Modified Makagonov's Method for Testing Word Similarity and its Application to Constructing Word Frequency Lists

Xavier Blanco,[1] Mikhail Alexandrov,[1,2] and Alexander Gelbukh [2]

[1] Department of French and Romance Philology, Autonomous University of Barcelona
dyner1950@mail.ru, Xavier.Blanco@uab.es

[2] Center for Computing Research, National Polytechnic Institute (IPN). Mexico
dyner@cic.ipn.mx; www.Gelbukh.com

Abstract. By (morphologically) similar wordforms we understand wordforms (strings) that have the same base meaning (roughly, the same root), such as *sadly* and *sadden*. The task of deciding whether two given strings are similar (in this sense) has numerous applications in text processing, e.g., in information retrieval, for which usually stemming is employed as an intermediate step. Makagonov has suggested a weakly supervised approach for testing word similarity, based on empirical formulae comparing the number of equal and different letters in the two strings. This method gives good results on English, Russian, and a number of Romance languages. However, his approach does not deal well with slight morphological alterations in the stem, such as Spanish *pensar* vs. *pienso*. We propose a simple modification of the method using n-grams instead of letters. We also consider four algorithms for compiling a word frequency list relying on these formulae. Examples from Spanish and English are presented.

1 Introduction

Given a large text or text corpus and a pair of wordforms (strings), we consider the task of guessing whether these two words have the same root and thus the same base meaning. We call this (morphological) *word similarity*: two words are *similar* if they have the same root. This relation permits grouping together the words having the same root, e.g., *sad, sadly, sadness, sadden, saddened,* etc. This task has numerous applications, such as constructing word frequency lists. Our motivation is to improve information retrieval and similar practical applications. Consequently, our goal is to provide a reasonably accurate statistical-based algorithm (tolerating certain error rate) and not a precise linguistic analysis.

For grouping together the words with the same root, two morphology-based methods are usually used: lemmatization and stemming. Lemmatization reduces words to the base form: *having* → *have*; stemming truncates words to their stems: *having* → *hav-* (often lemmatization task is also referred to as stemming).

Stemming or lemmatization can be used for testing the (morphological) similarity between two words: both words are first reduced to lemmas or stems; if the resulting strings are equal then the two given words are declared similar. This gives a symmetric, reflexive, and transitive relation that can be perfectly used for grouping together words with the same base meaning.

The stemming or lemmatizing algorithms using morphological rules and a large morphological dictionary provide practically 100% accuracy on known words and good accuracy on the words absent in the dictionary [4, 5]. The algorithms relying only on lists of suffixes and suffix removal rules, but no dictionaries, in spite of being much simpler, provide relatively high accuracy, often greater than 90%. The most popular algorithm is Porter stemmer [9] (actually, a lemmatizer). Both methods are strongly language-dependent. This becomes a problem in large-scale analysis of multilingual, multi-thematic document collections.

Makagonov *et al.* [7] suggested another approach to testing word similarity. It does not rely on a (language-dependent) intermediate step of reducing the two given words to a stem or lemma. Instead, it uses empirical formulae to compare the given strings directly. These formulae use the number of the coincident initial letters and non-coincident final letters in the two words. Such formulae are constructed by Ivakhnenko's [6] inductive method of model self-organization. In our paper [1] we have given more details on this method, investigated the sensibility of the formulae to different languages, and analyzed the typical errors of the method. The main advantage of this knowledge-poor approach is its simplicity and flexibility. It does not rely on any manually compiled morphological rules, dictionaries, suffix lists, or rule systems. To adapt the method to a new language or a new subject domain, only the parameters of the formula are to be automatically adjusted.

However, this method is sensitive to small differences in initial parts of similar words: an additional letter in one of these words may prevent the method from detecting their similarity. This affects mostly Romance languages with their irregular verbs: e.g., Spanish: *pienso* '(I) think' – *pensaba* 'thought', *entiendo* '(I) understand' – *entender* '(to) understand'. Obviously, this increases the rate of false negatives (failing to give a positive answer on a pair of words that are in fact similar). To avoid this, we propose here to use *n*-grams (3-grams) instead of letters in the original formula. *N*-grams have been already used for comparing words [10]. However, the work [10] concerns only evaluation of word importance but not their similarity. Nevertheless, it stimulated the research presented in this paper.

Another drawback of the approach from [7] is that the obtained relation is not transitive: if the formula reports the words *a* and *b* to be similar, as well as *b* and *c*, it does not necessarily report *a* and *c* to be similar. Strictly speaking, this prevents from using such a relation to group words together. However, [7] suggested a heuristic algorithm relying on these formulae for constructing word frequency lists (actually, for grouping words; the algorithm gives different results than a simple transitive closure).

We found, though, that this algorithm gives higher level of false negatives in comparison with the formulae themselves. We propose other algorithms, taking into account mentioned non-transitivity of our heuristic relation.

The paper is organized as follows. In Section 2, we explain the original algorithm and its empirical formulae. In Section 3, we present our modifications to the formula. We formally define the notion of *n*-gram as used by our algorithm and operations on such *n*-grams and then present the new formula and its training process. In Section 4, we introduce four algorithms for constructing word frequency lists and present experimental results. Section 5 concludes the paper and mentions some future work.

2 Testing Word Similarity Using Letters

2.1 Empirical Formulae

The empirical formulae from [7] test a hypothesis about word similarity. The proposed approach is applied only for suffixal inflective languages, i.e., the languages where the word base (morphologically invariant part) is located at the beginning of the word—which is generally true for the majority of European languages.

The formula relies on the following characteristics of the pair of words:

y: the length of the common initial substring (y standing for *yes*),
n: the total number of final letters differing in the two words (n for *no*),
s: the total number of letters in the two words (s for *sum*),

so that $2y + n = s$. For example, for the words *sadly* and *sadness*, the maximal common initial substring is *sad-*, thus the differing final parts are *-ly* and *-ness*, so that $n = 6$, $y = 3$, and $s = 12$.

The authors of [7] considered the following class of models for making decisions about word similarity: two words are similar if and only if

$$\frac{n}{s} \le F(y), \qquad F(y) = a + b_1 y + b_2 y^2 + \ldots + b_k y^k, \tag{1}$$

where $F(y)$ is the model function; a and b_i are constants (parameters of the formula). Such a function is general enough because any continuous function can be represented as a convergent polynomial series. The parameter k represents the model complexity.

To find the best model function, i.e., the model of optimum complexity, [7] used the inductive method of model self-organization. This method compares step-by-step the models of increasing complexity and stops this process when the optimum of an external criterion of model quality is reached [6]. This method relies on a set of examples, which are used to calculate the model parameters and evaluate the model quality; thus, the method is weakly supervised since the requited set of examples is very small. These examples are prepared by the user of the program for a specific language or genere.

With this method, the formulae for testing word similarity shown in Table 1 were constructed for different languages [1].

Table 1. Formulae for different languages.

French	Italian	Portuguese	Spanish
$n/s \le 0.48 - 0.024\,y$	$n/s \le 0.57 - 0.035\,y$	$n/s \le 0.53 - 0.029\,y$	$n/s \le 0.55 - 0.029\,y$

Basing on all examples prepared for four languages, the authors determined the generalized formula:

$$n / s \le 0.53 - 0.029\,y \tag{2}$$

This formula can be considered an initial approximation for further tuning on other romance languages.

2.2 Discussion

Since the empirical formula is based on statistical regularities of a language, it leads to the errors of the two kinds (false positive and false negative; this can be rephrased in terms of precision and recall—see Sections 3.4 and 4.2). Varying the threshold function F between -1 and $+1$ we can control the balance between precision and recall. Our goal is to find the function that gives their acceptable combination; for example, the formula (2) gives rather acceptable results. Note that the formula's parameters depend not only on the language but also on a specific genre or domain. E.g., the formula $n/s \leq 0.55 - 0.029\,y$ is optimal for general lexis in Spanish; however, for the texts on mortgage and crediting the best parameters proved to be $n/s \leq 0.53 - 0.026\,y$.

Certain questions arise as to the obtained model function

$$\frac{n}{s} \leq a + by \tag{3}$$

where $a > 0$, $b < 0$. This is a linear formula with two degrees of liberty where the threshold (the right-hand part) is lower for longer words. What does it mean from linguistics point of view?

(1) Our formula has in fact two degrees of liberty with respect to the parameters n, y, and s, since $s = n + 2y$. One can also consider a model in the form $n/s \leq F\,(y/s)$, which has only one degree of liberty since $y\,/\,s = (1 - n/s)\,/\,2$. It allows taking into account separately the statistics of both the initial and the final part of a given word.

(2) The linear approximation of the original formula $F(y) \sim a + by$ indicates high complexity of the real model we want to evaluate. Our model can reflect only the tendency (b is the first derivative of the model function) but not the shape of the function.

(3) All obtained formulas give lower threshold for longer words with the same relative number of non-coincident letters. This discrimination of long words reflects the following statistical property of a language: the length of the final part of similar words on average is similar for both long and short words. This property was noted in [8] and makes sense linguistically: the length of word endings (non-root morphemes) is the same for long and short words.

3 Testing Word Similarity Using *n*-grams

3.1 Limitation of the Original Approach

Our approach for testing word similarity is not applicable to words with irregular forms. Indeed, no simple formula can detect any similarity between irregular verbs such as *buy* and *bought*, because these words have only one common letter in the initial part of the string. Similar examples are there in Romance languages, e.g., Spanish *saber* 'know' vs. *supo* 'knew'. Obviously, this limitation leads to false negatives.

The other difficulty is related with the rigid comparison used in the formula. Namely, 'common part' means that both words have exactly equal initial substrings; slight changes in the common part dramatically reduce the size of the initial common substring recognized by the formula, so word similarity is not detected, e.g.: Spanish *pienso* vs. *pensar*, *parezco* and *parecer*.

In the paper, we address the latter problem. We assume that the common initial part of both words may have small differences. In the paper, we limit these differences to one letter.

3.2 3-grams and Operations with them

Speaking about possible differences in common part of two words, we face a task of determination of the boundary of this common part. If we deal with one-letter "defects", then this task can be solved with 3-grams.

Definition 1. *N*-gram of letters is a substring of *N* letters. A word of *n* has of $m = n - N + 1$ *N*-grams. Example: The 3-grams for the word *revolution* are {rev, evo, vol, olu, lut, uti, tio, ion}.

Definition 2. *N*-gram associated with *i*-th position of a given word *w* is *N*-gram starting at *i*-th position. Truncated *N*-grams are added by attaching new "undefined" symbols to the words. All undefined symbols are supposed to be different, even though we denote them with the same letter X; i.e., X ≠ X. A word of *n*-letters has exactly *n* different associated *N*-grams. Example: The 3-grams associated with the first and last letters of the word *revolution* are {rev, nXX}. Here X are the "undefined" symbols.

Definition 3. Full set of *N*-grams of letters for a given word *w* is all associated *N*-grams of this word. Example: The full set of 3-grams for the word *bill* are {bil, ill, llX, lXX}.

We will work with 3-grams, though similar definitions can be introduced for any N-grams. All comparisons are lexicographic. The definitions below can be rephrased in terms of Levenshtein distance, though we find them easier to understand in the form given here.

Definition 4. 3-grams ***A*** and ***B*** are equal with the level 1 if they are equal as strings. Example: 3-grams *pas* and *pas* are equal, while *pas* and *sap* are not.

Definition 5. 3-grams ***A*** and ***B*** are equal with the level 2/3 if two letters of one of them are found in the other one in the same order. Examples: 3-grams *pas* and *asp* as well as *ars* and *aps* are equal with the level 2/3, while *sra* and *aps* are not.

Definition 6. 3-grams ***A*** and ***B*** are equal with the level 1/3 if a letter of one of them is found in the other one. Examples: 3-grams *sra* and *ars* (*sra* and *ars*, *sra* and *ars*) are equal with the level 1/3, while *pas* and *wre* are not. Note that in the former case, both 3-grams contain the same letters but their order does not allow for equality with levels 2/3 or 1.

3-grams with undefined symbols X are compared taking into account that all such symbols are different: X ≠ X. Examples:

- 3-grams *kl*X and X*kl* are equal with the level 2/3.
- 3-grams *kl*X and *k*XX are equal with the level 1/3.
- 3-grams XXX and XXX are not equal.

Definition 7. Two strings are equal by 3-grams with the level Q if they have equal 3-grams for all positions with the level Q. Obviously, if $Q = 1$ then these two strings are just equal as strings.

Examples: Strings *assenr* and *reasse* are equal by 3-grams with the level 1/3 and *assenr* and *yssien* with the level 2/3.

3.3 Constructing of Empirical Formula for 3-grams

We use the model described in Section 2.1, but instead of letters, our model operates on 3-grams. I.e., we declare two words similar if and only if

$$\frac{n}{s} \leq F(y) \qquad (4)$$

where y is the total number of 3-grams of the common *initial* substring, n is the total number of 3-grams in *final* substrings of the two words, s ia the total number of 3-grams in the two words, and F is the model function, so that $2y + n = s$. Since we use here all associated 3-grams (see Definition 3), the total number of n-grams is equal to the total number of letters in the two words.

Common substring is defined as the maximum common part of the two initial substrings, which have the same first letter and which are equal by 3-grams with the level no less than 2/3 (Definition 7). According to Definitions 4 and 5, this allows having one incompatible letter in the common part.

Examples:

a) For *sadly* and *sadness* (Section 2.1), 3-grams are: *sadly* = {*sad, adl, dly, ly*X, *y*XX}, *sadness* = {*sad, adn, dne, nes, ess, ss*X, *s*XX }; $n = 8$, $s = 12$, $y = 2$.
b) For Spanish *comiendo* and *comer*, 3-grams are: *comiendo* = {*com, omi, mie, ien, end, ndo, do*X, *o*XX}, *comer* = {*com, ome, mer, er*X, *r*XX}; $n = 7$, $s = 13$, $y = 3$.

In this paper we will not try to find the optimum shape of the model function F by the inductive method of model self-organization. Instead, extrapolating the results of experiments with traditional model, we will assume that the model to be constructed will be the linear. Thus, we will look for the optimum parameters for the formula (3).

3.4 Training Procedure for Spanish

The formula to be constructed has two unknown parameters, a and b. To find them we should prepare the set of examples and use the least squares method. It is well known from the theory of experiments that the number of examples must be at least 3 times greater then the number of parameters to be evaluated, which provides the relative error of 10%. Thus for our case we will need at least 6 pairs of similar words.

The examples we used for model construction are pairs of similar words with: (a) short and long initial common part, (b) short and long final parts, and (c) "defects" at the beginning and at the end of words; see Table 2.

The solution of this system gives the criterion:

$$n / s \leq 0.63 - 0.036\, y \qquad (5)$$

Table 2. Examples for training formula.

Examples		Parameters	Equations
Circo	*Circense*	$n = 7,\ s = 13,\ y = 3$	$7/13 = a + 3b$
Creado	*Creacion*	$n = 8,\ s = 14,\ y = 3$	$8/14 = a + 3b$
Sentimentales	*Sentimentalismo*	$n = 8,\ s = 28,\ y = 10$	$8/28 = a + 10b$
Necesario	*Necesariamente*	$n = 9,\ s = 23,\ y = 7$	$9/23 = a + 7b$
Pensar	*Pienso*	$n = 6,\ s = 12,\ y = 3$	$6/12 = a + 3b$
Entender	*Entiendo*	$n = 6,\ s = 16,\ y = 5$	$6/16 = a + 5b$

This formula should be considered only as a first approximation and may be later tuned on the texts from a given domain, since our examples selected for training formula only reflect some general regularities of a language.

We compared the work of traditional and modified algorithm on document collection on economic problems; see Table 3. The total number of words considered was 320 (numbers and words with less than 4 letters were excluded). The original algorithm used the formula (2); the algorithm based on 3-grams used the formula (5). The algorithm compared the pairs of words adjacent in the alphabetically ordered list, i.e., 319 comparisons were made. By false positive (negative) rate P_p (P_n) we understand the number of pairs incorrectly reported by the program as similar (not similar), divided by the number of really similar pairs in the corpus. Viewing the task as retrieval of similar pairs among all considered pairs, we can also represent the quality of the algorithm via precision P and recall R; obviously, $R = 1 - P_n$; $P = R / (R + P_p)$.

Table 3. Comparison of both methods

		Traditional algorithm	Modified algorithm
False positive	P_p	4.9%	5.4%
False negative	P_n	12.9%	10.6%
Total errors	P_{err}	17.8%	16.0%
Precision	P	94.7%	94.3%
Recall	R	87.1%	89.4%
F-measure		90.7%	91.8%

3.5 Discussion

We did not consider the words containing less than 4 letters, since these are mostly prepositions, conjunctives, and pronouns.

We neither considered the other n-grams, for example, 2-grams, 4-grams, etc. The only reason was that we wanted to implement the simplest principle of voting while detecting one-letter "defects". 3-grams were the minimum n-gram that allows doing it. However, in the future it is necessary to check N-grams with other N.

The formula we used reflected the statistical regularities of a language, with the error rate given in Table 3. By using n-grams we tried to reveal similar words having a "defect" in their common part, which led to decreased rate of false negatives. Although false positives rate slightly increased, the experiments showed that the overall error rate decreased.

4 Constructing Word Frequency Lists

4.1 Main Algorithms

The first algorithm oriented on application of empirical formula was very simple and consisted of the following steps [7]. Initially, the text collection is considered as a bag of words. With every word in this sequence, a counter is associated and initially set to 1. The algorithm proceeds as follows:

1. All words are ordered alphabetically; literally equal words are joined together, and their counts are summed up (e.g., 3 occurrences of the string *ask* with counters 2, 3, and 1 are replaced by one occurrence with the counter 6).
2. The similarity for each pair of adjacent words is tested according the criterion described above; namely, the 1-st word is compared with the 2-nd one, 3-rd with the 4-th, etc. If a pair of words is similar then these two words are replaced with one new "word"—their common initial part, with the counter set to the sum of the counters of the two original words. If the list has an odd number of words, then the last word is compared with the immediately preceding one (or with the result of substitution of the last word pair).
3. Step 2 is repeated until no changes are made at Step 2.

This multi-pass algorithm worked quickly but it proved to have a defect: it often omitted similar words in adjacent pairs. Therefore, we had to consider the algorithm in more detail. Note that the word similarity relation implemented in the present paper is not transitive, i.e., that two words are similar to a third one does not mean that the first two ones are similar. Thus, our algorithms for constructing word frequency lists tare sensitive to the order of comparison. We consider here two algorithms, which are applied to lists of alphabetically ordered words:

1. Algorithm where joining of adjacent similar words is used;
2. Algorithm where both adjacent and not adjacent words are considered.

Both algorithms start with Step 1 of the algorithm described above.

Algorithm 1

It is one-pass algorithm consisting of the following steps:

1. Starting from the first word from the list, the algorithm searches for a pair of similar words, considering the words from the next one just after it.
   - If such a pair is not found, then the algorithm stops.
   - Otherwise, these two words are substituted by a new one—their initial common part. Its counter is the sum of the counters of the two joined words.
2. The procedure is repeated with the next position.

Algorithm 2

It is a multi-pass algorithm, which checks for similarity of each word to each another word. It consists of the following steps:

1. Starting from the first word in the list, the algorithm searches for the first similar word among all words with the same initial letter (not necessary adjacent).
   - If such a word is not found, the process is repeated from the second word.
   - In case of success, the first word is substituted by the new one—their common initial part. Its counter is the sum of the counters of the two joined words. The second word from the pair is eliminated from the list. From the position of the eliminated word, the algorithm searches for a word similar to the new first one among the words with the same initial letter.
2. The process is repeated from the second word of the corrected list.

