Lab 4: Sentiment analysis on Twitter

Use the html version[3] of this lab if you want to easily access the links or copy and paste commands.
Or use the pdf version[4] if you want nicer formatting or a printable sheet.

Goals and motivation of this lab, and what to submit

The goal of this lab is to give you some practice with using pointwise mutual information (PMI) to look at word collocations, which we will use for *sentiment analysis*. In particular, we will investigate sentiment in social media, where we can use PMI to explore whether people have a positive or negative view of particular things. For example, while it is clear that people on Twitter love Justin Bieber (the bane of any social media researcher) other sentiments are less obvious: What do people on Twitter think about food? Do people like themselves?

At the end of today's lab session, please email me a short message including the following information (If you are working with a partner, only one email is needed for both of you):

- your name and your partner's name;
- your answers to Q5, Q6, and Q7.

I will post solutions to each lab but will not provide individual feedback, I just want to make sure that you are actually making an effort.

Preliminaries

First, download the files lab4.py[5] *and* tweets_en.zip[6] into your `labs` folder (or folder of your choice).

Unzip `tweets_en.zip` and make sure you now have a file called `tweets_en.txt` in the same folder as `lab4.py`.

Start up Spyder and check that the code runs by first typing in your iPython window:

```
cd <foldername>
```

where `<foldername>` is replaced with the name of the folder where you put `lab4.py` and `tweets_en.txt`. Then run this week's lab as follows:

```
%run lab4.py
```

It will take a few seconds to run as it reads in the data and computes all the counts. It should then print out the following information:

- Whether the PMI function is correct (is isn't yet).
- The total number of tweets (also not yet correct).
- The counts for each sentiment and target word, and cooccurrence counts.

Do the following section of the lab before starting to look at the code.

[3] http://homepages.inf.ed.ac.uk/sgwater/teaching/lsa2015/labs/lab4.html
[4] http://homepages.inf.ed.ac.uk/sgwater/teaching/lsa2015/labs/lab4.pdf
[5] http://homepages.inf.ed.ac.uk/sgwater/teaching/lsa2015/labs/lab4.py
[6] http://homepages.inf.ed.ac.uk/sgwater/teaching/lsa2015/data/tweets_en.zip
[7] http://www.enchantedlearning.com/wordlist/positivewords.shtml
[8] http://www.enchantedlearning.com/wordlist/negativewords.shtml

Pointwise mutual information

This lab is based on work by Turney et al.[1], who propose that the "semantic orientation" of a word (whether it has a positive or negative connotation) can be measured by looking at whether that word co-occurs more with clearly positive words (e.g., *good*, *love*) or clearly negative words (e.g., *bad*, *hate*). Here, we will measure co-occurrence strength using PMI (Pointwise Mutual Information) and average it over a set of positive/negative words to get a positive/negative sentiment (semantic orientation) score.

Recall the definition of PMI:

$$\text{PMI}(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

As usual, we do not have access to the true probabilities of words, so we will need to estimate these from counts.

First, rewrite the PMI equation in terms of raw counts, assuming maximum-likelihood estimation of the probabilities. Use N for the total number observations (here, each observation will be a tweet), $C(x)$ for the number of tweets containing x, and $C(x, y)$ for the number of Tweets containing both x and y (i.e., the number of co-occurrences).

Now, using the toy data below, compute PMI(x, y), PMI(x, z), and PMI(y, z). (We'll use these to test the implementation of PMI later on):

$N = 12$
$C(x) = 6$
$C(y) = 4$
$C(z) = 3$
$C(x,y) = 2$
$C(x,z) = 1$
$C(y,z) = 2$

Q1: What do negative, zero and positive PMI values mean? (relate them to statistical independence).

About the data set

The data we will be using in this lab is a small sample of tweets sent during late 2009 and early 2010. We started with about 1.5 million tweets and preprocessed this data for you using the following steps:

- As a condition of using them, the Tweets have been anonymized: usernames have been replaced by strings of numbers, and the words in the Tweets have been reordered (so they are effectively bags of words).
- We filtered out non-English tweets using the method described by[2].
- We tokenized the tweets on whitespace, converted all words to lowercase, and removed all non-alphanumeric characters except @ and # (which have special meaning in tweets).

Although this may seem like a large data set, it is not big by Twitter standards, and is really only just big enough to allow us to analyze some of the more common words. We'll consider this issue further below.

(This lab is based on one using a dataset we couldn't redistribute, which contained 100 million tweets, still only a 1% sample of 2011 Tweets!)

Examining and running the code

Look at the code in lab4.py. Currently it does the following:

- Defines a list of positive words, currently just `'love'`.
- Defines a list of negative words, currently just `'hate'`.

[1] Turney, Peter D., and Michael L. Littman. "Measuring praise and criticism: Inference of semantic orientation from association." ACM Transactions on Information Systems (TOIS) 21.4 (2003): 315-346.

