Linguistically Fuelled Text Similarity

Björn Andrist
KTH CSC
Stockholm
andrist@kth.se

Martin Hassel
KTH CSC
Stockholm
xmartin@kth.se

Abstract

This paper describes TEXTSIM, a system for determining the similarity between texts. Further, we show the results of a comparison between two various configurations of TEXTSIM; one with and one without any deeper linguistic analysis. To evaluate and compare the two models of TEXTSIM we used two sets of examples: a set of automatically generated examples and a set of examples acquired from two assessors. Depending on the type of documents, we found the model using linguistic analysis to perform equally well or better than the model not using linguistic analysis.

1 Introduction

Many NLP systems utilize statistics for various tasks, and in many cases it is not desirable that several instances of the same text are represented in these statistics. In order to detect similar texts a similarity system named TEXTSIM has been implemented and evaluated. We aimed to develop TEXTSIM in such a way that it mirrors the human perception of similarity between documents. Examples of such systems have successfully been implemented with the use of two mathematical notions called *resemblance* and *containment* (Broder, 1997).

Since the context in which TEXTSIM is to operate is restricted to documents written in natural language, the use of linguistic analysis is quite intuitive in an attempt to improve the technique. In this project we have focused on texts written in Swedish. For linguistic analysis the grammar checker Granska (Domeij et al., 1999) has been utilized for *tokenizing*, *lemmatizing* and *part-of-speech tagging*.

Our hypothesis is that a system based on tokens, lemmas, and tags would outperform a system based solely on lexical tokens. This applies principally when comparing documents where lemmas and tags would provide additional information e.g. when documents are modified by synonyms or by tense.

2 Similarity between texts

We are interested in finding the document y in a document set D which is most similar to some document x, i.e.:

$$y = \underset{d \in D}{\text{argmax}}\, sim(x, d)$$

where sim is some arbitrary function that quantifies the similarity between two documents. In order for our similarity function sim to be useful, we claim that a *transitive* relation $\mathcal{R}$ on a document set D, given by $a\mathcal{R}_x b \Leftrightarrow sim(x, a) > sim(x, b)$ where $a, b \in D$, must exist for every document $x \in D$. The transitive property of $\mathcal{R}$ implies that if $a\mathcal{R}_x b$ and $b\mathcal{R}_x c \Rightarrow a\mathcal{R}_x c$. This property was of great importance when designing TEXTSIM, because it enabled us to construct reliable training data.

2.1 Training data

In order to mirror human perception of similarity between documents the training and evaluation data

should preferably be comprised of examples from human subjects. Furthermore, the training data ought to be reliable and preferably easy to generate. In this context, reliable data is data from a subject who produces approximately the same results when repeating a test. We used the following method for generating training examples. Give a subject three documents: a reference document R and two other documents A and B. Thereafter the subject specifies which of the documents A or B that he/she considers to be most similar to R.

This method generated reliable data and it was also possible to automatically generate large amounts of training examples by using hypotheses about the similarity between documents.

2.2 Resemblance and containment

Resemblance and containment quantify the similarity between two documents. The degree of resemblance and containment is represented by a real number $x \in [0, 1]$. A resemblance value close to 1 indicates a high level of similarity between two documents. A containment value close to 1 indicates that one of two documents nearly contains the other document. Resemblance and containment are clearly defined in (Broder, 1997). However, a short summary is given below.

Consider a document as a sequence of tokens. A token could be a word, a punctuation mark, etc. We call a subsequence of a document a *shingle*. Further, a shingle of a specific size w will be referred to as a *w-shingle*.

Let $S(A, w)$ represent the multiset of w-shingles for a document A. Given two documents A and B and a fixed shingle size of w, the resemblance can be computed by dividing the number of w-shingles in common with the total number of distinct w-shingles:

$$r_w(A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w) \cup S(B, w)|}.$$

To compute the containment of A in B, divide the number of w-shingles in common, with the number of w-shingles in A:

$$c_w(A, B) = \frac{|S(A, w) \cap S(B, w)|}{|S(A, w)|}.$$

Computing the resemblance and containment is a very time consuming task for large document sets and therefore we make an unbiased estimate of the resemblance and the containment. The estimate is performed by comparing sketches of documents, created by hashing a subset of the sequences of strings contained in the documents.

