

Re: word aware distance algorithm

Source: <http://coding.derkeiler.com/Archive/General/comp.programming/2007-06/msg00597.html>

- *From:* rem642b@xxxxxxxxxx (Robert Maas, see <http://tinyurl.com/uh3t>)
 - *Date:* Fri, 29 Jun 2007 23:48:05 -0700
-

From: rouadec <roua...@xxxxxxxxxx>
I'm ... trying to compute the similarity between two sentences ...
right now I'm using a levenshtein distance ...

You're talking at cross purposes. Computing similarity, and computing distance, are nearly opposite things. Computing distance usually implies a metric space, and is easy to define in a consistent precise way, usually Cartesian distance. Computing similarity is very different and rather difficult to define in any reasonable consistent manner. For example, consider:

aaaaaaabbbbbbb
aaaaaaa

Are those two strings 50% similar, or some other fraction similar, or do you get 100% in one direction but only 50% the other direction?

aaaaaaabbbbbbb
aaaaaaacccccc

Are those two strings 50% similar, or what? These two strings are less similar than the first pair, so if the first pair were 50% similar then these two must be less similar, so how do you compute that???

I guess it's apparent by now that I prefer vector distance methods.

but the levenshtein distance isn't word aware and this is causing issues when matching data, ie something like 'xxxxxx' is widely different in levenshtein terms to 'xxxxxx - yyyyyy' but not so for humans ;)

Any idea about a metrics which will give a greater weight to sentences with similar words (even better if it also weight the position of similar words) ?

Suppose you computed the trigram-frequency vector, and the measured

Re: word aware distance algorithm

the Cartesian distance between these vectors. Would that be close to what you need, or would an additional tuning of it be needed?

For a large corpus of sentences to be compared, for example a million different sentences, are you comparing only a few of the half-trillion possible pairs, or do you need in effect to compare nearly *all* of those possible pairs? If you need to compare nearly all pairs in order to perform clustering of sentences, it may be worth the extra work of pre-processing each sentence into an explicit vector, so that Cartesian distance between those pre-computed vectors can be performed very efficiently on vector processors. Something like my ProxHash may be useful, where each vector of trigrams is transformed to a much smaller vector (typically only 15 ordinates) whose pairwise distances can then be VERY quickly computed. Also, precomputed ProxHashes allow successive clustering on smaller numbers of ordinates, which greatly reduces the number of comparisons that need to be done.

.