Measuring the sentence level similarity

Ercan Canhasi
Faculty of Computer Science
University of Prizren
Prizren, Kosovo
ercan.canhasi@uni-prizren.com

Abstract - This article describes a method used to calculate the similarity between short English texts, specifically of sentence length. The described algorithm calculates semantic and word order similarities of two sentences. In order to do so, it uses a structured lexical knowledge base and statistical information from a corpus. The described method works well in determining sentence similarity for most sentence pairs, consequently the implemented method can be used in computer automated sentence similarity measurements and other text based mining problems. We encapsulated the implemented algorithm in a .NET library, to simplify the task of calculating sentence similarity for end users.

Keywords - Sentence similarity, short texts similarity, short texts comparison, semantic nets, corpus

I. INTRODUCTION

Sentence similarity measures are becoming increasingly more important in text-related research and application areas such as text mining, information extraction, automatic question-answering, text summarizing, text classification, machine translation and natural language processing (NLP). The goal is to implement an effective method to compute the similarity between very short texts, usually about one sentence length. These computed sentence similarities could be useful in Internet-related applications as well. For an example, in improving Web page retrieval effectiveness, where titles are used to represent documents instead of the actual text of the web page or for achieving more accurate Web page search results by comparing the search query with Web page titles. Another example could be retrieving or searching images from the web in which the image title or the short text surrounding the image, could be use to categorize and find the best matching results with higher precision than using the whole document in which the image is included. In document categorization instead of comparing the whole body of the document (which could be very large) the computed similarity of the document title and/or short description could be used to semantically categorize documents. These mentioned examples of usage show that the computing of short text similarity is becoming more and more important in text-related research and applications.

The traditional methods for detecting similarity between documents are based on analyzing and counting shared words. These methods are useful when dealing with large documents, because similar large documents usually contain a high degree of similar words. In short texts, there are not enough words for these methods to be useful. Two sentences could have similar meaning but are composed out of entirely different words. This is mostly due to the complexity and flexibility of natural language. Therefore a different method is needed to compare short texts.

This paper aims to develop and implement a dynamic method that can be used in applications that require an efficient and fast computation of short texts similarity. It is expected to be dynamic in only focusing on the sentences of concern, does not require user's manual input and is easily adaptable across potential application domains. It will be presented as a Microsoft .NET library, with an easy to understand and rich enough access interface. The source is expected to be well document and available publicly.

The remainder of this paper is laid out as follows. The next section reviews some related work. In Section 3 we describe the used external tools and knowledge databases, the usage of the information retrieved from these databases and the overall method for computing the similarity between short texts. Section 4 provides the method implementation description and the overall review and documentation of the developed .NET library with usage examples. Section 5 provides the testing results and computed similarity for a set of sentences obtained from various sources. Finally, Section 6 provides the short summarization, draws some conclusions, and proposes further related work.

II. RELATED WORK

There is extensive literature on measuring the similarity between long texts [2], [3], [4], [5], and increasingly more articles about measuring short texts similarity [6], [7], [8]. This section briefly describes some other techniques that ate related to our work. We consider three major categories of related methods: surface-matching methods, corpus-based methods and query-log methods.

A. *Word overlaps or Surface-matching methods*

Surface matching methods or also known as the "bag of words" methods, are mostly used in Information Retrieval systems. Given two text segments *(n, m)*, the idea of surface-matching methods is based on the number of words that occur in both text segments. This technique relies on the assumption that more similar texts share more of the same words, which is efficient only if both texts are of sufficient lengths. Let's say N and M are the sets of words in texts n and m. Common similarity measures presented in [9] are listed as follows:

Matching	$\lvert N \cap M \rvert$
Overlap	$\lvert N \cap M\rvert / min(\lvert N\rvert, \lvert M\rvert)$
Dice	$2 \lvert N \cap M \rvert /(\lvert N \rvert \cap \lvert M \rvert)$
Jaccard	$\lvert N \cap M \rvert / \lvert N \cup M \rvert$
Cosine	$\lvert N \cap S \rvert/\sqrt{\lvert N \rvert \times \lvert M \rvert}$

Instead of using the described set of operations, we could also describe texts as vectors in n-dimensional space, where n is the number of all meaningful words in a natural language, represented as a precompiled word list. It is therefore straightforward to extend these measures as operations with vectors, where each element may represent the frequency of the word. As mentioned before, these techniques are effective and usable only if we compare very long texts or texts with sufficient number of words. With short texts these techniques would have three obvious drawbacks:

1. Two sentences could have similar meaning but are composed out of entirely different words.

2. Surface-matching methods usually exclude function words like the, of, an, etc., because they are common to all documents and are therefore not very helpful. For computing sentence similarity these function words cannot be ignored, because they carry structural information, which is useful in interpreting sentence meaning.