Any of the two algorithms can be implemented in two variants:

- As a direct pass algorithm: the list is processed from the beginning to the end;
- As a reverse pass algorithm: the list is processed from the end to the beginning.

These implementations give different results.

4.2 Experimental Results

For the experiments, we took Spanish texts on mortgage and crediting. This topic is narrow enough to provide a representative set of similar words. The total number of words considered was 560 (numbers and words with less then 4 letters were excluded); again, only alphabetically adjacent pairs were considered. For the experiments, we used the formula (5). Both direct and reverse pass version of the algorithms were tested. The results proved to be rather similar; Table 4 shows the results for of Algorithm 1.

Table 4. Experimental results.

	Direct pass	Reverse pass
Recall R	92.5%	95.4%
Precision P	89.4%	95.0%
F-measure	90.9%	95.2%

4.3 Discussion

Reverse pass implementation proved gave better results for both algorithms. This is because the length of the common part of the similar words in an alphabetically ordered list increases on average. With a direct pass algorithm, the formula fails to detect similarity of words at the beginning and the end of a group of similar words. A reverse pass algorithm gives strong compensatory effect for truncation of common part of similar words, increasing the probability of their joining.

Algorithm 2 seems to give better results on texts from some narrow domains, where many similar words are separated by others in the lexicographic order and therefore can not be joined by Algorithm 1. However, testing this hypothesis requires more experiments.

5 Conclusions and Future Work

We have suggested a modification of Makagonov's method [7] for testing (morphological) word similarity. His approach is based on an empirical formula trained on a small number of examples. Our proposed modification uses 3-grams instead of letters in this formula. In the paper, we have introduced operations of comparison on *n*-grams, used by the algorithm. Experiments show improvement of the suggested modification over the original algorithm. The suggested modification keeps all properties of the original method: empirical formula does not require any morphological dictionaries of the given language and can be tuned manually (or trained on a small number of examples) on a given language or topic.

In the future, we plan to construct other empirical formulae taking into account statistical regularities of words extracted from the training corpus. We also plan to compare our results using *n*-grams with *n* other than 3.

References

1. Alexandrov, M., Blanco, X., Makagonov, P. (2004). Testing Word Similarity: Language Independent Approach with Examples from Romance. In *Natural Language Processing and Information Systems*, Lecture Notes in Computer Science 3136, Springer, pp. 223-234.
2. Baeza-Yates, R., Ribero-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley.
3. Cramer, H. (1946): *Mathematical methods of statistics*. Cambridge.
4. Gelbukh, A., Sidorov, G. (2002): Morphological Analysis of Inflective Languages through Generation. *Procesamiento de Lenguaje Natural*, No 29, 2002, p. 105–112.
5. Gelbukh, A., Sidorov, G. (2003): Approach to construction of automatic morphological analysis systems for inflective languages with little effort. In: *Computational Linguistics and Intelligent Text Processing* (CICLing-2003), Lecture Notes in Computer Science, Springer, No 2588, pp. 215–220.
6. Ivahnenko, A. (1980): *Manual on typical algorithms of modeling*. Tehnika Publ., Kiev (in Russian).
7. Makagonov, P., Alexandrov, M. (2002): Empirical Formula for Testing Word Similarity and its Application for Constructing a Word Frequency List. In: *Computational Linguistics and Intelligent Text Processing* (CICLing-2002), Lecture Notes in Computer Science, Springer, No 2276, pp. 425–432.
8. Makagonov, P., M. Alexandrov, M., Gelbukh, A. (2004): Formulae for Testing Word Similarity trained on examples. In: *Corpus Linguistics-2004*. Proc. of linguistics seminar of Sankt-Petersburg University, Russia, 15 pp. (in Russian).
9. Porter, M. (1980): An algorithm for suffix stripping. *Program*, 14, pp. 130–137.
10. Renz, I., Ficzay, A., Hitzler, H. (2003): Keyword Extraction for Text Categorization. In: In: *Natural Language Processing and Information Systems*, Lecture Notes in Informatics, GI-Edition Germany, No 129, pp. 228–234.