TEXT CLASSIFICATION USING SEMANTIC NETWORKS

Igor Kulev, Elizabeta Janevska, Milos Jovanovik, Riste Stojanov, Dimitar Trajanov
Faculty of Electrical Engineering and Information Technologies – Skopje
Ss. Cyril and Methodius University in Skopje, Republic of Macedonia

ABSTRACT

In the age of information overflow, we face with the challenge of categorizing the digital information we come across on a daily basis, in order to apply different operations and priorities to different types of information and to manage to use it in a more efficient manner. This issue introduces the challenge of automatic text classification. The problem of text classification can be defined as assigning one or more categories to a certain text, based on its contents. There are many different approaches for solving this problem: one of the solutions is the use of latent semantic analysis (LSA), statistical text analysis, etc.

This paper introduces an algorithm for text classification with the use of semantic networks. In this paper we present a method for knowledge representation needed for this type of text analysis. We also show how to create this knowledge representation and how to use it to assign one or more categories to a given text.

I. INTRODUCTION

The information stored in digital format can be represented as text, picture, video, etc. Each of these data elements uses different formatting in which the information is represented, but is also different in the knowledge type which is contained inside.

One challenging problem we commonly face is knowledge extraction [1]. It is very useful to have every data element represented by some metadata. Ideally, metadata should be created by experts in each domain, but having in mind that there is a huge amount of data in digital format (both on the Web and in our private digital data), it is not possible for people to describe every single piece of it.

Therefore, a method for metadata extraction from the data elements is needed; a method which will allow more efficient and faster searching, knowledge based indexing, summarization and short description of the data elements. It is very important to have metadata that is well organized and that forms a structure which will allow efficient information operations. Such operations can be searching, sorting, modifying, adding, deleting, etc.

Here we will present a method for knowledge extraction from text, getting the relevant information from the text using static or dynamic semantic database and using these relevant information in the process of text categorization.

II. RELATED WORK

Artificial intelligence and machine learning algorithms are related to the topic presented in this paper. More specifically, we use an online learning algorithm to create the semantic network. In machine learning, online learning is a model of induction which learns one instance at a time [2], and in our case the instance is a certain text. Metadata is represented by an RDF schema, which offers advantages over other technologies, especially because of its flexibility [3].

Another method for text classification is latent semantic indexing (LSI) [4], which will not be discussed in this paper. The main disadvantages of LSI are the storage and the efficiency. Our method does not give the full accuracy provided by LSI, but is efficient enough, provides good results, and stores the data in machine-readable structured format. Variants of the breadth-first search and dynamic programming are used in the algorithm for calculating the relevancies between different nodes in the graph. This will be discussed in details later in this paper.

III. GENERAL ARCHITECTURE

In order to get classification labels from a given text, we use a system for text classification. Our system consists of four main modules: a module for feature vector extraction, a knowledgebase (consisted of RDFS ontologies and RDF triples), a knowledge management engine and a classification engine (Fig. 1).

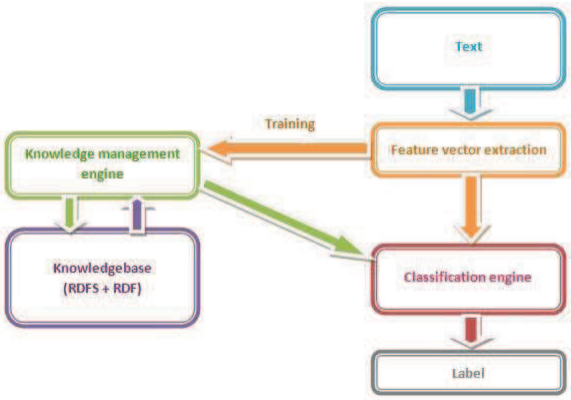

Figure 1. General architecture of the system for text classification. There are four modules: a module for feature vector extraction, a knowledgebase, a knowledge management engine and a classification engine. The system has two modes of work: training mode and classification mode.

The knowledge is stored in a semantic database (a knowledgebase) and is represented in RDF syntax. The process of the semantic database creation is completely independent of the process of text classification.