2.3 Features, weighting and learning

When comparing two documents, A and B, the following features are of interest: i) the resemblance of A and B, $r(A, B)$, which determines to what degree the documents resemble one another. ii) the containment of A in B $c(B, A)$, which is required to detect those documents which are contained in previously processed documents. iii) the containment of B in A, $c(B, A)$, which is required to detect documents which contain previously processed documents.

In order to build a system capable of learning from the training instances comprised of the three documents, R, A, and B, a perceptron (Mitchell, 1997) is used. When solely using tokens for document comparison, the perceptron needs six input nodes: $r(R, A)$, $c(R, A)$, $c(A, R)$, $r(R, B)$, $c(R, B)$, and $c(B, R)$. When using tokens, lemmas and tags for document comparison, the perceptron needs a total of 18 input nodes (six for each type). In either case, the perceptron only has one output node, where a positive output node indicates that document R is more similar to A than B.

Given a document R and a document set D, the document in D which is most similar to R can be found by iterating over D. In each iteration, the output of the perceptron determines the most similar document.

3 Evaluation of the text similarity system

We have evaluated two configurations of TEXTSIM: one based solely on tokens and one based on tokens, lemmas, and PoS tags. In order to evaluate and compare the models we used two sets of examples. First a set of automatically generated examples, and second, a set of examples acquired from two assessors. Furthermore, two evaluation experiments were conducted. The first experiment used a generated example set for training as well as testing.

The second experiment used the generated example set for training and the assessor example set for testing.

3.1 System configurations

Using TEXTSIM, two models for determining the similarity between documents were configured:

Model 1 was based on the perceptron with 6 input nodes. The input was collected from the resemblance and the containment of documents in token form.

Model 2 was based on the perceptron with 18 input nodes. The input was collected from the resemblance and the containment of documents using token, lemmas, and tags.

Both models were configured using 3-shingles where every fourth shingle formed the sketches. The learning rate and the number of maximum epochs was determined by evaluating various values of these parameters. The evaluation was conducted by means of one representative example set containing 400 examples.

3.2 Data sets

The reference documents that were used during training and testing were collected from two sources. The first was KTH News Corpus (Hassel, 2001), a digital corpus containing news articles from the web sites of widely spread Swedish newspapers. The second being the results from an experiment using Trace-it (Severinson-Eklundh & Kollberg, 1996) where a group of writers were asked to write short essays. Trace-it is a revision analysis tool that automatically stores a revision of a text after every modification of the text.

To automatically generate variants of the reference documents we substituted words with synonyms using Synlex and replaced words with misspellings using Missplel.

Missplel is a program for generating human like spelling errors (Bigert et al., 2003). It can be configured to produce a great variety of spelling errors. One can also configure Missplel to only substitute words so that the PoS tag of a word is changed and vice versa.

Synlex is an on-line Swedish dictionary of synonyms constructed by having people vote for the synonymity of possible synonym pairs (Kann & Rosell, 2005). We automatically replace a certain amount of the words in a document by synonyms in Synlex. Most likely this will lead to a lot of inappropriate replacements due to the ambiguity of many words. However, this method was deemed sufficient for generating training data.

3.3 Example types

The data used for training and evaluation was divided into the following five main types:

Type 1.a: A and B were generated from R with Missplel using the same type of errors, but with different amounts of misspelled words.

Type 1.b: A and B were generated from R with Synlex using different amounts of synonyms.

Type 2: A and B were generated from R by Missplel using different types of errors though the total sum of errors in A and B was equal.

Type 3: A was generated from R by substituting a certain amount of words with synonyms fetched from Synlex. B was generated from R with rearranged paragraphs and sentences.

Type 4: A was generated from R by introducing spelling errors with Missplel. B was generated from R by replacing a certain amount of words with synonyms from Synlex.

Type 5: A was generated from R by introducing spelling errors and/or by replacing words with synonyms from Synlex. B originated from R but was expressed in another tense.

Examples of types 1.a, 1.b, and 4 were not difficult to generate while examples of the types 2, 3, and 5 required manual editing.

3.3.1 The generated example set

The generated example set was created in order to acquire large amounts of training and evaluation data. It contained examples of the types 1.a, 1.b, and 4. Documents were sampled from the KTH News Corpus. The samples contained documents containing 200–300 tokens each. 1000 reference documents were used to generate 5850 examples.

3.3.2 The assessor example set

The assessor example set was acquired from two assessors and contained examples of all types, i.e. 1.a, 1.b, 2, 3, 4, and 5. We aimed to find test data that was both difficult to classify and reliable. This is an inconsistency since the less difficult the documents are to classify the more reliable the test data ought to be.