[2] Lui, Marco and Timothy Baldwin (2012) langid.py: An Off-the-shelf Language Identification Tool, In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012), Demo Session, Jeju, Republic of Korea

- Defines a list of target words (words you want to work out the sentiment of), currently just `'bieber'`. (Because of the size and preprocessing of the dataset, using Justin Bieber's full username, `@justinbieber`, won't work here. We are therefore assuming that mentions of `bieber` refer to Justin Bieber and not someone else.)
- Defines a function called PMI, which currently just returns 0. (This generates a warning which you can ignore for now.)
- Defines two functions, `filter_o_counts` and `filter_co_counts` that you can use to filter out words or word co-occurrences with low (or high) frequencies.
- Reads in the `tweets_en.txt` file and computes the occurrence counts for all words, and the co-occurrence counts of each target word with each sentiment word, putting them into `o_counts` and `co_counts` (see below). It then prints out the counts of each sentiment and target word, and the co-occurrences.
- Loops over target words to compute PMI with sentiment words, and prints out the results.

After running the file, you should be able to access the following two variables:

- `o_counts`: A Counter (a type of dictionary) where the key is a word and the value is the number of times the word occurs in the dataset.
- `co_counts`: A dictionary where the key is a target word and the value is a Counter that stores the number of co-occurrences of each sentiment word with the target. So, `co_counts['bieber']['love']` returns the co-occurrences of those two words. Note that counts are only stored for words defined in the target/sentiment words lists; if you try to access co-occurrence counts for any other pair of words, you'll just get 0.

To make sure you understand how to access the information you will need in the follwing sections, answer the following questions:

Q2: How big is the `o_counts` dictionary? What does this number represent?

Q3: Which word occurs more often in this dataset, `like` or `love`?

Do Twitter users like Justin Bieber?

To answer this question, you will need to do two things:

1. Modify the code so that the value of N (initialized around line 48) is updated to store the total number of tweets, which is needed to compute PMI values. (See where it is passed in to the PMI function around lines 88 and 92).
2. Fill in the correct definition of the PMI function. Before doing so, look back at the number you computed in the Pointwise Mutual Information section of the lab for PMI(y,z). Notice that our error-checking code (defined right after the PMI function) is based on the same input numbers, so you should have gotten the same answer that the error check is looking for. If you didn't, and you are not sure why, please ask one of us or another student for help. If you did, then go ahead and implement the PMI function.

   Note: When writing your own code, you should always be thinking about how to put this kind of error check in! Checking once by hand is a good start, but if you later change the code it is very easy to introduce bugs without realizing it, so having the error check run automatically every time is much better.

Q4: You should now be able to make a preliminary determination of whether Justin Bieber is seen positively on Twitter, by considering just two possible sentiment words. What is the answer?

Investigating other words

In this part of the lab we'll try adding other words to the sentiment word lists and target word list to do some further exploration of this dataset and Twitter users' opinions, as well as just what they like to talk about.

Before looking at any opinions, let's just examine the counts of two words: `'husband'` and `'wife'`.

Q5: Using `o_counts`, determine which of these two words occurs more in this dataset. By how much? What are two possible explanations for this difference?

Q6: Now, add these words to the target word list and re-run the file to look at their positive vs. negative sentiment scores. Are there noticeable differences in the sentiment of tweets in which people refer to husbands or wives? If so, what are they?

Now, consider your own words to investigate. Because of the size of the dataset, you'll need to choose relatively common words to test, otherwise there will be few or no co-occurrences with the sentiment words, and the results won't be very meaningful. You can check how frequent particular words are using `o_counts`. If you're having trouble thinking of good words, you can also use the `filter_o_counts` function to get a list of all words within a certain frequency range. For example,

```
filter_o_counts(o_counts, 1000, 10000)
```

will return a Counter with only the words with between 1k-10k occurrences. (Leaving off the 3rd argument will just use a minimum threshold---but consider why a maximum threshold might be useful too.) Look over this list to see if any words look interesting to try.

Some questions to consider in thinking of words to try: Are there any words that you would expect to have mostly negative connotations? What about words that have high PMI with *both* positive and negative sentiment words? What could this indicate? Are there words you might expect to have *negative* PMI with both positive and negative words? Why?

Once you've got a few target words that seem interesting, you may also want to consider adding other sentiment words to see whether it changes the results at all. You can think of words yourself or consult some existing positive[7] and negative[8] word lists. But remember that if your sentiment words are too low frequency, you may get unreliable results from small co-occurrence numbers. You could consider using the `filter_co_counts` function to filter out very low co-occurrence counts. Do you find that your results change much depending on the sentiment words and whether or not you include low co-occurrence counts? These are some of the thorny issues you would need to deal with if you were doing research in this area!

Q7: Pick a few words you looked at that you think are interesting and speak to some of the questions and issues above. Say what you expected to find and what you did find about these words, and what it might say about Twitter users' behavior and opinions. (You don't need to go into huge detail here if you did lots of comparisons, just give a few setnences about what you thought was interesting.)

Going Further

1. Twitter users often use character repetition to indicate exaggeration. For example, `looovvee` is an exaggerated form of `love`. Extend the code so that it searches not just for the exact words in your positive and negative lists, but also finds matches with extra repeated characters and counts these as additional instances of the sentiment word.

2. The J&M and M&S textbooks provide definitions of many different collocation measures as alternatives to PMI. Implement some of them and see if they give you any different results.

3. So far, we have been assuming that all positive (negative) words are created equal. But is it true? Perhaps some concepts with negative connotations don't attract words like `bad` but instead attract different kinds of negative words like `annoy`. Can you think of different kinds of positive/negative words that seem to fall into different classes and behave differently with respect to target words? Again, you may wish to consult some existing positive[7] and negative[8] word lists.