SAURABH PATEL Natural Language Processing FINAL PROJECT

Processing and Classification of Sentiment or other Data

Task Dataset:

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

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

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

> **train.tsv** contains the phrases and their associated sentiment labels. We have additionally provided a SentenceId so that you can track which phrases belong to a single sentence.
> **test.tsv** contains just phrases. You must assign a sentiment label to each phrase.

The sentiment labels are:

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

Following are steps for phrase sentimental classification:

1. **Read text from train.tsv file:**

   Read the kaggle training file (train.tsv), loop over lines in the file and use the first limit of them as passed as an argument to processkaggle method, ignore the first line starting with Phrase and only use lines up to the limit, each line has 4 items separated by tabs thus ignore the phrase and sentence ids and keep the phrase and sentence.

   Code:

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

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

  os.chdir(dirPath)

  f = open('./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. **Tokenization:**

In this step, I have done two types of tokenization, for one I have not applied pre-processing on sentence and then created pairs of (tokensOf(sentence), label) list for future classification. In second one I have used few pre-processing steps before start features selection and classification. Following steps are carried out for pre-processing steps.

Lower case:
Converted text to lower case words to remove upper case and lowercase sensitivity.

```
word_list = re.split('\s+', document.lower())
```

Remove punctuation and numbers:
Remove punctuation and numbers from text as that will deviate our result from correct analysis.

```
punctuation = re.compile(r'[-.?!,":;()|0-9]')
word_list = [punctuation.sub("", word) for word in word_list]
```

Removed stop words:
Started with the NLTK english stop word list, but I have removed some of the negationwords, or parts of words. As we know that “not”, “can not” these words are important when you are working on Classification of Sentiment task.

```
if word not in newstopwords:
        final_word_list.append(word)
```

Find attached below screenshot for pre processing document function from python script.

```
58
59  ##############################################################################
60  ###  Pre-processing the documents  ###
61  ##############################################################################
62  def pre_processing_documents(document):
63    # "Pre_processing_documents"
64    # "create list of lower case words"
65    word_list = re.split('\s+', document.lower())
66    # punctuation and numbers to be removed
67    punctuation = re.compile(r'[-.?!/\%@,":;()|0-9]')
68    word_list = [punctuation.sub("", word) for word in word_list]
69    final_word_list = []
70    for word in word_list:
71      if word not in newstopwords:
72        final_word_list.append(word)
73    line = " ".join(final_word_list)
74    return line
```

Possibly filter tokens:

I used two types of tokens list, in one I considered only words from train data which length is more than 3. In one normal case I considered all tokens.

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

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

3. **Features Selection:**

"bag-of-words" features

I used "bag-of-words" features to collect all the words in the corpus and select 300 number of most frequent words to be the word features. I have changed this number to analyze classification. I will discuss more in experiment section.

```
##############################################################
#### Get word-features, bag of words  ####
##############################################################
def get_word_features(wordlist):
  wordlist = nltk.FreqDist(wordlist)
  word_features = [w for (w, c) in wordlist.most_common(300)]
  return word_features
```

Example:

```
['movie', 'story', 'one', 'like', 'manipulative',
'film', 'lrb', 'rrb', 'makes', 'melodrama',
'disparate', 'not', 'found', 'really', 'pleasant',
'consequence', 'feel', 'service', 'script',
'suspense']
```

4. **Features Functions:**

4.1 Unigram features as baseline features

I defined the features for each document. The feature label will be 'contains(keyword)' for each keyword (aka word) in the word features set, and the value of the feature will be Boolean, according to whether the word is contained in that document.

I attached results later in this report.

Example of feature set item without pre-processing:

({'contains(of)': False, 'contains(movies)': False, 'contains(allows)': False, "contains(')": False, 'contains(woman)': False, 'contains(company)': False, 'contains(actorish)': False, 'contains(that)': False, 'contains(deeply)': True, 'contains(on)': False,
……..
……..

'contains(pokepie)': False, 'contains(sit)': False, 'contains(amount)': False}, 3)

Example of feature set item after pre-processing:

({'contains(terrifying)': False, 'contains(actorish)': False, 'contains(movies)': False, 'contains(allows)': False, 'contains(woman)': False, 'contains(stewart)': False, 'contains(company)': False, 'contains(deeply)': True, 'contains(movie)': False, 'contains(introduce)': False,
……..
……..
……..
'contains(tense)': False, 'contains(amount)': False}, 3)

```
# define features (keywords) of a document for a BOW/unigram baseline
# each feature is 'contains(keyword)' and is true or false depending
# on whether that keyword is in the document
# define a feature definition function here
def normal_features(document, word_features):
  document_words = set(document)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  return features


  featuresets = [(normal_features(d, word_features), s) for (d, s) in phrasedocs]
  writeFeatureSets(featuresets,"features_preprocessed.csv")
  print "-------------------------------------------------"
  print "Accuracy with pre-processed features : "
  accuracy_calculation(featuresets)
```

4.2 Sentiment Lexicon: Subjectivity Count features

We will first read in the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh in the MPQA project. Although these words are often used as features themselves or in conjunction with other information, we will create two features that involve counting the positive and negative subjectivity words present in each document. I copy and pasted the definition of the readSubjectivity function from the Subjectivity.py module which is provided by Professor. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength and polarity.

