- Log in
- Subscribe RSS Feed

Laurent Luce's Blog

- Technical blog on web technologies
- Home
- About

- ## Recent Posts
  - Cambridge city geospatial statistics
  - API to access the Cambridge city geospatial data
  - REST service + Python client to access geographic data
  - Massachusetts Census 2010 Towns maps and statistics using Python
  - Python, Twitter statistics and the 2012 French presidential election
  - Twitter sentiment analysis using Python and NLTK
  - Python dictionary implementation
  - Python string objects implementation
  - Python integer objects implementation
  - Python and cryptography with pycrypto
- ## Search

  GO
- ## Meta
  - Log in
  - Entries RSS
  - Comments RSS
  - WordPress.org

Twitter sentiment analysis using Python and NLTK

January 2, 2012

This post describes the implementation of sentiment analysis of tweets using Python and the natural language toolkit NLTK. The post also describes the internals of NLTK related to this implementation.

Background

The purpose of the implementation is to be able to automatically classify a tweet as a positive or negative tweet sentiment wise.

The classifier needs to be trained and to do that, we need a list of manually classified tweets. Let's start with 5 positive tweets and 5 negative tweets.

Positive tweets:

- I love this car.
- This view is amazing.
- I feel great this morning.
- I am so excited about the concert.
- He is my best friend.

Negative 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 enemy.

In the full implementation, I use about 600 positive tweets and 600 negative tweets to train the classifier. I store those tweets in a Redis DB. Even with those numbers, it is quite a small sample and you should use a much larger set if you want good results.

Next is a test set so we can assess the exactitude of the trained classifier.

Test tweets:

- I feel happy this morning. positive.
- Larry is my friend. positive.
- I do not like that man. negative.
- My house is not great. negative.
- Your song is annoying. negative.

Implementation

The following list contains the positive tweets:

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

The following list contains the negative tweets:

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

We take both of those lists and create a single list of tuples each containing two elements. First element is an array containing the words and second element is the type of sentiment. We get rid of the words smaller than 2 characters and we use lowercase for everything.

```
tweets = []
for (words, sentiment) in pos_tweets + neg_tweets:
    words_filtered = [e.lower() for e in words.split() if len(e) >= 3]
    tweets.append((words_filtered, sentiment))
```

The list of tweets now looks like this:

```
tweets = [
    (['love', 'this', 'car'], 'positive'),
    (['this', 'view', 'amazing'], 'positive'),
    (['feel', 'great', 'this', 'morning'], 'positive'),
    (['excited', 'about', 'the', 'concert'], 'positive'),
    (['best', 'friend'], 'positive'),
    (['not', 'like', 'this', 'car'], 'negative'),
    (['this', 'view', 'horrible'], 'negative'),
    (['feel', 'tired', 'this', 'morning'], 'negative'),
    (['not', 'looking', 'forward', 'the', 'concert'], 'negative'),
    (['enemy'], 'negative')]
```

Finally, the list with the test tweets:

```
test_tweets = [
    (['feel', 'happy', 'this', 'morning'], 'positive'),
    (['larry', 'friend'], 'positive'),
    (['not', 'like', 'that', 'man'], 'negative'),
    (['house', 'not', 'great'], 'negative'),
    (['your', 'song', 'annoying'], 'negative')]
```

Classifier

The list of word features need to be extracted from the tweets. It is a list with every distinct words ordered by frequency of appearance. We use the following function to get the list plus the two helper functions.

```
word_features = get_word_features(get_words_in_tweets(tweets))
```

```
def get_words_in_tweets(tweets):
    all_words = []
    for (words, sentiment) in tweets:
      all_words.extend(words)
    return all_words
```

```
def get_word_features(wordlist):
    wordlist = nltk.FreqDist(wordlist)
    word_features = wordlist.keys()
    return word_features
```

If we take a pick inside the function get_word_features, the variable ‘wordlist’ contains:

```
<FreqDist:
    'this': 6,
    'car': 2,
    'concert': 2,
    'feel': 2,
    'morning': 2,
    'not': 2,
    'the': 2,
    'view': 2,
    'about': 1,
    'amazing': 1,
    ...
>
```

