Tutorial 3. Sentiment analysis on the Twitter textual data

3.1 Introduction and setup

The purpose of this tutorial is to guide you through the construction of a very basic sentiment analysis tool. We aim at detecting the polarity of a given text: is the attitude of the writer **positive** or **negative**. Texts such as `'This view is amazing'` are considered positive, whereas text as `'This view is horrible'` will be considered negative. Such a tool can be useful, for example, to detect if products reviews are positive or negative. The tool that we will construct starts from a very simple hypothesis: *it a text contains mainly positive words, then the general sentiment is positive; if the text contains mainly negative words, then the sentiment is negative*.

Step 1: For this purpose, we will use a sentiment lexicon: a list of words together with their polarity (*i.e.*, positive or negative). We use the lexicon described in `Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA.`, and available for download here (http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html).

In order to speed up this tutorial, we have downloaded the lexicon and pre-treated it (*i.e.*, we removed the header). Save into your current folder the two lexicon files: sentiment-lexicon-positive-words.txt (https://wattlecourses.anu.edu.au/mod/resource/view.php?id=1130282) and sentiment-lexicon-negative-words.txt (https://wattlecourses.anu.edu.au/mod/resource/view.php?id=1215818). Please note that some words in the lexicon are intentionally misspelled in order to accommodate common errors found in social media.

Step 2: We will require to verify if the words in target text belong to the negative or to the positive lexicon. For this, we use the Python's powerful NLTK (Natural Language Toolkit) module (http://www.nltk.org/). NLTK contains a comprehensive number of tools needed for natural text processing: tokenizers, stemmers, lemmatizers *etc.* In the following, we will use a tokenizer (a tool which splits natural language text into tokens - words) and a lemmatizer (a tool to reduce a word to its lemma - singular, masculine form for nouns, infinitive for verbs *etc.*).

If you followed the installation instructions (https://wattlecourses.anu.edu.au/pluginfile.php/1412128/mod_resource/content/11/installation-and-usage-instructions.html), the *nltk* module should already be installed, alongside with the additional required data that the tokenizer requires. If you work on your own laptop, you need to download this additional data by executing the code below. A window will open letting you chose what to download:

- Select the "Corpora" tab, scroll down and select "wordnet";
- Select the "Models" tab, scroll down and select the "punkt". This will require a download of 23.3MB.

In [1]:

```
import nltk

nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pag
es/index.xml
```

Out[1]:

```
True
```

We are now set up to start building our sentiment analysis tool.

3.2 Constructing a basic sentiment analysis tool

We start by initializing a set of positive and negative examples to work with. The examples were taken from this sentiment analysis tutorial (http://www.laurentluce.com/posts/twitter-sentiment-analysis-using-python-and-nltk/).

In [2]:

```
pos_tweets = [ 'I love this cars',
               'This view is amazing',
                   'I feel great this morning',
                   'I am so excited about the concert',
                   'He is my best friend']

neg_tweets = ['I do not like this car',
              'This view is horrible',
              'I feel tired this morning',
              'I am not looking forward to the concert',
              'He is my worst enemy']
```

Next, we import the positive and the negative lexicons. The format of the lexicon files is one word per line. As we will lemmatize the words in the target text, we apply the same treatment to the words in the lexicon.

In [3]:

```
from nltk.stem.wordnet import WordNetLemmatizer

# define the lemmatizer
lmtzr = WordNetLemmatizer()

# read the positive and negative lexicon in lists of words
positive_words = [lmtzr.lemmatize(line.strip().decode('utf-8')) for line in
open('sentiment-lexicon-positive-words.txt')]
negative_words = [lmtzr.lemmatize(line.strip().decode('utf-8')) for line in
open('sentiment-lexicon-negative-words.txt')]

print "We have {:d} positive words and {:d} negative words.".format(len(positive
_words), len(negative_words))
```

```
We have 2006 positive words and 4783 negative words.
```

We are going to preprocess the target texts:

- the text is split into tokens (words) by using **nltk**;
- each token is transformed to lowercase;
- tokens are lemmatized;
- tokens with fewer than 3 characters are filtered out, since there are likely to be errors or non-sentiment related words.

In [4]:

```
for words in pos_tweets + neg_tweets:
    # tokenize and lemmatize the current tweet
    tokens = nltk.word_tokenize(words.decode('utf-8'))
    tweet = [lmtzr.lemmatize(x.lower()) for x in tokens if len(x) >= 3]

    # print the tweet
    print tweet
```

```
[u'love', u'this', u'car']
[u'this', u'view', u'amazing']
[u'feel', u'great', u'this', u'morning']
[u'excited', u'about', u'the', u'concert']
[u'best', u'friend']
[u'not', u'like', u'this', u'car']
[u'this', u'view', u'horrible']
[u'feel', u'tired', u'this', u'morning']
[u'not', u'looking', u'forward', u'the', u'concert']
[u'worst', u'enemy']
```

In the end, for each text we compute a score:

- if a given word is in the positive list, we add one to the score;
- if the word is in the negative list, we subtract one from the score.

In the end, if the score is greater than zero, the text is considered as positive. If it is less than zero, it is considered negative. If it is zero, the text is considered neutral.

The following code finds and outputs the positive and negative words in the texts. It computes the score and prints it out for each text.