3. The sentence representation as vectors is not very efficient. The vector dimension n is huge, compared to the number of words in a sentence. Therefore the resulting vector would have many empty or null components.

There are also several possible extensions of these surface-matching methods, for an example two words can be matched if they are synonyms according to some thesauruses like WordNet. Another possible extension of surface-matching techniques are pattern matching methods, which are commonly used in conversational agents and text mining. Pattern matching methods includes incorporating local structural information about words in the predicated sentences. A meaning is represented as a limited set of patterns, where each pattern is represented using regular expressions (usually consisting of parts of words and various wildcards). The similarity is then calculated using various pattern matching algorithms. There are some obvious and severe drawbacks of this technique. One of them is that this technique requires a complete pattern set for each meaning to avoid mismatches. At present, it is not possible to prove that a pattern set is complete. This leads to the conclusion that there is no automatic method for compiling such a pattern set and requires manual compilation, which is a major drawback.

Although different statistics and extensions for surface-matching methods have their own strengths and weaknesses, their quality of measuring sentence similarity is usually very poor and unreliable.

B. Corpus-based methods

One method to overcome the weakness of surface-matching methods is to leverage the information derived from a large corpus, such as Brown corpus or often even the Web. Corpus-based methods use this statistical information of words from a huge corpus, to calculate the sentence similarity. There are two popular corpus-based methods: the latent semantic analysis (LSA) [12], [13], [14] and the Hyperspace Analogues to Language (HAL) model [10], [11].

LSA was firstly proposed by Landauer (1998). In LSA, term co-occurrences in a corpus are captured by means of a dimensionality reduction operated by a singular value decomposition (SVD) on the term-by-document matrix T representing the corpus. The SVD decomposes the term-by-document matrix into three matrices $T = U \sum_k V^T$ where $\sum_k V^T$ is the k-dimension diagonal matrix containing the k singular values of T, and U and V are column-orthogonal matrices. To reduce the dimensionality, the diagonal singular matrix is then truncated by deleting small singular values. To re-compose the original term-by-document matrix, all three matrixes (U, V and T) are multiplied together. LSA can be used as a technique to overcome some of the drawbacks of surface-matching methods, which use standard vector space model. It reduces the high dimensionality and sparseness. The LSA similarity is computed in a reduced dimensional space, in which second-order relations among terms and texts are exploited. In this reduced dimensional space we have several options, we could just measure the similarity with the standard cosine similarity or some other similar similarity measure, or we could form a vector for each of the two comparing sentences and then measure similarity by computing the similarity of these two vectors. LSA does have a few drawbacks, one of them is that the dimension of term-by-document matrix T is fixed and limited to a few hundred, because of the computational limit of SVD. This causes that the vector is also fixed and is this likely to be a very sparse representation of a short text. LSA also ignores any syntactic information from the two comparing sentences. By taking these drawbacks into consideration, we conclude, that LSA is more appropriate for large texts than short texts.

The second most popular corpus-based method is the Hyperspace Analogues to Language (HAL) model, also known as semantic memory. It was firstly introduced and developed by Kevin Lund and Curt Burgress in 1996. Like the LSA, HAL relies on the basic premise that words with similar meaning repeatedly occur closely. For an example in a large corpus one could expect to see the words mouse, dog and cat appear often close to each other. These lexical co-occurrences are then used to produce a high-dimensional semantic space. In this semantic space, words are represented as points, and the position of each word along the axes is related to the word's meaning. Once this space is constructed, a distance measure can be used to determine relationships between words. HAL then builds a word-by-word matrix based on word co-occurrences within a moving window of a predefined width. This window moves over the entire corpus and records weighted lexical co-occurrences - words closer to the target word are given a higher weight than words farther away. These resulting weights are then recorded in an $n \times n$ result matrix (n is the number of words in the given vocabulary) with one row and one column for each unique word appearing in the corpus. Next we form a vector representing each word in 2n dimensional space by concatenating the transpose of a word's column to its row. Using these word vectors, we then form the sentence vector by adding together the word vectors for all words in the sentence. The similarity between two sentences is the calculated by using a metric such as Euclidean distance or something similar on the constructed sentence vectors. However the author's of the article "Explorations in context space: Words, sentences" [15] show that HAL was not as promising as LSA in computing the similarity between short texts. One of the HAL's drawbacks is due to the building of the memory matrix (the matrix badly captures sentence meanings) and its technique of forming sentence vectors. The sentence vector becomes diluted when a large number of words are added to it. This leads to the conclusion that, as with LSA, HAL is more appropriate for large texts than short texts [1].