A feature extraction function that has all the word features as before, but also has two features '**positivecount'** and '**negativecount'**. These features contain counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice.

I attached results later in this report.

```
SLpath = "./SentimentLexicons/subjclueslen1-HLTEMNLP05.tff"
SL = readSubjectivity(SLpath)
def SL_features(document, word_features, SL):
  document_words = set(document)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  # count variables for the 4 classes of subjectivity
  weakPos = 0
  strongPos = 0
  weakNeg = 0
  strongNeg = 0
  for word in document_words:
    if word in SL:
      strength, posTag, isStemmed, polarity = SL[word]
      if strength == 'weaksubj' and polarity == 'positive':
        weakPos += 1
      if strength == 'strongsubj' and polarity == 'positive':
        strongPos += 1
      if strength == 'weaksubj' and polarity == 'negative':
        weakNeg += 1
      if strength == 'strongsubj' and polarity == 'negative':
        strongNeg += 1
      features['positivecount'] = weakPos + (2 * strongPos)
      features['negativecount'] = weakNeg + (2 * strongNeg)

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


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

4.3 Negation features

Negation of opinions is an important part of sentimental classification. Here I tried a simple strategy which professor explained in Lab-10. I look for negation words "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t".

For example, my first document has the following words: if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself', One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation.

I followed the first strategy here, and I go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.

Example of feature set item:

({'contains(NOTeasy)': False, 'contains(movies)': False, 'contains(NOTfour)':

False, 'contains(deeply)': True, 'contains(NOTgripping)': False, 'contains(introduce)': False, 'contains(short)': False, 'contains(consequence)': False, 'contains(NOTcalm)': False, 'contains(drama)': ……..
……..
……..
False, 'contains(NOTtill)': False}, 3)

```
###  Negation words
# Negation words "not", "never" and "no"
# Not can appear in contractions of the form "doesn", "'", "t"
## if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself',
# One strategy with negation words is to negate the word following the negation word
#   other strategies negate all words up to the next punctuation
# Strategy is to 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
# Start the feature set with all 2000 word features and 2000 Not word features set to false
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']
def NOT_features(document, word_features, negationwords):
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = False
    features['contains(NOT{})'.format(word)] = False
  # go through document words in order
  for i in range(0, len(document)):
    word = document[i]
    if ((i + 1) < len(document)) and (word in negationwords):
      i += 1
      features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
    else:
      if ((i + 3) < len(document)) and (word.endswith('n') and document[i+1] == "'" and document[i+2] == 't'):
        i += 3
        features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
      else:
        features['contains({})'.format(word)] = (word in word_features)
  return features

NOT_featuresets = [(NOT_features(d, word_features, negationwords), c) for (d, c) in phrasedocs]
writeFeatureSets(SL_featuresets,"features_NOT.csv")
print "-------------------------------------------------"
print "Accuracy with NOT_featuresets : "
accuracy_calculation(NOT_featuresets)
```

4.4 Using Bigram features along with unigram features

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

Example of feature set item:

({'bigram(denying hardscrabble)': False, 'bigram(downsizing rock)': False, 'bigram(earn cheesy)': False, 'contains(seems)': False, 'contains(movies)': False, 'contains(lot)': False, 'contains(genuine)': False, 'bigram(concentrate city)': False, 'contains(deeply)': False,
…..
…….
……..
'bigram(carried bits)': False, 'bigram(earnest core)': False, 'bigram(circles crimeland)': False, 'bigram(clothed excess)': False, 'bigram(city ensemble)': False, 'bigram(credits metaphor)': False, 'bigram(ballistic ecks)': False}, 2)

