Lab 8: Sentiment on Twitter, and working with large files

Author: Luke Shrimpton
Author: Sharon Goldwater
Date: 2014-11-01, 2015-11-10

This lab is available as a web page[4] or pdf document[5].

Goals and motivation of this lab

The goals of this lab are two-fold:

1. To give you some practice with using pointwise mutual information (PMI) in a sentiment analysis setting. 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: Do people like the UK? What do people on Twitter think about food?
2. To introduce you to a few of the potential problems we run into when working with very large data sets (we use a dataset of ~100,000,000 tweets), and some ways to deal with them. Not all NLP tasks require such large datasets, but they are becoming more and more common so understanding these issues is important.

Preliminaries

As usual, create a directory for this lab inside your `labs` directory:

```
cd ~/anlp/labs
mkdir lab8
cd lab8
```

Download the files lab8.py[6] and load_map.py[7] into your `lab8` directory: From the Lab 8 web page[4], right-click on the link and select *Save link as...*, then navigate to your lab8 directory to save.

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

[3]http://creativecommons.org/licenses/by-nc/4.0/.
[4]http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab8.html
[5]http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab8.pdf
[6]http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab8.py
[7]http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/load_map.py
[8]http://www.inf.ed.ac.uk/teaching/courses/anlp/lectures/index.html
[9]http://www.enchantedlearning.com/wordlist/positivewords.shtml
[10]http://www.enchantedlearning.com/wordlist/negativewords.shtml

PMI for sentiment analysis

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.

As in last week's tutorial, we'll compute PMI here using MLE for the probability estimates. In the tutorial you should have found the expression for computing PMI from counts, and used it on some toy examples. We'll use this information below so if you can't remember it, please look back at the tutorial and solutions[8] now.

Examining the data files

The data we will be using in this lab is a 1% sample of all tweets sent during 2011, about 100 million tweets. We have preprocessed this data for you using the following steps:

- We filtered out non-English tweets using the method described by[2].
- We then tokenized the tweets on whitespace, converted all words to lowercase, and removed all non-alphanumeric characters except @ and # (which have special meaning in tweets).
- We removed stopwords, applied a stemmer to the remaining words, and then removed all words that occurred less than 100 times.
- We counted how often each word occurred in the data set and how often each word pair co-occurred in the same tweet. We filtered out all co-occurrences with counts less than 10.

Even after all this filtering, the data set is extremely large, and the amount of computer memory required to hold it all can be prohibitive. So, we will be using a trick to reduce the amount of memory we need: we will represent each word using a unique integer ID number, and store a mapping between the words and the numbers.

This trick works because a standard 32 bit integer can represent any number up to 2^31-1[*], or about 2.1 billion distinct word IDs, and still requires only 32 bits of memory. In contrast, each character in a word requires 1 byte (8 bits) of memory, so any word longer than four characters will require more memory to store than an integer ID. Using the integer mapping saves space for any word longer than four characters (which is most words).

Before moving on to the rest of the lab, make sure you understand what the data files look like. There are two data files, which are located in `/afs/inf.ed.ac.uk/group/teaching/anlp/`. Before doing anything with these files, please read the following important

WARNINGS:

1. **Do not copy these files** into your home directory on DICE, or onto your own computer. The `counts` file is over 200M, and we are not allowed to redistribute it anyway (see below).
2. **Do not try to open these files with gedit** or another text editor. Text editors struggle with files this big and you may find yourself waiting an awfully long time for the file to load, and not being able to do anything useful with it after that.
3. **Do not redistribute these files**. We have the right to use them within the University of Edinburgh but we do not have the right to redistribute them.

[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

[*]An *unsigned* integer (i.e., non-negative values only) can represent values up to 2^32-1, but a *signed* integer uses one bit to represent the sign.

Now that you have read the warnings, we will take a look at these files using our handy Unix commands, which do not require loading anything into a text editor.

Use `less` to take a look at the two files below, and make sure you understand what is in them:

1. `wid_word`: This file stores two tab separated columns listing each word ID and the word it represents.
2. `counts`: The first line of the file is the total number of tweets in the dataset (number of observations). The following lines contain more tab separated data, in the following format:

   w_0 <tab> c_0 <tab> w_1 <space> c_1 <tab> w_2 <space> c_2 ... w_n <space> c_n

   where w_0 is a word ID and c_0 is the number of times it occurs. Then, each of the pairs (w_i <space> c_i) is some other word ID and the number of times it co-occurs with w_0.

What word's cooccurrence statistics are listed on the second line of the `counts` file (the line that starts with `0`)? You will need to `grep` for the right information in the `wid_word` file.

After all our preprocessing, how many distinct words are left in our data set? *Hint:* use `wc`. Try this on each file and notice the difference in how long it takes to get a result.

Examining and running the code

Open the lab file with gedit and start up ipython:

```
gedit lab8.py &
ipython
```

You can run this week's lab as follows from iPython:

```
%run load_map.py # You only need to run this once when you first start ipython.
%run -i lab8.py # The -i ensures that the word id mappings are passed to this script
```

Look at the code in lab8.py. Currently it does the following:

- Defines a function called PMI, which currently just returns 0. (This generates a warning which you can ignore for now.)
- Defines a list of positive words. Currently just `'love'`
- Defines a list of negative words. Currently just `'hate'`
- Defines a list of target words (words you want to work out the sentiment of). Currently just `'@justinbieber'`.
- Opens the `counts` file and loads the occurrence counts and co-occurrence counts for the words defined in the above lists, putting them into the dictionaries o_counts and co_counts. (Loading counts for all the words in the data files would take too much time/memory so we only load the ones of interest.)
- Loops over target words to compute PMI with sentiment words. This code is incomplete, you will fill it in in the next section.