C. Query-log Methods

Search engines like Google or Bing process millions of search queries per day. The produced search query logs have become a great resource for measuring similarity between short texts. This information is then used for search query suggestion generation, which is becoming more and more accurate and useful. An example of generating query substitution is presented in the paper "Generating query substitutions" [16]. In this paper, their goal is to generate alternative query suggestions to a given query. Firstly they generated suggestions, which were based on the information about whether the target query and suggestion had appeared in the same session query log. These suggestions were then ranked based on a regression model trained with three types of features. The first type of feature is the usage of surface characteristics, such as number of characters of the query and resulting suggestion. The second type of feature is the usage of substitution statistic, such as the mutual information between the query and suggestion using their distribution in the logs. The final feature is the usage of syntactic difference between the query and the suggestion such as Levenshtein edit distance. Due to the fact, that candidate suggestions were selected using substitution statistic, they found that the only useful feature was based on syntactic differences. Compared to the other described related methods, the method proposed in "Generating query substitutions" [16] did not aim to provide a similar metric, but more of a generation task which is bound to measuring the similarity between the query and the suggestion. The limitation of this task as it was presented in the paper is that the coverage for pairs of short texts segments is limited, because subsets of the words in both segments must appear in the same user session query logs.

D. Probabilistic models

Probabilistic models are based on idea of estimating the probability that one sentence is a translation of another. This translation probability then serves as the basis of the similarity score for pairs of sentences. Statistical machine translation systems aim to generate high-quality translations of sentences between natural languages. Such systems make use of parametrized statistical language models of both source and target language, and a parametrized statistical translation model that estimates the probability that a given target sentence is a translation of the source sentence. Given these models and a parametrization, the system searches a space of possible translations and returns the sentence with the highest probability. In their paper, they propose using statistical translation models in much the same manner to estimate the probability that one sentence is a translation of another. However, as our problem is different from normal translation problems (both sentences are in the same language), we can make some assumptions. We will now briefly summarize their path from more general model to a model adequate to our problem. We will also add some motivation and description of some specific terms.

They start with IBM's Translation Model 1. IBM Model 1 is a generative model. Generative modeling means breaking up the process of generating the data into smaller steps, modeling the smaller steps with probability distributions, and combining the steps into a coherent story ([5]). They provide following similarity function, based on IBM model 1:

$$S(Q,R) = \frac{1}{(|R|+1)^{|Q|}} \prod_{i=1}^{|Q|} \sum_{j=1}^{|R|+1} P_t(q_i|r_j)$$

$|R|$ is the length of the sentence R

$|Q|$ is the length of the sentence Q

$P_t(q_i|r_j)$ is a probability that j-th word in R is a translation of i-th word in q

Then they made some additional assumptions. The original model assumes that each sentence has a special null term at position 1; this is the reason that the summation iterates through |R| +1 terms. The null term is used to represent the fact that the current term in Q doesn't align to any terms in R.

With that in mind, they make the distributional assumption that $P_t(q_i|r_j) = P(q_i|C)$, where C is the background model inferred from the collection as a whole. This precedes form intuition that – in the absence of any other evidence – an unaligned word is likely to be present in a sentence with a probability equal to its overall probability in the more generalized background language model.

The probability of aligning to the null term dictates the influence of the background language model on the resulting translation. Because IBM Model 1 assumes that all reordering are equally likely, the probability that a term in Q will align to the null term is $\frac{1}{|R|+1}$. Then they generalize the original model by assuming there exists μ null terms in each sentence, where μ is a non-negative integer. This results in each sentence having length |R| + μ, where |R| is the number of non-null terms in R. This model can be described as:

$$S(Q,R) = \frac{1}{(|R| + \mu)^{|Q|}} \prod_{i=1}^{|Q|} \left[\sum_{j=1}^{\mu} P(q_i|C) + \sum_{j=\mu+1}^{|R|+\mu} P_t(q_i|r_j) \right]$$

μ is the number of null terms in each sentence

$P_t(q_i|C)$ is a probability that i-th term in Q appears in some background model C

They simplify the model further, with assuming that each word translates to itself; that is $P(q_i|C) = 1$ if $q_i = r_i$.

This results in the following form:

$$S(Q,R) = \prod_{i=1}^{|Q|} \frac{tf_{q_i,R} + \mu P(q_i|C)}{|R| + \mu}$$

$tf_{q_i,R}$ is the frequency of i-th word in the sentence Q in sentence R

Above function is known as language modeling query likelihood ranking function using Dirichlet smoothing parameter μ. With μ=1, we get Berger and Lafferty's Translation Model 0. All models here assume that every term only translates to itself. We extended this model with synonyms and so incorporated a more refined estimate of the true translation probabilities. As parameter μ approaches 0, the model becomes word overlap measure that will likely be good at finding exact matches. At the other extreme, as μ gets large more background terms are allowed, which is likely (and known to be) good at finding topically relevant matches.