```
# define features that include words as before
# add the most frequent significant bigrams
# 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
def bigram_document_features(document, word_features,bigram_features):
  document_words = set(document)
  document_bigrams = nltk.bigrams(document)
  features = {}
  for word in word_features:
    features['contains({})'.format(word)] = (word in document_words)
  for bigram in bigram_features:
    features['bigram({} {})'.format(bigram[0], bigram[1])] = (bigram in document_bigrams)
  return features

def get_biagram_features(tokens):
  bigram_measures = nltk.collocations.BigramAssocMeasures()
  finder = BigramCollocationFinder.from_words(tokens,window_size=4)
  finder.apply_freq_filter(6)
  bigram_features = finder.nbest(bigram_measures.chi_sq, 3000)
  return bigram_features[:500]

  bigram_features = get_biagram_features(preprocessedTokens)
  bigram_featuresets = [(bigram_document_features(d, word_features,bigram_features), c) for (d, c) in phrasedocs]
  writeFeatureSets(bigram_featuresets,"features_biagram.csv")
  print "-------------------------------------------------"
  print "Accuracy with bigram featuresets : "
  accuracy_calculation(bigram_featuresets)
```

4.5 POS tag features

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

```
###  POS tag counts
# this function takes a document list of words and returns a feature dictionay
#   it depends on the variable word_features
# it runs the default pos tagger (the Stanford tagger) on the document
#   and counts 4 types of pos tags to use as features
def POS_features(document, word_features):
    document_words = set(document)
    tagged_words = nltk.pos_tag(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    numNoun = 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words:
        if tag.startswith('N'): numNoun += 1
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['nouns'] = numNoun
    features['verbs'] = numVerb
    features['adjectives'] = numAdj
    features['adverbs'] = numAdverb
    return features

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

5. NLTK Naive Bayes classifier:

I used NLTK Naïve Bayes classifier to train and test data. Initially taken 90 % of data as training set and 10% as test set. Printed also confusion matrix with it. We start by looking at the confusion matrix, which shows the results of a test for how many of the actual class labels (the gold standard labels) match with the predicted labels.

Code snapshot:

```
##############################################################################
####  Features   accuracy calculation####
##############################################################################
def accuracy_calculation(featuresets):
  print "_____________________________________________________"
  print "Training and testing a classifier "
  training_size = int(0.1*len(featuresets))
  test_set = featuresets[:training_size]
  training_set = featuresets[training_size:]
  classifier = nltk.NaiveBayesClassifier.train(training_set)
  print "Accuracy of classifier :"
  print nltk.classify.accuracy(classifier, test_set)
  print "_____________________________________________________"
  print "Showing most informative features"
  print classifier.show_most_informative_features(30)
  print "_____________________________________________________"
  print "precision, recall and F-measure scores"
  print_confusionmatrix(classifier,test_set)
  print ""

##############################################################################
## print ConfusionMatrix. ##
##############################################################################
def print_confusionmatrix(classifier_type, test_set):
  reflist = []
  testlist = []
  for (features, label) in test_set:
    reflist.append(label)
    testlist.append(classifier_type.classify(features))

  print " "
  print "The confusion matrix"
  cm = ConfusionMatrix(reflist, testlist)
  print cm
```

Sample Output, with confusion matrix :

Accuracy with normal features, without pre-processing steps :
Training and testing a classifier
Accuracy of classifier :
0.51

The confusion matrix
```
  |  0   1   2   3   4 |
--+--------------------+
0 | <.> 1   5   2   . |
1 |  3 <8>22   3   1 |
2 |  2   8<83> 8   1 |
3 |  4   4  17 <9> 1 |
4 |  2   4   5   5 <2>|
```

```
--+----------------+
(row = reference; col = test)


--------------------------------------------------
Accuracy with pre-processed features :
Training and testing a classifier
Accuracy of classifier :
0.545
--------------------------------------------------

The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  6  2  . |
1 |  1 <7>25  4  . |
2 |  .  4<93> 4  1 |
3 |  1  7 17 <8> 2 |
4 |  1  2  7  7 <1>|
--+----------------+
(row = reference; col = test)


--------------------------------------------------
Accuracy with SL_featuresets :
Training and testing a classifier
Accuracy of classifier :
0.55
--------------------------------------------------

The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  6  2  . |
1 |  1 <9>23  4  . |
2 |  .  5<89> 7  1 |
3 |  1  6 15<11> 2 |
4 |  1  1  7  8 <1>|
--+----------------+
(row = reference; col = test)


--------------------------------------------------
Accuracy with NOT_featuresets :
Training and testing a classifier
Accuracy of classifier :
```

```
0.51
--------------------------------------------------

The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  5  3  . |
1 |  5 <7>22  3  . |
2 |  2 10<79> 8  3 |
3 |  4  7 12<12> . |
4 |  2  1  4  7 <4>|
--+----------------+
(row = reference; col = test)


--------------------------------------------------
Accuracy with bigram featuresets :
Training and testing a classifier
Accuracy of classifier :
0.545
--------------------------------------------------

The confusion matrix
  |  0  1  2  3  4 |
--+----------------+
0 | <.> .  6  2  . |
1 |  1 <7>25  4  . |
2 |  .  4<93> 4  1 |
3 |  1  7 17 <8> 2 |
4 |  1  2  7  7 <1>|
--+----------------+
(row = reference; col = test)
```

