NLP Lab Session

Week 10

**Sentiment Analysis in NLTK and Python**

**Getting Started**

For this lab session, download the examples and put them in your class folder for copy/pasting examples.

LabWeek10.sentimentfeatures.sents.txt

Subjectivity.py

subjclueslen1-HLTEMNLP05.tff.zip

Unzip the subjclues file and remember the location and start your Python interpreter session.

>>> import nltk

The goal of this week’s lab is to show how to conduct a sentiment analysis in the NLTK.

**SentiWordNet, a sentiment lexicon in NLTK**

Let’s look at SentiWordNet, a sentiment lexicon in NLTK. [here shows the NLTK resources http://www.nltk.org/howto/corpus.html] The documentation for SentiWordNet is on this HowTo page: http://www.nltk.org/howto/sentiwordnet.html

In this sentiment lexicon, each word is judged to be made up of partly positive, negative and objective meaning, and 3 scores are given as to how much of each, where the scores must sum to 1.

>>>from nltk.corpus import wordnet as wn

>>> from nltk.corpus import sentiwordnet as swn

Note that there are not very many functions listed, but the trick is that the synsets in SentiWordNet are the same as in WordNet, so we can use the functions to find the synonyms, definitions and examples in wordnet.

>>> list(swn.senti\_synsets('breakdown'))

>>> wn.synsets('breakdown')

Following the example in the HowTo page, we look at the third sense of the word breakdown

>>> breakdown3 = swn.senti\_synset('breakdown.n.03')

The print function for senti\_synsets will give the positive and negative scores.

>>> print (breakdown3)

There are functions to access these two scores separately, and a third function to access the objective scores.

>>> breakdown3.pos\_score()

>>> breakdown3.neg\_score()

>>> breakdown3.obj\_score()

Now let’s look at this term *swn.senti\_synset('breakdown.n.03')* closely. In this naming convention *‘breakdown.n.03’,* the first item is the word we are interested in, the second item *n* stands for noun, and the third item shows that this synset is the third one in the synset list for *breakdow*n synsets. Below is a list that shows how different word classes are represented in this naming convention.

n - NOUN

v - VERB

a – ADJECTIVE

s – ADJECTIVE SATELLITE

r - ADVERB

You can find some discussions about the meaning of “Adjective Satellite” (also called “Adjective Cluster”): <http://stackoverflow.com/questions/18817396/what-part-of-speech-does-s-stand-for-in-wordnet-synsets>.

Recall that you can find here a list of POS tags for the default tagger of nltk.pos\_tag(text) (text is a list of tokens) <http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html>

In Python environment, you can also use the following command to find the tagset.

>>>nltk.help.upenn\_tagset()

**Exercise:**

Use SentiWordNet to get the senti\_synset of the sense of a word that you pick. Show the positive, negative and objective sentiment scores for that word, if any.

**Identify Positive and Negative Words with SentiWordNet**

Now let’s see how we can use SentiWordNet to identify the positive and negative words from a given text. Here is the input text:

>>>text = ‘What a beautiful but horrible person.’

We first create the tokens of this text and then tag the tokens with the POS tagger provided by NLTK.

>>> text\_tokens = nltk.word\_tokenize(text)

>>> text\_tokens

>>> text\_pos = nltk.pos\_tag(text\_tokens)

>>> text\_pos

[Note: what will happen if you use split() function instead of word\_tokenize()?]

Next, we want to examine the sentiment score of each token based on SentiWordNet. To do so, we need to first know the correct naming convention for a token. We will always use the first synset from the synset list of the token. We will use the POS tag information to figure out the second item in the naming convention.

>>> text\_positive = []

>>> text\_negative = []

>>> for i in range(0, len(text\_pos)):

... word = text\_pos[i][0]

... tag = text\_pos[i][1]

... condition = 0

... if (tag == 'NN' or tag == 'NNS'): condition = 1

... elif (tag == 'VB' or tag == 'VBD' or tag == 'VBN' or tag == 'VBP' or tag == 'VBZ'): condition = 2

... elif (tag == 'JJ' or tag == 'JJR' or tag == 'JJS'): condition = 3

... elif (tag == 'RB' or tag == 'RBR' or tag == 'RBS'): condition = 4

... print (word + " " + tag + " " + str(condition))

Now with the above approach, we can identify the naming convention for each token and use this to obtain the synset.