We end up with the following list of word features:

```
word_features = [
    'this',
    'car',
    'concert',
    'feel',
    'morning',
    'not',
    'the',
    'view',
    'about',
    'amazing',
    ...
]
```

As you can see, 'this' is the most used word in our tweets, followed by 'car', followed by 'concert'…

To create a classifier, we need to decide what features are relevant. To do that, we first need a feature extractor. The one we are going to use returns a dictionary indicating what words are contained in the input passed. Here, the input is the tweet. We use the word features list defined above along with the input to create the dictionary.

```
def extract_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

As an example, let's call the feature extractor with the document ['love', 'this', 'car'] which is the first positive tweet. We obtain the following dictionary which indicates that the document contains the words: 'love', 'this' and 'car'.

```
{'contains(not)': False,
 'contains(view)': False,
 'contains(best)': False,
 'contains(excited)': False,
 'contains(morning)': False,
 'contains(about)': False,
 'contains(horrible)': False,
 'contains(like)': False,
 'contains(this)': True,
 'contains(friend)': False,
 'contains(concert)': False,
 'contains(feel)': False,
 'contains(love)': True,
 'contains(looking)': False,
 'contains(tired)': False,
 'contains(forward)': False,
 'contains(car)': True,
 'contains(the)': False,
 'contains(amazing)': False,
 'contains(enemy)': False,
 'contains(great)': False}
```

With our feature extractor, we can apply the features to our classifier using the method apply_features. We pass the feature extractor along with the tweets list defined above.

```
training_set = nltk.classify.apply_features(extract_features, tweets)
```

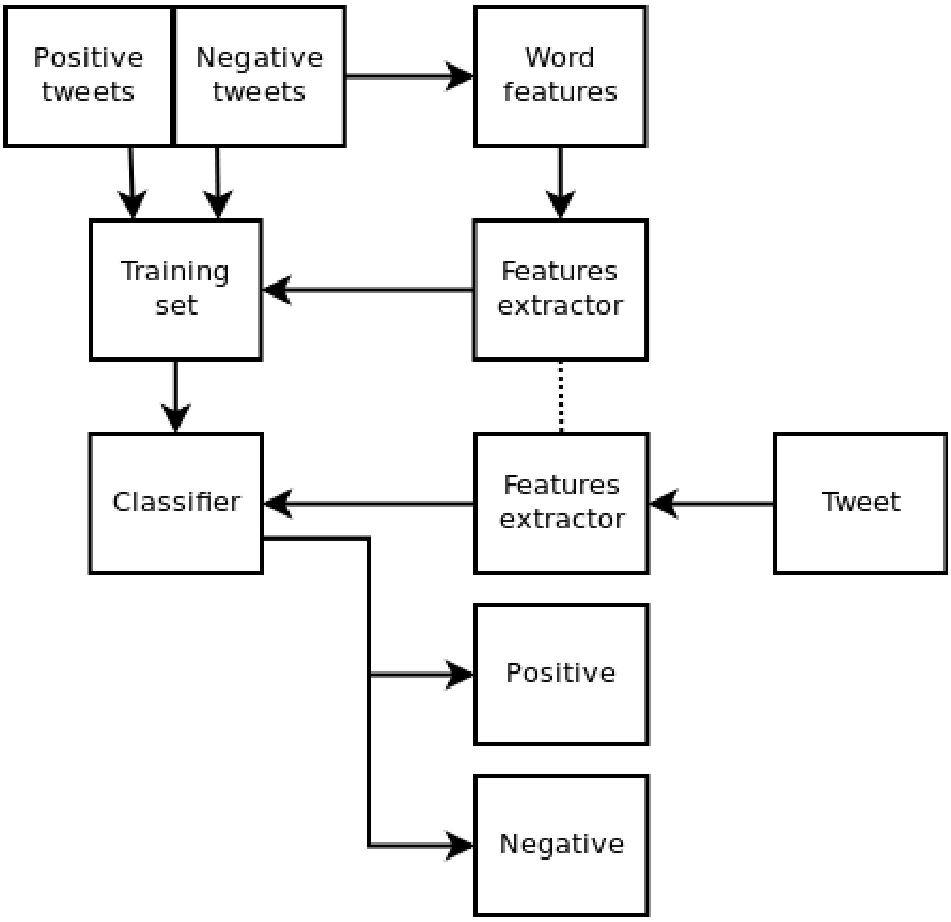
The variable 'training_set' contains the labeled feature sets. It is a list of tuples which each tuple containing the feature dictionary and the sentiment string for each tweet. The sentiment string is also called 'label'.