The test was carried out with two assessors. The examples were randomly ordered prior to giving them to the assessors. The assessors were only informed about the context in which the examples would be used. The assessors were given 25 examples, five of each main type. Each example was comprised of a document triple (R, A, B). For each example the assessors were asked to read the documents R, A, and B in order to determine which of the documents A or B that they thought was most similar to R.

3.4 Performance

The performance of the two systems was evaluated and compared with each other using the generated example set as well as the assessor example set. However, consideration should be taken to the fact that not every type of the examples in the assessor example set was included in the training examples. Preferably, all example types should be properly represented in both the training set as well as the test set.

3.4.1 Performance on generated examples

The reported error rates are the mean values of 10-fold cross-validations repeated ten times using the generated example set. Table 1 presents the error rates for each type and the overall error rate when evaluating the entire example set.

Type	Model 1	Model 2	Examples
Type 1.a	14.5%	13.3%	1970
Type 1.b	0.4%	0.4%	1940
Type 4	29.4%	5.6%	1940
All types	14.2%	8.0%	5850

Table 1: Error rates for each type of text for Model 1 (tokens) and Model 2 (tokens, lemmas and PoS tags) respectively.

The most significant difference between Model 1 and Model 2 appears in the examples of Type 4. This corresponds well with our hypothesis. Model 1 has shown not to be efficient in separating document pairs with the same amount of token overlap. Further, the differences in performance between Model 1 and Model 2 are negligible on the examples of types 1 and 2. Changes that have been initiated in the examples of types 1 and 2 are mirrored equally in the token representation, the lemma representation and the tag representation of the documents. Thus the tags and lemmas are superfluous and do not improve the performance of Model 2.

The reason for the substantial differences between Type 1 and Type 2 is a result of the more accurate data obtained from the method using Synlex compared to the method using Missplel. However, our attention is not drawn to the differences in values between the various types but to the differences between Models 1 and 2.

Finally, a significant difference between Models 1 and 2 was apparent when using the entire set of generated examples.

3.4.2 Performance on the assessor example set

Model 1 and Model 2 were trained on the generated example set and thereafter compared with the assessors perception of similarity. The assessors agreed on 19 of the 25 examples. As in the evaluation using the generated example set our attention is drawn to the differences between Models 1 and 2, and not the absolute error rates. By using the examples that the assessors agreed upon, the error rate for Model 1 is 6/19 while the error rate on Model 2 is 4/19.

Type	Assessors	Model 1	Model 2
Type 1	4/5	1	0
Type 2	5/5	1	1
Type 3	1/5	0	0
Type 4	5/5	2	0
Type 5	4/5	2	3
Total	19/25	6	4

Table 2: The number of errors for each type of text for Model 1 and Model 2 respectively. The assessors column indicates the agreement between the assessors.

4 Conclusions

The system using tokens, lemmas, and PoS tags outperformed the system using only tokens, both when evaluation was performed on the generated example set and the examples acquired from assessors. However, this is at the cost of performance with respect to memory and speed. The most obvious differences between the systems appeared on documents with the same amount of token overlap but with different types of variations.

The evaluation was performed on a small amount of assessor examples. A more reliable method would have been to acquire a substantially larger amount of assessor examples in order to facilitate the performance of a stratified k-fold cross-validation on the assessor examples without the generated example set.

References

J. Bigert, L. Ericson, and A. Solis. 2003. Missplel and AutoEval: Two generic tools for automatic evaluation. *Proc. of NODALIDA 2003.*

A. Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES 97)*, 21–29. IEEE Computer Society, 1998.

R. Domeij, O. Knutsson, J. Carlberger, and V. Kann. 1999. Granska – an efficient hybrid system for Swedish grammar checking. *Proc. of NODALIDA 1999.*

M. Hassel. 2001. Automatic construction of a Swedish news corpus. *Proc. of NODALIDA 2001*.

V. Kann and M. Rosell. 2005. Free Construction of a Free Swedish Dictionary of Synonyms. *Proc. of NODALIDA 2005.*

T. Mitchell. 1997. *Machine Learning.* McGraw Hill.

K. Severinson-Eklundh, P. Kollberg. 1996. A computer tool and framework for analysing on-line revisions. *Levy, C. M. and Ransdell, S. (eds), The science of writing: Theories, Methods, Individual Differences, and Applications, 163–188, Lawrence Erlbaum Ass.*