>>>for i in range(0, len(text\_pos)):

word = text\_pos[i][0]

tag = text\_pos[i][1]

synlist = []

if (tag == 'NN' or tag == 'NNS'):

synlist = list(swn.senti\_synsets(word, 'n'))

elif (tag == 'VB' or tag == 'VBD' or tag == 'VBN' or tag == 'VBP' or tag == 'VBZ'):

synlist = list(swn.senti\_synsets(word, 'v'))

elif (tag == 'JJ' or tag == 'JJR' or tag == 'JJS'):

synlist = list(swn.senti\_synsets(word, 'a'))

elif (tag == 'RB' or tag == 'RBR' or tag == 'RBS'):

synlist = list(swn.senti\_synsets(word, 'r'))

if (len(synlist) > 0):

pscore = synlist[0].pos\_score()

nscore = synlist[0].neg\_score()

oscore = synlist[0].obj\_score()

if (pscore > nscore and pscore > oscore):

text\_positive.append(word)

elif (nscore > pscore and nscore > oscore):

text\_negative.append(word)

>>>text\_positive

>>>text\_negative

While this approach seems to be straightforward, we usually don’t use it in sentiment analysis. One problem is that if the POS tagger incorrectly tagged a word then we will have a problem of identifying its sentiment.

[Note: if you are interested, you can use this approach to find the sentiment words in this input: “She is so lovely.”]

**Sentiment Classification – Words as Features**

Now let’s look at two ways to add features that are sometimes used in various sentiment or opinion classification problems. We will illustrate the process on a corpus of sentences from the Movie Review corpus, where each sentence has been labeled with positive or negative sentences.

We start by loading the sentence\_polarity corpus and creating a list of documents where each document represents a single sentence with the words and its label.

>>> from nltk.corpus import sentence\_polarity

>>> import random

Look at sentences from the entire list of sentences.

>>> sentences = sentence\_polarity.sents()

>>> len(sentences)

>>> type(sentences)

>>> sentence\_polarity.categories()

The movie review sentences are not labeled individually, but can be retrieved by category. We first create the list of documents where each document(sentence) is paired with its label.

>>> documents = [(sent, cat) for cat in sentence\_polarity.categories()

for sent in sentence\_polarity.sents(categories=cat)]

In this list, each item is a pair (sent,cat) where sent is a list of words from a movie review sentence and cat is its label, either ‘pos’ or ‘neg’.

>>> documents[0]

>>> documents[-1]

(Note: negative index in Python means you count from the end of the list, so [-1] is the last element in the list)

Since the documents are in order by label, we mix them up for later separation into training and test sets.

>>> random.shuffle(documents)

We need to define the set of words that will be used for features. This is essentially all the words in the entire document collection, except that we will limit it to the 2000 most frequent words. Note that we lowercase the words, but do not do stemming or remove stopwords.

>>> all\_words\_list = [word for (sent,cat) in documents for word in sent]

>>> all\_words = nltk.FreqDist(all\_words\_list)

>>> word\_items = all\_words.most\_common(2000)

>>> word\_features = [word for (word, freq) in word\_items]

Now we can define the features for each document, using just the words, sometimes called the BOW or unigram features. 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.

>>> def document\_features(document, word\_features):

document\_words = set(document)

features = {}

for word in word\_features:

features['contains({})'.format(word)] = (word in document\_words)

return features

Define the feature sets for the documents.

>>> featuresets = [(document\_features(d,word\_features), c) for (d,c) in documents]

We create the training and test sets, train a Naïve Bayes classifier, and look at the accuracy, and this time we’ll do a 90/10 split of our approximately 10,000 documents.

>>> train\_set, test\_set = featuresets[1000:], featuresets[:1000]

>>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>>> print (nltk.classify.accuracy(classifier, test\_set))

The function show\_most\_informative\_features shows the top ranked features according to the ratio of one label to the other one. For example, if there are 20 times as many positive documents containing this word as negative ones, then the ratio will be reported as 20.00: 1.00 pos:neg.

>>> classifier.show\_most\_informative\_features(30)

**Sentiment Classification - Subjectivity Count features**

Let’s look at another way of adding features for sentiment classification. 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.

Copy and paste the definition of the readSubjectivity function from the Subjectivity.py module. We’ll look at the function to see how it reads the file into a dictionary.

Create a path variable to where you stored the subjectivity lexicon file. If you started your python in the same directory, you can just type

>>>SLpath = 'subjclueslen1-HLTEMNLP05.tff'

Now run the function that reads the file. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength, POStag, whether it is stemmed and the polarity. (See more details in the Subjectivity.py file.)

>>> SL = readSubjectivity(SLpath)

Now the variable SL (for Subjectivity Lexicon) is a dictionary where you can look up words and find the strength, POS tag, whether it is stemmed and polarity. We can try out some words.

>>> SL['absolute']

>>> SL['shabby']

Or we can use the Python multiple assignment to get the 4 items:

>>> strength, posTag, isStemmed, polarity = SL['absolute']

Now we create a feature extraction function that has all the word features as before, but also has two features ‘positivecount’ and ‘negativecount’. These features contains 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. Note that this is only one of the ways in which people count up the presence of positive, negative and neutral words in a document.

>>> def SL\_features(document, word\_features, SL):

document\_words = set(document)

features = {}

for word in word\_features:

features['contains(%s)' % 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)

return features

Now we create feature sets as before, but using this feature extraction function.

>>> SL\_featuresets = [(SL\_features(d, word\_features, SL), c) for (d,c) in documents]

# features in document 0

>>> SL\_featuresets[0][0]['positivecount']

>>> SL\_featuresets[0][0]['negativecount']

>>> SL\_featuresets[0][1]

>>> SL\_features[0][0]['contains(bother)']

>>> train\_set, test\_set = SL\_featuresets[1000:], SL\_featuresets[:1000]

>>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>>> print nltk.classify.accuracy(classifier, test\_set)

In my random training, test split, these particular sentiment features did improve the classification on this dataset. But also note that there are several different ways to represent features for a sentiment lexicon, e.g. instead of counting the sentiment words, we could get one overall score by subtracting the number of negative words from positive words, or other ways to score the sentiment words. Also, note that there are many different sentiment lexicons to try.

**Negation features**

Negation of opinions is an important part of opinion classification. Here we try a simple strategy. We 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.

We follow the first strategy here, and we 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.

Here is one list of negation words, including some adverbs called “approximate negators”:

no, not, never, none, rather, hardly, scarcely, rarely, seldom, neither, nor,

couldn't, wasn't, didn't, wouldn't, shouldn't, weren't, don't, doesn't, haven't, hasn't, won't, hadn't

The form of some of the words is a verb followed by n’t. Now in the Movie Review Corpus itself, the tokenization has these words all split into 3 words, e.g. “couldn”, “’”, and “t”. (and I have a NOT\_features definition for this case). But in this sentence\_polarity corpus, the tokenization keeps these forms of negation as one word ending in “n’t”.

>>> for sent in list(sentences)[:50]:

... for word in sent:

... if (word.endswith("n't")):

... print(sent)

>>> negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor']

Start the feature set with all 2000 word features and 2000 Not word features set to false. If a negation occurs, add the following word as a Not word feature (if it’s in the top 2000 feature words), and otherwise add it as a regular feature word.

>>> 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) or (word.endswith("n't"))):

i += 1

features['contains(NOT{})'.format(document[i])] = (document[i] in word\_features)

else:

features['contains({})'.format(word)] = (word in word\_features)

return features

Create feature sets as before, using the NOT\_features extraction funtion, train the classifier and test the accuracy.

>>> NOT\_featuresets = [(NOT\_features(d, word\_features, negationwords), c) for (d, c) in documents]

>>> NOT\_featuresets[0][0]['contains(NOTlike)']

>>> NOT\_featuresets[0][0]['contains(always)']

>>> train\_set, test\_set = NOT\_featuresets[200:], NOT\_featuresets[:200]

>>> classifier = nltk.NaiveBayesClassifier.train(train\_set)

>>> print nltk.classify.accuracy(classifier, test\_set)

>>> classifier.show\_most\_informative\_features(20)

In my random split, using the negation features did improve the classification.

Now let’s use this classifier to classify the polarity of a new sentence.

>>>text = 'I had a good day at the office'

>>>texttokens = nltk.word\_tokenize(text)

>>>inputfeatureset = NOT\_features(texttokens, word\_features, negationwords)

>>>classifier.classify(inputfeatureset)

**Sentiment Classification - Other features**

There are other types of possible features. For example, sometimes people use bigrams in addition to just words/unigrams or use the words together with a POS tagger. Also, there are many other forms of negation features.

For some problems, the word features can be pruned with a stop word list, but care should be taken that the list doesn’t remove any negation or useful function words. A very small stop word list is probably better than a large one.

**Exercise**

Let’s try using a stopword list to prune the word features. We’ll start with the NLTK stop word list, but we’ll remove some of the negation words, or parts of words, that our negation filter uses. This list is still pretty large.

>>> stopwords = nltk.corpus.stopwords.words('english')

>>> len(stopwords)

>>> stopwords

>>> newstopwords = [word for word in stopwords if word not in negationwords]

>>> len(newstopwords)

>>> newstopwords

Now take the new stop words out of the collection of all words and then take the top 2000 to be the word features.

>>> new\_all\_words\_list = [word for word in all\_words\_list if word not in newstopwords]

Now continue to get new word features of length 2000 after the stopwords are removed:

>>> new\_all\_words = nltk.FreqDist(new\_all\_words\_list)

>>> new\_word\_items = new\_all\_words.most\_common(2000)

>>> new\_word\_features = [word for (word,count) in new\_word\_items]

>>> new\_word\_features[:30]

Now choose to re-run one of the classifiers with the word\_features having stop words removed. Noting that the definition of the feature functions uses the word\_features variable, choose to redefine either

featuresets

SL\_featuresets

or NOT\_featuresets

Rerun the training and test sets, train the classifier and report on classifier accuracy in the discussion. Be sure to post the baseline accuracy that you got for that type of feature set when you first ran it and the new accuracy score with stopwords removed.

Another option would be to re-define the SL\_features function to have just one numeric feature that would subtract the number of negative words from positive words. Again you would post a baseline accuracy score for the original SL\_features and a new accuracy score for the new definition of features.