6. Most informative features:

Displayed top 30 most informative features by using **"show_most_informative_features()"** function

Code snapshot:

```
print "Showing most informative features"
print classifier.show_most_informative_features(30)
```

Example from any one of the build classifier:

```
Showing most informative features
Most Informative Features
        contains(good) = True              4 : 2      =     20.0 : 1.0
      contains(script) = True              0 : 2      =     17.8 : 1.0
         contains(can) = True              0 : 2      =     17.8 : 1.0
        contains(make) = True              4 : 2      =     13.8 : 1.0
       contains(whole) = True              4 : 2      =     12.8 : 1.0
      contains(actors) = True              4 : 2      =     12.8 : 1.0
        contains(well) = True              4 : 2      =     12.8 : 1.0
      contains(cinema) = True              4 : 2      =     12.8 : 1.0
        contains(gets) = True              4 : 2      =     12.8 : 1.0
         contains(man) = True              4 : 2      =     12.8 : 1.0
      contains(brings) = True              4 : 2      =     12.8 : 1.0
contains(entertaining) = True              0 : 2      =     10.7 : 1.0
        contains(back) = True              0 : 2      =     10.7 : 1.0
       contains(great) = True              0 : 2      =     10.7 : 1.0
       contains(thing) = True              0 : 2      =     10.7 : 1.0
        contains(rise) = True              0 : 2      =     10.7 : 1.0
     contains(neither) = True              0 : 2      =     10.7 : 1.0
       contains(story) = True              0 : 2      =     10.7 : 1.0
       contains(three) = True              0 : 2      =     10.7 : 1.0
        contains(know) = True              0 : 2      =     10.7 : 1.0
       contains(seems) = True              0 : 2      =     10.7 : 1.0
    contains(material) = True              0 : 2      =     10.7 : 1.0
      contains(series) = True              0 : 2      =     10.7 : 1.0
         contains(guy) = True              0 : 2      =     10.7 : 1.0
       contains(place) = True              0 : 2      =     10.7 : 1.0
      contains(school) = True              0 : 2      =     10.7 : 1.0
       contains(women) = True              0 : 2      =     10.7 : 1.0
      contains(nature) = True              0 : 2      =     10.7 : 1.0
         contains(end) = True              0 : 3      =     10.3 : 1.0
     contains(nothing) = True              1 : 2      =      8.0 : 1.0
```

7. **Write csv features file for testing in weka or sklearn:**

   I have created csv file and use this file in Weka or Sci-Kit Learn to train and test a classifier, using cross-validation scores from those packages. I have created files for every features and will test in Weka or Sci-kit Learn program which is provided by Professor.

I created following files, for every feature sets. I created with 1000,2000,10000 and for whole training dataset. I have not attached those features csv files in project zip folder as it size get increased.

features_biagram.csv
features_normal.csv
features_NOT.csv
features_preprocessed.csv
features_SL.csv

8. Experiments:

1. Comparison before and after applying Filter by stopwords or other pre-processing methods

Number of records	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	SL Features	Not Features	Bigram Features	POS Tag Features (1000 Records)
(1000 Records) Accuracy	0.48	0.55	0.56	0.5	0.55	0.54
(10000 Records) Accuracy	0.504	0.535	0.537	0.46	0.536	NA
(156060 Records) Accuracy	0.524093297	0.542547738	0.551967192	0.558951685	0.542547738	NA

Observation:

Applying pre-processing and stopwords improved accuracy in all classifier. This is because pre processing and stopwords filter removed unnecessary words from classifier. We can see that after applying pre-processing and stopwords unigram features and bigram features give good accuracy.

2. Comparison on different sizes of vocabularies

Number of Vocabulary	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	SL Features	Not Features	Bigram Features	POS Tag Features (1000 Records)
300	0.48	0.55	0.56	0.5	0.55	NA
500	0.51	0.545	0.55	0.51	0.545	NA
1000	0.539	0.551	0.556	0.498	0.551	NA
2000	0.572	0.556	0.575	0.556	0.585	NA

Observation:

Increase in size of vocabularies show increase in accuracy. This is because of bigger feature sets to classify data.

3. Using Ski-Learn python script, run on features.csv file for comparison using cross-validation, Precision, Recall and F- measures score in all three classifier

Number of records	Before filter and preprocessing	After filter and preprocessing(Unigram Features)	SL Features	Not Features	Bigram Features
Precision score	0.53	0.51	0.55	0.55	0.51
Recall score	0.51	0.49	0.52	0.52	0.49
F-measure score	0.52	0.5	0.53	0.53	0.5

SL features is performance better in classification other features functions because of few words are unseen in train data as features. Those words or tokens covers on Lexicon dictionary. Plus, we observed recall score lesser than F-measure which is less than Precision.

