NLP Lab Session
Week 13, November 19, 2014
Text Processing and Twitter Sentiment for the Final Projects

Getting Started

In this lab, we will be doing some work in the Python IDLE window and also running some stand-alone python programs. All of the programs and data are zipped together in one zip file on Blackboard:

LabExamplesWeek13FinalProject.zip

Download this zip file to your NLP class folder in the lab and unzip it there.

General Text Processing

In this section, we just review the recommended text processing tools in the NLTK. Given some plain text, we'll break it into sentences, tokenize it, assign POS tags to the tokens and final find named entity chunks. For parsing, we would load the Stanford parser jar file and models, and use the NLTK Stanford parser interface.

Just for the sake of examples, we define some text (found in an online news story).

```
>>> text = 'Senator Elizabeth Warren from Massachusetts announced her support of Social Security in Washington, D.C. on Tuesday. Warren joined other Democrats in support.'
```

First, the NLTK tokenize package has a generically named sentence tokenizer, which is actually the Punkt sentence tokenizer. This function produces a list of text strings, each of which is a sentence.

```
>>> import nltk
>>> from nltk import tokenize
>>> sentences = tokenize.sent_tokenize(text)
>>> sentences
```

Next we separate each sentence into tokens, using the NLTK Penn Treebank tokenizer, and we apply it to each sentence.

```
>>> from nltk.tokenize import TreebankWordTokenizer
>>> texttokens = []
>>> for sent in sentences:
        texttokens.append(TreebankWordTokenizer().tokenize(sent))
>>> texttokens
```

Similarly, we can then apply the Penn Treebank POS tagger to each sentence's list of tokens.

```
>>> from nltk.tag import pos_tag
>>> taggedsentences = []
>>> for sentencetokens in texttokens:
        taggedsentences.append(pos_tag(sentencetokens))
>>> taggedsentences
```

Then we can apply the named entity chunker in order to find and label each named entity in the sentences.

```
>>> chunksentences = []
>>> for sent in taggedsentences:
        chunksentences.append ( nltk.chunk.ne_chunk(sent))
>>> for sent in chunksentences:
        print sent
```

The resulting lists and trees can then be used for further processing. As an example, I have written a python program called processTextPOS.py. It is called on the command line with a directory argument:

$ processTextPOS.py datatext (or on a Mac: $ python processTextPOS.py datatext)

- processes all text files in the directory
- writes a file with POS tags on all the words for each file
- writes a file where it has found all the verbs (selecting words whose tag starts with 'V')
- writes a file where it has found all the named entities of type PERSON

Some parts of this type of processing will be useful in working on projects with the email Spam detection or the Kaggle movie reviews.

Twitter Processing

Although it's possible to use the above general text language processing tools on Twitter text, it is more useful to use language processing tools that have been adapted for the Twitter genre. The Twitter tokenizing from Tweet Motif that we used earlier is part of a general Twitter language processing package called ARK from CMU.

The Part-of-Speech tagging has been extensively modified from Penn Treebank to Twitter. Tagsets for nouns and verbs have been collapsed to fewer tags, and extra tags have been introduced by the special types of tokens often found in Tweets.

Here is a list of tags (from Gimpel et al, Part-of-Speech Tagging for Twitter, ACL 2011):

Tag Description	Examples	%
Nominal, Nominal + Verbal		
N common noun (NN, NNS)	books someone	13.7
O pronoun (personal/WH; not possessive; PRP, WP)	it you u meeee	6.8
S nominal + possessive	books' someone's	0.1
^ proper noun (NNP, NNPS)	lebron usa iPad	6.4
Z proper noun + possessive	America's	0.2
L nominal + verbal	he's book'll iono(= I don't know)	1.6
M proper noun + verbal	Mark'll	0.0
Other open-class words		
V verb incl. copula, auxiliaries (V*, MD)	might gonna ought couldn't is eats	15.1
A adjective (J*)	good fav lil	5.1
R adverb (R*, WRB)	2 (i.e., too)	4.6
! interjection (UH)	lol haha FTW yea right	2.6
Other closed-class words		
D determiner (WDT, DT, WP$, PRP$)	the teh its it's	6.5
P pre- or postposition, or subordinating conjunction (IN, TO)	while to for 2 (i.e.,to) 4 (i.e., for)	8.7
& coordinating conjunction (CC)	and n & + BUT	1.7
T verb particle (RP)	out off Up UP	0.6
X existential there, predeterminers (EX, PDT)	both	0.1
Y X + verbal	there's all's	0.0
Twitter/online-specific		
# hashtag (indicates topic/category for tweet)	#acl	1.0
@ at-mention (indicates another user as a recipient of a tweet)	@BarackObama	4.9
~ discourse marker, indications of continuation of a message across multiple tweets	RT and : in retweet construction RT @user : hello	3.4
U URL or email address	http://bit.ly/xyz	1.6
E emoticon	:-) :b (: <3 o O	1.0
Miscellaneous		
$ numeral (CD)	2010 four 9:30	1.5
, punctuation (#, $, ", (,), ,, ., :, ``)	!!! .... ?!?	11.6
G other abbreviations, foreign words, possessive endings, symbols, garbage (FW, POS, SYM, LS)	ily (I love you) wby (what about you) 's _ -->awesome...I'm	

To run this Twitter specific POS tagger, unzip the .zip file (which was downloaded from

the ARK project) to get the directory
ark-tweet-nlp-0.3.2

In a terminal window, navigate to this folder and run the tagger on one of the example text files provided and redirect the output to a file in that folder.

