TextBlob Sentiment: Calculating Polarity and Subjectivity

Sunday June 7, 2015

The TextBlob package for Python is a convenient way to do a lot of Natural Language Processing (NLP) tasks. For example:

```
from textblob import TextBlob

TextBlob("not a very great calculation").sentiment
## Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769)
```

This tells us that the English phrase "not a very great calculation" has a *polarity* of about -0.3, meaning it is slightly negative, and a *subjectivity* of about 0.6, meaning it is fairly subjective.

But where do these numbers come from?

Let's find out by going to the source. (This will refer to sloria/TextBlob on GitHub at commit eb08c12.)

After digging a bit, you can find that the main default sentiment calculation is defined in _text.py, which gives credit to the pattern library. (I'm not sure how much is original and how much is from `pattern`.)

There are helpful comments like this one, which gives us more information about the numbers we're interested in:

```
# Each word in the lexicon has scores for:
# 1)     polarity: negative vs. positive    (-1.0 => +1.0)
# 2) subjectivity: objective vs. subjective (+0.0 => +1.0)
# 3)    intensity: modifies next word?      (x0.5 => x2.0)
```

The lexicon it refers to is in en-sentiment.xml, an XML document that includes the following four entries for the word "great".

```
<word form="great" cornetto_synset_id="n_a-525317" wordnet_id="a-01123879" pos="JJ" sense
<word form="great" wordnet_id="a-01278818" pos="JJ" sense="of major significance or impor
<word form="great" wordnet_id="a-01386883" pos="JJ" sense="relatively large in size or nu
<word form="great" wordnet_id="a-01677433" pos="JJ" sense="remarkable or out of the ordir
```

In addition to the polarity, subjectivity, and intensity mentioned in the comment above, there's also "confidence", but I don't see this being used anywhere. In the case of "great" here it's all the same part of speech (`JJ`, adjective), and the senses are themselves natural language and not used. To simplify for readability:

```
word   polarity  subjectivity  intensity
great       1.0           1.0        1.0
great       1.0           1.0        1.0
great       0.4           0.2        1.0
great       0.8           0.8        1.0
```

When calculating sentiment for a single word, TextBlob uses a sophisticated technique known to mathematicians as "averaging".

```
TextBlob("great").sentiment
## Sentiment(polarity=0.8, subjectivity=0.75)
```

At this point we might feel as if we're touring a sausage factory. That feeling isn't going to go away, but remember how delicious sausage is! Even if there isn't a lot of magic here, the results can be useful—and you certainly can't beat it for convenience.

TextBlob doesn't not handle negation, and that ain't nothing!

```
TextBlob("not great").sentiment
## Sentiment(polarity=-0.4, subjectivity=0.75)
```

Negation multiplies the polarity by -0.5, and doesn't affect subjectivity.

TextBlob also handles modifier words! Here's the summarized record for "very" from the lexicon:

```
word   polarity  subjectivity  intensity
very        0.2           0.3        1.3
```

Recognizing "very" as a modifier word, TextBlob will ignore polarity and subjectivity and just use intensity to modify the following word:

```
TextBlob("very great").sentiment
## Sentiment(polarity=1.0, subjectivity=0.9750000000000001)
```

The polarity gets maxed out at 1.0, but you can see that subjectivity is also modified by "very" to become $0.75 \cdot 1.3 = 0.975$.

Negation combines with modifiers in an interesting way: in addition to multiplying by -0.5 for the polarity, the inverse intensity of the modifier enters for both polarity and subjectivity.

```
TextBlob("not very great").sentiment
## Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769)
```

How's that?

$$\text{polarity} = -0.5 \cdot \frac{1}{1.3} \cdot 0.8 \approx -0.31$$

$$\text{subjectivity} = \frac{1}{1.3} \cdot 0.75 \approx 0.58$$

TextBlob will ignore one-letter words in its sentiment phrases, which means things like this will work just the same way:

```
TextBlob("not a very great").sentiment
## Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769)
```

And TextBlob will ignore words it doesn't know anything about:

```
TextBlob("not a very great calculation").sentiment
## Sentiment(polarity=-0.3076923076923077, subjectivity=0.5769230769230769)
```

TextBlob goes along finding words and phrases it can assign polarity and subjectivity to, and it averages them all together for longer text.

And while I'm being a little critical, and such a system of coded rules is in some ways the antithesis of machine learning, it is still a pretty neat system and I think I'd be hard-pressed to code up a better such solution.

Check out the source yourself to see all the details!

If you sign up, I'll send you an email update at most monthly with new stuff.