Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model

Grigori Sidorov[1], Alexander Gelbukh[1], Helena Gómez-Adorno[1], and David Pinto[2]

[1] Centro de Investigación en Computación,
Instituto Politécnico Nacional, México D.F.,
Mexico

[2] Facultad de Ciencias de la Computación,
Benemérita Universidad Autónoma de Puebla, Puebla,
Mexico

{sidorov,gelbukh}@cic.ipn.mx, helena.adorno@gmail.com, dpinto@cs.buap.mx

Abstract. We show how to consider similarity between features for calculation of similarity of objects in the Vector Space Model (VSM) for machine learning algorithms and other classes of methods that involve similarity between objects. Unlike LSA, we assume that similarity between features is known (say, from a synonym dictionary) and does not need to be learned from the data. We call the proposed similarity measure **soft similarity**. Similarity between features is common, for example, in natural language processing: words, n-grams, or syntactic n-grams can be somewhat different (which makes them different features) but still have much in common: for example, words "play" and "game" are different but related. When there is no similarity between features then our soft similarity measure is equal to the standard similarity. For this, we generalize the well-known cosine similarity measure in VSM by introducing what we call **"soft cosine measure"**. We propose various formulas for exact or approximate calculation of the soft cosine measure. For example, in one of them we consider for VSM a new feature space consisting of pairs of the original features weighted by their similarity. Again, for features that bear no similarity to each other, our formulas reduce to the standard cosine measure. Our experiments show that our soft cosine measure provides better performance in our case study: entrance exams question answering task at CLEF. In these experiments, we use syntactic n-grams as features and Levenshtein distance as the similarity between n-grams, measured either in characters or in elements of n-grams.

Keywords. Soft similarity, soft cosine measure, vector space model, similarity between features, Levenshtein distance, n-grams, syntactic n-grams.

1 Introduction

Computation of similarity of specific objects is a basic task of many methods applied in various problems in natural language processing and many other fields. In natural language processing, text similarity plays crucial role in many tasks from plagiarism detection [18] and question answering [3] to sentiment analysis [14–16].

The most common manner to represent objects is the Vector Space Model (VSM) [17]. In this model, the objects are represented as vectors of values of features. The features characterize each object and have numeric values. If by their nature the features have symbolic values, then they are mapped to numeric values in some manner. Each feature corresponds to a dimension in the VSM. Construction of VSM is in a way subjective, because we decide which features should be used and what scales their values should have. Nevertheless, once constructed, the calculation of similarity of the vectors is exact and automatic. VSM allows comparison of any types of objects represented as values of features. VSM is especially actively used for representing objects in machine learning methods.

In the field of natural language processing, the objects usually are various types of texts. The most widely used features are words and n-grams. In particular, recently we have proposed a concept of syntactic n-grams, i.e., n-grams constructed by following paths in syntactic trees [19, 21]. These n-grams allow taking into account syntactic information for VSM representation (and, thus, for use with machine learning algorithms as well). There are various types of n-grams and syntactic n-grams according to types of elements they are built of: lexical units (words, stems, lemmas), POS tags, SR tags (names of syntactic relations), characters, etc. Depending on the

Computación y Sistemas Vol. 18, No. 3, 2014 pp. 491–504
ISSN 1405-5546
DOI: 10.13053/CyS-18-3-2043

task, there are also mixed syntactic n-grams that are combinations of various types of elements, for example, ones that include names of syntactic relations [20]. The values of the features are usually some variants of the well-known tf-idf measure.

In Vector Space Model, traditional cosine measure [17] is commonly used to determine the similarity between two objects represented as vectors. The cosine is calculated as normalized dot-product of the two vectors. The normalization is usually Euclidean, i.e., the value is normalized to vectors of unit Euclidean length. For positive values, the cosine is in the range from 0 to 1. Given two vectors a and b, the cosine similarity measure between them is calculated as follows: the dot product is calculated as

$$a \cdot b = \sum_{i=1}^{N} a_i b_i, \tag{1}$$

the norm is defined as

$$||x|| = \sqrt{x \cdot x}, \tag{2}$$

and then the cosine similarity measure is defined as

$$cosine(a, b) = \frac{a \cdot b}{||a|| \times ||b||}, \tag{3}$$

which given (1) and (2) becomes

$$cosine(a, b) = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2} \sqrt{\sum_{i=1}^{N} b_i^2}}. \tag{4}$$

Applied to a pair of N-dimensional vectors, this formula has both time and memory complexity $O(N)$.

In a similar way, the same VSM is used by machine learning algorithms. They are applied, for example, for grouping, separation, or classification of objects by learning weights of features or by choosing most significant features.

Traditional cosine measure and traditional similarity measures consider VSM features as independent or in some sense completely different; mathematically speaking, the formula (1) considers the vectors in the VSM as expressed in an orthonormal basis. In some applications this is a reasonable approximation, but far too often it is not so.

For example, in the field of natural language processing the similarity between features is quite intuitive. Indeed, words or n-grams can be quite similar, though different enough to be considered as different features. For example, words “play” and “game” are of course different words and thus should be mapped to different dimensions in SVM; yet it is obvious that they are related semantically. This also can be interpreted in information-theoretic way (when one speaks of playing, speaking of a game is less surprising) or in probabilistic way (conditional probability of the word “game” increases in the context of “play”). That is, these dimensions are not independent. Similarity between words is very important in many applications of natural language processing and information retrieval.

In this paper, we propose considering such similarity of features in VSM, which allows generalization of the concepts of cosine measure and similarity. We describe our experiments that show that the measure that takes into account similarity between features yields better results for a question answering task we worked with.

Some methods, such as LSA, can learn the similarity between features from the data. In contrast, we assume that similarity between features is known—say, from a synonym dictionary—and does not need to be learned from the data. Thus our method can be used even when there is no sufficient data to learn the similarity between features from statistics.

The rest of this paper is organized as follows. Section 2 introduces the soft cosine measure and the idea of the soft similarity. Section 3 describes the question answering task for entrance exams at CLEF and the method that we applied in it. Section 4 presents application of the soft cosine similarity (the experiments) and discussion of the results. Section 5 concludes the paper.

2 Soft Similarity and Soft Cosine Measure

Consider an example of using words as features in a Vector Space Model. Suppose that we have two texts: (1) *play, game*, (2) *player, gamer*. It defines a 4-dimensional VSM with the following features: *play, player, game, gamer*. We have two vectors a and b: $a = [1, 0, 1, 0]$ and $b = [0, 1, 0, 1]$. The traditional cosine similarity of these two vectors is 0. But if we take into consideration the similarity of words, it turned out that these vectors are quite similar. There is special procedure called ‘stemming’ in natural language processing aimed to take into account this kind of similarity between words, but it is a specific *ad hoc* procedure. A more general question is: how can we take into account the similarity between features in Vector Space Model? The traditional similarity does not consider this question, i.e., all features are considered different.

