Serdica J. Computing **11** (2017), No 2, 183–198

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A WEB APPLICATION FOR TEXT DOCUMENT CLASSIFICATION BASED ON k-NEAREST NEIGHBOR ALGORITHM*

Adelina Aleksieva-Petrova, Emilyan Minkov, Milen Petrov

ABSTRACT. The paper gives insight on how the text document categorization problem can be solved and implemented in a software product. On that score, it specifies how input data are provided, processed and transformed into output data. The goal of the paper is not only to suggest a simple theoretical solution to the text document categorization problem but to provide a real-life implementation as part of a software system.

1. Introduction. Classification can be defined as the "systematic arrangement in groups or categories according to established criteria" [3]. When texts become the subject of classification the process can be referred to as text classification or text categorization. Yang and Joachims describe it as "the task of assigning predefined categories to free-text documents" [11].

ACM Computing Classication System (1998): H.3.3, H.3.5, I.7.5.

Key words: clustering, document analysis, web-based services.

*The research presented in this paper was partially supported by the project FNI-SU-2017/80 10-128 (St. Kliment Ohridski University of Sofia, Bulgaria) *Secure and re-usable software architectures for Technology-enhanced learning.*

Text classification is a common task and can be required in various situations. Examples include determining the type of a news article, e. g., politics, sports, culture, etc.; filtering spam in e-mails; managing archives containing different types of documents; determining the most appropriate section for a book in a library; determining the language when converting text to speech; organizing fiction texts by genre; and others. These examples prove that the need of text document classification may occur in completely different situations and may involve different types of people. In addition, text categorization may concern diverse text types, each of which may necessitate a specific processing approach.

There are classifiers which are specifically designed to work with short texts such as instant messages, blog and news comments as testing instances. Such texts are characterized by small length of a couple of hundred characters, frequent use of non-standard terms and misspellings—cf. [4]. These must be properly handled in order to deliver satisfactory results. In contrast to short texts, scientific or literature texts may require heavier focus on appropriate feature selection, since they are larger in size and are semantically quite different. One can draw the conclusion that text classification is a complex task which requires that many aspects be taken into account. It is also practically impossible to create a solution which guarantees optimal results for all cases.

The object of this paper is to propose a method for document categorization and describe the development of a web application which categorizes text documents according to user-specified categories. The document categorization is achieved with an adaptation of the k-nearest neighbor algorithm, whereby the documents are the test samples, the categories are the classes and the keywords are the training set. The purpose of the application is to offer a means of grouping large amounts of data into separate semantic units which would enable their easier further processing and use. Each category is specified via a set of keywords. Each keyword is additionally supplied with a weight which determines how strongly it categorizes its respective category.

2. Text categorization. Nowadays, several theoretical models for performing automatic text classification exist. These are based on different

techniques and algorithms: decision trees (cf. [2]), support vector machines (cf. [8]), naïve Bayes classifiers (cf. [4, 6]), etc. The solution proposed in this paper uses the popular k-nearest neighbor algorithm. This particular mechanism has been employed before to provide a means of performing text categorization (cf. [9]), thus proving that k-nearest neighbors can be successfully implemented to fulfil this type of linguistic task.

2.1. Classification algorithm. The core task of the application is to classify each item in a set of text documents in a user-defined category. This necessitates the use of a specific algorithm. For this purpose, the k-nearest neighbor classification was selected, which "finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood" [10]. This choice is based on the fact that k-nearest neighbors is easily comprehensible and performs effectively. Thirumuruganathan [7] states that it is "very simple to understand but works incredibly well in practice" and "is surprisingly versatile".

Going back to the definition from the previous paragraph, the documents to classify are the test objects and the user-specified categories form the training set. An accurate classification presupposes that each of them is transformed into a suitable representation in a feature space.