They defined similarity spectrum, where at one end there is exact identity and at the other general topic relation. They divided this spectrum into 5 parts: exact match, minor edit, same facts, specific topic match, general topic match. They found out that at the general and specific topic level, query likelihood function with μ =2500 gives the best results. This was expected, because past research has shown query likelihood to be effective at identifying topicality. At other levels the relative performance difference between techniques was small, but Translation Model 0 (μ =1) was consistently the most effective.

III. METHOD DESCRIPTION

This chapter describes the method we used for measuring sentence similarity based on semantic knowledge databases and corpus statistics. It is based on the article titled "Sentence Similarity Based on Semantic Nets and Corpus Statistic" [1] which we used as a basis for our implementation.

A. *Similarity between words*

The basis for calculating sentence similarity is calculating word similarity. For helping us to achieve this task we used a semantic knowledge base. It consists of a hierarchical structure that models usage of the general English language. In this database, words are organized into sets of synonyms called synsets. Each synset represents a node in the hierarchical structure.

To calculate similarity between two words we must first determine the length of the path between the two synsets containing these two words. There are three possible cases for path length between two words:

1. both words are in the same synset

2. words are not in the same synset, but their respective synsets contain one or more common words

3. words are not in the same synset and their synsets do not contain any common words

In the first case, both words have the same meaning, so we assign the path length between them to 0. The second case implies that both words share some common features; in this case we assign the path length to 1. Finally, in the third case, the words do not have the same semantic meaning, so we find the shortest path length between the synsets containing each word [26].

We must also take into account that words closer to the root of the hierarchical structure have more general concepts than words further from the root. Because of this we must determine the depth of the subsumer of both words. Subsumer is the node, closest to the root of the hierarchical structure, on the path between two synsets, containing our respective words.

The final formula for calculating words similarity is:

$$s(w_1, w_2) = e^{-\alpha l} * \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

where $w_1$and $w_2$are two words we want to calculate the similarity of, α is in the range of [0, 1], β is in the range of (0, 1], l is the determined path length between both words and h is the depth of the subsumer of both words. Factor α determines how much path length contributes to the overall word similarity. As α increases, so does the contribution of path length. Similarly β factor determines the contribution of subsumer depth. Contrary to α, as β increases, the relative contribution of subsumer depth decreases. The value of $s(w_1, w_2)$ is in the range of [0, 1].

B. *Similarity between sentences*

In order to calculate similarity between two sentences T1 and T2, each containing their own set of words, we must first define a joint word set of both sentences T. This joint word set contains all the distinct words from both sentences. We keep the word form as it appears in the sentence, for example: cat, cats, mouse and mice are four distinct words and are all included in our joint word set. We also include all function words, because function words contain syntactic information that we cannot ignore if the text is very short, or even sentence length.

1) *Semantic sentence similarity*

To calculate the semantic sentence similarity we must first calculate the lexical semantic vector of both sentences, denoted as s_i. Each entry in this vector corresponds to a word in the joint word set. To calculate the entire vector for the first sentence s1, we do the following for each word w in the joint word set:

1. If the first sentence contains w, the entry in the semantic vector š is set to 1. Then we multiply this

value with the value for w from the Corpus statistics to the power of two, and so

$$s_i = \hat{s} * I(w)^2$$

2. If the first sentence does not contain w, we calculate the similarity between w and every word in the first sentence to find the most similar word, denoted as ~w. If the similarity exceeds a pre-set threshold, the entry in the semantic vector is set to the calculated similarity; otherwise the entry is set to 0. Then we multiply this value with the value for w from the Corpus statistics and with the value for ~w from the Corpus statistics, and so

$$s_i = \hat{s} * I(w) * I(\sim w)$$

The final value for semantic sentence similarity is the cosine coefficient between both lexical semantic vectors:

$$S_s = \frac{s_1 * s_2}{\|s_1\| * \|s_2\|}$$

The value for Ss is in the range [0, 1].