The semantic database can be created either statically or dynamically. Static creation is used when experts from the domain import the knowledge into the knowledgebase. This is the best way to create the knowledgebase, but when the domain and the data is very large, a lot of time and resources

will be spent in order to create the knowledgebase and to secure its correctness and consistency. That is the reason why the second method sometimes has to be used.

A dynamic knowledgebase creation means that the machine extracts knowledge from previously created training data sets and that knowledge is brought in the knowledgebase without any human interaction. Additionally, the human factor can participate in this process of knowledgebase creation through its modification, in order to eliminate possible mistakes of the knowledge extraction algorithm.

This method is useful when a large semantic database needs to be created and there are not enough resources for the first approach, but the risk is that there can be unforeseen mistakes in the process of knowledge extraction, which will cause errors in the text classification process later.

The process of text classification is done in the following steps: extraction of the relevant data from a given text, finding the meaning of the data using the semantic database and using the newly created knowledge in order to classify the text.

IV. KNOWLEDGE EXTRACTION APPROACH

Each text consists of words which have some meaning. The meaning of the entire text is composed of the meanings of each word which can be found in the text. Here we will use a heuristic approach to find the meaning of the text as a whole. By increasing the frequency of the word in the text, the probability that the meaning of the word is more connected to the meaning of the entire text increases.

The meaning of the text can be approximated by a sum of the meanings of the most frequently used words in the text. The vector of pairs (word, appearance frequency) will be called *feature vector of the text*. This vector can have a size which depends on the demands and possibilities of the actual implementations it will be used for. It is better for the vector to have more components, because by increasing the number of components the knowledge which is carried by the vector approximates to the knowledge which is carried by the entire text.

However, even if the vector consists of all the possible words which appear in the text, the entire meaning and knowledge inside the text cannot be covered – with this approach of knowledge representation, using a vector of most frequently used words, we do not take into account the connections between words. But, regardless of that, this method of knowledge representation is good enough to get a general picture of what is being described within the text. Before the algorithm was implemented, an experiment was performed in which a set of texts was analyzed. A frequency value was assigned to each word, and the words such as conjunctions and prepositions were removed. When reading the most frequently used words, we were able to get a general picture of what the text was about.

Meta-knowledge

There are two types of data representation in our system: the first one consists of the characteristic words which summarize the text (the feature vector), and the second type consists of the relations between these characteristic words, which are stored within the semantic database, i.e. the knowledgebase.

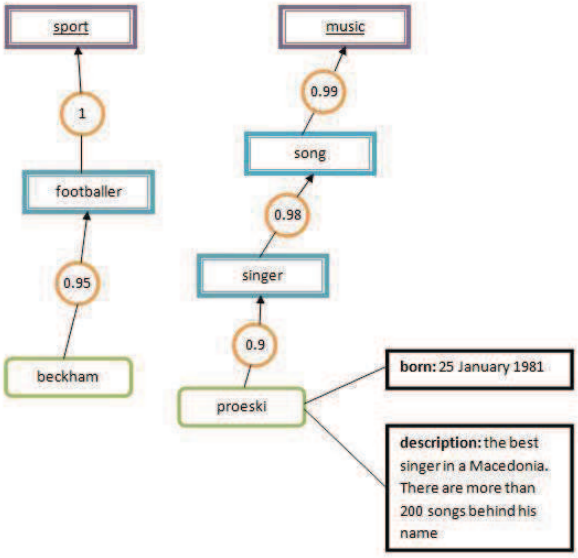

Figure 2. A simple semantic network which contains two categories: sport and music. There are some tags which are connected using one-directional arcs that are represented by their relevance. Also, the figure depicts two instances - "proeski" and "beckham", along with two properties of the instance "proeski".

V. STRUCTURE AND ORGANIZATION OF THE SEMANTIC DATABASE

The semantic database is designed and built around a few fundamental concepts: category, tag, instance, relation and property. We can see the structure of the knowledgebase on Fig. 3.

"Category" is a class from which all the categories for classification are derived. The number of categories is fixed and is often small. Experts from the domain should decide which categories will be included by the classification algorithm. Categories should be as independent from each other as possible, and should cover as many domains as possible. All categories can be seen as subclasses of the class "Category".

"Tag" is a class from which all the tags (characteristic words) are derived. Tags are the main nodes in the semantic graph of the knowledgebase. The rest of the nodes in the graph are the categories and instances.

"Instance" is a type of tag. It should be possible for one instance to belong to one or more tags. An example of an instance is "Audi", which can be seen as an instance of the tag "automobile". But it can also be seen as an instance of the tag "automobile brand". Within the knowledgebase we can state that certain instances representing people belong to tags such as "singer", "politician", "model", etc. Instances are the most direct facts within the knowledgebase.

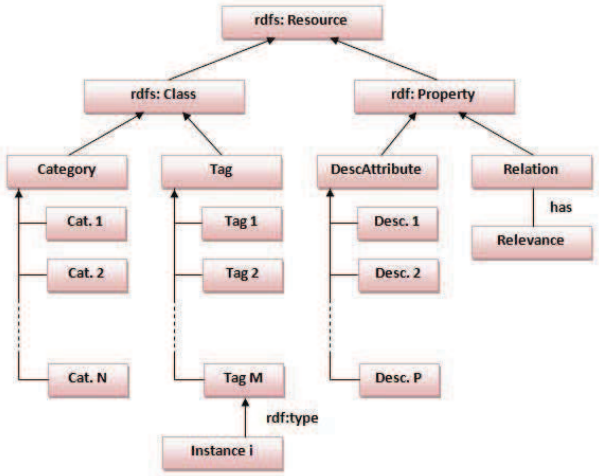

Figure 3. The knowledgebase. It consists of RDFS ontologies and RDF triples.

The most important concept is the property "Relation". All of the direct connections between two nodes in the graph are instances of the property "Relation", which contains one attribute – relevance. The value of this attribute is a real number in the interval from 0 to 1. The relevance is a heuristic measure for the strength of the connection between two given nodes. If the value is closer to 1 the connection is stronger, and if it is closer to 0 the connection is weaker. This relation is not symmetric. In order to provide symmetry, two relations between the nodes should be created.

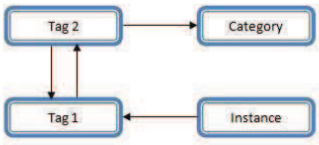

Figure 4. The three types of relevance connections are shown here: from an instance to a tag, from a tag to another tag, and from a tag to a category.

We define a membership factor of a given instance or tag A, to a given category B, as the probability that A belongs to B. If there is a relation (here and further in the text, we consider the instance of the property "Relation" as a relation) from one instance or tag A_1 to some tag A_2 with relevancy r, this means that the membership factor of A_1 to category B is equal to r multiplied by the membership factor of A_2 to B.

We use non-symmetric relevance relations because two concepts are not always equally relevant to each other in different directions, i.e. in different contexts. Let's take a look at the following example: the "car" and "vehicle" concepts are similar and every property of a vehicle is also a property of a car, so we can state that the value of relevance between a car and a vehicle is 1.0 (in that direction). However, the relevance from vehicle to car is smaller than 1.0, because there are specific properties for a car which are not related to all vehicles.

We use three types of relations: relations from tags to categories (direct belongings), relations between tags, and relations from instances to tags (Fig. 4). An instance should belong to a tag, in order to form a relation from the instance to the tag.

Every tag or instance can have description attributes. These attributes provide more information (facts) about the concept represented by the tag or instance. There is a main property "DescriptionAttribute" within the knowledgebase, and all other description attributes are its subproperties. On Fig. 2 we can see that the instance "proeski" has two description attributes: "born" and "description".

VI. Text Classification Using Semantic Database

We use a fixed number of categories in which we want to classify the given text with our system. The output of the system is a vector which consists of ordered coefficients (real numbers) in the interval (0, 1). The number of coefficients is the same with the number of categories. The coefficient in the vector is ordered so that the i^{th} coefficient corresponds to the i^{th} category. Every coefficient in the output vector is calculated independently from the others.