The choice of a proper strategy to achieve this must take into account several characteristics of the classification process itself and the expected results. First, each document should be categorized independently from the other documents which are part of the same classification process. Unlike a task such as ranking documents, classification does not require that the individual test items be taken into consideration in the context of the remaining items. The only thing which is shared by each document is the training set used to determine the category to which it should be assigned. Second, it is not mandatory that each category should have a document assigned to it at the end of the classification, which in turn discloses the possibility that all items end up in the same category. Although this may not be a desired result in all cases, it does provide a good idea about the semantics of the document, bearing in mind that the user may not be acquainted with its contents. If the document set in question must definitely result in multiple

categories having at least one associated item, then this knowledge can be used to set the individual categories more precisely. A new classification run with those is bound to lead to more satisfactory results. Thirdly, it is essential that the means of representing the documents guarantee a balance between accuracy and performance. In other words, the document processing should be carried out via a technique which is quick to perform and does not use up too many resources, whilst ensuring a sufficient and accurate representation of the specified document.

Considering the specificities, the main approach selected for performing a document representation is term frequency—the number of times each token appears in a specific text document. On one hand, this type of weighting is, in comparison to schemes such as the more complex tf-idf weighting, completely independent on the remaining documents which are being classified. On the other hand, term frequency can be easily and automatically obtained during a single iteration over the document set, during which each of them is read and analyzed.

In regard to optimization of the term frequency approach, several adjustments are applied. When using term frequency, one must contemplate what to consider a token. This task is language-specific and mainly dependent on the use of punctuation and other non-alphanumeric symbols in texts. In the framework of the proposed application, only English is taken into account due to the fact that the current implementation of the classifier application works with English texts. In this context, words are considered as tokens, with words defined as the character sequences between whitespaces. Punctuation is not considered part of the word when it is adjacent to a whitespace character, but is when it is surrounded by alphanumeric characters.

The application of the term frequency approach in its current form requires that each of the documents to be classified is split into separate words. This is achieved by reading each document and performing a split operation and word count whilst keeping count of all the words. At the end of this process the following parameters are known:

- total number of words in the document;
- number of occurrences of each word in the document;

- word with the most occurrences in the document;
- number of occurrences of the most common word.

These values are further used for transforming each document into its classification representation.

The actual representation of each document is an n-dimensional vector where n is the number of set keywords for a specific category. Each keyword can be thought of as a separate feature and constitutes a dimension in this n-dimensional space. Both keywords and documents are represented as points in this space where the vector of each item holds the coordinate for each axis. Keywords' vectors are unit vectors which have a value of 1 for the axis which is represented by the keyword itself and a value of 0 along the other axes.

To utilize the set of training points, each document is represented as a vector in the same n-dimensional space. This is achieved by doing the following: For each keyword in the category a check is performed to see how many times this word occurs in the text document. The number of occurrences are then divided by the most common word in the document, which results in a value in the range [0; 1]. The most common word in the text would hence have a value of 1, and a word which does not appear in the text, a value of 0. The calculated value determines the vector coordinate of the document along the axis specified by the current keyword.

2.2. Category keywords. In order to provide a more accurate description of each category, each keyword is supplied with a specific weight by the user. This weight is used during the document vector calculation to increase to a certain extent the importance of a particular word in the document. The value is determined with the help of a function and the user-specified keyword weight. The current implementation of the classifier application uses common multiplication as a weighting function—the number of occurrences of the keyword in the text is multiplied by the keyword weight. In other words, the keyword weight is used to create an effect, as if the word were n times more common and thus specified the text better.

The current implementation of the classifier application uses a set of standard stop words (such as *a*, *an*, *the*, *in*, *on*, *of*, *I*, *you*, etc.) to determine which words should be ignored when estimating the document stats which are

relevant for the categorization process. These are taken from Ranks NL [5] and represent the most common words in English which are considered stop words. They are stored as a constant and put into action during the classification process. It is possible, however, that the documents to be classified are more specific and require special attention to certain words which are normally considered stop words. In particular, it may be desired to ignore specific words which may cause a more inaccurate document vector representation, hence an inaccurate categorization. Considering this, the user has the option of extending the stop word list by entering additional words. That means that specific non-default words will also be ignored when performing the classification.

After estimating the keyword and document vectors, both the documents and the category keywords can be represented as points in space. This situation is a standard starting position for performing a *k*-nearest neighbors classification. For each test sample, i. e., each document the distance to each training set point is calculated. Euclidean distance, defined by

$$S(d,\ k) = \sqrt{\left(d_1 - k_1\right)^2 + \left(d_2 - k_2\right)^2 + \ldots + \left(d_n - k_n\right)^2}, \qquad (1)$$

is considered a good distance measuring technique, since it corresponds to the geometric notion of distance and is easy to grasp, visualize and calculate. The rooting part of the formula can be skipped to reduce the computational effort, since concrete values are not of importance to the algorithm, but are used solely as a means of comparison to allow determining the nearest neighbors.