2) Syntactic (word order) sentence similarity

After we have calculated the semantic sentence similarity, we must calculate the word order similarity of both sentences. This is important, because different word order can significantly change the meaning of a sentence. For example: Dogs can swim, but chicken cannot. We can change the word order so the sentence looks like: Chicken can swim, but dogs cannot. As a result, the meaning of both sentences is completely different. We must therefore form word order vectors for both sentences, namely r_1 and r_2. For the first sentence, this is achieved by doing the following for each word w in the joint word set:

1. If the first sentence contains w, we fill the entry in r_1 with the corresponding index of w in the first sentence.
2. If the first sentence does not contain w, we find the word from the first sentence, which is most similar to w. This word is denoted as ~w. If the similarity is greater than the preset threshold, we fill the first sentence's vector entry with the corresponding index of ~w in the first sentence. If the similarity is not greater than the threshold, we fill the vector's entry with 0.

Threshold is important because we are calculating the word similarity of different words and therefore the similarity measures could be very low. Since that means the words are not similar, we do not want to introduce such noise into our calculation. If we increase the threshold, we could potentially introduce more noise to our calculations, which is not desirable. We repeat the process for both sentences, so we obtain word order vectors for both sentences. The final value for the word order similarity measure is calculated using the following formula:

$$S_r = 1 - \frac{\|r_1 - r_2\|}{\|r_1 + r_2\|}$$

Word order similarity measure between two sentences is calculated as a normalized difference of word order. The measure is sensitive to the distance between two words of the word pair. If the distance increases, the measure decreases.

3) Final formula for calculating sentence similarity

The final formula for calculating sentence similarity is a combination of semantic sentence similarity and word order similarity measure. The final formula looks like this:

$$S(T_i, T_j) = \delta * S_s + (1 - \delta) * S_r$$

where factor δ determines the relative contribution of semantic similarity and word order similarity measure to the overall sentence similarity. Factor δ should be in the range of (0.5, 1], because semantic similarity is more important than word order similarity [25]. The value of $S(T_i, T_j)$ is in the range of [0, 1].

C. Corpus statistics

Corpus statistics is important for calculating sentence similarity, because we need to weigh the importance of different words that occur in a sentence [24]. Different words contribute differently to the meaning of a sentence. This is especially important, because we need to keep all function words (for example *"of", "the", "as" …)*, which contribute a lot less to the meaning of the sentence than other words. Words that occur more frequently in a corpus contains less information than words that occur less frequently [3].
Therefore the information content of a word has to be derived from the probability that a word is contained in a corpus. Information content of a word is calculated as

$$I(w) = 1 - \frac{\log(n+1)}{\log(N+1)}$$

where *n* is the frequency of the word *w* in corpus and *N* is the total number of words in the corpus increased by 1 to avoid division by zero). Finally, the I(w) value of is contained within the interval [0, 1].

IV. METHOD IMPLEMENTATION

In this section we present methods and models used in our experiments, data used in our measures and evaluation of these methods.

A. Methods

We decided to implement the described method in C# and .NET 4.0 using Visual Studio 2010. As previously described, two databases were used, namely WordNet [17] and the Brown Corpus [18]. We combined and grouped all of our implemented methods in a .NET library, which can then be used in various applications. This section briefly describes the two databases, how we implemented the methods for retrieving the desired data, provides the visualization of the overall flow of the method and describes the final library and the demo GUI.

1) WordNet

WordNet is a large lexical database of English language, developed at Princeton University by a group led by Miller [19]. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms called synsets, each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. In other words synset represents a group of words, in which all words have a similar meaning. All synset also include pointers to other related synsets and thus forming a tree-like hierarchical structure ranging from many specific terms at the lower levels to a few generic terms at the top (see Fig.1 for an example). Each node or synset is represented in one word sense. If a word has more than one sense, it will appear in multiple synsets at various locations in the tree. WordNet defines relations between synsets and relations between word senses. A relation between synsets is a semantic relation, and a relation between word senses is a lexical relation. In this paper we are only interested in the semantic relation. WordNet is also freely and publicly available for download [20]. The version used in this paper is WordNet 3.0, which has 206,941 words, organized into 117,659 synsets. We used WordNet as the main semantic knowledge base for calculating of semantic similarity.

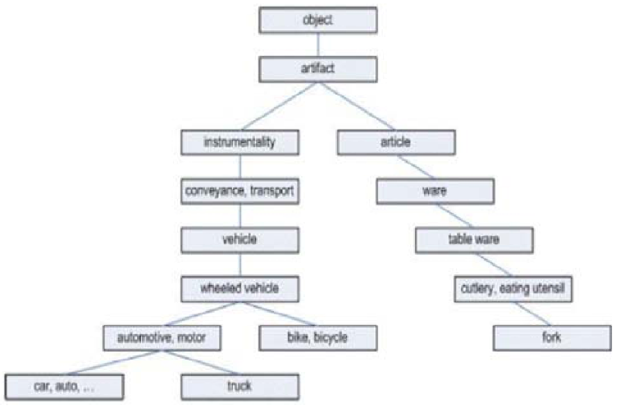

Fig. 1. Example of WordNet tree-like hierarchical structure

To extract the required information from WordNet we used an already developed public .Net Framework library called WordNet.Net [21], which is maintained and managed by Troy Simpson. Malcolm Crowe is the author of the legacy library code which is superseded by several WordNet database versions and library enhancements/bug fixes.

As described in the previous chapter we used WordNet to calculate the semantic similarity between words. To calculate the desired similarity using the proposed formula between two words w_1 and w_2, we performed a search of the semantic net for the shortest path between the synsets containing the compared words w_1 and w_2 including the depth of the first common subsumer of both synsets. The easiest way to accomplish this is to treat tree-like structure as an undirected graph and measure the shortest path length between the found synsets. The path length is measured in links/edges, where the shorter the path from one synset to another, the more similar they are. A shared parent of two synsets is known as a subsumer. After extracting the required information from WordNet, the similarity is calculated using the proposed formula, described in the previous chapter.

For example, the shortest path between "automotive" and "bicycle" as presented in Figure 2 is "automotive-wheeled vehicle-bicycle", where the minimal path length is 2 and the common subsumer is "wheeled vehicle" with a depth of 5 from the root node. The path length and the depth of the common subsumer give us a simple way to compute the relatedness distance between two word senses. However WordNet does have a few limitations, the first being that it is possible for two synsets from the same part of speech to have no common subsumer, which means that a path cannot be always found between the two synsets. We resolved this limitation by using a unique root node, which then means that a path will always exist between any two noun/verb synsets. The second limitation is that we only compare the word senses which have the same part of speech. This means that we do not compare a noun and a verb because they are located in different databases. When considering a word, we first check if it is a noun and if so we will treat it as a noun and if it is a verb or an adjective it will be ignored. Finally we check if it is not a noun, if so we will then check if it is a verb.

2) Brown Corpus

We used Brown Corpus to extract and calculate all of the necessary statistical information need for our calculations. Brown Corpus was compiled in 1960s by Henry Kucera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus. The Corpus consists of 500 samples, distributed across 15 text categories and originally (1961) contained 1,014,312 words [23].

The original Brown Corpus is available in plain text format. For easier access we used a reformatted XML version [23], which contains the same samples as the original Brown Corpus represented in XML format. The required information from the corpus is the number of times a selected word appears in the corpus. This number of word repetition is then used in the described formula. To retrieve the required information from the Corpus, we read the complete Brown Corpus in to the RAM memory. This is possible, because of a relatively low number of words, approximately 49 thousand distinct words and because of the fact that nowadays computer RAM memories are relatively large and thus have enough space to read the complete corpus. Another advantage of using the in-memory database instead of an ordinate disk-based database is that it speeds up the overall process of the similarity calculation. To store corpus words into memory, we used a C# dictionary object which is an equivalent to a hash-table. Dictionaries are very convenient for retrieving random elements quickly, which make them very appropriate for our purpose. Each word in the dictionary has a counter which counts the number of repetition of this exact word in the corpus.

3) .NET library

We encapsulated all of the calculations in a .NET library, in order to make the usage as simple as possible for the end user. Usage of the library is very simple, since there is only one entry point which initializes all of the required components in order to calculate the similarity value. The entry point for the calculations is in the class named CalculateSentenceSimilarity. The main method, named run, takes two strings as arguments (two sentences) and returns a value of type double, which represents the sentence similarity of given sentences.

An example call to the library looks like:

double sentenceSimilarity = CalculateSentenceSimilarity.run(sentence1, sentence2);

Value of the variable sentenceSimilarity is between 0 and 1. A larger value means more similarity between the compared sentences. Value 1 is returned if sentences are the same and 0 is returned if sentences are completely dissimilar (i.e. not similar at all).

4) *Program flow*

The following figure Fig. 2 describes the basic program flow for calculating sentence similarity:

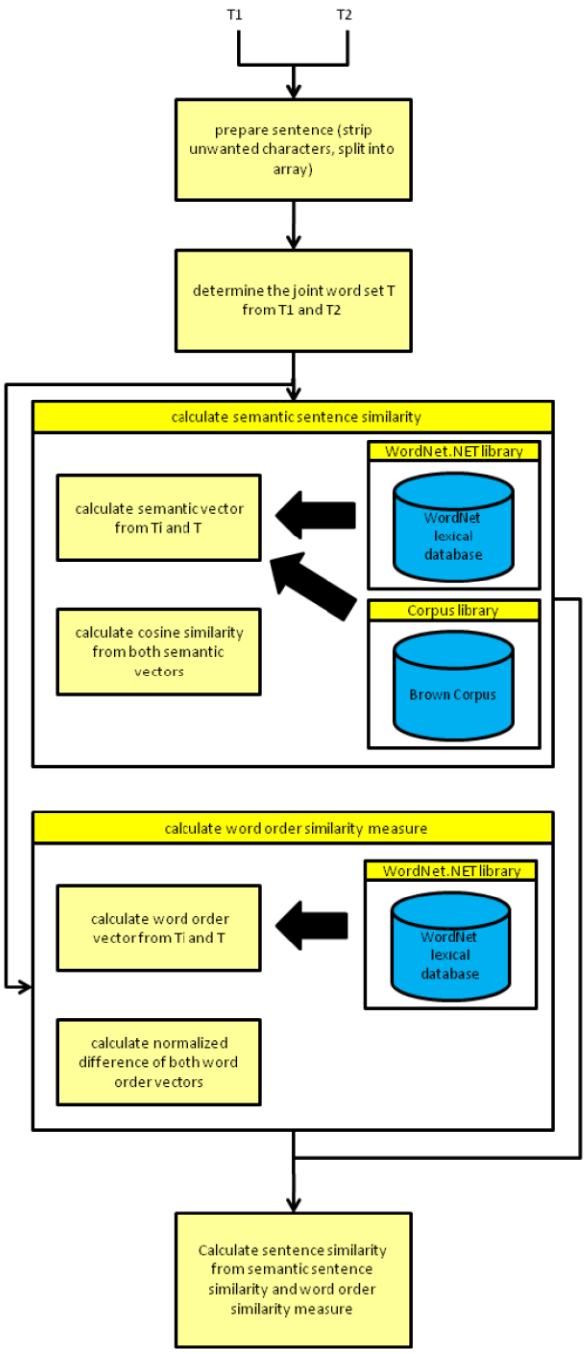
T1
T2
prepare sentence (strip unwanted characters, split into array)
determine the joint word set T from T1 and T2
calculate semantic sentence similarity
WordNet.NET library
WordNet lexical database
calculate semantic vector from Ti and T
calculate cosine similarity from both semantic vectors
Corpus library
Brown Corpus
calculate word order similarity measure
WordNet.NET library
WordNet lexical database
calculate word order vector from Ti and T
calculate normalized difference of both word order vectors
Calculate sentence similarity from semantic sentence similarity and word order similarity measure

Fig.2. Basic program flow for calculating sentence similarity

V. RESULTS

Although a few related articles have been published, there are currently no suitable benchmark text sets for the evaluation of sentence similarity methods. In order to evaluate our sentence similarity measure, we compared our results with our reference article results [1]. We used the same 16 sentence pairs as they were presented in the reference article and the results are shown in Table 1. We used the following constants in our described method, which yielded the best sentence similarity results: Alpha = 0.2; Beta = 0.5; Delta = 0.7; Threshold = 0.4

To get these parameters, we estimated the similarity of 8 sentence pairs by hand. Using these human estimations we then tuned the parameters to get the best match between calculated similarity and estimated similarity.

Table 1 - Results comparison

First sentence	**Second sentence**	**Our results**	**[1]**
I like that bachelor.	I like that unmarried man.	0,558	0,561
I have a pen.	Where do you live?	0,277	0
John is very nice.	Is John very nice?	0,599	0,997
Red alcoholic drink.	A bottle of wine.	0,665	0,585
It is a dog.	That must be your dog.	0,701	0,739
Red alcoholic drink.	Fresh orange juice.	0,721	0,611
It is a dog.	It is a log.	0,182	0,623
Red alcoholic drink.	An English dictionary.	0	0
It is a dog.	It is a pig.	0,179	0,790
Dogsare animals.	They are common pets.	0,756	0,738
I have a hammer.	Take some nails.	0,621	0,508
Canis familiaries are animals.	Dogsare common pets.	0,806	0,362
I have a pen.	Whereis ink?	0,102	0,129
Red alcoholic drink.	Freshapple juice.	0,474	0,420
A glass of cider.	A full cup of apple juice.	0,253	0,678

Results shown in Table 1 are in most cases very similar in both methods. There are also cases where results differ by a small margin.

VI. CONCLUSION

Nowadays sentence similarity is becoming more and more important. A lot of different applications and system require an automated and efficient method for computing the similarity between short texts. Our presented method and library should fulfill most of these needs and can be used and adopted to various domains and applications. As presented in the previous chapter our implemented method gives quite good sentence similarity results, which is comparable to a human's interpretation. We believe that our presented method is comparable to other implementations found online. However, as described in the chapter related work, there are still a lot of other different possibilities of measuring short texts similarity, where each has its advantages and disadvantages.