The cosine measure is widely applied and usually is taken for granted. We found two papers that suggest its modification. In [7] the authors claim that the cosine

similarity is overly biased by features with higher values and does not care much about how many features two vectors share, i.e., how many values are zeroes. They propose a modification of the cosine measure called the distance weighted cosine measure (dw-cosine). The dw-cosine is calculated by averaging the cosine similarity with the Hamming distance. The Hamming distance counts how many features two vectors do not share. In this way, they try to decrease the similarity value of the two vectors that share less features or have high values. It is an attempt "to tune" the traditional cosine measure.

The other paper, Mikawa et al. [8], proposes "extended cosine measure", which is in a some way similar to our proposal: they consider Mahalanobis distance for similarity of features. They do not generalize the concept of similarity, they just modify the cosine for this specific measure. Also, it is not obvious how to measure Mahalanobis distance in many situations (we propose to use much more clear Levenshtein distance, see below).

Our idea is more general: we propose to modify the manner of calculation of similarity in Vector Space Model taking into account similarity of features. If we apply this idea to the cosine measure, then the **"soft cosine measure"** is introduced, as opposed to traditional "hard cosine", which ignores similarity of features. Note that when we consider similarity of each pair of features, it is equivalent to introducing new features in the VSM. Essentially, we have a matrix of similarity between pairs of features and all these features represent new dimensions in the VSM.

Note that if the same idea is applied to similarity while using machine learning algorithms in Vector Space Model, then the similarity is transformed into **"soft similarity"**. Again, new dimensions (features) are added to the VSM. The values of the new features can be obtained, say, by taking the mean value of the two features of the same vector multiplied by the similarity of these two features. It is the most obvious suggestion, other possibilities can be explored.

The idea to take into account the similarity of features was also proposed in [4, 5], but it was applied to the concept of cardinality. The authors introduced the "soft cardinality", i.e., the cardinality that can obtain different values depending on similarity of features. In their case, this idea does not have the clear manner of calculation of the new cardinality and does not generalize any important concept. We use the term "soft" following their idea of soft cardinality.

The next question is how to measure similarity between features. In general, the measuring of similarity depends on the nature of the features. In our case, we compare features using the Levenshtein distance [6], taking advantage of the fact that they are usually strings in case of natural language processing.

Recall that the Levenshtein distance is the number of operations (insertions, deletions, rearrangements) needed to convert a string into another sting. In our case, the Levenshtein distance is a good measure for string comparison, but other measures can be exploited as well. So, if our objects are texts then the traditional features are words, n-grams or syntactic n-grams and their corresponding values are based on the tf-idf measure. In case of the Levenshtein distance if we use n-grams or syntactic n-grams then there are two possibilities for string comparison: directly compare character transformations or consider each element of n-grams as a unit for comparison. We explored both possibilities in the experiments. In case of words as features, only first possibility is applicable. In case of word, say, WordNet similarity functions can be used.

In what follows we will present several formulas that take into account similarity between features. We will show that the soft cosine measure performs better than the conventional cosine (4) in most cases for a version of question answering task—a classical natural language processing problem. Namely, both our exact and simplified expressions for soft cosine measure obtained better experimental results as compared to the standard cosine measure in the majority of cases.

2.1 Feature Similarity as a Non-orthogonality of the VSM Basis

We assume that we deal with objects, say, documents, that are modeled in a VSM as vectors whose coordinates correspond to features, say, words. For example, in the bag-of-words model, documents

a: *a player will play a game they like to play*
b: *they play the game they like*

are represented by vectors

$$a = (2, 1, 1, 2, 1, 1, 1, 1, 0), \tag{5}$$
$$b = (0, 0, 0, 1, 1, 2, 1, 0, 1), \tag{6}$$

where the coordinates correspond to the frequencies of the words *a, player, will, play, game, they, like, to, the* in each document; e.g., *a* and *play* appear in the first document twice. Now we can measure the similarity between these texts as $cosine(a, b)$.

However, the basis vectors of this representation, i.e.,

$$e^1 = (1, 0, 0, \ldots, 0)$$
$$e^2 = (0, 1, 0, \ldots, 0)$$
$$\ldots$$
$$e^N = (0, 0, 0, \ldots, 1)$$

are also objects (one-word documents, i.e., words) in this vector space. Saying that they are orthogonal,

$$cosine(e^i, e^j) = 0, \tag{7}$$

is equivalent to saying that they are independent, or that there is no similarity whatsoever between them.

However, as we have discussed, in fact almost always there is some similarity, which can be identified independently from the VSM—e.g., using a dictionary of synonyms for words. In our example, *game* is obviously related to *play*.

In this paper we make a natural assumption that this similarity can be modeled as cosine between these objects:

$$cosine(e^i, e^j) = s_{ij} = sim(f_i, f_j), \tag{8}$$

where f_i and f_j are the features corresponding to these basis vectors, and $sim(\cdot)$ is a similarity measure, such as synonymy. There exist numerous ways of quantitatively measuring similarity (or relatedness) between words. For example, the well-known `WordNet::Similarity` provides eight different ways of calculating word relatedness.

Thus, we consider the basis in which we initially obtained the vectors, such as (5) in the above example, to be non-orthogonal.

Still our goal is to be able to calculate the cosine $cosine(a, b)$ between vectors initially given in such a basis:

$$a = \sum_{i=1}^{N} a_i e^i, \tag{9}$$

$$b = \sum_{i=1}^{N} b_i e^i. \tag{10}$$

Since dot product is bilinear, we have:

$$a \cdot b = \left(\sum_{i=1}^{N} a_i e^i\right) \cdot \left(\sum_{i=1}^{N} b_i e^i\right) \tag{11}$$

$$= \sum\sum_{i,j=1}^{N} a_i b_j (e^i \cdot e^j) \tag{12}$$

$$= \sum\sum_{i,j=1}^{N} s_{ij} a_i b_j, \tag{13}$$

where s_{ij} are given by (8). We obtain instead of the classical (4) our main formula:

$$soft\_cosine_1(a, b) = \frac{\sum\sum_{i,j}^{N} s_{ij} a_i b_j}{\sqrt{\sum\sum_{i,j}^{N} s_{ij} a_i a_j}\sqrt{\sum\sum_{i,j}^{N} s_{ij} b_i b_j}}, \tag{14}$$

where $s_{ij} = sim(f_i, f_j)$.

Obviously, if there is no similarity between features ($s_{ii} = 1$, $s_{ij} = 0$ for $i \neq j$), (14) is equivalent to the conventional formula (4).

This formula computes the cosine of a pair of vectors in time $O(N^2)$.

2.2 Simplified Formula

While the formula (14) gives mathematically correct result, in practice existing software packages for machine learning and natural language processing, such as WEKA, might not be designed to handle comparisons of the data vectors via a matrix (s_{ij}). Instead, they apply a built-in expression (4) with dot-product (1).

Our goal now will be to transform the data vectors in such a way that the cosine measure (14) be calculated via the conventional expression of the form (4). One way of achieving it is to map our data vectors a, b to a space with orthonormal basis.