Since the number of keywords in a category determine what the vector representation of a document is, each document would have a different vector for each individual category with a different set of training points. Due to the fact that keyword and document vectors have scaled values for their coordinates, this would not cause any undesired deviation when determining the nearest neighbors from different categories. In the end, it is known for each document which are its *k*-nearest neighbors. These neighbors are composed of the individual keyword points of possibly different categories. The category whose keywords are the most common in the neighbors set, is the one to which the document is assigned. If two or more categories have the same number of keywords among the *k*-nearest neighbors, then the distances are also taken into

consideration. The category for which the sum of its keywords that are countable neighbors is the smallest is the one which gets chosen finally.

An important question to consider is how to choose an appropriate value for k or in other words—how many neighbors to contemplate when determining the appropriate category for a document. Since the training set consists of points represented by the category keywords it is logical to derive an appropriate value for k from the user-specified keywords. k is shared among the individual categories, so the keywords of each of them should be taken into account. Due to k being a quantitative measure and the fact that keywords are assumed to be of equal importance (k should not be influenced by their weights) a function based on the number of keywords in each category provides a suitable solution. A function meeting this requirement is the number of keywords of the category which has the fewest keywords set. This suggestion guarantees that each category would have a chance to have a document assigned to it. This is assumed as an initial goal, otherwise it would be pointless to have such a category set in the first place. That would not be the case if the category with the most keywords or the average number of category keywords was used.

3. Practical solution of the problem.

3.1. Format of application input data. As with any software system, input data is supplied, then processed and returned with the goal of providing a basis for new information which can be further transformed into knowledge. In order for this flow to work, the format of all input and output data should be explicitly specified and known to both the user and the system. The classifier application requires the following pieces of input data: text documents to classify, category names, category keywords with weights, specific stop words (optional), max keyword weight (optional) and some additional settings. Output data consists of information about the category to which each text document from the input was assigned.

The documents to classify are the main component of the input data. The current implementation of the application restricts them to PDF format. The choice for this particular format is conditioned by the fact that PDF documents are the most used type over the web. A statistic published by Duff

Johnson [1] reveals that in spite of the slight decrease in the last years PDF remains with 71.7% the most used format on the web. Furthermore, PDF documents can be easily created and read, they are lightweight and many tools for their processing exist. Due to the desired extensibility, kept in mind during the development phase, the application is designed in a way which allows support for other formats to be easily added in the future when desired.

Categories with their respective keywords and weights should be provided, so that classification classes are known, as the current implementation of the classifier application does not define default categories. The weighting provides a means of determining how strongly a specific word is taken into account during the categorization process. The default settings specify that weights should be non-negative integers. A weight of 0 means that the specified word should be completely ignored during the classification, and a larger weight, that it should be considered with high priority. The right endpoint can be explicitly provided as a piece of input data. For example, setting a value of 20 would provide better granularity when specifying the importance of different keywords, in contrast to a value of 5. For common use cases and datasets, a default weight range of [0; 10] should provide satisfactory results. Words in the text which are not explicitly specified by a keyword are considered to have a standard weight of 1. Since categories and keywords are small pieces of data, e. g., strings or numbers, they can be easily supplied via the application's user interface. Keywords are limited to character sequences which contain no whitespace characters. Category names, on the other hand, are allowed to consist of one or several words. Category and keyword names are treated as case-insensitive.

The last obligatory item in the input data list are additional settings which comprise different available adjustments to the business logic of the application. These include locations of different files on the server, database properties, input data for the classification algorithm, etc. They are not intended to be exposed to the user but can be modified by the developer in order to optimize the quality and performance of the application and manage ingoing and outgoing data. Due to their nature, such settings are stored in a configuration file on the server where they can be easily accessed and extended by the application.

3.2. Format of application output data. The output data includes the result of the classification process. Likewise, it requires presenting in an appropriate format, so that it is as useful as possible for further use. The category association information by itself is helpful but not sufficient for the desired result. For instance, let's consider a case where 100 documents should be classified. Let the object of the classification be the division of the document set in several groups, so that each group can then be handed out to a separate person for further reading or research. When only the appropriate category is returned as output data the user will have an idea about the contents of the documents but due to the relatively large amount of documents an extra time and effort may be required to actually execute the document grouping and have them distributed to the other people. A better way to expose the output is to have the documents themselves as part of the output and have them grouped before returning them to the user. This can be achieved easily by having a separate folder for each category and putting each document in the appropriate folder according to the classification result. It is, however, convenient to have these folders combined in one chunk. This can be accomplished by putting all folders in an archive.

This will allow having the documents sorted in one convenient location and will, likewise, enable that classifications from different sessions are kept apart and be distinguished from one another. The core of the application lies in its backend which is responsible for the document categorization once all required input data is supplied. This involves the execution of a classification algorithm and the preparation of the results for output.

3.3. Application workflow. Fig. 1 presents the basic workflow of the classifier application. Since it is a web application it is launched by making an HTTP request. Once the request is processed, the graphic user interface (GUI) is loaded and can be accessed by the user to specify which documents to classify. Once selected, the documents are immediately uploaded to the server where they become available for processing by the application. Categories with their respective keywords and weights are also set through the GUI.

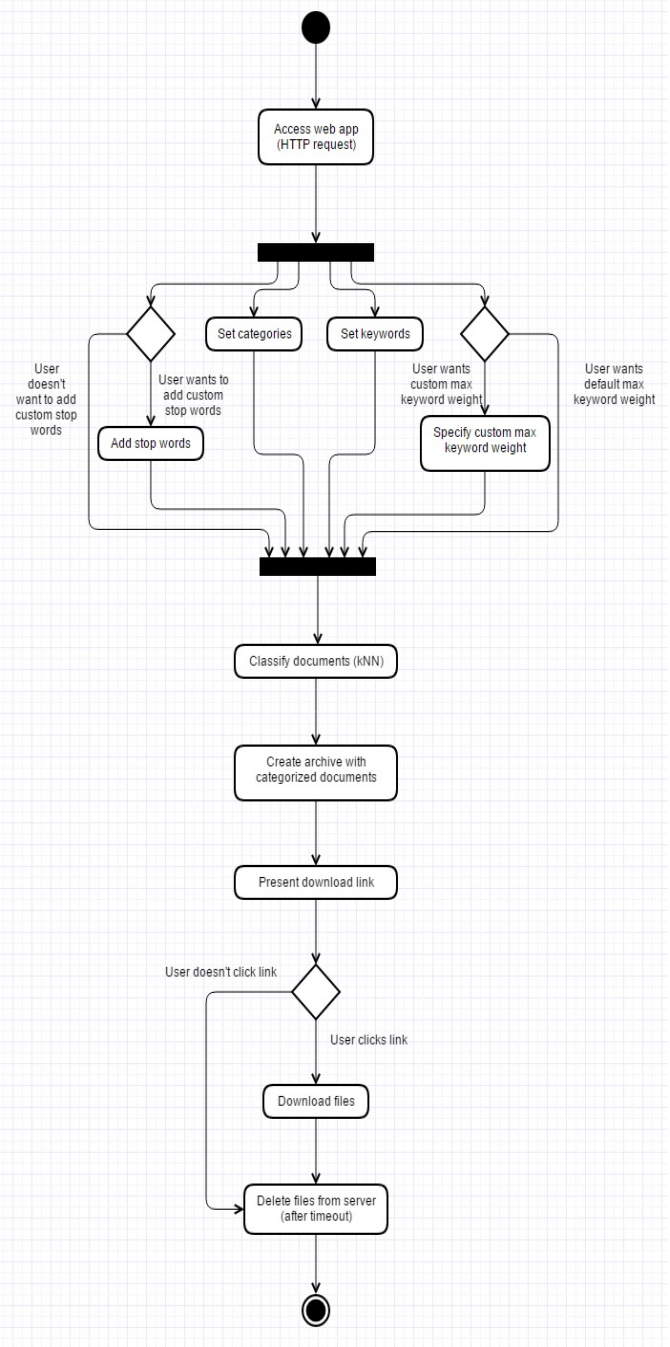

Fig. 1. Basic workflow of the classifier application

During this stage the user is free to specify a custom keyword maximum weight and additional stop words in addition to the default ones, in accordance with the considerations outlined earlier in this section. After the documents and categories are set, all input data is collected and the processing is transferred to the backend of the application where the classification itself takes place. The categorization is carried out along the lines of the algorithm described above. The result of the execution of the algorithm is information about each document's assignment to a category. For each category which has at least one document assigned to it a folder is created. All files are moved to the folder of their respective category. Once all documents are in their right location all category folders and their contents are added to an archive. A link to the archive is presented to the user. Once clicked the archive is downloaded to the user's machine and the user has possession over the classified documents.