In [5]:

```
# define the function that computes the sentiment score
def get_sentiment_score(text):
    # tokenize and lemmatize the current tweet
    tokens = nltk.word_tokenize(text)
    tweet = [lmtzr.lemmatize(x.lower()) for x in tokens if len(x) >= 3]

    # calculate the sentiment score
    score = 0
    for word in tweet:
        if word in positive_words:
            score = score + 1
            print "+1", word
        if word in negative_words:
            score = score - 1
            print "-1", word

    return score

# apply it on our example tweets
for words in pos_tweets + neg_tweets:
    print "Tweet: '{:s}', score: {:d}".format(words, get_sentiment_score(words))
    print "-----------------------------------"
```

```
+1 love
Tweet: 'I love this cars', score: 1
-----------------------------------
+1 amazing
Tweet: 'This view is amazing', score: 1
-----------------------------------
+1 great
Tweet: 'I feel great this morning', score: 1
-----------------------------------
+1 excited
Tweet: 'I am so excited about the concert', score: 1
-----------------------------------
+1 best
Tweet: 'He is my best friend', score: 1
-----------------------------------
+1 like
Tweet: 'I do not like this car', score: 1
-----------------------------------
-1 horrible
Tweet: 'This view is horrible', score: -1
-----------------------------------
-1 tired
Tweet: 'I feel tired this morning', score: -1
-----------------------------------
Tweet: 'I am not looking forward to the concert', score: 0
-----------------------------------
-1 worst
-1 enemy
Tweet: 'He is my worst enemy', score: -2
-----------------------------------
```

We observe that all the positive examples have been detected as positive, because they contain positive words. Only 3 out of 5 negative examples were detected as such. In the case of '*I do not like this car*', the word **like** is in the positive list, but in the text it is prefixed by **not** which changes the polarity. **like** is positive, but **not like** is negative. '*I am not looking forward to the concert*' does not contain any positive or negative words, even if the expression **not looking forward** has a negative connotation. This shows the limitations of our naive approach. These special cases should be taken into account.

3.3 Applying the analysis on real data: assignments for you

(1 point) Assignment question #3.1: Use the `get_sentiment_score(text)`, the sentiment scoring function defined before, and calculate the sentiment polarity of the tweets in the Twitter JSON dataset used in tutorial 2 (https://wattlecourses.anu.edu.au/pluginfile.php/1510225/mod_resource/content/14/tutorial-2-construct-network-real-twitter-dump.html). The dataset is available to download here (https://wattlecourses.anu.edu.au/mod/resource/view.php?id=1215757). Print the text of the 10 most positive and the 10 most negative tweets. We consider that a tweet t_1 is more positive than another tweet t_2 when score of the former is higher than the score of the latter ($score(t_1) > score(t_2)$). Similarly, a tweet t_1 is more negative than t_2 when $score(t_1) < score(t_2)$.
HINT: Load the tweets one by one as seen in tutorial 2 and extract the text, which is found in the field *text* of each tweet.

(1 point) Assignment question #3.2: Based on the scores calculated in **Assignment #3.1**, determine the 3 most positive users. A user u_1 is more positive than a user u_2 if the dataset contains more positive tweets emitted by u_1 than tweets emitted by u_2. Formally:
$positivity(u_1) > positivity(u_2) \iff |\{t \mid author(t) = u_1 \wedge score(t) > 0\}| > |\{t \mid author(t) = u_2 \wedge scor$

(1 point) Assignment question #3.3: We have discussed earlier that our system is fragile to negations: it will score the expression *not beautiful* as positive because it only detects the word beautiful as positive. More generally, we consider that the token **not** changes the polarity of a given token: **not beautiful** becomes negative, while **not bad** becomes positive.
Modify the function `get_sentiment_score(text)` to detect the changes of polarity due to the token **not**.

Bonus assignments questions

Bonus assignment questions earn you extra marks if you solve this assignment correctly, no penalty is inflicted if you do not solve it. Note that the total grade of SMA assignments cannot exceed 10 points, therefore the bonus point can only be used to compensate for another question which you did not solve correctly.

(0.25 additional points) Bonus question #1. Based on the scores calculated in **Assignment #3.1**, plot the temporal evolution of the counts of positive and negative tweets. The date a tweet was emitted is found in the field *created_at*. Divide the temporal extent of your dataset into 100 timeslices. The temporal extent of the dataset is from the creation date of the first tweet to the creation date of the last tweet. Count how many positive and how many negative tweets you have in each timeslice. Plot these counts on a graphic resembling this one:

caption

HINT: the graphic above is not based on real data. Your actual curves might **NOT** look like this one. Its purpose is just to show you the expected form of the graphic.

(0.25 additional points) Bonus question #2: Polarized communities We want to know if there are polarized communities in our social graph. Are there closely linked groups of users who have similar sentiment polarities? We calculate the sentiment polarity of a user as the numbre of positive tweets emitted, from which we substract the number of negative tweets. Formally:

$$sentiment\_polarity(u) = |\{t \mid author(t) = u \wedge score(t) > 0\}| - |\{t \mid author(t) = u \wedge score(t) < 0\}| \quad .$$

To visually detect the polarized communities, we want to plot the same graph as in tutorial 1 (https://wattlecourses.anu.edu.au/pluginfile.php/1510161/mod_resource/content/13/tutorial-1-construct-social-graph.html) at *Step 3*, but with the colors of nodes representing their polarity: `blue` for a negative polarity and `red` for a positive polarity. Are nodes colored similarly clustered close together?

In [ ]: