentiment analysis and classification. This is because pre-processed version removed unnecessary words**.**

Different size of vocabulary-

We will use pre-processed version of dataset and following classifier for this study:

1.) Naïve Bayes (Baseline)
2.) Random Forest (Best accuracy)
3.) SGDC (Worst accuracy)

Also, we will limit our study on following feature sets:

1.) Unigram(baseline)
2.) Subjectivity + LIWC (best accuracy)
3.) Opinion (accuracy between Unigram and Subjectivity)

Other parameters will remain same i.e. 90/10 split, 5 folds

For Vocabulary size-300

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5473	0.5387	0.5638
SL+ LIWC	0.5545	0.5362	0.5739
Opinion	0.5486	0.5442	0.5680
Average	**0.5528**		

For Vocabulary size-500

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5545	0.5516	0.5707
SL+ LIWC	0.5612	0.5438	0.5768
Opinion	0.5561	0.5503	0.5732
Average	**0.5598**		

For Vocabulary size -1000

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5664	0.5662	0.5799
SL+ LIWC	0.5779	0.5665	0.5854
Opinion	0.5699	0.5718	0.5831
Average	**0.5741**		

For Vocabulary size -2000

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5747	0.5780	0.5845
SL+ LIWC	0.5803	0.5713	**0.5861**
Opinion	0.5790	0.5784	0.5856
Average	**0.5797**		

Observation-Increasing vocabulary size also increases accuracy. We also got our new best accuracy of 0.5861 with SL_LIWC feature set (best feature set as we predicted earlier) and random forest classifier (best classifier as we predicted earlier)

```
Random Forests Classifier
Start Fold 0
0 0.5845
Start Fold 1
1 0.5824
Start Fold 2
2 0.582
Start Fold 3
3 0.5848
Start Fold 4
4 0.5969
Done with cross-validation
mean accuracy- 0.58612
0.376953125
0.25630810092961487
0.4546446596644387
0.3759305210918114
0.6674371084983084
0.7865545555029212
0.4949373979314099
0.43204713932712413
0.4405727923627685
0.30552797087057265
average precision 0.48690901669138514
average recall    0.43127365754440883
F-score   0.4522000466881613
```

Different size of dataset-

Till now we were limiting our dataset size to 30000 phrases for single fold, 50000 phrases for cross validation (5 fold) now we will train and test on entire dataset (1,56,060 phrases) with other parameters (best we found so far) as follows:
Vocabulary size: 2000
Feature sets: SL+LIWC (Best till now), Opinion (in between other two), Unigram(Baseline)
Classifiers: Naïve Bayes(Baseline), SGDC (worst accuracy), Random Forest(Best)
Version: Pre-processed
No. of folds- 5
Split ratio :90/10

For Dataset size- 30,000 phrases

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5413	0.5297	0.5549
SL+ LIWC	0.5502	0.5335	0.5620
Opinion	0.5446	0.5302	0.5595
Average	**0.5451**		

For Dataset size- 50,000 phrases

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5473	0.5387	0.5638
SL+ LIWC	0.5545	0.5362	0.5739
Opinion	0.5486	0.5442	0.5680
Average	**0.5528**		

For Dataset size-1,56,060 phrases

	Naïve Bayes	SGDC	Random Forest
Unigram	0.5886	0.5793	0.6282
SL+ LIWC	0.5944	0.5847	**0.6312**
Opinion	0.5912	0.5837	0.6294
Average		**0.6011**	

Note- I have added screenshot for above experiment (entire dataset part) in corpus/Multifold output folder. Also in order to train entire dataset, I had to run code feature wise by selecting one feature at a time and commenting other features. This avoided memory errors.

Observation- Increasing dataset size also increases accuracy. We also got our new best accuracy of 0.6312 with SL_LIWC feature set (best feature set as we predicted earlier) and random forest classifier (best classifier as we predicted earlier).

```
Random Forests Classifier
Start Fold 0
0 0.629917980264
Start Fold 1
1 0.630526720492
Start Fold 2
2 0.633762655389
Start Fold 3
3 0.631808278867
Start Fold 4
4 0.630462642573
Done with cross-validation
mean accuracy- 0.631295655517
0.44597181137985037
0.3624151583710407
0.5303050310312953
0.44175558244417557
0.7043857921344626
0.8088638134251463
0.554325428701149
0.4820360190724937
0.48396039603960395
0.39821855311753207
average precision 0.5437896918572723
average recall    0.4986578252860777
F-score   0.5174945285678328
```

WEKA Classifier-
We have already covered Sci kit classifier and one extra sentiment lexicon (opinion lexicon) for our advanced level tasks but it would be interesting to try GUI based classifier like WEKA especially on our optimized parameters that we found till now as mentioned below-
Vocabulary size: 2000
Dataset: all phrases (156060 phrases)
Feature sets: SL+LIWC, Opinion, Unigram
Version: Pre-processed

Note-Only SL_LIWCfeaturesets_with_preprocessing, opinionsets_with_preprocessing, and unigramfeaturesets_with_preprocessing has all phrases in csv format and rest all csv files are built for 50,000 phrases.