Finally, the category and keywords data is persisted to the database for future reference. No history of documents is kept as it is considered unnecessary for the current version of the application. This is subject to change, if in the future a means of optimizing the application's performance based on the documents themselves is developed. Currently, however, the documents are deleted from the server after a predefined amount of time and the download link expires. The session timeout is set in the additional settings file and has a default value of 30 minutes, which can be changed if desired. A user therefore has 30 minutes to accomplish the data input and to collect the categorized documents. If this does not happen in time, the progress is lost and the process has to be restarted.

4. Summary and conclusions. The classifier web application which forms the primary focus of this paper employs a relatively simple scheme for categorizing documents, based on the k-nearest neighbors classification algorithm. It does, however, deliver satisfactory results and is a useful tool in many situations where large amounts of data have to be processed. It is designed in a way which gives the user complete freedom for setting the categories to be used. It is also aimed to be user-friendly, i. e., assist the user as much as possible and be pleasant to work with. Since it is implemented in the form of a web application, it can be accessed easily via a browser and does

not require any installation or updates. This makes it desirable for a larger group of users, thus achieving a high degree of practicability.

During the development phase multiple test runs were executed to ensure that the algorithm performs appropriately. One of those sample runs had the following input parameters:

- ten documents to be categorized, including the following: short personal resume, music tutorial on chords, lecture in information retrieval, rule book of curling, rule book of golf, lyrics of a rock song, router manual, story by Oscar Wilde, clarification document for a university assignment, electronic ticket for a plane and a bus;
- four categories with their respective weighted keywords:
    - Music: *music* = 10; *song* = 8; *guitar* = 8; *piano* = 8; *pop* = 9; *rock* = 9; *hip-hop* = 9; *artist* = 6; *album* = 5; *sound* = 4; *noise* = 3; *loud* = 5; *fun* = 2; *culture* = 4; *theory* = 1; *singer* = 7; *love* = 5
    - Culture: *music* = 8; *art* = 9; *sports* = 4; *ritual* = 3; *history* = 8; *sight* = 5; *museum* = 7; *gallery* = 7; *nationality* = 3; *people* = 7; *mentality* = 4; *language* = 7
    - Sports: *sport* = 10; *sports* = 10; *football* = 9; *tennis* = 9; *basketball* = 9; *volleyball* = 9; *athlete* = 9; *fitness* = 5; *healthy* = 3; *watch* = 2; *television* = 4; *politics* = 2; *team* = 3; *ball* = 5; *coach* = 6; *player* = 6; *money* = 2; *fame* = 1
    - IT: *programming* = 9; *IT* = 10; *language* = 7; *software* = 9; *hardware* = 9; *computer* = 9; *innovation* = 4; *mobile* = 6; *device* = 6; *company* = 3; *money* = 3; *business* = 4; *engineer* = 5; *work* = 1; *project* = 2; *network* = 6; *code* = 7
- no additional stop words specified;
- default function for determining k.

The documents ranged in size from 1 page to 215 pages and had a total size of 11.2 MB. The two rule books were assigned to the Sports category, which was anticipated since they concern particular sports. The router manual and the information retrieval lecture were assigned to the IT category, which

was likewise expected. This category also contained the resume due to the mentions of information technologies in it and the clarification document due to its reference to a programming-based task. The Music category had a single member which was the chord tutorial. The lyrics document was not assigned to it but to the Culture category instead. The application cannot determine the type of a text solely by its contents, hence the song text was considered more appropriate for another category, in contrast to the Music category which one might expect. Two of the documents – the ticket and the literature work – had no matching keywords for any category, so they were assigned to the IT category, which happened to be the first one in the category list specified by the user. The results of the classification, as well as the preparation of the output, took approximately 5 seconds.

