Inspiratron.org - Natural language processing, machine learning and cybersecurity

How Sentiment analysis tool for English language was built

by Nikola Miloševi? - Saturday, March 02, 2013

https://inspiratron.org/blog/2013/03/02/how-sentiment-analysis-for-english-language-was-built/

Firstly I would like to anounce that this blog will go English from now on. This is quite a big change, but since topics of blog changed from time I started it, when I was in high school, at that time at some hotmail site service, that was later ported to wordpress.com, then I migrated it to inspiratron, it is better this way. Most of my posts are related to computer security and natural language processing, so there is quite small audience for it in Serbia (we did not have even stemmer built except mine and one with too many rules by Kešelj and Šipka). So I decided to continue with English.

This article is about sentiment analyzer for English language I have built. So let's start with it. So I had to tell first that I had a base, because I built sentiment analyzer for Serbian language as part of my master thesis. So basicly most was done, I had only to port it to other language. Actually it helps a lot, because main algorithm was already written.

Most of text classification problems, and sentiment analysis is a text classification problem, are based on Naive Bayes algorithm. It is simple machine learning algorithm that uses statistics and Bayesian rule to predict classes of some text, or data of any other type. It takes in count total number of documents, total number of documents of each class and word counts for each class. I would not go into details, there are many books written since 19th century about it (from the time Thomas Bayes lived). So I had already implemented algorithm using Naive Bayes for Serbian language. So I just copyed and renaimed file, change DB table names in which learning was writing it's data, and almost that's it.

Almost? Why almost? Because sentiment analysis cannot be implemented just by one algorithm. Actually it can, but there is overhead in learning. I would not get it precise as it is, if I just put data in Naive Bayes, and try to use it. Also I would need much more data in training set. So what are the problems? First of all language flections. This is problem present in almost all natural language processing task, and it can be solved in two ways. One is stemming, the other one is lemmatization. In linguistic morphology and information retrieval, **stemming** is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Lemmatization has a dictionary of all flections and it's roots, so it always returns root. For lemmatization it is needed a large database of flections, but for stemmer, only few rules (about 100-500, depending on language). Performance is basicly the same (gain that they add to machine learning of NLP tasks). So logical chose was to use stemmer. Most used and well known stemmer for English language is Porter's stemmer, and many implementations can be found on Porter's webpage. So I used implementation in PHP and before going to Naive Bayes, I did stemming. Then there are word's that are very often and does not add value to sentiment analysis. This words are in NLP called stopwords. I had corpus of stopwords from Natural Language Processing online class held by professor Manning and Jurafsky from Stanford University on Coursera. So I little modify it, and used it (they have to be filtered from text, before stemming). Also there is problem with negation handling, because it is quite big sentiment switcher. So I also added NOT_ prefix to each word after no, not, and similar negation words, and before next punction mark. In this way algorithm will also better and easier learn from smaller data. Then it was time to add some training data.

I had a corpus of 1000 positive and 1000 negative movie reviews from IMDB, but to add it by hand looked quite painfull. So all this was implemented as API, you can learn or get sentiment from API call using JSON. Also stemming is implemented as API call. Then I could write separate application that would read file by file, and add it to learning algorithm, and teach it. So I wrote a simple application in C# that was reading pos and neg folder from my PC, file by file and send it to the API and teaching requests. The files was too long for JSON, so I made it read first 200 words, make from it text with postive or negative context (depending on location of file) and send it to API. Programming took around 2 hours, and there was few hours working of algorithm, since I was all executing in one thread, also I did not want to server ban me, thinking I was doing some flooding attack, so there was even a short wait between request sending. But still it was much much faster, counting programming overhead, then to enter so many texts by hand.

At the end I had a corpus with more then 600 000 word enties (in words table, that contains word, word count in some sentiment, and sentiment, so each word could have 2 entires - positive and negative). Quite largde corpus, and it worked. Almost perfectly. You can try it : http://www.inspiratron.org/EnglishSentiment.php

Just write sentence in English, and it will show the sentence is positive or negative.