REFERENCES

[1] Y. Li, D. McLean, Z. A. Bandar, J. D. O'Shea and K. Crockett, "Sentence Similarity Based on Semantic Nets and Corpus Statistic" IEEE Transaction on knowledge and data engineering, Vol. 18, No. 8, August 2006

[2] V. Hatzivassiloglou, J. Klavans, and E. Eskin, "Detecting Text Similarity over Short Passages: Exploring Linguistic Feature Combinations via Machine Learning," Proc. Joint SIGDAT Conf. Empirical Methods in NLP and Very Large Corpora, 1999.

[3] C.T. Meadow, B.R. Boyce, and D.H. Kraft, Text Information Retrieval Systems, second ed. Academic Press, 2000.

[4] F. RYO, Y. TSUYOSHI, K HIROAKI, M. TETSUYA, T. YOSHINORI and O. NOBORU, "Measuring Similarity between Documents using Term Frequency and Concept Dictionary", Joho Shori Gakkai Kenkyu Hokoku, VOL.2003, NO.4(NL-153)

[5] H. Huang, H. Yang and Y. Kuo, "A Fuzzy-Rough Set Based Semantic Similarity Measure Between Cross-Lingual Documents", 2008 3rd International Conference on Innovative Computing Information and Control

[6] D. Metzler, S. Dumais and C. Meek, "Similarity Measures for Short Segments of Text"

[7] Wen-tau Yih and Christopher Meek, "Improving Similarity Measures for Short Segments of Text", Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

[8] M. Sahami and T. D. Heilman,"A Webbased Kernel Function for Measuring the Similarity of Short Text Snippets", WWW 2006, May 23–26, 2006, Edinburgh, Scotland.

Manning, C. D., and Sch¨utze, H. 1999. Foundations of Statistical Natural Language Processing. The MIT Press.

[9] Lund, K., Burgess, C. & Atchley, R. A. (1995). Semantic and associative priming in a high-dimensional semantic space. Cognitive Science Proceedings (LEA), 660-665.

[10] Lund, K. & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. Behavior Research Methods, Instruments & Computers, 28(2),203-208.

[11] Susan T. Dumais (2005) "Latent Semantic Analysis". Available: http://onlinelibrary.wiley.com/doi/10.1002/aris.1440380105/pdf

[12] Thomas K Landauer and Susan Dumais, "Latent semantic analysis", Scholarpedia 2008, 3(11):4356. Available: http://www.scholarpedia.org/article/Latent_semantic_analysis

[13] Thomas Landauer, Peter W. Foltz, & Darrell Laham (1998). "Introduction to Latent Semantic Analysis". Available: http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

[14] BURGESS, C., LIVESAY, K., AND LUND "Explorations in context space: Words, sentences", 1998, discourse. Disc. Proc. 25, 2–3, 211–257.

[15] Jones, R.; Rey, B.; Madani, O.; and Greiner, W., "Generating query substitutions", 2006, In Proc. of WWW '06.

[16] G.A. Miller, "WordNet: A Lexical Database for English," Comm. ACM, vol. 38, no. 11, pp. 39-41, 1995.

[17] Brown Corpus Information, http://clwww.essex.ac.uk/w3c/corpus_ling/content/corpora/list/private/brown/brown.html, 2005.

[18] Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press

[19] George A. Miller (Emeritus), Christiane Fellbaum, Randee Tengi, Helen Langone, Adam Ernst and Lavanya Jose, "WordNet", 2006, http://wordnet.princeton.edu/wordnet/

[20] Simpson T., Crowe M., 2005, WordNet.Net http://opensource.ebswift.com/WordNet.Net

[21] Wikipedia, "Brown Corpus", 15. March 2011. http://en.wikipedia.org/wiki/Brown_Corpus

[22] Nltk, "Natural language toolkit development", 17. March 2011 http://nltk.googlecode.com/svn/trunk/nltk_data/packages/corpora/brown_tei.zip

[23] P.Resnik, "Using Information Content to Evaluate Semantic Similarity in a Taxonomy," Proc. 14th Int'l Joint Conf. AI, 1995.

[24] P. Wiemer-Hastings, "Adding Syntactic Information to LSA," Proc. 22nd Ann. Conf. Cognitive Science Soc., pp.989-993, 2000.

[25] R.Rada, H.Mili, E.Bichnell, and M.Blettner, "Development and Application of a Metric on Semantic Nets," IEEE Trans. System, Man, and Cybernetics, vol.9, no.1, pp. 17-30, 1989