Experiment 1 - Weka Classifier Vs Others (Cross validation 5 folds)

a.) Weka Classifier Vs NLTK

	NLTK Naïve Bayes	Weka Naïve Bayes
Unigram	0.5886	0.5870
SL+ LIWC	0.5944	0.5792
Opinion	0.5912	0.5821
Average	**0.5914**	**0.5827**

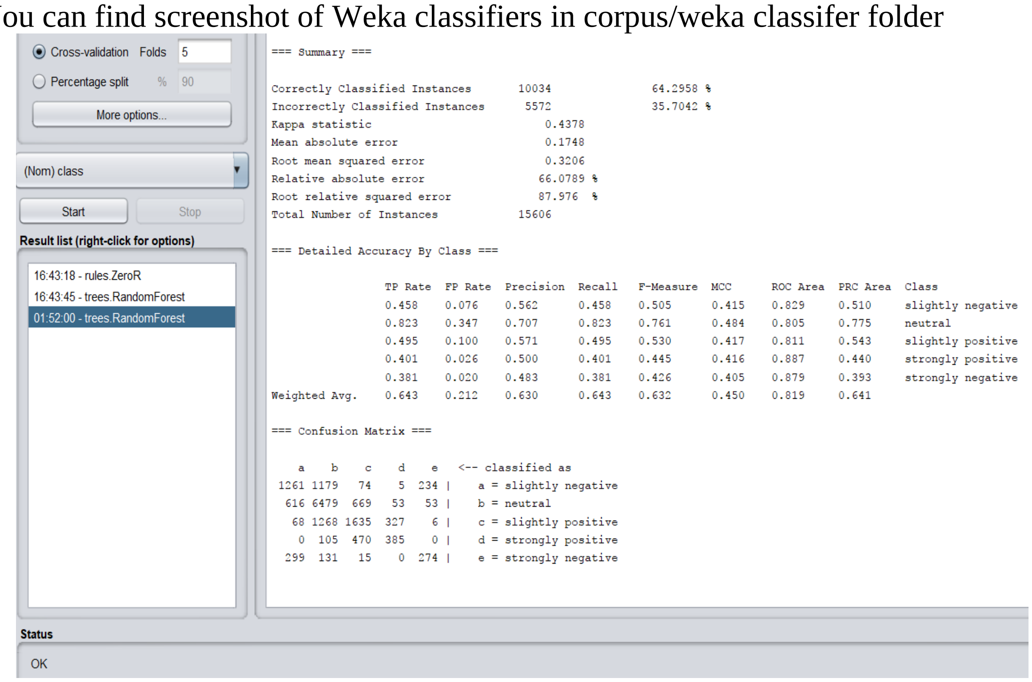
Observation-NLTK Naive Bayes outperformed Weka naive Bayes classifier with higher accuracy

b.) Weka Classifier Vs Sci Kit learn

	Weka Random Forest	Sci Kit Random Forest
Unigram	0.6325	0.6282
SL+ LIWC	**0.6429**	0.6312
Opinion	0.6325	0.6294
Average	**0.6359**	**0.6296**

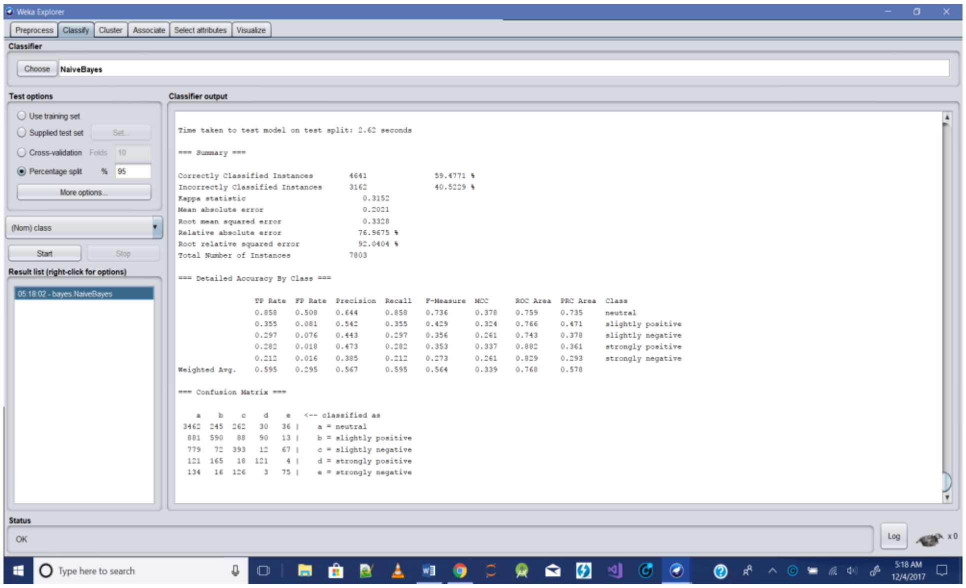
Observation-Weka's Random Forest version outperformed its counterpart-NLTK Random Forest classifier with higher accuracy. **We also obtained our new best accuracy score of 0.6429 again with or combined SL+LIWC feature set and random forest classifier with only difference is being that this time Random Forest version is from Weka**