9. Experiments output screenshots: I have done following experiment on these csv file.

1. LogisticRegression classifier on 10000 records features csv file. Here features are created using bigram features. I used here 10 folds cross validation.

	precision	recall	f1-score	support
neg	0.23	0.39	0.29	497
neu	0.68	0.63	0.65	5100
pos	0.24	0.42	0.31	557
sneg	0.32	0.33	0.32	1725
spos	0.42	0.33	0.37	2121
avg / total	0.51	0.49	0.50	10000

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	194	106	29	124	44	497
neu	285	3208	266	760	581	5100
pos	21	81	235	46	174	557
sneg	244	636	100	564	181	1725
spos	103	706	349	256	707	2121
All	847	4737	979	1750	1687	10000

2. LogisticRegression classifier on 10000 records features csv file. Here features are created using normal bag of words (unigram) features where pre processing has not done. I used here 10 folds cross validation.

	precision	recall	f1-score	support
neg	0.24	0.40	0.30	497
neu	0.69	0.66	0.68	5100
pos	0.25	0.43	0.32	557
sneg	0.35	0.33	0.34	1725
spos	0.41	0.34	0.37	2121
avg / total	0.53	0.51	0.52	10000

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	201	92	39	124	41	497
neu	238	3381	229	650	602	5100
pos	31	77	240	38	171	557
sneg	248	637	80	566	194	1725
spos	113	695	360	238	715	2121
All	831	4882	948	1616	1723	10000

3. LogisticRegression classifier on 10000 records features csv file. Here features are created using NOT features. I used here 10 folds cross validation.

	precision	recall	f1-score	support
neg	0.24	0.42	0.31	497
neu	0.72	0.66	0.69	5100
pos	0.26	0.46	0.33	557
sneg	0.36	0.36	0.36	1725
spos	0.45	0.37	0.41	2121
avg / total	0.55	0.52	0.53	10000

Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	211	73	26	147	40	497
neu	262	3342	213	697	586	5100
pos	22	51	254	45	185	557
sneg	289	560	87	614	175	1725
spos	101	627	382	216	795	2121
All	885	4653	962	1719	1781	10000

4. LogisticRegression classifier on 10000 records features csv file. Here features are created using features as like bag of words but here tokens are pre processed. I used here 10 folds cross validation.

	precision	recall	f1-score	support
neg	0.23	0.39	0.29	497
neu	0.68	0.63	0.65	5100
pos	0.24	0.42	0.31	557
sneg	0.32	0.33	0.32	1725
spos	0.42	0.33	0.37	2121
avg / total	0.51	0.49	0.50	10000

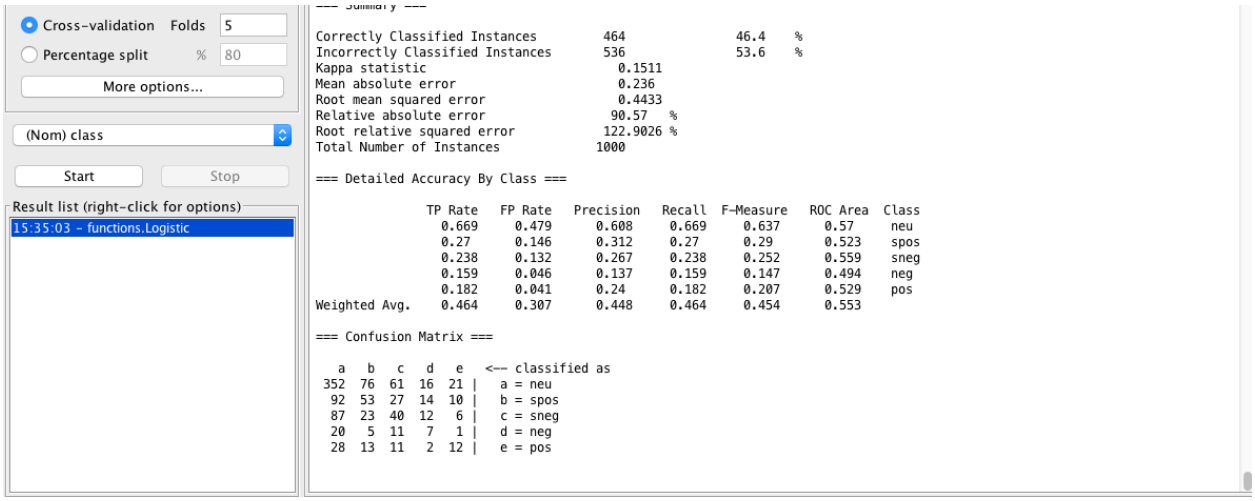
Predicted	neg	neu	pos	sneg	spos	All
Actual						
neg	194	106	29	125	43	497
neu	285	3209	266	760	580	5100
pos	21	81	235	46	174	557
sneg	244	636	100	564	181	1725
spos	104	704	349	256	708	2121
All	848	4736	979	1751	1686	10000

5. LogisticRegression classifier on 10000 records features csv file. Here features are created using SL. I used here 10 folds cross validation.