```
[({'contains(not)': False,
   ...
   'contains(this)': True,
   ...
   'contains(love)': True,
   ...
   'contains(car)': True,
   ...
   'contains(great)': False},
  'positive'),
 ({'contains(not)': False,
   'contains(view)': True,
   ...
   'contains(this)': True,
   ...
   'contains(amazing)': True,
   ...
   'contains(enemy)': False,
   'contains(great)': False},
  'positive'),
  ...]
```

Now that we have our training set, we can train our classifier.

```
classifier = nltk.NaiveBayesClassifier.train(training_set)
```

Here is a summary of what we just saw:



The Naive Bayes classifier uses the prior probability of each label which is the frequency of each label in the training set, and the contribution from each feature. In our case, the frequency of each label is the same for 'positive' and 'negative'. The word 'amazing' appears in 1 of 5 of the positive tweets and none of the negative tweets. This means that the likelihood of the 'positive' label will be multiplied by 0.2 when this word is seen as part of the input.

Let's take a look inside the classifier train method in the source code of the NLTK library. 'label_probdist' is the prior probability of each label and 'feature_probdist' is the feature/value probability dictionary. Those two probability objects are used to create the classifier.

```
def train(labeled_featuresets, estimator=ELEProbDist):
    ...
    # Create the P(label) distribution
    label_probdist = estimator(label_freqdist)
    ...
    # Create the P(fval|label, fname) distribution
    feature_probdist = {}
    ...
    return NaiveBayesClassifier(label_probdist,
feature_probdist)
```

In our case, the probability of each label is 0.5 as we can see below. label_probdist is of type ELEProbDist.

```
print label_probdist.prob('positive')
0.5
print label_probdist.prob('negative')
0.5
```

The feature/value probability dictionary associates expected likelihood estimate to a feature and label.

We can see that the probability for the input to be negative is about 0.077 when the input contains the word ‘best’.

```
print feature_probdist
{('negative', 'contains(view)'): <ELEProbDist based on 5 samples>,
 ('positive', 'contains(excited)'): <ELEProbDist based on 5 samples>,
 ('negative', 'contains(best)'): <ELEProbDist based on 5 samples>, ...}
print feature_probdist[('negative', 'contains(best)')].prob(True)
0.076923076923076927
```

We can display the most informative features for our classifier using the method show_most_informative_features. Here, we see that if the input does not contain the word ‘not’ then the positive ration is 1.6.

```
print classifier.show_most_informative_features(32)
Most Informative Features
           contains(not) = False          positi : negati =      1.6 : 1.0
         contains(tired) = False          positi : negati =      1.2 : 1.0
       contains(excited) = False          negati : positi =      1.2 : 1.0
         contains(great) = False          negati : positi =      1.2 : 1.0
       contains(looking) = False          positi : negati =      1.2 : 1.0
          contains(like) = False          positi : negati =      1.2 : 1.0
          contains(love) = False          negati : positi =      1.2 : 1.0
       contains(amazing) = False          negati : positi =      1.2 : 1.0
         contains(enemy) = False          positi : negati =      1.2 : 1.0
         contains(about) = False          negati : positi =      1.2 : 1.0
          contains(best) = False          negati : positi =      1.2 : 1.0
       contains(forward) = False          positi : negati =      1.2 : 1.0
        contains(friend) = False          negati : positi =      1.2 : 1.0
      contains(horrible) = False          positi : negati =      1.2 : 1.0
...
```

Classify

Now that we have our classifier initialized, we can try to classify a tweet and see what the sentiment type output is. Our classifier is able to detect that this tweet has a positive sentiment because of the word ‘friend’ which is associated to the positive tweet ‘He is my best friend’.