Note- You can find screenshot of Weka classifiers in corpus/weka classifer folder

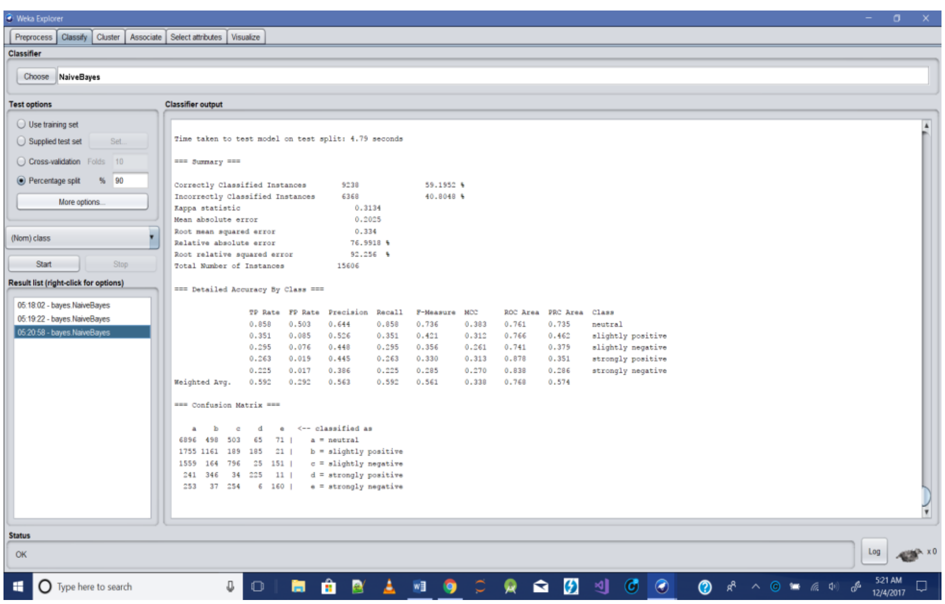


Experiment 2- Varying split ratio
Classifier-Naive Bayes
Feature set-Unigram

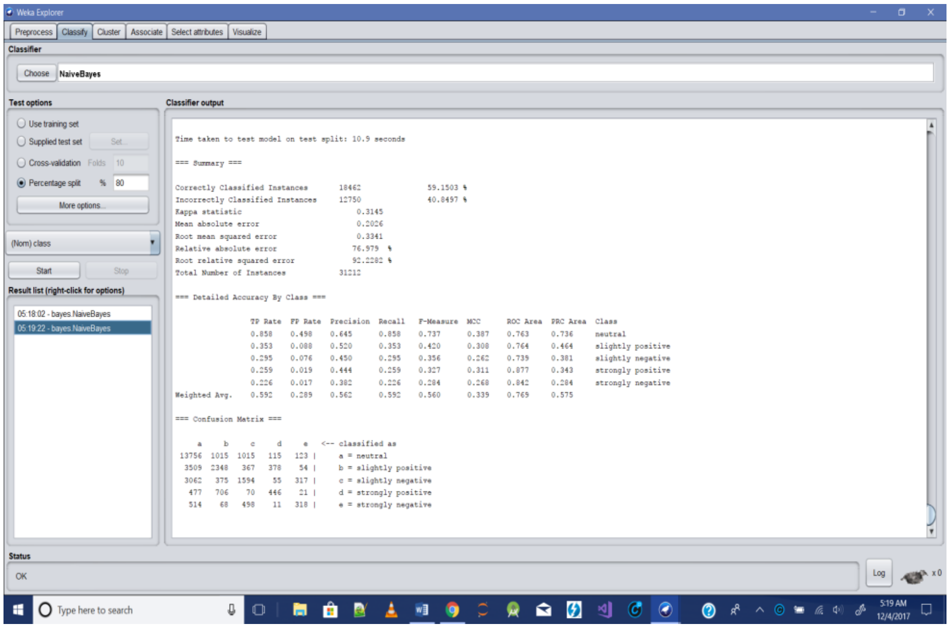
a.)95/5 ratio



b.)90/10



b.)80/20



Observation-Reducing training size of data, reduces accuracy(slightly).

Summary

I obtained **best accuracy of 0.6429 on a scale of 1** with Weka's random forest and SL+LIWC feature set. Although I tried to cover as many experiments as possible but still I had to limit my study to certain parameters because some experiments took 9+ hours to complete as the size of dataset was very big.