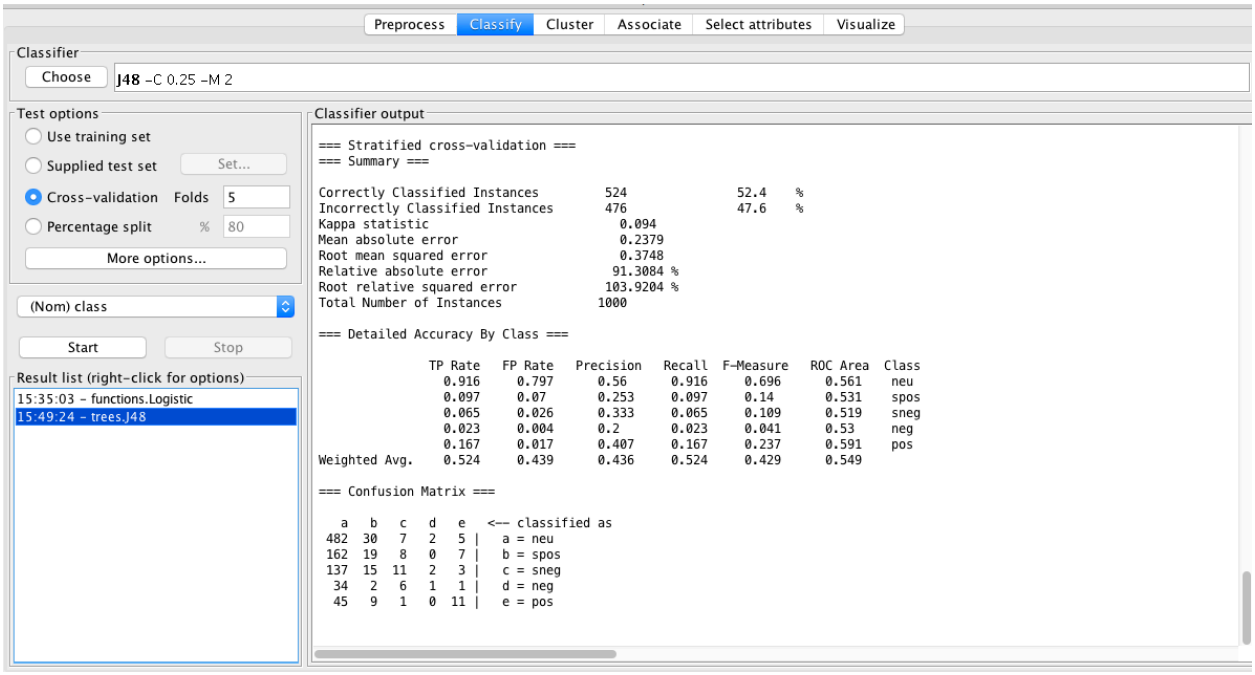
	precision	recall	f1-score	support
neg	0.24	0.42	0.31	497
neu	0.72	0.66	0.69	5100
pos	0.26	0.46	0.33	557
sneg	0.36	0.36	0.36	1725
spos	0.45	0.37	0.41	2121
avg / total	0.55	0.52	0.53	10000

Predicted Actual	neg	neu	pos	sneg	spos	All
neg	211	73	26	147	40	497
neu	262	3342	213	697	586	5100
pos	22	51	254	45	185	557
sneg	289	560	87	614	175	1725
spos	101	627	382	216	795	2121
All	885	4653	962	1719	1781	10000