WARNING: Be careful what variables you try to print. For example, if you type `print wid_word` or just `wid_word` in the interpreter, it will take a very long time to format and print to the screen. You will probably want to cancel the command by pressing `Ctrl-c`.

After running the initial commands there are four variables loaded that will be very useful:

- `wid_word`: A dictionary that contains the mapping from word id to the actual word.
- `word_wid`: A dictionary that contains the mapping from word to word id.

- `o_counts`: A dictionary where the key is a word id and the value is the number of times the word occurs in the dataset.
- `co_counts`: A dictionary where the key is a word id (`word0`) and the value is another dictionary with word id as the key (`word1`) and the co-occurrence count of (`word0`, `word1`) as the value. If you try to access word pairs that don't co-occur it will return an error.

To make sure you understand how to access the information you will need in the follwing sections, answer the following questions (using Python now, not Unix shell commands):

- How big is the `wid_word` dictionary?
- What is the id of `'@bbcnews'` and what word has the id `234`?
- Check whether the following words exist in the dataset: `'bob'`, `'toast'`, `'insektors'`, `'looovvee'`, `'you'`, `'Norman'`, `':-)'`. For the ones that don't exist, why don't they? (*Hint*: look back at our preprocessing steps.)
- What data type is `o_counts[word_wid['love']]` and `co_counts[word_wid['love']]`?
- How many times does the word `'@justinbieber'` occur in the dataset?
- How many times does the world `'@justinbieber'` co-occur with `'love'`?
- How many times does the world `'@justinbieber'` co-occur with `'hate'`?

Do Twitter users like Justin Bieber?

To answer this question, you will need to do the following in `lab8.py`:

1. Fill in the correct definition of the PMI function. Before doing so, look back at the number you computed for PMI(y,z) in last week's tutorial. 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 a demonstrator 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.

2. Fill in the code in the two `for` loops at the bottom, which are intended to compute the PMI between each target word and each positive/negative word. The loops should build up the list of positive and negative PMI values for each target word (currently, the lists will only have one item each). Once you have the correct values in the lists, ucomment the last line of the file, which prints out the average value of each list for each target item.

   Remember all the counts are stored using word ids so you will need to use `word_wid` to look up the id for the words.

When you have finished these tasks, you should be able to determine whether Justin Bieber is seen positively on Twitter, at least if we only consider two possible sentiment words. What is the answer?

Family members

Now add the words `husband` and `wife` to the target word list and re-run the file. Before considering their sentiment, answer the following questions:

Which of these two words occurs more in this dataset? By how much? What are some possible explanations for this difference? (There are quite a few!)

Now look at the positive vs. negative sentiment scores of these words. Are there noticeable differences in the sentiment of tweets in which people refer to husbands or wives? If so, what are they?

What about other family members like `son` and `daughter`, or some of your own choice? Do people like themselves? (Answering the last question isn't entirely straightforward. What target words might help?)

In class there was a question about synonyms, and I pointed out that supposedly synonymous words often have different connotations. What would you predict about the sentiment of the synonymous words `kid` and `child`? Does the Twitter data support your prediction?

PMI gives us a very high-level overview of the data, but doesn't give us a lot of detail. If we're trying to explain *why* sentiment differs for different words, it helps to look at examples from the raw data. If you want to do that, here are two example queries to search the Twitter website for Tweets with the same date range as our corpus. The first brings back Tweets with the word `husband`, and the second looks for both `husband` and `hate`:

https://twitter.com/search?vertical=default&q=%22husband%22%20since%3A2011-01-01%20until%3A2011-12-30&src=typd

https://twitter.com/search?vertical=default&q=husband%20hate%20since%3A2011-01-01%20until%3A2011-12-30&src=typd

You can modify these to look for other words if you want--just change the relevant part of the URL. (*Warning*: reading too many random Tweets may melt your brain.)

What else do Twitter users like?

You can also add more words to the target list to try to see what other opinions Twitter users have. Can you think of some words that you would expect to have negative connotations? What about words that have high (or low) PMI with *both* positive and negative sentiment words? What could this indicate?

Feel free to choose words that interest you, though if you are unsure try: `facebook`, `school`, `@stephenfry`, `@comcast`, `@oprah`, `food`, `toothbrush`, `bf`, `gf` . Do any of the results surprise you?

The current system is very simplistic as it only uses one positive/negative word. Try adding some other positive and negative words of your own and see whether the results change. You may wish to consult some existing positive[9] and negative[10] word lists.

Note: you may find that some words you would expect to be in the dictionaries are not, and cause key errors. The most likely reason is stemming: for example, `terrible` is not in the dictionary but the stemmed version `terribl` is. You can check `word_wid` to see if each word is there, though in some cases you may still get an error from `co_counts` if the number of co-occurrences was too low and got filtered out.

Going Further

1. On Twitter people can reference a user either by an @ mention or by name (eg: `@justinbieber` and `Bieber` refer to the same person). Think of some examples of this and compare the use of name references and @ mentions. Do different mention methods result in different sentiments?

2. Twitter users often use character repetition to indicate exaggeration. For example, `aaawwesome` is an exaggerated form of `awesome`. 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.

3. 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.

4. 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[9] and negative[10] word lists.