```
tweet = 'Larry is my friend'
print classifier.classify(extract_features(tweet.split()))
positive
```

Let’s take a look at how the classify method works internally in the NLTK library. What we pass to the classify method is the feature set of the tweet we want to analyze. The feature set dictionary indicates that the tweet contains the word ‘friend’.

```
print extract_features(tweet.split())
{'contains(not)': False,
 'contains(view)': False,
 'contains(best)': False,
 'contains(excited)': False,
 'contains(morning)': False,
 'contains(about)': False,
 'contains(horrible)': False,
 'contains(like)': False,
 'contains(this)': False,
 'contains(friend)': True,
 'contains(concert)': False,
 'contains(feel)': False,
 'contains(love)': False,
 'contains(looking)': False,
 'contains(tired)': False,
 'contains(forward)': False,
 'contains(car)': False,
 'contains(the)': False,
 'contains(amazing)': False,
 'contains(enemy)': False,
 'contains(great)': False}
```

```
def classify(self, featureset):
    # Discard any feature names that we've never seen before.
    # Find the log probability of each label, given the features.
    # Then add in the log probability of features given labels.
    # Generate a probability distribution dictionary using the dict logprod
    # Return the sample with the greatest probability from the probability
    # distribution dictionary
```

Let’s go through that method using our example. The parameter passed to the method classify is the feature set dictionary we saw above. The first step is to discard any feature names that are not know by the classifier. This step does nothing in our case so the feature set stays the same.

Next step is to find the log probability for each label. The probability of each label (‘positive’ and ‘negative’) is 0.5. The log probability is the log base 2 of that which is -1. We end up with logprod containing the following:

```
{'positive': -1.0, 'negative': -1.0}
```

The log probability of features given labels is then added to logprod. This means that for each label, we go through the items in the feature set and we add the log probability of each item to logprod[label]. For example, we have the feature name ‘friend’ and the feature value True. Its log probability for the label 'positive' in our classifier is -2.12. This value is added to logprod['positive']. We end up with the following logprod dictionary.

```
{'positive': -5.4785441837188511, 'negative':
-14.784261334886439}
```

The probability distribution dictionary of type DictionaryProbDist is generated:

```
DictionaryProbDist(logprob, normalize=True, log=True)
```

The label with the greatest probability is returned which is 'positive'. Our classifier finds out that this tweets has a positive sentiment based on the training we did.

Another example is the tweet 'My house is not great'. The word 'great' weights more on the positive side but the word 'not' is part of two negative tweets in our training set so the output from the classifier is 'negative'. Of course, the following tweet: 'The movie is not bad' would return 'negative' even if it is 'positive'. Again, a large and well chosen sample will help with the accuracy of the classifier.

Taking the following test tweet 'Your song is annoying'. The classifier thinks it is positive. The reason is that we don't have any information on the feature name 'annoying'. Larger the training sample tweets is, better the classifier will be.