The function for calculating the relevance between a tag or an instance and a category returns a real number that is proportional with the membership factor of that tag or instance to the particular category. Here, different algorithms for finding the relevance can be used. All of these algorithms use the measure for relevance between nodes in the graph.

It should be mentioned that if there is no relation from node A to node B, then the relevance from A to B is equal to 0. The choice of an algorithm depends on the needs of the system and on the computational resources which are available. We use a heuristic formula for the calculation of the relevance between a category and some given node: if there are N nodes, where node 1 is an instance or a tag, nodes 2 to (N-1) are tags, and node N is category, and node i is connected to node (i+1) with relevance r[i], then the membership factor of node 1 to node N is equal to the product M (1).

$$M_{1,N} = \prod_{i=1}^{N-1} r[i] \qquad (1)$$

The graph can have the form of a tree – from each node there can be exactly one or no path to each of the categories, or there can be more than one path. First, a general algorithm used in the second case will be described – the situation when there are more than one paths and cycles.

Let N be the number of nodes in the graph and let r[i][j] be the relevance from the i^{th} to the j^{th} node. We define the algorithm recursively:

```
relevance(int k, int category) {
    if (k == category)
        return 1.0
    sum = 0.0
    counter = 0
    for each i from 0 to N {
        if there is relation from k to i then {
```

```
            sum += r[k][i]*relevance(i, category)
            counter++
        }
    }
    if counter == 0
        return 0.0
    sum /= counter
    return sum
}
```

It should be noted that the best way to implement this algorithm is iteratively. This can be accomplished by using a queue structure and additional arrays for keeping information. By using the general algorithm, all of the paths are calculated and the number of calculations is significantly large.

There is a possible optimization: an increase in the speed of calculations and a relative decrease of the correctness can be made by simplifying the recursive function:

```
relevance(int k, int category) {
    if (k == category)
        return 1.0
    for each i from 0 to N {
        if there is relation from k to i then {
            return r[k][i]*relevance(i, category)
    }
    return 0.0
}
```

If there is at most one path between any two nodes, then both of the recursive functions provide the same results.

First, the characteristic words should be extracted from a given text. For each tag or instance we calculate its relevance to all of the categories from the knowledgebase. That is how the vectors of relevance are calculated for all of the tags and instances. The feature vector of the text and the vectors of relevance for each word that appears in the feature vector are used for normalization and the final classification of the entire text within a category.

One of the possible formulas would be a sum of the relevance vectors of all the words, after which some normalization could be performed.

Implementation

This method for text classification was implemented in a link sharing web system. The system allows registered users to publish links. The web location at which the links point to are checked by the system, the feature vector is extracted from the text on the web page, the relevant information connected to the tags is shown and the text is categorized within a category from the knowledgebase of the system.

The focus in this web application was put on links which contain text in Macedonian language. It should be noted that frequently used words like conjunctions and preposition are ignored and not taken into account, i.e. they are not placed as components of the feature vector for the text.

VII. Dynamic relevance creation

Here we present one possible machine learning algorithm which can be used with the system. In the first phase, we extract the characteristic words (the most frequently used words) form each of the texts in the training data set. In this phase, the words that do not bring real information, such as conjunctions, pronouns, prepositions, etc., are ignored. Also, we use the Porter Stemming algorithm [5] to find the principal form of the words. After that, we assemble the text feature vector.

Not all of the characteristic words used here exist in the knowledgebase. The ones that already exist within the knowledgebase will not be processed by the learning algorithm in the phase where we add new tags and new connections. If a word does not exist in the knowledgebase, it is added as a new tag. The problem here is to determine where exactly the word will be placed in the semantic network – with which words will it be connected and what will the value of the relation be. Here we use online learning, which we defined earlier in the text.

At this stage, before the new text is processed, we can use the connections of the tags in the semantic network. The heuristic that the characteristic words in the text have similar meaning are also used (we assume that the text belongs to one domain).

In order to add a new tag in the network, there should exist at least one characteristic word in the knowledgebase. Better results can be obtained if the number of characteristic words that already exist in the knowledgebase is larger. When the threshold is exceeded, the new word is added within the knowledgebase. This is the first filter used for selection of the texts which can be processed by the learning algorithm. When this condition is satisfied, the characteristic words are separated into two groups: the first group of words exists in the knowledgebase, and the other group of words should be added in the knowledgebase.

The knowledge management engine (KME) which operates with the semantic network contains a method for tag and category relevance calculation. Let the set of characteristic words in the text which exist in the semantic network be X, and the set of words that do not exist in the knowledgebase is represented with Y. Let the number of words in X be N, and the number of words in Y be M. A similarity matrix of size N*M is created, where the element with index (i, j) is the similarity between the word i from the set X and the word j from the set Y.

In our implementation we are using the Normal Google Distance [6] method for word similarity calculation. There are other possible variants that can be used, but we are using this method as an extension of our previous work [7].

Here we have defined a similarity threshold for adding new words into the knowledgebase. If the new word is similar enough to some subset of characteristic words which exist in the knowledgebase, new relations are created from the new word to all of the other words for which the similarity threshold is exceeded. The relevance of the relation is proportional to the similarity – more similar words have larger relevance between them.

After the new words are added, the relevancies between each pair of characteristic words which previously existed in the knowledgebase are changed. If the two words appear in the text more frequently, the relevance between those words in the semantic network is increased by the value of k*similarity, where k is a learning speed coefficient. It should be noted that the relevance should not exceed 1.0. Therefore, normalization of the values is performed.

This is how the semantic database is automatically expanded. Attention should be paid on a couple of things: first, each of the texts should belong in a certain domain. The biggest problem which appears is the (a)symmetry of the relations. We treat the semantic network as a directed graph, and using the technique explained above we cannot explicitly find the direction of the relevance. It is simpler to implement the scenario when the relations are symmetric – the relevance from X to Y and the relevance from Y to X are equal. Other methods should be considered for asymmetric relations, as future work.

One of the problems with this approach in dynamic creation of relevance is that we cannot distinguish instances from tags. Instances have their advantages because they define the most direct facts, but it is difficult to label some words as instances without human interaction.

VIII. Conclusion and Future Work

In this paper, we describe a method for extraction of relevant information from a given text and we show how classification can be performed with the use of the semantic network which contains the information. We also describe the structure and the organization of the semantic database, i.e. the knowledgebase that is used for finding the relevance between tags.

The approach and the algorithm are implemented in a web application used for link sharing. The application gives satisfactory accuracy and performance (static creation of relevance). However, in order to obtain more relevant results from the use of this approach, we need to test the system with the use of larger amounts of data.

We also plan to analyze the system for extreme cases, such as cycles, larger number of relations, larger depth in the inference process, etc. We believe that better results can be obtained when the texts are on same subject, because the knowledgebase will contain a lot of details on the entities from the domain.

References

[1] Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H. and Shadbolt, N. R., *Automatic Ontology-Based Knowledge Extraction from Web Documents*, IEEE Intelligent Systems, 18 (1). pp. 14-21, 2003

[2] Ethem Alpaydın, *Introduction to Machine Learning, Second Edition.* The MIT Press, 2010, page 241

[3] Hans-Jörg Happel and Stefan Seedorf, *Applications of Ontologies in Software Engineering*, 2006

[4] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, *Indexing by latent semantic analysis*, Journal of the American Society for Information Science, Volume 41, Issue 6, pages 391–407, September 1990

[5] M.F.Porter, *An algorithm for suffix stripping,* Program no. 3, pp 130-137

[6] Rudi L. Cilibrasi and Paul M.B. Vitanyi*, The Google Similarity Distance*, IEEE transactions on knowledge and data engineering, vol. 19, no.3, March 2007

[7] Riste S., Danco D., *Evaluation of the Normal Google Similarity Distance method for semantic word categorization*, ICT Innovations 2009, Ohrid, September 2009