Our first attempt is to use a space of dimension N^2. We will map the data vectors $a = (a_i)$, $b = (b_i)$ to a new N^2-dimensional vectors (a_{ij}), (b_{ij}) by averaging different coordinates:

$$a_{ij} = \sqrt{s_{ij}}\frac{a_i + a_j}{2}, \quad b_{ij} = \sqrt{s_{ij}}\frac{b_i + b_j}{2}, \tag{15}$$

where $s_{ij} = sim(f_i, f_j)$ are given by (8). In these new coordinates, we compare our data points using the classical expression (4), which in this case takes the form

$$soft\_cosine_2(a, b) = \frac{\sum\sum_{i,j=1}^{N} a_{ij} b_{ij}}{\sqrt{\sum\sum_{i,j=1}^{N} a_{ij}^2}\sqrt{\sum\sum_{i,j=1}^{N} b_{ij}^2}}. \tag{16}$$

This formula has a simple interpretation: we consider each pair of features as a new feature with a "weight" or "importance" being the similarity of the two features, thus the normalizing coefficients in (15).

An advantage of the formula (16) over (14) is its simplicity and the fact that it can be used with existing machine-learning tools without change, by only recalculating the data vectors.

A disadvantage is the size of the obtained vectors: N^2 instead of N, which makes it suitable only for small

feature sets. Time complexity of this formula is still $O(N^2)$.

Another disadvantage is that this simplified formula does not provide correct value for the cosine, as we will see in Section 2.4. However, our experiments show that this approximate value gives quite good results, comparable within the natural fluctuation of data to the correct formula (14). This suggests that the simplified formula (16) is worth trying in practice.

In particular, if there is no similarity between features, (16) is equivalent to the conventional formula (4).

In our experiments reported below, we used this formula as $soft\_cosine_2$.

2.3 Dimensionality Reduction

In practice, VSM often has high dimensionality N; for example, in natural language processing tasks hundreds of thousands of dimensions is common. In case of n-grams as features the dimensionality of the VSM can be especially high. In this case, forming N^2-dimensional vectors or operating with $N \times N$ matrices is impractical.

However, to use our formulas it is not necessary to operate with all N^2 elements of matrices or feed all N^2 coordinates of vectors (15) into a machine learning software package such as WEKA. Instead, it is enough to identify a small number of highly similar pairs of features, that is, only keep s_{ij} greater than some threshold t, otherwise consider $s_{ij} = 0$.

If the similarity between features is given by some list, such as a dictionary of synonyms, then automatically only a small number of $s_{ij} \neq 0$. For n-grams with the Levenshtein distance as a similarity measure (see Section 2.6), only a small number of n-grams will have any nonzero similarity.

With this, the matrix s_{ij} can be easily stored and operated upon as a sparse matrix. What is more, in our simplified formula, only a small number of additional dimensions is to be stored: those for that $s_{ij} > t$, $i \neq j$ (since $s_{ii} \equiv 1$, at least N dimensions are always stored).

While the simplified formula apparently implies a frightening N^2 dimensions of vectors, in practice one can choose to add very few data columns to the feature vectors, which makes this simple formula quite affordable in practice.

2.4 Corrected Formula, Dimensionality N^2

As we have mentioned, the expression (16) does not exactly compute cosine between two vectors. Indeed, substituting a_{ij}, b_{ij} in (16)

$$a \cdot b = \sum\sum_{i,j}^{N} a_{ij} b_{ij} \tag{17}$$

with (15) and removing parentheses, we obtain (taking into account that the similarity values are symmetric, $s_{ij} = s_{ji}$):

$$\begin{aligned}
a \cdot b &= \sum\sum_{i,j}^{N} a_{ij} b_{ij} \\
&= \frac{1}{4} \sum\sum_{i,j}^{N} s_{ij} (a_i + a_j)(b_i + b_j) \\
&= \frac{1}{2} \Big(\sum\sum_{i,j}^{N} \frac{s_{ij} + s_{ji}}{2} a_i b_i \\
&\quad + \sum\sum_{i,j}^{N} \frac{s_{ij} + s_{ji}}{2} a_i b_j \Big) \\
&= \frac{1}{2} \Big(\sum\sum_{i,j}^{N} s_{ij} a_i b_i + \sum\sum_{i,j}^{N} s_{ij} a_i b_j \Big) \\
&= \frac{1}{2} \left(\sum_{i}^{N} \left(\sum_{k}^{N} s_{ik} \right) a_i b_i + \sum\sum_{i,j}^{N} s_{ij} a_i b_j \right) \\
&= \frac{1}{2} \left(\sum\sum_{i,j}^{N} \left(\begin{cases} \sum_k^N s_{ik}, & i = j \\ 0, & i \neq j \end{cases} \right) a_i b_j \right. \\
&\quad \left. + \sum\sum_{i,j}^{N} s_{ij} a_i b_j \right) \\
&= \frac{1}{2} \sum\sum_{i,j}^{N} \left(\begin{cases} s_{ij} + \sum_k^N s_{ik}, & i = j \\ s_{ij}, & i \neq j \end{cases} \right) a_i b_j .
\end{aligned}$$

Comparing this with (14), we see that the supposed value of the dot product is half the correct one, but this is not important because due to normalization in (3) it does not affect the cosine. What *is* important is that the coefficient at $a_i b_i$ has an extra summand equal to the sum of a row (or column) in the matrix (s_{ij}), thus the value of the expression is, generally speaking, incorrect.

This is simple to repair by changing the coefficients in (15): let it be now

$$a_{ij} = \sqrt{2c_{ij}} \frac{a_i + a_j}{2}, \quad b_{ij} = \sqrt{2c_{ij}} \frac{b_i + b_j}{2} \tag{18}$$

(2 is here to compensate for $\frac{1}{2}$ in the derivation above, though it does not affect the cosine in any way). Comparison of the last obtained equation (using in it c_{ij} instead of s_{ij}) with the correct formula (14) gives the equations

$$\begin{cases} c_{ii} + \sum_{k=1}^{N} c_{ik} = s_{ii} \equiv 1, \\ c_{ij} = s_{ij}, & i \neq j, \end{cases} \tag{19}$$

which gives

$$c_{ij} = \begin{cases} \frac{1-\sum_{k\neq i} s_{ik}}{2}, & i = j, \\ s_{ij}, & i \neq j. \end{cases} \quad (20)$$

Together with (18) and (16), this formula gives a data transformation almost as simple as our $soft\_cosine_2$, but as correct as $soft\_cosine_1$. It still has a disadvantage of dimensionality N^2 and time complexity $O(N^2)$, and it is only applicable when all $c_{ii} \geq 0$ (improbable for large N).

Obviously, in terms of precision and recall all experimental results for this formula are the same as for $soft\_cosine_1$.

2.5 Formula with Dimensionality N

It is possible, however, to construct a transformation of the basis of the vector space that gives the correct result with the same dimensionality N of the vectors and with linear complexity $O(N)$ of computing the cosine of a pair of vectors.

Namely, we will transform the input data vectors to be expressed in an orthonormal basis. For this, we need to represent the basis vectors $\{e^j\}$ in some orthonormal basis of the same space. The matrix of the coordinates of these basis vectors $e^j = (e_1^j, \ldots, e_N^j)$ in the orthonormal basis is the transition matrix to re-calculate the input data points.

Denote this matrix $E = (e_i^j)$ and the matrix of similarities $S = (s_{ij})$. Since we interpret the similarities as cosines between the basis vectors and given that they have unit length, we have a non-linear system of N^2 equations on N^2 unknown values e_i^j:

$$e^i \cdot e^j = s_{ij}, \quad (21)$$

or

$$EE^T = S, \quad (22)$$

where E^T is the transpose of E. The algorithm that we describe below finds a solution of this system in the form of a triangular matrix E.

From the geometric point of view, we will implicitly construct an orthonormal basis and find the coordinates of our initial basis vectors e^i in this new basis. If we know the coordinates of the basis vectors, we can easily find the coordinates of each data point in this new orthonormal basis by multiplying the data vectors by the transition matrix (e_i^j).

We consider an orthonormal basis such that for each k, the first k vectors of this basis form an (orthonormal) basis or the space generated by the first k original basis vectors e^i.

Obviously, in such a basis matrix E of coordinates of the vectors e^i is a lower-triangular matrix:

$$\begin{aligned} e^1 &= (e_1^1,\ 0,\ 0,\ 0, \ldots, 0\ ) \\ e^2 &= (e_1^2,\ e_2^2,\ 0,\ 0, \ldots, 0\ ) \\ e^3 &= (e_1^3,\ e_2^3,\ e_3^3,\ 0, \ldots, 0\ ) \\ e^4 &= (e_1^4,\ e_2^4,\ e_3^4,\ e_4^4, \ldots, 0\ ) \\ &\ldots \\ e^N &= (e_1^N, e_2^N, e_3^N, e_4^N, \ldots, e_N^N). \end{aligned}$$

Assume that we have found the coordinates of first k vectors e^j. The next vector e^{k+1} forms the angles with the known cosines $s_{ij} = sim(f_i, f_j)$ with the first k vectors e^j. Denote $x_i = e_i^{k+1}$. All vectors e^j have unit length, thus $cosine(e^i, e^j) = e^i \cdot e^j$. Since by construction all coordinates $e_i^j = 0$ for $i > k$, we have:

$$e^j \cdot e^{k+1} = \sum_{i=1}^{k} e_i^j x_i = s_{i,k+1}, \quad j = 1, \ldots, k. \quad (23)$$

Recall that we consider the similarity values s_{ij} to be the cosines between the basis vectors.

This is a square system of linear equations. It has a solution given that the coefficients are linearly independent, which is by assumption because they are coordinates of our basis vectors; we assume that the similarity measure obeys the triangle inequality: a thing cannot be too similar to two very different things at the same time.

This system can be solved, for example, by Gaussian elimination in time $O(N^3)$.

Since the vector e^{k+1} has unit length, we have

$$\sum_{i=1}^{k+1} x_i^2 = 1, \quad (24)$$

As soon as we found the first k of $x_i = e_i^{k+1}$ from (23), we obtain

$$e_{k+1}^{k+1} = x_{k+1} = \sqrt{1 - \sum_{i=1}^{k} \left(e_i^{k+1}\right)^2}. \quad (25)$$

Finally, all the coordinates $e_i^{k+1} = 0$ for $i > k+1$. This completely defines the coordinates of the vector e^{k+1}.

The above expressions naturally give

$$e^1 = (1, 0, \ldots, 0) \quad (26)$$

for $k = 0$. Starting from this value, in N steps we can find the coordinates of all e^i, that is, the transition matrix to transform our input data to an orthonormal basis, in which the cosine measure has the usual expression (4) built into existing machine-learning programs. The complexity of the algorithm is $O(N^4)$. However, the transition matrix does not depend on the data and thus can be

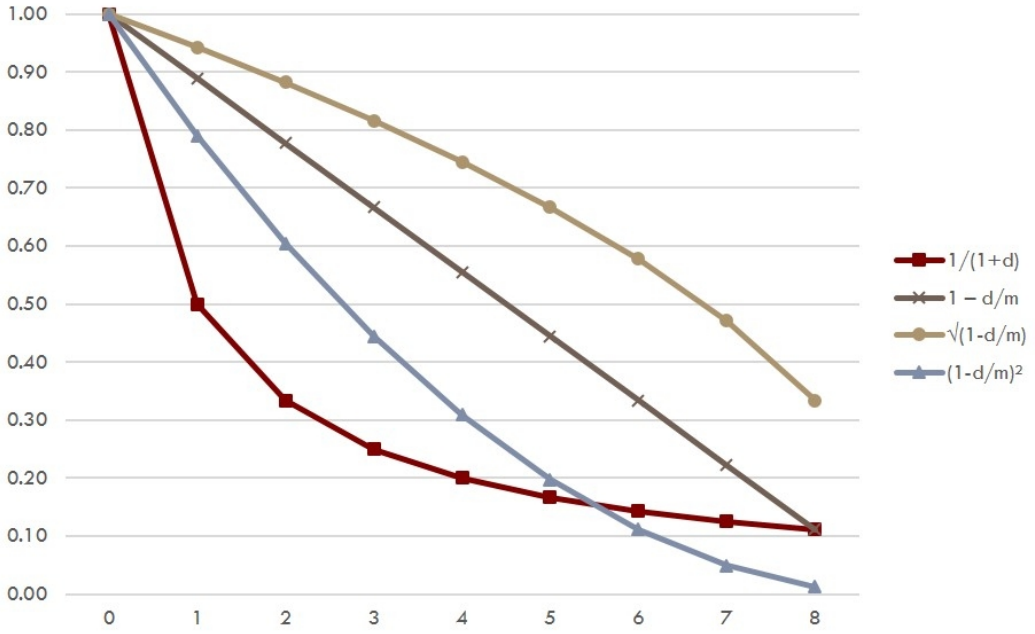

Fig. 1. Behavior of the similarity functions based on Levenshtein distance: sim as a function of d ($m = 9$).

computed once for the given feature set and feature similarities. Then this matrix is used as a pre-processing step to re-calculate the input data vectors before using any machine-learning software package.

The dimensionality of the data vectors remains the same, N, and once the vectors have been transformed, cosine is computed in time $O(N)$, so that our soft cosine measure has no effect on the complexity of the algorithms that use it.

We believe that this is the most promising way of calculating the soft cosine measure (14) for large datasets of high dimensionality, though it is somewhat more complicated in implementation.

2.6 Similarity between n-grams

We experimented with a natural language processing task using n-grams as features. Given that n-grams can share elements, it is important to take into account similarity between them.

As a measure of difference, or distance, between two n-grams f_i, f_j we used their Levenshtein distance (edit distance). We tried various ways to convert this distance into its inverse, similarity:

$$sim(f_i, f_j) = \frac{1}{1+d}, \tag{27}$$

$$sim(f_i, f_j) = 1 - \frac{d}{m}, \tag{28}$$

$$sim(f_i, f_j) = \sqrt{1 - \frac{d}{m}}, \tag{29}$$

$$sim(f_i, f_j) = \left(1 - \frac{d}{m}\right)^2, \tag{30}$$

where $d = Levenshtein\_distance(f_i, f_j)$ and m is the maximum possible Levenshtein distance for two strings of the same length as the two given ones, which is the length of the longer of the two strings.