Several conclusions can be drawn from the test run described in the foregoing paragraph. First, providing a sufficient number of keywords for each category and using a proper technique for determining the classification algorithm parameter k is a key premise for an accurate categorization. Second, the contents of a text document are the single factor which is currently used to determine its semantics. This may cause unforeseen results in certain situations. An appropriate way to handle this scenario is to extend the classification algorithm by having it inspect the metadata and description of a document, if present, and incorporate them in the estimation of the document vector. These two additional data types provide essential information for the document and their exploitation may prove crucial for a more adequate classification. Third, documents which do not match any of the keywords may cause inaccurate associations and thus noise in a certain category. To avoid such undesired effects, a too wide spectrum of documents should not be used in the same run. Since the user may be completely unaware of the contents of the documents, this may not always be an option. In such cases providing a greater number of categories, described by many keywords, should lead to fewer erroneous classifications. Fourth, the execution time can be considered satisfactory. Nevertheless, techniques for its reduction can be implemented in the future, thus contributing to an overall improved performance of the application.

A wider range of tests would help drawing further important conclusions. Testing is further required for unveiling any weak points or areas where the classifier application can be optimized. This includes runs with different types of documents, different number of documents, documents of different size, different number of categories, different strategy for choosing k, different weighting functions, different edge cases, etc. An iterative process, consisting of a test creation phase and a subsequent phase where new features and improvements are added, may prove beneficial for better usability. The design of the application offers a good basis for adopting such an approach for future updates.

Acknowledgements. The research presented in this paper was partially supported by the FNI-SU-2017/80-10-128 project (St. Kliment Ohridski University of Sofia, Bulgaria) *Secure and re-usable software architectures for technology-enhanced learning.*

REFERENCES

[1] Duff Johnson Strategy & Communications. The 8 most popular document formats on the Web in 2015. `http://duff-johnson.com/2015/02/12/the-8-most-popular-document-formats-on-the-web-in-2015/`, 17 February 2018.

[2] Lewis D. D., M. Ringuette. A Comparison of Two Learning Algorithms for Text Categorization. In: Third Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, 11–13 April 1994. 81–93.

[3] Merriam-Webster Online. Dictionary, s. v. *classification.* `https://www.merriam-webster.com/dictionary/classification`, 18 February 2018.

[4] Nigam K., A. K. McCallum, S. Thrun, T. Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, **39** (2000), 103–134.

[5] Ranks.nl: Default English Stopwords list. `http://www.ranks.nl/stopwords`, 18 February 2018.

[6] Song G., Y. Ye, X. Du, X. Huang, S. Bie. Short Text Classification: A Survey. *Journal of Multimedia*, **9** (2014), No 5, 635–643.

[7] Thirumuruganathan S. A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm. 15 May 2010. `https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/`, 18 February 2018.

[8] Tong S., D. Koller. Support Vector Machine Active Learning with Applications to Text Classification. *Journal of Machine Learning Research*, **2** (2001), 45–66.

[9] Trstenjak B., S. Mikac, D. Donko. KNN with TF-IDF based Framework for Text Categorization. *Procedia Engineering*, **69** (2014), 1356–1364.

[10] Wu X., V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, **14** (2008), No 1, 1–37. doi: 10.1007/s10115-007-0114-2

[11] Yang Y., T. Joachims. Text categorization. *Scholarpedia* 3(5):4242, 2008. `http://www.scholarpedia.org/article/Text_categorization`, 18 February 2018. doi:10.4249/scholarpedia.4242

Adelina Aleksieva-Petrova
Computer Systems Department
Faculty of Computer Systems and Technologies
Technical University of Sofia
8, St. Kliment Ohridski Blvd
1000 Sofia, Bulgaria
e-mail: aaleksieva@tu-sofia.bg

Emilyan Minkov
Faculty of German Engineering Education and Industrial Management
Technical University of Sofia
8, St. Kliment Ohridski Blvd
1000 Sofia, Bulgaria
e-mail: emilyan.minkov@fdiba.tu-sofia.bg

Milen Petrov
Department of Software Engineering
Faculty of Mathematics and Informatics
St. Kliment Ohridski University of Sofia
5, J. Baurchier Blvd
1164 Sofia, Bulgaria
e-mail: milenp@fmi.uni-sofia.bg

Received September 11, 2017
Final Accepted November 27, 2017