$ ./runTagger.sh examples/example_tweets.txt > example_tweets_results.txt
(or equivalent on PC)

I also modified their script show.py to show the output already stored in a file:

$ python showfile.py > ../example_tweets_pretty.txt
(or equivalent on PC)

From this project, there is also a model file for the POS tagger that gives Penn Treebank style tags, and there is a dependency parser called Tweebo.

Classifying Twitter Sentiment using Sentiment 140 data for training

There have been quite a few research papers in recent years that discuss various aspects of finding Twitter Sentiment. I have given details from

- Koulompis, Wilson and Moore "Twitter Sentiment Analysis: The Good the bad and the OMG!", in ICWSM (International Conference on Weblogs and Social Media) in 2011, and
- Mohammad, Kiritchenko and Zhu, NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013),* June 2013, Atlanta, Georgia, USA.

These papers are available on-line and also in the Content -> Readings and Resources.

There are two sets of training data for Twitter sentiment. First, we are going to look at the Sentiment 140 Emoticon data set, available at http://help.sentiment140.com/, where you can click on the link "For Academics" to read about the data and their process. The data set was collected by searching Twitter for positive and negative emoticons. The emoticons were removed from the tweets, and each tweet was labeled positive (4) or negative (0) according to its emoticon. The data set has 1,600,000 tweets. This dataset is good for experiments, but note that in a practical situation where we want to detect sentiment in tweets, we would want to have training data for three polarities: positive, negative and neutral. Also, since this training set has emoticons removed, they can't be used for features.

In order to randomly select a smaller training set, I wrote a program called sentiment__classify_select_training_data.py that selects 8,000 tweets of each polarity and saves them in a .csv file called training.16000.mixed.csv. I have included the program for your information and use if you wish to have larger or smaller training sets. (We can

look at some examples in this file, either using Excel or NotePad.) Note that to run the program, you should use easy_install to install the python package csv.

First, if we don't already have easy_install, we should install it. Next we open a terminal (command prompt) window and type

% easy_install csv

This installs the python csv package in its folder of external packages.
And then copy the Sentiment140 folder into the same folder as the other sentiment classification programs.

$ python sentiment__classify_select_training.py
(or whatever command you need to use on a PC)

In order to use the SemEval training data, you should write a similar program that reads that data. Since there are only about 8,000 of them, you do not need to select a subset.

Next, I have written a baseline sentiment classifier program that uses NLTK to train a classifier using word features and subjectivity features from the training set. The baseline classifier uses a twitter tokenizer from Tweet Motif and does some pruning to remove non-ascii characters, tokens of length 1 and tokens on a tweet stopword list. In order to run this program, open a terminal window (or command prompt window) and navigate to the folder where you put all the programs from this week's lab examples.

Now run the classify train model program (note that this uses the full 1600000 training set and not the smaller one):

$ python sentiment__classify_train_model.py
(or whatever command you need to use on a PC)

This will take a little time to make the feature sets and train the model.

Now to work on this classification task for the final project, you can extend this program in several ways:

- Work on preprocessing to improve the tokens
- Work on features to add useful ones
- Experiment with different classifiers in weka

For ideas on preprocessing and features, we can look at a summary of the two papers did. Note that the baseline program already uses the three subjectivity lists as they did, but that you could experiment with using the LIWC or ANEW lists instead of, or in addition to, the subjectivity lexicon. We also have available the NRC created lexicon if you would like to work on it.

Here are the main points for successful preprocessing and features from the Wilson paper:

Preprocessing:

- Tokenize emoticons, hashtags, mentions and URLs and replace them by their token type (HAPPY, SAD, URL, etc.)
- Expand abbreviations to their meanings, brb -> "be right back"
- Normalize character repetitions to two, "happyyyyy" -> "happyy"
- Note intensifiers such as all caps, "I LOVE this show"

Features:

- Vocabulary: removed stopwords, replaced words that either directly preceded or directly succeeded a negation word with the word with NOT, "happy" -> "NOT_happy" and keep only the top occurring 1000 words
- Subjectivity Lexicon: three features denoting the presence or absence of positive, neutral or negative emotion words.
- (Part-of-speech features were tried, but didn't add to the accuracy.)
- Micro-blogging features: emoticons, presence of intensifiers (using the Internet Lingo Dictionary and other internet slang dictionaries).

Now we'll refer to the NRC Canada paper to see their features and get additional ideas. In particular, note how they handled features from the sentiment lexicons.

Note that if you want to try using positive and negative sentiment words from the LIWC lexicon, I have given a python program that will read them in sentiment__read_pos_neg_words.py in the zip file.

Lab Exercise:

For an exercise this week, choose one of the ways to process text, either general text or tweets. Create a file with with either 2 regular text sentences or 2 tweet messages. Run the appropriate processing to get POS tagged sentences. Put your sentences and the resulting POS tagged text into a post in the Week 13 Discussion thread.