Sentiment Analysis using Textblob and VSM

Duy Quy VO

June 2021

1. Introduction

With the rapid development of modern technology, Deep Learning is one of the fastest growing branches. One of the easiest applications to come across is Sentiment Analysis. Sentiment analysis can be encountered when customers send feedback about products or services to suppliers, based on customer feedback analysis to evaluate service quality. This is Sentiment Analysis.

With the rapid development of modern technology, Deep Learning is the fastest growing branch. One of the easiest applications to come across is Sentiment Analysis. Sentiment analysis can be encountered when customers send feedback about products or services to suppliers, based on customer feedback analysis to evaluate service quality. This is Sentiment Analysis.

For example, if we want to evaluate the positive/negative and objective level of a dialogue comedy, we need to calculate the positive/negative and objective index of each word in the sentence. Then calculate the total index of each sentence and then the whole document.

The **Textblol** library provides us with built-in functions to quickly calculate the required slope. Apply VSM (Vector Space Model) in parallel to find the document that is close to the document we require and get the positive/negative and objective index from that document.

2. Vector Space Model(VSM)

In short, the Vector space model is an algebraic model that represents textual information as a vector, the elements of which represent the importance of a given vector. word and also its Bag of words in a document.

This model represents text as points in n-dimensional Euclidean space, each of which corresponds to a word in the set of words. The *i* element, which is d_i of the text vector, indicates the number of times the *i* word occurs in the text. The similarity of two texts is defined as the distance between points, or the angle between vectors in space.

Each word in the vector space will have a weight, there are many different ranking methods.

3. Benchmarks

For the benchmarks, I use 50 comedy texts to read as a database. These documents have been through the steps of data cleaning, removing low-meaning words, numbers, etc. Using the function available in the Textblob library to calculate the polarity and subjectivity of 50 documents.

From a new document, I clean the data of the document. Apply VSM to the 30 most used words of the document and find the document that matches the text you just entered.

I compare parallel and non-parallel search running algorithms.

CPU Time Sum of time on all threads in ms.
Real Time A real time from the beginning of the algorithm until the algorithm is finished in ms.

	No Parallelization		Parallelization	
	CPU Time [ms]	Real Time [ms]	CPU Time [ms]	Real Time [ms]
Mean time:	1	4176	5	4175

4. Conclusion

Threading is not always quicker and in many cases can be slower (such as the case seen here). There are plenty of reasons why but the two most significant are

Creating a thread is a relatively expensive OS operation. Context switching (where the CPU stops working on one thread and starts working on another) is again a relatively expensive operation.

Creating 200 threads will take a fair amount of time (with the default stack size this will allocate 200MB of memory for stacks alone), and unless you have a machine with 200 cores the OS will also need to spend a fair amount of time context switching between those threads too.

The end result is that the time that the machine spends creating threads and switching between them simply outstrips the amount of time that the machine spends doing any work. You may see a performance improvement if you reduce the number of threads being used.

Multithreading where you have more threads than cores is generally only useful in scenarios where the CPU is hanging around waiting for things to happen.