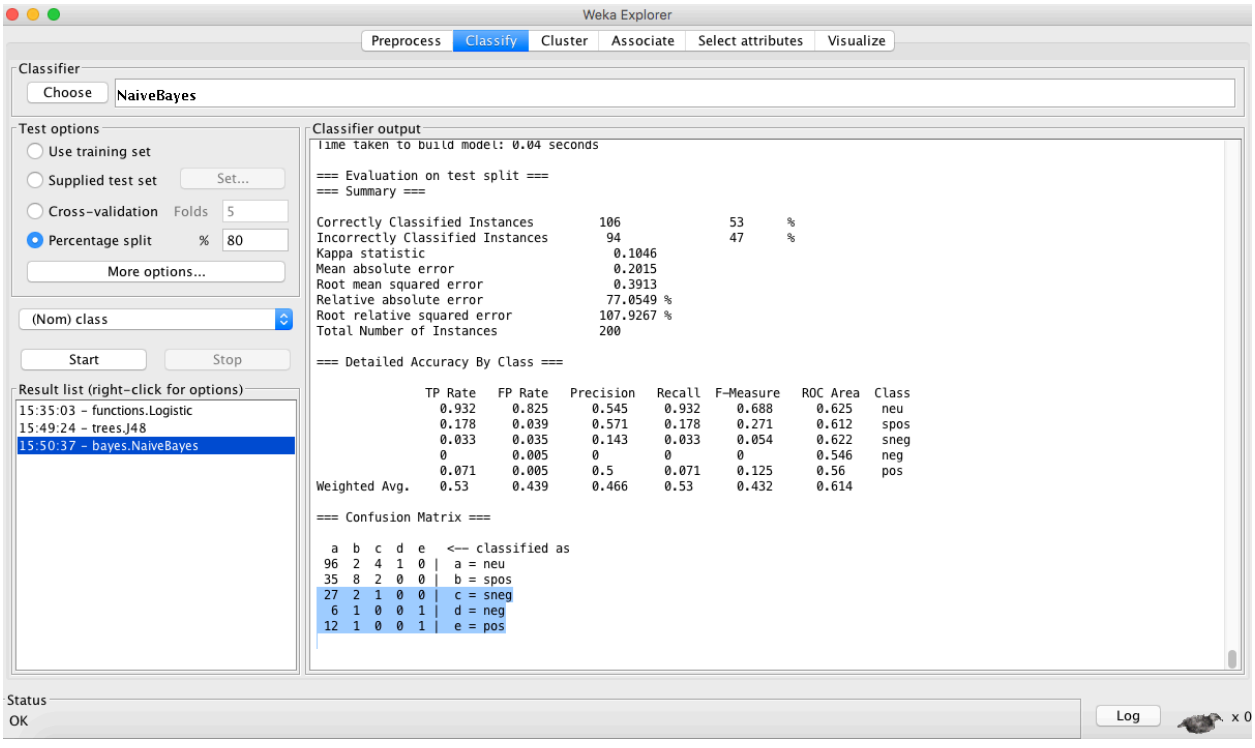
6. Logistic classifier with cross validation of 5 folds.



7. J48 Decision tree classifier and 5 folds' cross validation.



8. Naive Bayes classifier and use percentage split of 80%.



10. Advanced Task:

Train the classifier on the entire training set and test it on a separately available test set. Submitted submission file on Kaggle website to get accuracy and place on leaderboard.

Code Screenshot:

```
# generate test csv file.
f = open('./test.tsv', 'r')
# loop over lines in the file and use the first limit of them
testphrasedata = []
for line in f:
  # ignore the first line starting with Phrase and read all lines
  if (not line.startswith('Phrase')):
    # remove final end of line character
    line = line.strip()
    # each line has 4 items separated by tabs
    # ignore the phrase and sentence ids, and keep the phrase and sentiment
    tempList = []
    if len(line.split('\t')) == 3:
      tempList.append(line.split('\t')[0])
      tempList.append(line.split('\t')[2])
      testphrasedata.append(tempList)
    else :
      tempList.append(line.split('\t')[0])
      tempList.append("")
      testphrasedata.append(tempList)

phraselist = testphrasedata
print('Read', len(testphrasedata), 'phrases, using', len(phraselist), 'test phrases')

#for phrase in phraselist[:10]:
  #print (phrase)

# create list of phrase documents as (list of words)
phrasedocs = []
# add all the phrases
for id,phrase in phraselist:
  # with pre processing
  tokenizer = RegexpTokenizer(r'\w+')
  phrase = pre_processing_documents(phrase)
  tokens = tokenizer.tokenize(phrase)
  phrasedocs.append((id,tokens))

preprocessedTestTokens = get_words_from_test(phrasedocs)
test_word_features = get_word_features(preprocessedTestTokens)

test_featuresets = [(normal_features(d, test_word_features), id) for (id, d) in phrasedocs]
create_test_submission(featuresets,test_featuresets,"sample.csv")
```

Submission.csv creation:

```
##############################################################################################
####  create test submission file, predict label using classifier ####
##############################################################################################
def create_test_submission(featuresets,test_featuresets,fileName):
  print "-------------------------------------------------"
  print "Training and testing a classifier "
  test_set = test_featuresets
  training_set = featuresets
  classifier = nltk.NaiveBayesClassifier.train(training_set)

  fw = open(fileName,"w")
  fw.write("PhraseId"+','+"Sentiment"+'\n')
  for test,id in test_featuresets:
    fw.write(str(id)+','+str(classifier.classify(test))+'\n')
  fw.close()
```

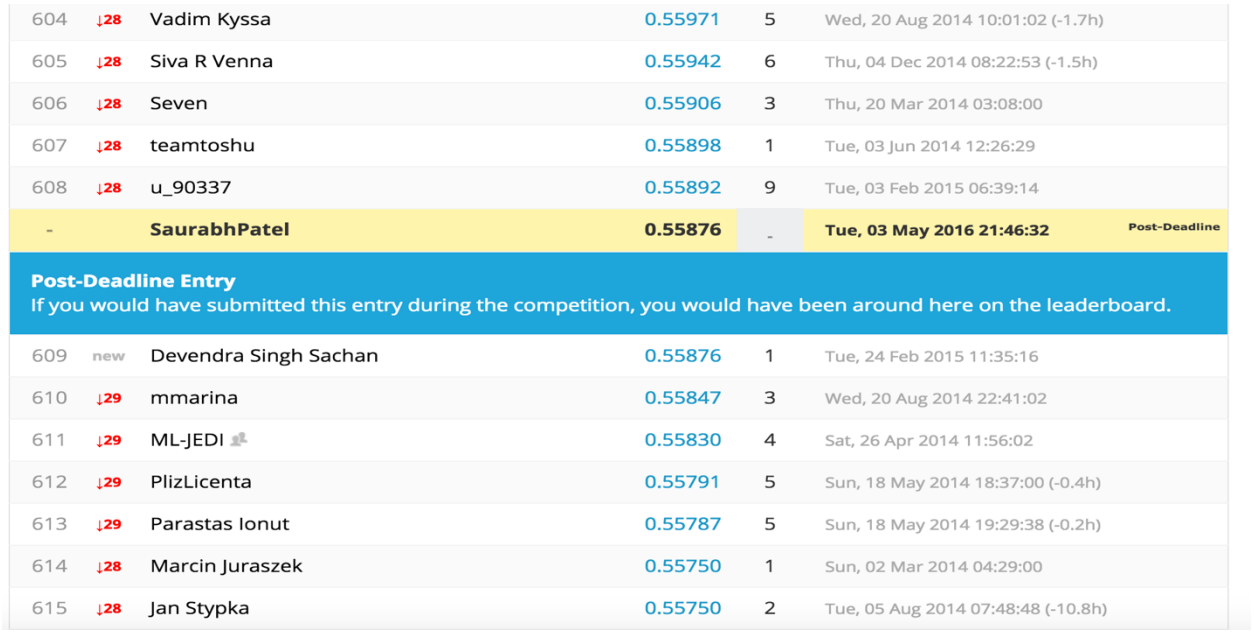
Program Run:

```
Saurabhs-MacBook-Pro:kagglemoviereviews saurabhpatel$ python classifyKaggle.py corpus/ 156060
('Read', 156060, 'phrases, using', 156060, 'random phrases')
-------------------------------------------------
Accuracy with pre-processed features :
Training and testing a classifier
Accuracy of classifier :
0.559400230681
-------------------------------------------------

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


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

Results: Got 0.55876 Accuracy, please find attached screenshot from the Kaggle website.



604	↓28	Vadim Kyssa	0.55971	5	Wed, 20 Aug 2014 10:01:02 (-1.7h)
605	↓28	Siva R Venna	0.55942	6	Thu, 04 Dec 2014 08:22:53 (-1.5h)
606	↓28	Seven	0.55906	3	Thu, 20 Mar 2014 03:08:00
607	↓28	teamtoshu	0.55898	1	Tue, 03 Jun 2014 12:26:29
608	↓28	u_90337	0.55892	9	Tue, 03 Feb 2015 06:39:14
-		**SaurabhPatel**	**0.55876**	-	**Tue, 03 May 2016 21:46:32** Post-Deadline

Post-Deadline Entry
If you would have submitted this entry during the competition, you would have been around here on the leaderboard.

609	new	Devendra Singh Sachan	0.55876	1	Tue, 24 Feb 2015 11:35:16
610	↓29	mmarina	0.55847	3	Wed, 20 Aug 2014 22:41:02
611	↓29	ML-JEDI	0.55830	4	Sat, 26 Apr 2014 11:56:02
612	↓29	PlizLicenta	0.55791	5	Sun, 18 May 2014 18:37:00 (-0.4h)
613	↓29	Parastas Ionut	0.55787	5	Sun, 18 May 2014 19:29:38 (-0.2h)
614	↓28	Marcin Juraszek	0.55750	1	Sun, 02 Mar 2014 04:29:00
615	↓28	Jan Stypka	0.55750	2	Tue, 05 Aug 2014 07:48:48 (-10.8h)

References:

1. https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews
2. NLP Class Lab Exercises.
3. https://en.wikipedia.org/wiki/Weka_(machine_learning)
4. http://scikit-learn.org/stable/