In our experiments, the expression (27) gave slightly better results, though the choice of the best expressions still needs more research. Below we report the experimental results for similarity calculated using the expression (27).

A graphical representation of these expressions is given in Figure 1, which shows the similarity (Y axis) vs. the Levenshtein distance d (X axis). In the future we plan to try other expressions for calculation of similarity from distance.

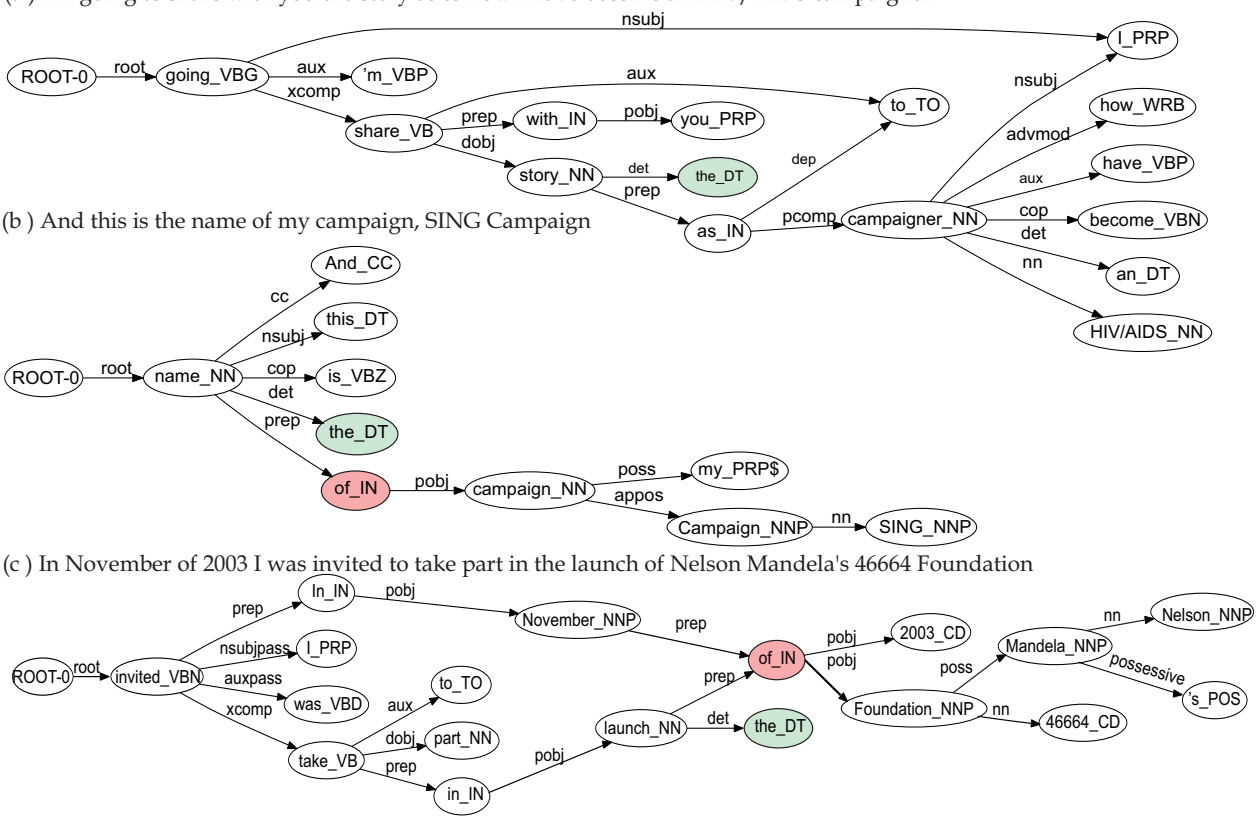

Fig. 2. Dependency trees of the first three sentences of the text using word_POS combination for the nodes and dependency labels for the edges

3 Case Study: Entrance Exams Question Answering Task

3.1 Description of the Entrance Exams Task

In this section we describe the task, where we applied the proposed soft cosine measure. The only reasons why we chose this task are: (1) we applied in it the traditional cosine measure (so we can compare it), and (2) we participated in it in 2014, thus, we had all the data at hand. Our aim was to apply the soft cosine measure with various parameters and compare its performance with the traditional cosine measure.

The entrance exam task was first proposed in 2013 as a pilot task [12] in the Question Answering for Machine Reading Evaluation (QA4MRE) lab, which has been offered at the CLEF conference[1] since 2011 [10, 11]. The entrance exam task evaluates systems in the same situation, in which high school students are evaluated for entering a university. The challenge consists in reading a small document (≈500-1,000 words) and identifying answers (from multiple choices) for a set of questions about the information that is expressed or implied in the text. The task is very difficult for an automatic system and ideally implies deep semantic analysis of the text. We proposed a methodology for its solution, built the corresponding system, and participated in the task (evaluation).

The test set 2013 based on the entrance exams task is composed of tests for reading comprehension taken from the Japanese Center Test (a nation-wide achievement

[1] Conference and Labs of the Evaluation Forum; see http://www.clef-initiative.eu/

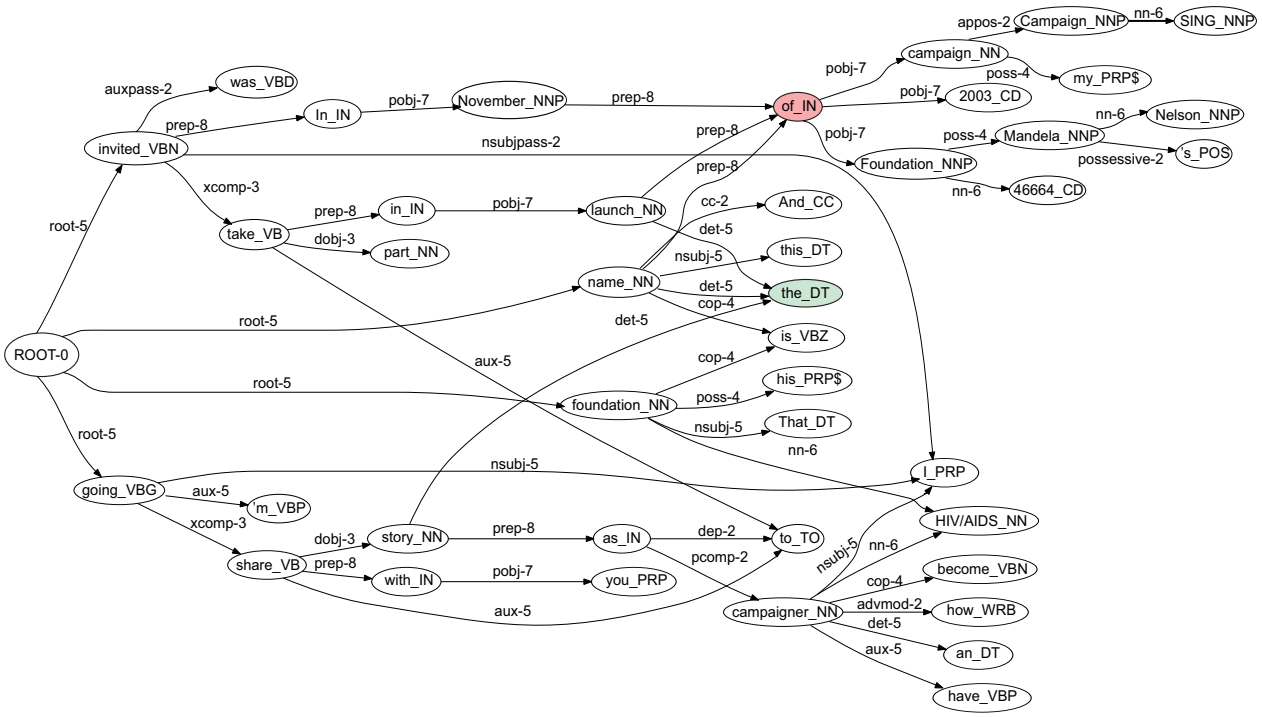

Fig. 3. Integrated Syntactic Graph of the paragraph presented in Figure 2

test for admission in Japanese universities). The data set is composed of the following elements:

— 12 test documents,

— 60 questions (5 questions for each document),

— 240 choices/options (4 for each question).

The principal measure used in the evaluation of the task is c@1, which is defined as shown in (31). This measure was defined in the QA4MRE task at CLEF 2011 with the purpose of allowing the systems to decide whether or not to answer a given question. The aim of this procedure is to reduce the amount of incorrect answers, maintaining the number of the correct ones, i.e., a system is penalized for answering incorrectly and it is better not to answer at all if not sure:

$$c@1 = \frac{1}{n}\left(n_R + n_U \frac{n_R}{n}\right), \tag{31}$$

where n_R is the number of the correctly answered questions, n_U is the number of the unanswered questions, and n is the total number of questions.

3.2 Our Method for the Entrance Exam Task

Our system formulates several candidate "answer hypotheses" as the improved versions of the original question, removing the cue words associated with the questions, such as *who*, *where*, *which*, and replacing them with one of the possible answers given in the test data. So, we have several "answer hypotheses", which are then validated in order to determine the one that best matches, i.e., has the major similarity with the document itself. In the task we applied the traditional cosine measure, which we now substitute with the soft cosine.

For performing this, the text is transformed into an Integrated Syntactic Graph (explained in Section 3.3) for both the reference document and each of the answer hypotheses. We validated each one of these answer hypotheses by comparing their similarity with the reference document. The hypothesis that obtains the highest score is the one selected as the correct answer for the given question. Previously, we used the traditional cosine measure for the task evaluation and now we tried both equations for calculation of the soft cosine measure. The results show that the soft cosine similarity performs better in most cases.

Table 1. Configurations of the graphs representation and the extracted features

	Representation schemes					Extracted features		
System	**Words**	**POS tags**	**Dependency tags**	**Stemming**	**Frequency**	**Words**	**POS tags**	**Dependency tags**
cicnlp-1	✓	✓	✓				✓	✓
cicnlp-2	✓	✓	✓	✓			✓	✓
cicnlp-3	✓	✓	✓		✓		✓	✓
cicnlp-4	✓	✓	✓	✓	✓		✓	✓
cicnlp-5	✓	✓	✓			✓	✓	✓
cicnlp-6	✓	✓	✓	✓		✓	✓	✓
cicnlp-7	✓	✓	✓		✓	✓	✓	✓
cicnlp-8	✓	✓	✓	✓	✓	✓	✓	✓

In the next sections we present a brief description of the Integrated Syntactic Graph (ISG) and the feature extraction process that we used in the task.

3.3 Construction of the Integrated Syntactic Graph

We construct the Integrated Syntactic Graph following the methodology presented in the research work [13]. The ISG can represent a sentence, a paragraph, a document or a collection of documents. Figure 2 shows the dependency trees of the first three sentences of the text. The construction of the graph starts with the first sentence of the text. We apply the dependency parser[2] and obtain the dependency tree of the first sentence. The parser generates a tree with the generic root node (ROOT), in which the rest of the sentences are attached in order to conform the integrated syntactic graph. Each node of the tree is augmented with other annotations, such as the combination of lemma (or word) and POS tags (lemma_POS).

After this, we perform similar actions for the second sentence of the text (Fig. 2b), using the dependency parser and attaching the obtained parsed tree to the ROOT. If there exists a repeated node, then, at the time of attaching the new tree, this node is not duplicated. Instead, the repeated node contracts with the existing node (i.e., in Figure 3, the node "of_IN" appears only once and all the relationships in the second and third sentences are compressed into it). In this way, we create new connections of nodes (containing the same lemmas (or words) and POS tags) from different sentences that would not have existed otherwise.

In addition, we can expand the ISG using paradigmatic semantic relations. The graph expansion results in the overlap of the higher nodes, when we compare two different graphs. The most widely used paradigmatic semantic relations are: antonymy, synonymy, inclusion of classes, part-whole, and case [1]. In order to extract semantic relations of words in a text, we used the WordNet taxonomy [9]. For instance, in Figure 3, the node "foundation_NNP" can be expanded with the synonyms: "institution_NNP" and "endowment_NNP", which are then linked to the same vertices in the graph corresponding to the original node "foundation_NNP", direction of the edges is kept.

3.4 Feature Extraction from the ISG

We use shortest path walks in the graph for feature extraction. Note that the idea of using syntactic paths is similar to extraction of syntactic n-grams [19, 21]. So, we use syntactic n-grams of various types as features. The procedure starts by selecting the root node of the graph as the initial node for the path traversal, whereas the final nodes correspond to the remaining nodes of the graph reachable from the initial node.

We used the Dijkstra algorithm [2] for finding the shortest path between the initial and the final nodes. While traversing the shortest paths, we construct the syntactic n-gram with the linguistic features found in

[2] In this work, we used the output generated by the Stanford parser: http://nlp.stanford.edu/software/lex-parser.shtml

Table 2. Results (c@1) using the Dijkstra path

	Levenshtein dist. (for characters)		Levenshtein dist. (for n-grams)		
Run	$soft\_cosine_1$	$soft\_cosine_2$	$soft\_cosine_1$	$soft\_cosine_2$	$cosine$
cicnlp-1	0.17	0.28	0.27	**0.30**	0.23
cicnlp-2	0.23	0.23	0.23	0.22	**0.27**
cicnlp-3	0.13	0.27	0.20	**0.28**	0.23
cicnlp-4	0.20	0.20	0.22	0.17	**0.23**
cicnlp-5	0.30	0.27	0.22	**0.32**	0.23
cicnlp-6	0.27	0.27	0.25	**0.28**	0.23
cicnlp-7	**0.30**	0.25	0.20	**0.30**	0.23
cicnlp-8	0.22	0.22	0.23	**0.28**	0.23

Table 3. Results (c@1) using all shortest paths

	Levenshtein dist. (for characters)		Levenshtein dist. (for n-grams)		
Run	$soft\_cosine_1$	$soft\_cosine_2$	$soft\_cosine_1$	$soft\_cosine_2$	$cosine$
cicnlp-1	0.20	**0.25**	0.22	**0.25**	0.22
cicnlp-2	0.25	**0.30**	0.20	0.27	**0.30**
cicnlp-3	0.10	0.25	0.20	**0.28**	0.22
cicnlp-4	**0.23**	0.22	**0.23**	0.18	**0.23**
cicnlp-5	**0.28**	0.27	0.17	**0.28**	0.23
cicnlp-6	**0.32**	0.27	0.25	**0.30**	0.25
cicnlp-7	0.32	0.22	0.17	0.30	0.23
cicnlp-8	0.23	0.18	0.22	0.30	0.23

the current path (including words (or lemmas), POS tags and dependency tags). For example, in Figure 3 the shortest path between the node "*ROOT-0*" and the node "*the_DT*" is the path: "ROOT-0", "name_NN" and "the_DT". So, the syntactic n-gram is: *ROOT-0 root name_NN nsubj the_DT*. Note that we include relation names (dependency tags) in syntactic n-grams, i.e, in fact, we use mixed syntactic n-grams with names of syntactic relations [20].

4 Experimental Results

We presented eight versions of our method/system (eight runs) in the competition using various sets of features. Table 1 shows the features included in the ISG and which of those features were used in the feature extraction process for each of the eight configurations.

In the system **cicnlp-1** we included for the nodes of the graph combination of words and POS tags and the dependency tags for the edges. The system **cicnlp-2** uses stems of words for the nodes. The systems **cicnlp-4** and **cicnlp-5** contain in addition the frequency of the repeated pair of the initial and the final nodes, which are counted at the moment of the graph construction. The configurations of the systems **cicnlp-5** to **cicnlp-8** differ from the first one in the type of features extracted. The systems **cicnlp-1** to **cicnlp-4** use only POS tags and dependency tags for the n-gram conformation, while the systems **cicnlp-5** to **cicnlp-8** in addition include words for forming n-grams.

In Table 2, we observe the results, when the features were extracted using the Dijkstra algorithm for obtaining the shortest paths. The values are obtained using traditional cosine measure and the two soft cosine measures with both variants of the Levenshtein distance: measured in characters and in elements of n-grams. It shows that the soft cosine similarity obtain better results in most of the experiments. There were only two systems that could not achieve better performance than the traditional cosine measure: cicnlp-2 and cicnlp-4.

Table 4. Results (c@1) using the expansion with synonyms

	Levenshtein dist. (for characters)		Levenshtein dist. (for n-grams)		
Run	$soft\_cosine_1$	$soft\_cosine_2$	$soft\_cosine_1$	$soft\_cosine_2$	$cosine$
cicnlp-1	**0.35**	0.32	0.28	**0.35**	0.28
cicnlp-2	**0.45**	0.37	0.33	0.37	0.33
cicnlp-3	0.27	0.28	0.18	**0.35**	0.25
cicnlp-4	0.30	**0.40**	0.28	0.35	0.25
cicnlp-5	**0.37**	0.30	0.30	0.33	0.22
cicnlp-6	**0.42**	0.40	0.33	0.35	0.25
cicnlp-7	**0.40**	0.30	0.37	0.33	0.23
cicnlp-8	**0.40**	0.38	0.35	0.37	0.25

Table 5. Results (c@1) using the expansion with hypernyms

	Levenshtein dist. (for characters)		Levenshtein dist. (for n-grams)		
Run	$soft\_cosine_1$	$soft\_cosine_2$	$soft\_cosine_1$	$soft\_cosine_2$	$cosine$
cicnlp-1	0.30	0.35	0.28	**0.40**	0.37
cicnlp-2	0.33	**0.40**	0.30	0.35	**0.40**
cicnlp-3	0.28	0.35	0.28	**0.37**	0.33
cicnlp-4	0.32	**0.35**	0.25	**0.35**	0.27
cicnlp-5	**0.37**	**0.37**	0.33	0.32	0.30
cicnlp-6	0.27	0.30	0.32	**0.37**	0.27
cicnlp-7	**0.38**	0.37	0.28	0.33	0.28
cicnlp-8	0.25	0.30	0.30	**0.32**	0.25

There can be several shortest paths in the graphs between the initial and the final nodes, i.e., several paths have the same length. Table 3 shows the results, when we use all shortest paths. In this set of experiments the systems built with the soft cosine measure were better (or equal) than the systems using the traditional cosine in all cases.

As we mentioned before, the graphs can be expanded with semantic paradigmatic relations. We expanded the graph with synonyms and conducted more experiments. We used the Dijkstra algorithm for obtaining the shortest paths. Table 4 presents the obtained results. The systems that use the soft cosine overcome the systems with traditional cosine.

We obtained here a very promising result: the best result of the competition was 0.42 [12], while one of our systems got 0.45. So, this system got the best performance in the competition. Let us remind that it was obtained only by using the soft cosine measure instead of the traditional cosine measure.

We also expanded the graphs used in these experiments with hypernyms. Here we also used the Dijkstra algorithm for obtaining the shortest paths. The results are shown in Table 5. The soft cosine measure is better (or equal) than the traditional cosine in all cases.

5 Conclusion and Future Work

Calculation of similarity is the most used metric in information retrieval and natural language processing. Usually, it is used with the representation based on the Vector Space Model. In this case, if we calculate the similarity directly between objects, then the cosine measure is used. Note that we can also apply machine learning algorithms that interpret the Vector Space Model as a metric for similarity.

These models are quite effective, but it turns out that they can be improved if we take into account the similarity of features in Vector Space Model. Traditional VSM considers that all features are completely different. It is not true in many tasks, for example, in natural language

processing: words and n-grams can have a certain degree of similarity.

In this paper, we have proposed the concepts of the soft similarity and the soft cosine measure, which are calculated precisely taking into account similarity of features. Essentially, we add to the Vector Space Model new features by calculation of similarity of each pair of the already existing features, i.e., we construct the matrix of similarity for each pair of features and then use it. We proposed two equations for the soft cosine measure and tried several manners for measuring similarity using the Levenshtein distance. Note that if the features are similar only to themselves, i.e., the matrix of similarity has 1s only at the diagonal, then these equations are equal to the traditional cosine equation.

We made a study of applicability of the soft cosine measure for a specific question answering task: entrance exams task at CLEF. The soft cosine measure obtained better results as compared with the traditional cosine measure in the majority of cases. In our experiments, this measure even obtained the best score for the competition (it is not the official score, though, because it was obtained after the competition).

In future, we would like to experiment with more types of similarity functions, for example, add the well-known WordNet similarity functions. It should be analyzed also which features benefit from the soft cosine measures, and which do not.

References

1. **Bejar, I., Chaffin, R., & Embretson, S.** (**1991**). *Cognitive and psychometric analysis of analogical problem solving*. Recent research in psychology. Springer-Verlag.
2. **Dijkstra, E. W.** (**1959**). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269–271.
3. **Gmez-Adorno, H., Sidorov, G., Pinto, D., & Gelbukh, A.** (**2014**). Graph-based approach to the question answering task based on entrance exams. **Cappellato, L., Ferro, N., Halvey, M., & Kraaij, W.**, editors, *Notebook for PAN at CLEF 2014. CLEF 2014. CLEF2014 Working Notes*, volume 1180 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 1395–1403.
4. **Jimenez, S., Gonzalez, F., & Gelbukh, A.** (**2010**). Text comparison using soft cardinality. **Chavez, E. & Lonardi, S.**, editors, *String Processing and Information Retrieval*, volume 6393 of *Lecture Notes in Computer Science*, Springer, pp. 297–302.
5. **Jimenez Vargas, S. & Gelbukh, A.** (**2012**). Baselines for natural language processing tasks based on soft cardinality spectra. *International Journal of Applied and Computational Mathematics*, 11(2), 180–199.
6. **Levenshtein, V. I.** (**1966**). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8), 707–710.
7. **Li, B. & Han, L.** (**2013**). Distance weighted cosine similarity measure for text classification. **Yin, H., Tang, K., Gao, Y., Klawonn, F., Lee, M., Weise, T., Li, B., & Yao, X.**, editors, *IDEAL*, volume 8206 of *Lecture Notes in Computer Science*, Springer, pp. 611–618.
8. **Mikawa, K., Ishida, T., & Goto, M.** (**2011**). A proposal of extended cosine measure for distance metric learning in text classification. *Systems, Man, and Cybernetics (SMC)*, IEEE, pp. 1741–1746.
9. **Miller, G. A.** (**1995**). WordNet: A lexical database for English. *Communications of the ACM*, 38, 39–41.
10. **Peñas, A., Hovy, E. H., Forner, P., Rodrigo, Á., Sutcliffe, R. F. E., Forascu, C., & Sporleder, C.** (**2011**). Overview of qa4mre at clef 2011: Question answering for machine reading evaluation. *CLEF (Notebook Papers/Labs/Workshop)*, pp. 1–20.
11. **Peñas, A., Hovy, E. H., Forner, P., Rodrigo, Á., Sutcliffe, R. F. E., Sporleder, C., Forascu, C., Benajiba, Y., & Osenova, P.** (**2012**). Overview of qa4mre at clef 2012: Question answering for machine reading evaluation. *CLEF (Online Working Notes/Labs/Workshop)*, pp. 1–24.
12. **Peñas, A., Miyao, Y., Forner, P., & Kando, N.** (**2013**). Overview of qa4mre 2013 entrance exams task. *CLEF (Online Working Notes/Labs/Workshop)*, pp. 1–6.
13. **Pinto, D., Gómez-Adorno, H., Ayala, D. V., & Singh, V. K.** (**2014**). A graph-based multi-level linguistic representation for document understanding. *Pattern Recognition Letters*, 41, 93–102.
14. **Poria, S., Agarwal, B., Gelbukh, A., Hussain, A., & Howard, N.** (**2014**). Dependency-based semantic parsing for concept-level text analysis. *15th International Conference on Intelligent Text Processing and Computational Linguistics, CICLing 2014, Part I*, number 8403 in Lecture Notes in Computer Science, Springer, pp. 113–127.
15. **Poria, S., Gelbukh, A., Cambria, E., Hussain, A., & Huang, G.-B.** (**2015**). EmoSenticSpace: A novel framework for affective common-sense reasoning. *Knowledge-Based Systems*.

16. **Poria, S., Gelbukh, A., Hussain, A., Howard, N., Das, D., & Bandyopadhyay, S.** (**2013**). Enhanced SenticNet with affective labels for concept-based opinion mining. *IEEE Intelligent Systems*, 28, 31–38.
17. **Salton, G.**, editor (**1988**). *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
18. **Sanchez-Perez, M., Sidorov, G., & Gelbukh, A.** (**2014**). The winning approach to text alignment for text reuse detection at pan 2014. **Cappellato, L., Ferro, N., Halvey, M., & Kraaij, W.**, editors, *Notebook for PAN at CLEF 2014. CLEF 2014. CLEF2014 Working Notes*, volume 1180 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 1004–1011.
19. **Sidorov, G.** (**2013**). Syntactic dependency based n-grams in rule based automatic English as second language grammar correction. *International Journal of Computational Linguistics and Applications*, 4(2), 169–188. Methods and Applications of Artificial and Computational Intelligence.
20. **Sidorov, G.** (**2014**). Should syntactic n-grams contain names of syntactic relations? *International Journal of Computational Linguistics and Applications*, 5(1), 139–158.
21. **Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernndez, L.** (**2014**). Syntactic n-grams as machine learning features for natural language processing. *Expert Systems with Applications*, 41(3), 853–860. Methods and Applications of Artificial and Computational Intelligence.

Grigori Sidorov received his B.Sc., M.Sc., and Ph.D. in structural and applied linguistics from the Lomonosov Moscow State University, Russia. He is currently a Research Professor of the Natural Language Processing Laboratory of the Centro de Investigación in Computación (CIC) of the Instituto Politécnico Nacional (IPN), Mexico, and the Elected President of the Mexican Society of Artificial Intelligence (SMIA). He is a Member of the Mexican Academy of Sciences and National Researcher of Mexico (SNI) at excellence level 3. He is author, co-author, or editor of more than 200 research publications in computational linguistics and natural language processing.

Alexander Gelbukh received his M.Sc. in mathematics from the Lomonosov Moscow State University, Russia, and Ph.D. in computer science from VINITI, Russia. He is currently a Research Professor and Head of the Natural Language Processing Laboratory of the Centro de Investigación in Computación (CIC) of the Instituto Politécnico Nacional (IPN), Mexico, and the President of the Mexican Society of Artificial Intelligence (SMIA). He is a Member of the Mexican Academy of Sciences and National Researcher of Mexico (SNI) at excellence level 2. He is author, co-author, or editor of more than 500 research publications in natural language processing and artificial intelligence.

Helena Gómez-Adorno graduated as the Bachelor of Information Systems Analysis from the National University of Asuncion, Paraguay. She completed the Masters in Computer Science at the Autonomous University of Puebla, Mexico. She is currently pursuing the Ph.D. in Computer Science at the Centro de Investigación in Computación (CIC) of the Instituto Politécnico Nacional (IPN) in Mexico City. Her main research interests include natural language processing, computational linguistics and information retrieval.

David Pinto obtained his Ph.D. in computer science in the area of artificial intelligence and pattern recognition from the Polytechnic University of Valencia, Spain in 2008. At present he is a full time professor at the Faculty of Computer Science of the Benemérita Universidad Autónoma de Puebla (BUAP). His areas of interest include clustering, information retrieval, crosslingual NLP tasks and computational linguistics in general.

Article received on 25/07/2014; accepted on 12/09/2014.