```
tweet = 'Your song is annoying'
print classifier.classify(extract_features(tweet.split()))
positive
```

There is an accuracy method we can use to check the quality of our classifier by using our test tweets. We get 0.8 in our case which is high because we picked our test tweets for this article. The key is to have a very large number of manually classified positive and negative tweets.

Voilà. Don't hesitate to post a comment if you have any feedback.

tags: Python
posted in Uncategorized by Laurent Luce

Follow comments via the RSS Feed | Leave a comment | Trackback URL

37 Comments to "Twitter sentiment analysis using Python and NLTK"

1. *Koray Sahinoglu* wrote:

   Very nice example with detailed explanations. Good work, thank you.

   Link | January 2nd, 2012 at 11:16 pm

2. *Patrick* wrote:

   Hi,
   very good article. But I found two liitle errors:
   1.) Your function get_word_features() does only need one argument.
   2.)apply_features() needs to be called upon nltk.classify.util instead of only nltk.classify

Thanks for sharing.

Link | January 3rd, 2012 at 4:40 am

3. *Arulalan.T* wrote:

   Thanks a lot. Very very useful coding stuff.

   Link | January 3rd, 2012 at 6:12 am

4. *Elliott* wrote:

   Really great article ! You say in the real implementation, you use around 1200 tweets; is there a GUI, or is it possible to automate the process of adding all these tweets into their respective lists? It just seems kind of long to have to do it by hand.

   Link | January 3rd, 2012 at 9:36 am

5. *Hywel M* wrote:

   Thanks very much. Excellent timing for me as I was looking how to do this – but with Welsh language tweets. On a quick read I haven't spotted anything that limits this to working only with English – or have I missed anything?

   Link | January 3rd, 2012 at 1:34 pm

6. *mark* wrote:

   very nice posting! it makes me wondering what else is possible. i'll definitely add your blog to my blogroll.

   Link | January 4th, 2012 at 6:14 am

7. *Hywel M* wrote:

   Excellent guide. Just what I was looking for as I'm just starting to explore sentiment analysis. Thanks very much.

   I can't see what 'num_word_features' is doing in the 3rd line of code under Classifier.

   Link | January 5th, 2012 at 5:00 pm

8. *Laurent Luce* wrote:

   @Patrick Thanks for the feedback. I fixed the call to get_word_features(). Regarding apply_features, it says nltk.classify.apply_features in the nltk online book: http://nltk.googlecode.com/svn/trunk/doc/book/ch06.html. It has been working for me so I am not sure what might have changed.

   Link | January 7th, 2012 at 10:53 am

9. *Laurent Luce* wrote:

@Elliott I built a simple web interface to help me classifying the tweets. I also stored the tweets in a Redis DB in 2 different lists: positive and negative. For the article, I am using hard coded list to simplify things.

Link | January 7th, 2012 at 10:55 am

10. *Laurent Luce* wrote:

@Hywel You are correct. The same can apply to other languages. I am actually using that method to classify tweets written in French. I am not satisfied with the results yet because the sample I classified manually is too small. An issue with other languages than English is that it is difficult to find a large corpus of classified tweets.

Link | January 7th, 2012 at 11:08 am

11. *Laurent Luce* wrote:

@ Hywel I fixed the call to get_word_features. I am trimming the number of word features in the real application but I keep it simpler for the post.

Link | January 7th, 2012 at 11:13 am

12. *Elliott* wrote:

@Laurent Luce The web interface is quite a good idea. If anyone is looking for a pre-classified test corpus, I found one here: http://sandersanalytics.com/lab/twitter-sentiment/.

Link | January 7th, 2012 at 3:53 pm

13. *Laurent Luce* wrote:

@Elliott Thanks for the link to that pre-classified test corpus. Here is another one using emoticons: http://www.stanford.edu/~alecmgo/cs224n/twitterdata.2009.05.25.c.zip
I had to manually classified the tweets on my side because they are in French and I didn't find a pre-classified corpus in French.

Link | January 14th, 2012 at 11:12 am

14. *Elliott* wrote:

@Laurent Luce Pour les liens en français, je sais que ça peut sembler un peu arbitraire, mais prendre un corpus en anglais et le faire passer par Google Translate, je vois pas de raison pour que ça marche pas!

Link | January 14th, 2012 at 9:48 pm

15. *Ultraviolet* wrote:

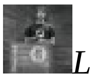
Very useful Laurent! thanks for your post

Link | February 18th, 2012 at 3:25 pm

16. *Phil Day* wrote:

Thanks so much for this, great blog. 😃

Link | March 7th, 2012 at 1:24 pm

17. *Luca* wrote:

The algorithm used by NLTK is highly inefficient. With about 9000 positive tweets and 9000 negative tweets on a 8gb ram quad core server (2ghz each) it takes 45 minutes to be trained (running on pypy). I highly suggest to write your own implementation of the Baesian Classifier; mine takes about 1.5 minutes to train with 9000/9000 tweets on the same machine.

Link | March 13th, 2012 at 11:51 am

18. *Mark* wrote:

What would be a reasonable number of manually classified tweets to use in order to train the classifier in a real world system?

Would there be a point at which manually training more tweets becomes pointless or detrimental?

Link | March 26th, 2012 at 4:02 am

19. *Ravikiran Janardhana* wrote:

The clarity of your post and the brilliant explanation is just amazing ! Great work Sir 😃 !

Link | April 15th, 2012 at 8:19 am

20. *Arbie Samong* wrote:

For those getting the error
nltk.classify has no module apply_features

make sure your nltk version is correct:
PyYAML==3.09
nltk==2.0.1rc1

Link | May 11th, 2012 at 4:01 am

21. *Laurent Luce* wrote:

@Mark. There are few papers out there on training sets. Size of 200k-300k is not uncommon. Not sure where the upper limit is.

Link | June 1st, 2012 at 6:09 pm

22. *Ana* wrote:

Hi…it's great explanation. Thanks for sharing…

Right now I'm working in this kind of work, and having great challenge in converting tweet language to standard form. Any advice? Thanks 😀

Link | June 10th, 2012 at 9:40 am

23. *Thiago* wrote:

Thanks a lot for the tutorial. I followed it with some modifications for my purpose. But, like Luca wrote, it's very inefficient, but it works efficiently. With your base I got 0.81 of accuracy. With a movie review database I got an 0.77. And with another tweet database selected by my group I got a lot less: 0,69 of accuracy. I calculated these accuracy using a 3-fold cross validation.

Link | July 2nd, 2012 at 10:39 am

24. *Jaimin* wrote:

Very good information, and detailed explanation. Thanks for sharing it.

Link | September 3rd, 2012 at 7:37 pm

25. *Pramod Gupta* wrote:

Hi Laurent,

Can you please let me know some good references for Sentiment Analysis especially implementation issues e.g., choosing word features, feature extractions etc.

Thanks

Link | September 13th, 2012 at 2:25 pm

26. *Links* wrote:

This is a really great walk through of sentiment classification using NLTK (especially since my Python skills are non-existent), thanks for sharing Laurent!

Just an FYI- the apply_features function seems to be really slow for a large number of tweets (e.g. 100,000 tweets have taken over 12 hours and still running). any tips to improve the performance?, this is not even half of my training set!

Link | September 14th, 2012 at 1:24 am

27. *Deyan* wrote:

I don’t understand.What is the purpose of test_tweets??

Link | September 24th, 2012 at 6:02 am

28. *Laurent Luce* wrote:

@Deyan: test_tweets is our manually classified list of tuples: words + positive/negative.

Link | October 16th, 2012 at 9:20 am

29. *Bonnie* wrote:

nice blog ! it’s very easy to follow. I wrote my own naive bayes python classifier before but think it’s time to move on to playing with other libraries.

btw does the theme you’re using make it easy to share code? or did you do the highlighting yourself? I’ve been meaning to write some posts where I can share code for people.

Link | October 19th, 2012 at 11:39 am

30. *koda* wrote:

Really nice article. @luca can you share the links or blog post for your implementation ?

Link | November 9th, 2012 at 10:00 pm

31. *Laurent Luce* wrote:

@Bonnie: I use SyntaxHighlighter Evolved.

Link | November 21st, 2012 at 7:35 am

32. *jutky* wrote:

Very helpful article.
Helped me to solve exactly the problem I had.

Link | January 8th, 2013 at 3:59 am

33. *EngineeringDuniya* wrote:

Really useful article!

Link | March 3rd, 2013 at 1:24 pm

34. *Andrew Martin* wrote:

Thanks for this excellent tutorial. I do have to ask though, is there an alternative to the classifier you use? I tried using it, but my dataset is 1.5 million tweets and I just don’t think it’s feasible.

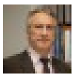
It’s taking far too long.
Is there other ready-build libraries you know of that I could substitute?

Link | August 9th, 2013 at 3:45 pm

35. *Eduard* wrote:

Thanks.
You are inspiring me for writing my bachelor degree project.

Link | May 9th, 2014 at 6:04 pm

36. *Paulo Oliveira* wrote:

Hi Laurent

Is there a corpus of classified tweets in Brazilian Portuguese?

Thanks in advance.

Paulo

Link | May 19th, 2014 at 7:42 am

37. *Santosh Regmi* wrote:

great job. Thanks for article.

Link | November 26th, 2014 at 7:31 am

Leave Your Comment

Name (required)

Mail (will not be published) (required)

Website

Post Comment

Powered by Wordpress and MySQL. Theme by Shlomi Noach, openark.org